J.N. Oliveira

MÉTODOS FORMAIS DE PROGRAMAÇÃO

Departamento de Informática
Universidade do Minho
1998

Índice

Lista de Figuras

Lista de Exercícios

Parte I

Especificação Funcional

0

Capítulo 1

Fundamentos de Especificação Algébrica

1.1 Introdução

A vida real está povoada de "coisas" a que chamamos, em linguagem comum, *objectos*. Alguns objectos, estruturalmente mais elaborados, podem merecer o nome (mais "ambicioso") de *sistemas*. A nossa interacção com as coisas pressupõe alguma forma de "linguagem" de comunicação com elas. Tal linguagem deverá, no mínimo, ser capaz de descrever a *estrutura* das coisas de que desejamos falar. Tecnicamente, daremos o nome de *sintaxe* a essas estruturas.

Por outro lado, é preciso ter uma ideia do que as coisas, assim descritas, significam para nós. A esta segunda componente "epistomológica" da nossa interacção com o mundo real chamaremos *semântica* (das coisas). Temos então:

$$\text{Objecto}\left\{\begin{array}{l}\text{Sintaxe}\\ \text{Semântica}\end{array}\right.$$

Por exemplo, a Figura 1.1 descreve a estrutura sintática da frase

$$f = \textit{“Maria gosta de quem gosta dela”}$$

A semântica de frases como esta, tirada da nossa linguagem *natural*, pode ser bastante difícil de descrever. Mas podemos tentá-lo, no que diz respeito à frase f: a sua semântica refere-se a uma relação binária *gosta* definível entre pessoas,

$$gosta \subseteq Pessoas \times Pessoas$$

a que f acrescenta o seguinte facto, ou propriedade:

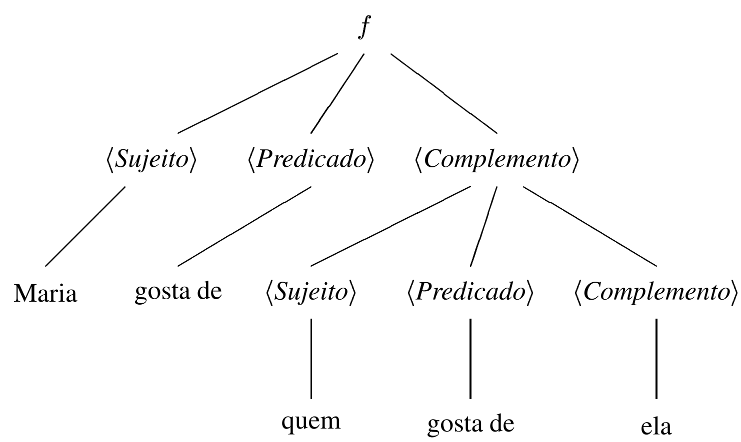


Figura 1.1: Estrutura sintática de uma frase.

$$gosta(x, Maria) \Rightarrow gosta(Maria, x)$$

Historicamente, foi a Linguística a primeira ciência (humana) que se dedicou ao estudo da linguagem (*natural*) e da sua caracterização sintática e semântica. Com o advento da Informática, tornou-se necessário avançar para uma caracterização dos objectos muito mais precisa, no sentido de transportar conhecimentos do nível humano para o nível da máquina. Isto porque um computador — como qualquer outra máquina — não tem intuições nem inteligência [1] e, portanto, não consegue lidar com o discurso ambíguo.

Por exemplo, como ensinar a um computador a organização e funcionamento de uma instituição bancária? Teremos de lhe falar de coisas como *quantias de dinheiro*, *contas de clientes*, seus *titulares*, *cheques etc.* Portanto, essas coisas fazem parte da "sintaxe" de um banco. Mas teremos ainda que lhe dizer, por exemplo, que não faz sentido depositar dinheiro numa conta que ainda não foi aberta! Ou mais ainda, ensinar-lhe que o balanço de uma conta após o depósito de x escudos é $x + y$, onde y é o balanço da conta *antes* desse depósito. Claramente, estes factos têm agora a ver com a "semântica" do serviço prestado pelo dito banco. E que dizer quanto ao modo de funcionamento das portas do edifício bancário? Será de

[1] *inteligência* $<$ *inter+legere* (="ler nas entrelinhas").

dizer alguma coisa? Se o fizermos, alguém poderá dizer que estamos a ser muito "específicos", ou seja, a "especificar pormenores irrelevantes". É óbvio que a noção de "pormenor irrelevante" é relativa: o que é irrelevante num contexto pode ser relevante noutro. No exemplo dado, o modo de funcionamento das portas do edifício bancário será sem dúvida um pormenor relevante no respectivo projecto de construção civil. Quer dizer, especificar significará sempre "ter uma visão intensionalmente parcial" do mundo que nos rodeia.

Somos assim conduzidos ao conceito de *especificação*. Um dicionário [2] diz:

- *especificação* (substantivo feminino) — acto ou efeito de especificar;

- *especificar* (verbo transitivo) — indicar a espécie de; particularizar; mencionar.

Em resumo, especificar um objecto, sistema *etc.* significa particularizá-lo (tanto sintatica como semanticamente), mas não demasiado!

Outro aspecto a discutir quanto a *especificações* tem a ver com outra das suas dimensões, a da *redundância*. Vamos supor que alguém nos aparece com um programa em PASCAL que implementa o problema do sistema bancário acima referido. Poderá esse programa ser considerado uma boa *especificação* do problema? Facilmente se vê que não, mas agora numa outra dimensão: esse programa está "sobre-especificado", *i.é* cheio de informação redundante. Por exemplo, o programa poderá definir uma estrutura de informação global que registe todas as contas, baseada *e.g.* no tipo de dados [3]

```
type contas = ^registo;
     registo = record
                Nr: Numero de conta;
                St: Estado de conta;
                Next: ^registo
               end;
```

ou noutro qualquer que lhe seja equivalente.

Para além de exigir algum conhecimento de pormenores de codificação em PASCAL — *e.g.* a implementação de *listas de x* por apontadores ($\hat{x}$) — este programa impõe uma *ordem* de acesso sobre contas. Trata-se de uma redundância pois, abstractamente, nenhum cliente do banco "vem primeiro que outro", ou "tem prioridade sobre outro" (com igual *status*). Por conter esta redundância (e talvez outras) a referida codificação diz-se uma *implementação* (e não uma *especificação*) do problema.

[2] Pág. 581 do *Dicionário de Língua Portuguesa*, 1975, Porto Editora, por J.A.Costa e A.S. Melo.

[3] Trata-se de solução académica impensável numa situação real, o que contudo não prejudica a sua eficácia enquanto mera ilustração.

Temos, em resumo, um binómio

$$\begin{cases} \text{especificação — O QUE o sistema deve fazer} \\ \text{implementação — COMO o sistema o faz} \end{cases}$$

Como é previsível que haja mais do que uma *implementação* para uma dada *especificação* (*e.g.* por escolha de diferentes estruturas de dados, de diferentes linguagens de codificação *etc.*), temos uma relação de *um para muitos* entre especificações e suas implementações. Essa relação é uma relação de abstracção:

uma especificação

(abstracção)

. . . . . .

as suas implementações

A maior parte da redundância em implementações é introduzida por razões de eficiência, *i.é* no sentido de tornar os programas mais curtos e/ou mais rápidos. O que se ganha em eficiência (em 'run-time') perde-se em clareza (de leitura). Em grandes sistemas de 'software' essa perda de clareza pode vir a ser, ao fim e ao cabo, anti-económica, já que pode dificultar (se não inviabilizar) a fase de *manutenção* do sistema, a mais onerosa e importante de todas as fases do ciclo de vida de um produto de 'software'. Se acrescentarmos a este estado de coisas o facto de o 'software' poder ter erros por detectar —isto porque não é, tradicionalmente, feito qualquer esforço no sentido de *justificar* as implementações — temos identificada a *crise* de produtividade que se instalou na indústria do 'software', crise essa cujos contornos foram determinados há, pelo menos, duas décadas.

É hoje do consenso geral a ideia de que o 'software' deve ser concebido a partir de *especificações* tão abstractas quanto possível, que captem apenas a *essência* dos sistemas a construir, que constituam a base da sua *documentação* (quer para o utilizador, quer para a equipa de manutenção) e que sejam o ponto de partida para a sua implementação.

O principal objectivo é que as especificações se possam escrever numa linguagem não-ambígua para evitar erros de interpretação. Este objectivo conduz ao conceito de *especificação formal*, isto é, a especificação de sistemas usando a linguagem matemática. Além de ser não-ambígua, a linguagem matemática abre caminho para a *justificação de correcção* das implementacções, impossível de outra forma [4]. Neste contexto, o desenvolvimento de uma implementação pode

[4] Para uma introdução mais detalhada a este assunto, porventura o mais importante dos métodos formais de programação, podem ser consultadas, na parte III deste texto, as secções 7.1 e 7.2.

ser feito em simultâneo com a demonstração matemática da sua correcção face à sua especificação:

$$\begin{array}{rccl} & \text{especificação} & - & \text{O QUÊ} \\ \textit{justificação} & \uparrow & - & \text{O PORQUÊ} \\ & \text{implementação} & - & \text{O COMO} \end{array} \tag{1.1}$$

Os métodos ditos *formais* de programação têm por objectivo fazer cumprir o desiderato do diagrama 1.1. Desde já se anuncia que há variadas maneiras de o fazer cumprir, sobretudo dependentes da teoria matemática que se adopta por base [5]. Começa-se neste capítulo por fazer uma introdução ao estudo das técnicas de especificação de 'software' (nas suas vertentes *sintática* e *semântica*) usando uma linguagem baseada na notação matemática "convencional" [6].

1.2 A Sintaxe

Voltemos ao problema-exemplo da "informatização de um sistema bancário" referido na secção anterior e comecemos por enumerar algumas entidades sintáticas que porventura nos ocorram sobre ele [7]:

Quantia — de dinheiro

Titular — o dono de uma conta

IdConta — a identificação (*e.g.* um número) de uma conta

Sistema — o conjunto de todas as contas

abertura — de uma nova conta no sistema, indicando o seu titular

fecho — de uma conta que se pretende eliminar do sistema

depósito — de uma determinada quantia numa conta do sistema

levantamento — inversa da anterior

co-titular — adição de mais um titular a uma conta do sistema

balanço — inquérito ao saldo de uma dada conta do sistema.

[5] Por exemplo, a Lógica, a Álgebra, a Matemática Discreta, a Teoria das Categorias, *etc.*

[6] Sugere-se o livro de texto [AKM81] como possível referência de consulta para os conceitos matemáticos elementares que o presente texto assume à partida como sendo do conhecimento do leitor.

[7] Naturalmente que temos em mente um sistema de gestão bancária à escala "brinquedo", já que o problema na sua extensão real seria impraticável como exemplo introdutório.

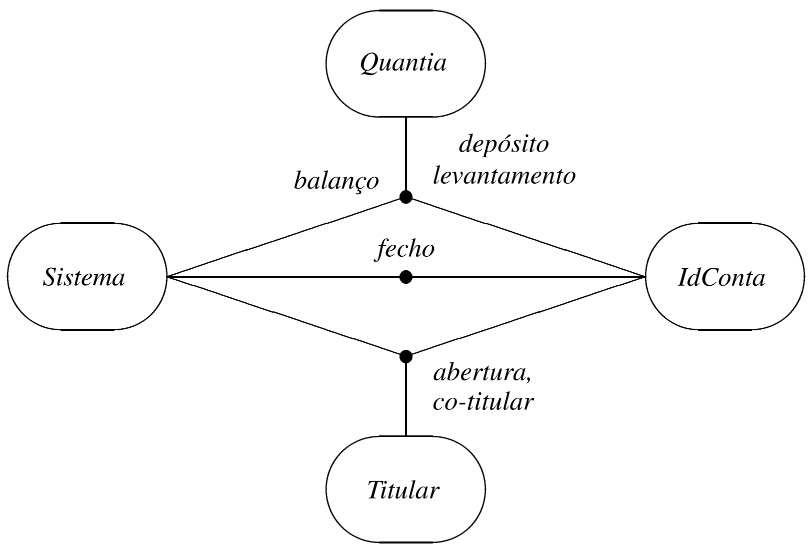


Figura 1.2: Relação entre espécies e operadores.

É intuitiva a constatação de que as entidades acima explicitadas são de duas naturezas possíveis. As quatro primeiras são designações de classes de objectos presentes no problema. Mais correctamente, são nomes de *tipos*, ou *espécies* de objectos. As restantes entidades são nomes de "coisas" que podem acontecer num banco. Dar-lhe-emos o nome técnico de *operações*, ou *operadores*. Em resumo,

$$\text{entidades sintáticas} \left\{ \begin{array}{l} \text{espécies} \\ \text{operadores} \end{array} \right.$$

Notemos ainda que as espécies e os operadores estão relacionados entre si, pois é possível identificar o conjunto de espécies que cada operador envolve:

$$\left\{ \begin{array}{lll} \textit{abertura} & \rightarrow & \{Titular, IdConta, Sistema\} \\ \textit{fecho} & \rightarrow & \{IdConta, Sistema\} \\ \textit{depósito} & \rightarrow & \{Quantia, IdConta, Sistema\} \\ \textit{levantamento} & \rightarrow & \{Quantia, IdConta, Sistema\} \\ \textit{co-titular} & \rightarrow & \{Titular, IdConta, Sistema\} \\ \textit{balanço} & \rightarrow & \{IdConta, Sistema, Quantia\} \end{array} \right.$$

o que se representa no grafo da Figura 1.2.

É possível melhorar o relacionamento acima, se nos apercebermos que o conjunto de espécies manipuladas por um operador se pode desdobrar num par que exprime a sua *funcionalidade*, ou seja, o conjunto das suas espécies-argumento e a sua espécie-resultado:

$$\left\{\begin{array}{lll} \textit{balanço} & \rightarrow & \langle\{IdConta, Sistema\}, Quantia\rangle \\ \textit{co-titular} & \rightarrow & \langle\{IdConta, Titular, Sistema\}, Sistema\rangle \\ \textit{levantamento} & \rightarrow & \langle\{IdConta, Quantia, Sistema\}, Sistema\rangle \\ \textit{depósito} & \rightarrow & \langle\{IdConta, Quantia, Sistema\}, Sistema\rangle \\ \textit{fecho} & \rightarrow & \langle\{IdConta, Sistema\}, Sistema\rangle \\ \textit{abertura} & \rightarrow & \langle\{IdConta, Titular, Sistema\}, Sistema\rangle \end{array}\right. \tag{1.2}$$

Sejam S e Ω os nomes dados aos conjuntos:

$$\begin{array}{lll} S & = & \{Quantia, IdConta, Sistema, Titular\} \\ \Omega & = & \{\textit{balanço, co-titular, levantamento, depósito, fecho,abertura}\} \end{array}$$

Então o sistema (1.2) pode ser caracterizado matematicamente como sendo uma aplicação de $\Omega \rightarrow 2^S \times S$. Porque a ordem dos argumentos de um operador $\sigma \in \Omega$ é relevante e pode envolver repetição de espécies, somos convidados a substituir os conjuntos ($\{\ldots\}$) de (1.2) por sequências ($<\ldots>$) de espécies, *e.g.*

$$\textit{balanço} \rightarrow \langle <IdConta, Sistema>, Quantia\rangle$$

Teremos então que ajustar a aplicação (1.2) a

$$\Omega \rightarrow S^\star \times S$$

obtendo

$$\left\{\begin{array}{lll} \textit{balanço} & \rightarrow & \langle <IdConta, Sistema>, Quantia\rangle \\ \textit{co-titular} & \rightarrow & \langle <IdConta, Titular, Sistema>, Sistema\rangle \\ \textit{levantamento} & \rightarrow & \langle <IdConta, Quantia, Sistema>, Sistema\rangle \\ \textit{depósito} & \rightarrow & \langle <IdConta, Quantia, Sistema>, Sistema\rangle \\ \textit{fecho} & \rightarrow & \langle <IdConta, Sistema>, Sistema\rangle \\ \textit{abertura} & \rightarrow & \langle <IdConta, Titular, Sistema>, Sistema\rangle \end{array}\right.$$

A toda a aplicação deste tipo chamaremos *assinatura*.

1.2.1 Assinaturas

Definição 1.1 (Assinatura) *Seja S um conjunto finito não vazio de símbolos de* espécie *e Ω um conjunto finito de símbolos de* operação*. Daremos o nome de* assinatura *a toda a aplicação*

$$\Sigma : \Omega \rightarrow S^\star \times S$$

□

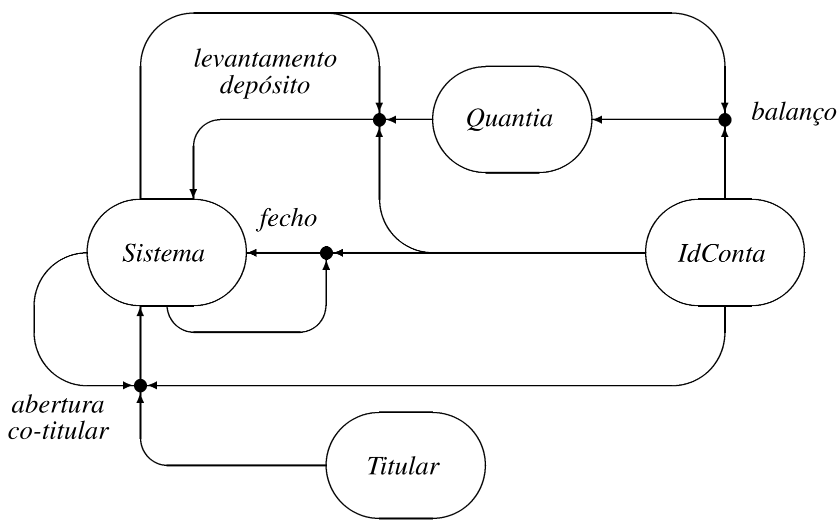


Figura 1.3: Diagrama ADJ para Σ_{SGIB}

Para todo o $\sigma \in \Omega$, dizemos que $\Sigma(\sigma)$ é o valor da sua *funcionalidade*.

Notação 1.1 (Funcionalidade) *Dada uma assinatura Σ, é vulgar escrever-se a expressão*

$$\sigma : s_1 \times \ldots \times s_n \to s \tag{1.3}$$

para exprimir o facto de que $\Sigma(\sigma) = \langle <s_1, \ldots, s_n>, s \rangle$. $\Box$

De acordo com a notação acima, podemos finalmente escrever a assinatura Σ_{SGIB} do nosso "sistema-brinquedo" para gestão de uma instituição bancária como se segue:

$$\Sigma_{SGIB} \left\{ \begin{array}{lcllr} \textit{balanço} & : & IdConta \times Sistema & \to & Quantia \\ \textit{co-titular} & : & IdConta \times Titular \times Sistema & \to & Sistema \\ \textit{levantamento} & : & IdConta \times Quantia \times Sistema & \to & Sistema \\ \textit{depósito} & : & IdConta \times Quantia \times Sistema & \to & Sistema \\ \textit{fecho} & : & IdConta \times Sistema & \to & Sistema \\ \textit{abertura} & : & IdConta \times Titular \times Sistema & \to & Sistema \end{array} \right. \tag{1.4}$$

o que pode ser ilustrado pelo diagrama "em aranha" da Figura 1.3. A este tipo de diagramas também se dá, na literatura, o nome de "diagrama ADJ" [8].

[8] Designação derivada do grupo de investigação que tornou popular o uso destes diagramas, *cf.* Secção 1.5.

Definição 1.2 (Constantes) *Dada uma assinatura Σ, o conjunto C_s de todas as* constantes *de Σ, de espécie s, define-se por*

$$C_s \stackrel{\text{def}}{=} \{\sigma \in \Omega \mid \ \Sigma(\sigma) = \langle <>, s \rangle\}$$

□

Reparemos que uma constante σ é declarada numa assinatura escrevendo $\sigma : <> \to s$ ou, simplesmente, $\sigma :\to s$. Quer dizer, uma constante pode ser considerada um operador "zero-ádico" [9] ou seja, a designação de um "valor" de uma álgebra.

Por exemplo, consideremos uma estrutura algébrica conhecida — o monóide — e tentemos caracterizar a correspondente assinatura. Em Álgebra, dizemos que é um monóide todo o conjunto M munido de uma operação binária $\theta : M \times M \to M$, associativa, e com um elemento neutro ϵ. Notemos que só existe uma espécie numa álgebra deste tipo. Logo $S = \{M\}$ na assinatura de um monóide. Como θ é binária, escreveremos $\Sigma(\theta) = \langle <M, M>, M \rangle$. A especificação de um monóide exige, ainda, a presença do elemento neutro ϵ; trata-se de uma constante tal que $\Sigma(\epsilon) = \langle <>, M \rangle$. Em resumo, teremos

$$\begin{array}{rcl} S & = & \{M\} \\ \Omega & = & \{\theta, e\} \\ \Sigma(\theta) & = & \langle <M, M>, M \rangle \\ \Sigma(\epsilon) & = & \langle <>, M \rangle \end{array}$$

ou, usando a notação mais vulgar (1.3):

$$\Sigma \qquad \left\{ \begin{array}{l} \theta : M \times M \to M \\ \epsilon :\to M \end{array} \right.$$

Note-se que nem a associatividade de θ nem a neutralidade de ϵ são reflectidas na assinatura, já que são propriedades *semânticas* dos símbolos de operação θ e ϵ, e uma assinatura só especifica a sua *estrutura sintática*.

Exercício 1.1 Especifique a assinatura de um espaço vectorial, e represente o respectivo diagrama ADJ.
□

Definição 1.3 (Assinaturas Homogéneas/ Heterogéneas) *Uma assinatura $\Sigma : \Omega \to S^\star \times S$ tal que $|S| = 1$ diz-se* homogénea*; sempre que $|S| > 1$, a assinatura diz-se* heterogénea. □

[9] Um operador diz-se *monádico* ou *unário* se tem apenas um argumento, *diádico* ou *binário* se tem dois, *etc.*

Claramente, a assinatura de um monóide é homogénea [10]; já a nossa assinatura Σ_{SGIB} acima é *heterogénea*. A maior parte dos problemas da vida real, quando especificados, conduzem a assinaturas heterogéneas. Daí a necessidade desta extensão à álgebra "clássica".

Exercício 1.2 Suponha que os requisitos do sistema de gestão de uma instituição bancária acima estudado (SGIB) se alteram no seguinte sentido:

- todas as transacções sobre contas passam a fazer-se apenas por cheque;
- a "história" de cada conta (*i.é* todos os cheques que foram movimentados desde que ela foi criada) fica registada na base de dados.

Faça as alterações à assinatura Σ_{SGIB} que julgue, como consequência, convenientes.
□

Exercício 1.3 O Departamento de Planeamento da Produção (DPP) de uma fábrica de equipamento electrónico pretende uma base de dados contendo informação sobre a estrutura de qualquer equipamento em produção, e níveis do 'stock' existente. Cada equipamento deverá ser registado com base nos seus blocos estruturais, *i.é*, seguindo os esquemas técnicos definidos pelos departamentos de 'design' e manutenção. Segundo estes, qualquer equipamento é um bloco constituído por um determinado número de:

- circuitos integrados;
- outros componentes electrónicos (díodos, transistores, resistências, conectores *etc.*.);
- outros (sub)blocos.

O DPP pretende usar a base para relacionar níveis futuros de produção com investimentos a fazer, face aos níveis de 'stock' existentes. Deverá, assim, ser registado o custo unitário (de produção ou compra) de todos os componentes electrónicos elementares. Em particular, pretende-se ter disponível uma operação que determine a "explosão em componentes" (relação *componente elementar* $\mapsto$ *número*) de qualquer dos equipamentos que a fábrica produz.

Escreva uma assinatura que especifique de forma detalhada a camada sintática da aplicação que o DPP acima pretende ver desenvolvida.
□

1.2.2 Gramáticas versus Assinaturas

É agora relevante mostrar como, por detrás de toda a gramática generativa dita *independente de contexto* ou *de contexto livre* ('context-free') — exprimível na clássica notação BNF, por exemplo — está uma assinatura algébrica, heterogénea na maior parte dos casos.

Seja $G = \langle N, T, S, P \rangle$ uma gramática independente de contexto, onde N é o conjunto dos seus símbolos *não-terminais*, T é um conjunto de símbolos *terminais*

[10] E não só: em Álgebra, grupos, grupos abelianos, anéis *etc.* continuam a ter assinaturas homogéneas.

(disjunto de N), $S \in N$ é o seu símbolo *inicial* e $P \subseteq N \times (N \cup T)^\star$ é o conjunto das *produções* ou *regras sintáticas* de G. A inferência da assinatura Σ_G determinada por G segue dois critérios básicos:

1. a cada símbolo *não-terminal* da gramática G corresponde uma *espécie* na assinatura Σ_G;

2. a cada *produção* da gramática G corresponde um *operador* na assinatura Σ_G;

   (a) esse operador constrói-se da seguinte forma: seja a produção em causa da forma
   $$\langle NT \rangle ::= \alpha \langle NT_1 \rangle \beta \langle NT_2 \rangle \ldots \langle NT_n \rangle \gamma$$
   onde os símbolos não-terminais se representam entre $\langle \ldots \rangle$ e α, β, γ, ... são sequências de símbolos *terminais*; então o operador correspondente a essa produção tem a funcionalidade
   $$\sigma : \langle NT_1 \rangle \times \langle NT_2 \rangle \times \ldots \times \langle NT_n \rangle \to \langle NT \rangle$$

Observações:

- o símbolo σ que se escolhe por produção é arbitrário, desde que produções diferentes correspondam a símbolos diferentes;

- a assinatura que se constrói "perde informação" quanto aos símbolos *terminais* da gramática, que são ignorados;

- embora o símbolo inicial $S \in N$ se possa considerar uma "espécie distinguida", isso não tem qualquer implicação formal na construção da assinatura.

Em resumo, a assinatura retém a essência generativa da gramática, de forma *abstracta* — abstracta no sentido em que "filtra" a sintaxe *concreta*, *i.é* os símbolos que só estão na gramática por uma de duas razões: ou desambiguam a gramática para efeito de *reconhecimento* por um autómato (*e.g.* da classe LL(1), LR(0) *etc.*), ou são símbolos que presumivelmente aumentam a "legibilidade" da linguagem gerada pela gramática [11].

Vejamos um exemplo: seja dada a gramática G de uma pequena linguagem de comandos cujos símbolos não-terminais são: ⟨*Cmd*⟩ (comando), ⟨*Var*⟩ (variável)

[11] A esta componente de uma gramática é vulgar dar o nome sugestivo de "açúcar sintático". Este é particularmente notório em linguagens comerciais como, por exemplo, o COBOL.

e $\langle \mathit{Nato} \rangle$ (números naturais incluindo o zero). As suas produções são as seguintes:

$$\begin{array}{rcl} \langle \mathit{Nato} \rangle & ::= & \texttt{0} \mid \texttt{suc(}\langle \mathit{Nato} \rangle\texttt{)} \mid \langle \mathit{Var} \rangle \\ \langle \mathit{Var} \rangle & ::= & \texttt{x} \mid \texttt{y} \\ \langle \mathit{Cmd} \rangle & ::= & \texttt{skip} \mid \\ & & \langle \mathit{Cmd} \rangle\texttt{;}\langle \mathit{Cmd} \rangle \mid \\ & & \langle \mathit{Var} \rangle\texttt{:=}\langle \mathit{Nato} \rangle \mid \\ & & \texttt{if } \langle \mathit{Nato} \rangle \texttt{ then } \langle \mathit{Cmd} \rangle \texttt{ else } \langle \mathit{Cmd} \rangle \mid \\ & & \texttt{while } \langle \mathit{Nato} \rangle \texttt{ do } \langle \mathit{Cmd} \rangle \end{array} \tag{1.5}$$

Notemos que cada cláusula BNF da forma

$$\langle \mathit{NT} \rangle ::= p_1 \mid p_2 \mid \ldots \mid p_n$$

abrevia de facto *n* produções:

$$\begin{array}{rcl} \langle \mathit{NT} \rangle & ::= & p_1 \\ \langle \mathit{NT} \rangle & ::= & p_2 \\ & \vdots & \\ \langle \mathit{NT} \rangle & ::= & p_n \end{array}$$

Isto permite-nos adiantar que a assinatura correspondente à gramática (1.5) terá três operadores com resultado em $\langle \mathit{Nato} \rangle$, outros dois com resultado em $\langle \mathit{Var} \rangle$, e cinco com resultado em $\langle \mathit{Cmd} \rangle$. Seguindo a ordem das produções, teremos:

$$\begin{array}{lcllr} \sigma_1 & : & & \rightarrow & \langle \mathit{Nato} \rangle \\ \sigma_2 & : & \langle \mathit{Nato} \rangle & \rightarrow & \langle \mathit{Nato} \rangle \\ \sigma_3 & : & \langle \mathit{Var} \rangle & \rightarrow & \langle \mathit{Nato} \rangle \\ \sigma_4 & : & & \rightarrow & \langle \mathit{Var} \rangle \\ \sigma_5 & : & & \rightarrow & \langle \mathit{Var} \rangle \\ \sigma_6 & : & & \rightarrow & \langle \mathit{Cmd} \rangle \\ \sigma_7 & : & \langle \mathit{Cmd} \rangle \times \langle \mathit{Cmd} \rangle & \rightarrow & \langle \mathit{Cmd} \rangle \\ \sigma_8 & : & \langle \mathit{Var} \rangle \times \langle \mathit{Nato} \rangle & \rightarrow & \langle \mathit{Cmd} \rangle \\ \sigma_9 & : & \langle \mathit{Nato} \rangle \times \langle \mathit{Cmd} \rangle \times \langle \mathit{Cmd} \rangle & \rightarrow & \langle \mathit{Cmd} \rangle \\ \sigma_{10} & : & \langle \mathit{Nato} \rangle \times \langle \mathit{Cmd} \rangle & \rightarrow & \langle \mathit{Cmd} \rangle \end{array}$$

ou, adoptando nomes mais sugestivos:

$$\begin{array}{llll} zero & : & & \rightarrow \langle \mathit{Nato} \rangle \\ incremente & : & \langle \mathit{Nato} \rangle & \rightarrow \langle \mathit{Nato} \rangle \\ valor & : & \langle \mathit{Var} \rangle & \rightarrow \langle \mathit{Nato} \rangle \\ x & : & & \rightarrow \langle \mathit{Var} \rangle \\ y & : & & \rightarrow \langle \mathit{Var} \rangle \\ nada & : & & \rightarrow \langle \mathit{Cmd} \rangle \\ seq & : & \langle \mathit{Cmd} \rangle \times \langle \mathit{Cmd} \rangle & \rightarrow \langle \mathit{Cmd} \rangle \\ atrib & : & \langle \mathit{Var} \rangle \times \langle \mathit{Nato} \rangle & \rightarrow \langle \mathit{Cmd} \rangle \\ se & : & \langle \mathit{Nato} \rangle \times \langle \mathit{Cmd} \rangle \times \langle \mathit{Cmd} \rangle & \rightarrow \langle \mathit{Cmd} \rangle \\ ciclo & : & \langle \mathit{Nato} \rangle \times \langle \mathit{Cmd} \rangle & \rightarrow \langle \mathit{Cmd} \rangle \end{array} \tag{1.6}$$

Note-se que se escolheram os nomes “x” e “y” como operadores, sendo `x` e `y` símbolos terminais, o que não constitui inconveniente nenhum.

Exercício 1.4 Dada a seguinte gramática independente de contexto:

$$\begin{array}{rcl} G & = & \langle NT, T, \langle \mathit{pilha} \rangle, P \rangle \\ NT & = & \{\langle \mathit{pilha} \rangle, \langle \mathit{elem} \rangle, \langle \mathit{bool} \rangle\} \\ T & = & \{\mathtt{push}, \mathtt{pop}, \mathtt{onto}, \mathtt{empty}, \mathtt{top}, \mathtt{true}, \mathtt{false}, \mathtt{(}, \mathtt{)}, \mathtt{?}\} \\ P & = & \left\{ \begin{array}{lcl} \langle \mathit{pilha} \rangle & ::= & \mathtt{pop}\langle \mathit{pilha} \rangle \mid \mathtt{push}\langle \mathit{elem} \rangle\mathtt{onto}\langle \mathit{pilha} \rangle \mid \\ & & \mathtt{empty} \\ \langle \mathit{elem} \rangle & ::= & \mathtt{top}\langle \mathit{pilha} \rangle \\ \langle \mathit{bool} \rangle & ::= & \mathtt{true} \mid \mathtt{false} \mid \mathtt{empty(}\langle \mathit{pilha} \rangle\mathtt{)?} \end{array} \right. \end{array}$$

construir uma assinatura Σ_G correspondente.

□

Vamos agora supor que uma dada gramática de contexto-livre G contém produções da forma

$$\langle \mathit{NT} \rangle ::= \alpha\beta^+\gamma \tag{1.7}$$

onde α,β e γ são sequências de símbolos terminais ou não-terminais, $\langle \mathit{NT} \rangle$ é um símbolo não-terminal e β^+ designa “uma ou mais ocorrências de β”. Como se traduz a produção acima na assinatura Σ_G correspondente?

Reparemos que, em Teoria das Linguagens, β^+ apresenta-se como “abreviatura” do par de produções

$$\langle L \rangle ::= \beta \mid \beta\langle L \rangle \tag{1.8}$$

Bastará assim acrescentar a G um *novo* não-terminal $\langle L \rangle$ e substituir a produção (1.7) por

$$\begin{array}{rcl} \langle \mathit{NT} \rangle & ::= & \alpha\langle L \rangle\gamma \\ \langle L \rangle & ::= & \beta \mid \beta\langle L \rangle \end{array}$$

construindo-se a assinatura Σ_G a partir da gramática assim estendida.

Esta extensão merece alguns comentários. Primeiro, as produções a acrescentar poderiam ter sido

$$\langle L\rangle ::= \beta \mid \langle L\rangle\beta$$

em lugar de (1.8), o que apenas alteraria a ordem dos argumentos do segundo construtor da espécie $\langle L\rangle$ em Σ_G. Mas também poderíamos ter introduzido uma infinidade de produções,

$$\langle L\rangle ::= \beta \mid \beta\beta \mid \beta\beta\beta \mid \ldots \mid \underbrace{\beta\ldots\beta}_{i \text{ vezes}} \mid \ldots$$

gerando em Σ_G um operador i-ário ($i > 0$) por cada número de vezes que β se repete em β^+ [12].

Exercício 1.5 Suponha que, em (1.7), β^* (="nenhuma, uma ou várias ocorrências de β") aparece em lugar de β^+. Faça um tratamento análogo ao anterior com vista a construir a assinatura correspondente.

$\square$

Exercício 1.6 Considere as produções seguintes de uma gramática independente de contexto G.

$$\begin{array}{lcl} \langle A\rangle & ::= & \mathtt{a}\,\langle B\rangle \mid \mathtt{b}\,\langle A\rangle\,\mathtt{a}\,\langle B\rangle^* \\ \langle B\rangle & ::= & \mathtt{a}\ \mathtt{b} \mid \mathtt{a}\,\langle A\rangle^+\,\mathtt{c} \end{array}$$

Construa a assinatura Σ_G correspondente.

$\square$

Exercício 1.7 O quadro que se segue descreve duas transformações conhecidas para resolver conflitos LL(1) em gramáticas independentes de contexto:

Transformação ($G \longrightarrow G'$)	**Antes** (G)	**Depois** (G')
Factorização à esquerda	$\langle A\rangle ::= \alpha\beta \mid \alpha\gamma$	$\langle A\rangle ::= \alpha\langle A'\rangle$ $\langle A'\rangle ::= \beta \mid \gamma$
Eliminação de recursividade à esquerda	$\langle A\rangle ::= \langle A\rangle\alpha \mid \beta$	$\langle A\rangle ::= \beta\langle A'\rangle$ $\langle A'\rangle ::= \alpha\langle A'\rangle \mid \epsilon$

Caracterize e comente as correspondentes transformações entre as assinatauras Σ_G e $\Sigma_{G'}$.

$\square$

[12] Só no Capítulo 2 se estudará a teoria matemática que explica esta multiplicidade de extensões. Quando dizemos que β^+ é "abreviatura" de (1.8) estamos a esconder esse processo, que nos levará a encarar (1.8) como uma *equação* da qual β^+ é uma solução.

1.2.3 Termos versus Árvores Sintáticas

Põe-se agora uma questão de fundo: a qualquer gramática G de contexto livre corresponde uma *linguagem* L_G, que é o conjunto de todas as *frases* geradas por G; quando substituimos G pela sua correspondente assinatura Σ_G, o que é que obtemos para L_G? Por exemplo, dada a frase

`x:=0; while x do skip` (1.9)

de L_G, *cf.* (1.5), como a representar ao nível de Σ_G? Veremos de seguida que o conjunto de todas as *árvores sintáticas* das frases geradas por G é a construção que, partindo de Σ_G, corresponde a L_G. Para isso, precisamos de uma definição básica:

Definição 1.4 (Σ-termo) *É dada uma assinatura $\Sigma : \Omega \to S^\star \times S$. Para cada espécie $s \in S$, é possível construir o conjunto de todos os "Σ-termos" de espécie s, que será o menor conjunto que satisfaz as seguintes cláusulas:*

1. *toda a constante $\sigma :\to s$ é um Σ-termo de espécie s;*
2. *para todo o operador $\sigma : s_1 \times \ldots \times s_n \to s$, se, para $1 \leq i \leq n$, t_i é um Σ-termo de espécie s_i, então*

$$\sigma(t_1, \ldots, t_i, \ldots, t_n)$$

é um Σ-termo de espécie s.

O conjunto de todos os Σ-termos de espécie s designa-se por $W_{\Sigma,s}$. Um Σ-termo também se designa Σ-palavra ('word') ou Σ-frase. $\square$

É fácil fazer corresponder à definição indutiva anterior a seguinte expressão recursiva:

$$W_{\Sigma,s} \stackrel{\text{def}}{=} C_s \cup \{\sigma(t_1, \ldots, t_n) \mid \left(\begin{array}{ll} \sigma \in \Omega & \wedge \\ \Sigma(\sigma) = \langle <s_1, \ldots, s_n>, s\rangle & \wedge \\ \forall 1 \leq i \leq n : t_i \in W_{\Sigma,s_i} & \end{array}\right)\} \quad (1.10)$$

Quer dizer, a cada espécie $s_i \in S$ corresponde uma definição:

$$\left\{\begin{array}{lcl} W_{\Sigma,s_1} & \stackrel{\text{def}}{=} & \ldots \\ W_{\Sigma,s_2} & \stackrel{\text{def}}{=} & \ldots \\ \vdots & & \\ W_{\Sigma,s_n} & \stackrel{\text{def}}{=} & \ldots \end{array}\right.$$

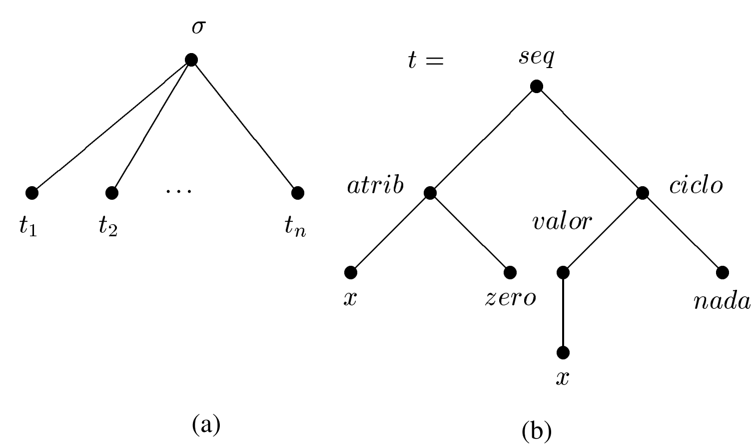


Figura 1.4: Representação de termos por árvores

Uma maneira prática de entender o conceito de Σ-termo é encarar cada Σ-termo $\sigma(t_1, \ldots, t_n)$ ou como o 'string' de texto $'\sigma(t_1, \ldots, t_n)'$, ou como uma árvore simbólica, tal como se ilustra na Figura 1.4(a).

Por exemplo, a árvore desenhada na Figura 1.4(b) representa um Σ-termo da assinatura Σ_G correspondente à linguagem G acima referida. Escrito linearmente, teremos:

$$t = seq(atrib(x, zero), ciclo(valor(x), nada)) \tag{1.11}$$

É imediato ver no diagrama da Figura 1.4(b) — ou na expressão (1.11) — a árvore sintática (sintaxe *abstracta*) da frase *concreta* (1.9).

Em resumo, a maneira mais abstracta (*i.é* económica, sem pormenores desnecessários) de descrever a estrutura sintática de um objecto complicado da vida real (*e.g.* uma instituição bancária), de uma linguagem (de comunicações, de programação *etc.*) *etc.* é através da construção de uma assinatura Σ.

Exercício 1.8 Dada uma assinatura $\Sigma : \Omega \to S^\star \times S$ onde

$$\begin{array}{rcl} S & = & \{s, r\} \\ \Omega & = & \{1, 0, \sigma, 2\} \\ \Sigma(0) & = & \langle <>, s \rangle \\ \Sigma(1) & = & \langle <>, s \rangle \\ \Sigma(2) & = & \langle <>, r \rangle \\ \Sigma(\sigma) & = & \langle <s, s>, r \rangle \end{array}$$

construa o respectivo diagrama-ADJ e enumere todos os termos da correspondente Σ-linguagem.

□

Exercício 1.9 Estudou-se acima como uma gramática de contexto livre G pode ser abstractamente caracterizada por uma assinatura Σ_G, bem como a relação que existe entre L_G — a linguagem gerada por G — e o conjunto de todos os termos em W_{Σ_G}.

É sabido da Teoria das Linguagens que a mesma linguagem pode ser caracterizada por mais do que uma gramática, *i.é*

$$L_{G_1} = L_{G_2} \not\Rightarrow G_1 = G_2$$

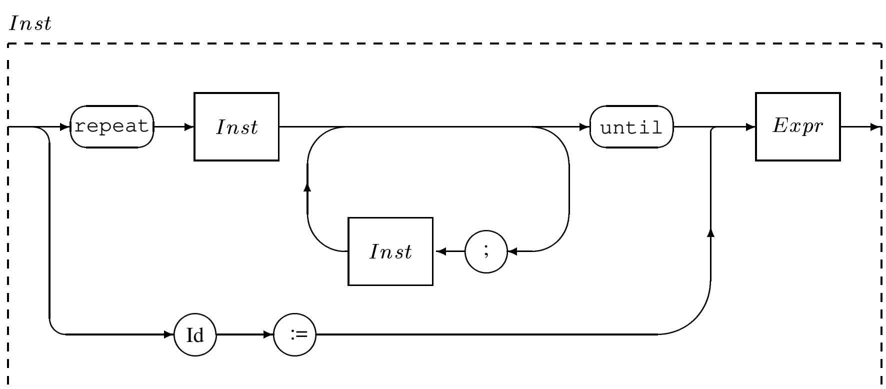
Em que medida se pode afirmar que

$$W_{\Sigma_{G_1}} = W_{\Sigma_{G_2}} \Rightarrow G_1 = G_2 \ ?$$

Justifique.

□

Exercício 1.10 Construa a assinatura algébrica correspondente ao seguinte fragmento da gramática da linguagem de programação PASCAL, expresso sobre a forma de um grafo de sintaxe, onde $Inst$ é o não-terminal para *instruções* e Exp é o não-terminal para *expressões*:



□

Para terminar, refira-se que toda a espécie $s \in S$ sem construtores numa assinatura Σ conduz a $W_{\Sigma,s} = \emptyset$, *cf.* (1.10). Isso corresponde, em termos gramaticais, a não apresentarmos quaisquer produções para um dado não-terminal. Por exemplo, em Σ_{SGIB} teremos

$$W_{\Sigma_{SGIB},IdConta} = W_{\Sigma_{SGIB},Titular} = W_{\Sigma_{SGIB},Quantia} = \emptyset$$

Quer dizer, para a especificação de $SGIB$ ficar completa, será necessário enriquecer o diagrama da Figura 1.3 com os construtores dessas espécies. Mas subsistem problemas em Σ_{SGIB}: também $W_{\Sigma_{SGIB},Sistema} = \emptyset$. Isso acontece porque nos esquecemos de uma constante

$$inic :\rightarrow Sistema$$

que corresponde ao "banco vazio", *i.é* ainda sem contas, ou seja, à primeira configuração de um banco especificado por Σ_{SGIB}. A razão pela qual, sem quaisquer constantes, a espécie $Sistema$ fica vazia de termos entender-se-á resolvendo o exercício que se segue.

Exercício 1.11 Com base na fórmula (1.10), concorda ou discorda da seguinte afirmação: *"Numa assinatura, é vazia de termos toda a espécie para a qual não são fornecidas quaisquer constantes."* ?
□

1.2.4 Representação de Termos por Expressões-S

Vamos agora ver uma estratégia simples para "programarmos" directamente na *linguagem abstracta* que nos é transmitida por W_Σ, o que faremos com recurso à linguagen LISP para computação simbólica.

Em LISP existe um único suporte para representação de informação, designado por *expressão-S* — abreviatura de "expressão simbólica" — de acordo com a sintaxe seguinte:

$$\langle\text{Expressão-S}\rangle \quad ::= \quad \langle\text{Átomo}\rangle \mid \texttt{(}\langle\text{Expressão-S}\rangle^{*}\texttt{)} \tag{1.12}$$

onde se considera um ⟨Átomo⟩ toda a unidade de informação indivisível, não-estruturada (*i.é* "atómica"). Por exemplo, são átomos os inteiros e os 'strings' alfanuméricos, *e.g.* `10`, `-5`, `x12`, `Sun3@160`. Dão-se a seguir exemplos de ⟨Expressão-S⟩ não atómicas:

```
()
(1)
(1 um 2 dois)
(1 (2 (3 (4))))
```

Vejamos agora como é simples representar qualquer termo $t \in W_\Sigma$ por uma expressão-S, para uma dada assinatura $\Sigma : \Omega \rightarrow S^\star \times S$. Em primeiro lugar, todo o símbolo de operação $f \in \Omega$ é representado pelo correspondente átomo `f`. Vamos designar por $\{t\}$ a representação em expressão-S do termo t. Seja agora $t = f(t_1, \ldots, t_n)$. Teremos a seguinte regra de tradução:

$$\{f(t_1, \ldots, t_n)\} = \texttt{(f } \{t_1\} \cdots \{t_n\} \texttt{)}$$

Por exemplo,

$$\begin{array}{rcl} \{sqrt(17)\} & = & \texttt{(srqt 17)} \\ \{2 \times (3 + x)\} & = & \texttt{(* 2 (+ 3 x))} \end{array}$$

e a representação em expressão-S do termo (1.11) atrás será

```
(seq (atrib x zero) (ciclo (valor x) nada))
```

Capitalizando tudo o que até aqui se viu, podemos dizer que qualquer frase de qualquer linguagem (independente de contexto) pode ser representada por uma expressão-S. Esta propriedade "universal" das expressões-S é explorada no próprio LISP, cujos programas são — eles próprios — expressões-S. Por exemplo, a expressão-S

```
(defun fac (n) (if (eq n 0) 1 (* n (fac (- n 1)))))
```

designa, em LISP, a definição recursiva da função *factorial*:

$$fac(n) \stackrel{\text{def}}{=} \begin{cases} n = 0 & \Rightarrow & 1 \\ n > 0 & \Rightarrow & n \times fac(n-1) \end{cases}$$

Daqui resulta uma economia conceptual apreciável: uma linguagem, uma única estrutura de dados, a que não é alheio o sucesso crescente da linguagem nos dias de hoje.

1.3 A Semântica

Em tudo o que acima se expôs, a especificação que fazemos de um sistema — atribuindo-lhe uma assinatura e, por inerência, definindo-lhe uma linguagem — é meramente simbólica, ou seja, carece de "significado". O termo "semântica" costuma ser usado para designar a fase de atribuição de significados às entidades sintáticas [13]. É nesta fase que se procura uma construção da forma

$$\text{sintaxe} \xrightarrow[\text{significado}]{} \text{semântica}$$

Como se verá, há várias formas de dotar uma Σ-linguagem de significado. A que se vai ver neste capítulo introdutório é talvez a mais intuitiva e mais simples,

[13] O estudo da Semântica, na Linguística, data do século passado: *"... c'est sur le corps et sur la forme des mots que la plupart des linguistes ont exercé leur sagacité; les lois qui president à la transformation des sens, ..., ont eté laissées dans l'ombre... Comme cette étude, aussi bien que la phonetique et la morphologie, mérite d'avoir son nom, nous l'appelerons la sémantique... c'est-à-dire, la science des significations."* (Michel Bréal, 1883).

por se limitar a seguir a tradição científica secular (*e.g.* da Física, das Ciências da Engenharia *etc.*) de raciocinar sobre o mundo físico com base em *modelos* matemáticos. A principal diferença é que, na Física, Ciências da Engenharia *etc.*, o universo de discurso é "contínuo" e "infinito" (*cf. e.g.* domínios como *espaço* e *tempo*), o que contrasta com a natureza *discreta* do 'software', imposta pela finitude das máquinas onde este está condenado a correr.

Isto explica que os formalismos requeridos pelos modelos semânticos do 'software' pertençam ao domínio da Álgebra e da Matemática Discreta e, no nosso caso e neste contexto, à teoria "ingénua" dos conjuntos, e não ao clássico cálculo diferencial e integral.

Começaremos assim por rever e sistematizar os conceitos da teoria "ingénua" dos conjuntos que serão úteis em especificação formal.

1.3.1 Noções de Teoria "Ingénua" de Conjuntos

À classe de todos os conjuntos finitos (ou infinitos numeráveis) daremos o nome de *Sets* [14].

Dados dois conjuntos A e B, entenderemos como elementar a noção de função, ou aplicação f de A para B, que se designará por $A \xrightarrow{f} B$ ou $f : A \to B$. A aplicação identidade sobre um conjunto A, que será designada por 1_A ou simplesmente por id quando A estiver subentendido, define-se por

$$\begin{array}{rcl} 1_A & : & A \longrightarrow A \\ 1_A(a) & \stackrel{\mathrm{def}}{=} & a \end{array} \tag{1.13}$$

A composição de duas aplicações $f : A \to B$ e $g : C \to A$ é a aplicação $f \circ g : C \to B$ tal que

$$(f \circ g)(c) = f(g(c)) \tag{1.14}$$

para qualquer $c \in C$. Os conjuntos A e B dir-se-ão *isomorfos*, escrevendo-se $A \cong B$, se e só se fôr possível estabelecer uma aplicação de A para B, ou de B para A, que seja uma bijecção.

Construções Primitivas

Assumiremos que, se A e B são conjuntos em *Sets*, então fazem sentido as construções seguintes:

- o seu *produto cartesiano*:

$$A \times B = \{\langle a, b\rangle \mid a \in A \wedge b \in B\} \tag{1.15}$$

[14] Seguindo a terminologia de [AKM81], por um conjunto *infinito numerável* entendemos um conjunto cujo cardinal $|A|$ é igual a $\aleph_0$, *i.é* ao cardinal de $I\!N$.

- a sua *união disjunta*:

$$A + B = (\{1\} \times A) \cup (\{2\} \times B) \tag{1.16}$$

- a *exponenciação*:

$$B^A = \{f \mid \ f : A \to B\} \tag{1.17}$$

onde aqui A é um conjunto finito e $f : A \to B$ é a designação de uma qualquer aplicação de A em B, *i.é* tal que

$$\begin{array}{rcl} a \in A & \Rightarrow & f(a) \in B \\ f(a) \neq f(a') & \Rightarrow & a \neq a' \end{array}$$

Admite-se ainda o conjunto vazio $\emptyset$ (também designável por $\{\}$) e, para todo o número natural $n \in I\!N_0$, o conjunto

$$\overline{n} = \{1, 2, \ldots, n\}$$

designado *segmento inicial* de $I\!N$ induzido por n. Note-se que

$$\begin{array}{rcl} \overline{0} & = & \emptyset \\ \overline{n+1} & = & \{n+1\} \cup \overline{n} \end{array}$$

Sempre que for claro pelo contexto, escreveremos "n" em lugar de "$\overline{n}$". É, pois, natural designarmos por 0 o conjunto vazio $\emptyset$. Note-se ainda que o conjunto dos valores booleanos é isomorfo de $\overline{2}$,

$$\{V, F\} \cong \overline{2}$$

o que explica que, quando for claro pelo contexto, escrevamos "2" como abreviatura de $\{V, F\}$. Designaremos por *condição* ou *predicado* toda a aplicação $p : A \longrightarrow 2$, para A qualquer.

PRODUTO CARTESIANO ($A \times B$)

Ao produto cartesiano (1.15) estão associadas as respectivas projecções π_1 e π_2,

$$A \xleftarrow{\pi_1} A \times B \xrightarrow{\pi_2} B \tag{1.18}$$

tais que

$$\begin{array}{rcl} \pi_1(\langle a, b\rangle) & = & a \\ \pi_2(\langle a, b\rangle) & = & b \end{array}$$

Da mesma forma que, dados dois conjuntos A e B, se pode construir o seu produto $A \times B$, também faz sentido construir o produto $f \times g$ de duas aplicações

$f : A \longrightarrow C$ e $g : B \longrightarrow D$, e que corresponde à aplicação de f e de g "em paralelo":

$$\begin{aligned} f \times g &\quad : \quad A \times B \longrightarrow C \times D \\ f \times g(\langle a, b \rangle) &\quad \stackrel{\text{def}}{=} \quad \langle f(a), g(b) \rangle \end{aligned} \tag{1.19}$$

Outra agregação funcional associada ao produto cartesiano [15] é a chamada *construção de funções*, designada $\langle f, h \rangle$ e que, para $f : A \longrightarrow C$ e $h : A \longrightarrow D$, tem o significado seguinte:

$$\begin{aligned} \langle f, h \rangle &\quad : \quad A \longrightarrow C \times D \\ \langle f, h \rangle(a) &\quad \stackrel{\text{def}}{=} \quad \langle f(a), h(a) \rangle \end{aligned} \tag{1.20}$$

Para além da sua utilidade na construção de especificações, estas construções funcionais gozam de propriedades que se revelarão muito úteis nos cálculos que quereremos fazer sobre essas especificações. Mais do que listar essas propriedades é útil perceber a sua origem. Atente-se, para esse efeito, no diagrama que se segue:

$$\begin{tikzcd}
B & B \times E \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & E \\
C \arrow[u, "j"] & C \times D \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[u, "j \times i"'] & D \arrow[u, "i"'] \\
& A \arrow[ul, "f"] \arrow[u, "\langle f, h \rangle"] \arrow[ur, "h"'] &
\end{tikzcd}$$

Repare-se, antes de mais, que as funções presentes no diagrama estão coerentemente tipadas. As seguintes quatro propriedades elementares,

$$\pi_1 \circ \langle f, h \rangle = f \tag{1.21}$$
$$\pi_2 \circ \langle f, h \rangle = h \tag{1.22}$$
$$\pi_1 \circ (j \times i) = j \circ \pi_1 \tag{1.23}$$
$$\pi_2 \circ (j \times i) = i \circ \pi_2 \tag{1.24}$$

[15] As construções que estamos a apresentar não surgem ao acaso — correspondem a construções universais que se explicam facilmente recorrendo à Teoria das Categorias [Mac71] mas cuja fundamentação, contudo, preferimos ocultar para já.

correspondem à leitura dos quatro pares de caminhos alternativos entre "nós" do grafo. Por exemplo, para "irmos" de A para C temos duas alternativas: directamente através de f, ou passando por $\langle f, h\rangle$ e depois por π_1. Daqui infere-se a primeira lei e assim sucessivamente.

Se combinarmos verticalmente as setas do diagrama obtemos o facto, mais elaborado, que se segue:

$$(j \times i) \circ \langle f, h\rangle \quad = \quad \langle j \circ f, i \circ h\rangle \tag{1.25}$$

Se acrescentarmos a função $k : F \longrightarrow A$ ao diagrama e nele representarmos as composições $f \circ k$ e $h \circ k$ seremos conduzidos a:

$$\langle f, h\rangle \circ k \quad = \quad \langle f \circ k, h \circ k\rangle \tag{1.26}$$

O produto binário $A \times B$ é extensível ao produto finitário $A_1 \times \ldots \times A_n$, para $n \in \mathbb{N}$, também designável por $\Pi_{i=1}^{n} A_i$, ao qual estão associadas tantas projecções π_i quantos os factores envolvidos.

Na prática da especificação formal, usam-se muitas vezes produtos cartesianos prefixados por *selectores*, para melhor legibilidade. O efeito é o seguinte: se definirmos $C = A \times B$ temos imediatamente disponíveis os operadores canónicos de selecção π_1 e π_2,

$$A \xleftarrow{\pi_1} C \xrightarrow{\pi_2} B$$

como acabamos de ver. Alternativamente, podemos definir

$$C \cong P : A \times S : B$$

Neste caso, os selectores "anónimos" π_1 e π_2 passarão a designar-se

$$A \xleftarrow{P} C \xrightarrow{S} B$$

Como a escolha dos selectores (P,S, *etc.*) é livre, desde que se não repitam no mesmo produto, a legibilidade do texto pode aumentar significativamente, sem contudo perturbar o seu significado matemático.

UNIÃO DISJUNTA $(A + B)$

Para a união disjunta $A + B$ o diagrama (1.18) passa a ser

$$A \xrightarrow{i_1} A + B \xleftarrow{i_2} B \tag{1.27}$$

onde duas cláusulas

$$\begin{array}{rcl} i_1(a) & = & \langle 1, a\rangle \\ i_2(b) & = & \langle 2, b\rangle \end{array} \tag{1.28}$$

definem as correspondentes "injecções". Dualmente ao que fizemos acima para o produto cartesiano, faz sentido a construção funcional $f + g$ definida como se segue:

$$\begin{array}{rcl} f+g & : & A+B \longrightarrow C+D \\ f+g(x) & \stackrel{\mathrm{def}}{=} & \begin{cases} x = i_1(a) & \Rightarrow \quad i_1(f(a)) \\ x = i_2(b) & \Rightarrow \quad i_2(g(b)) \end{cases} \end{array} \tag{1.29}$$

Da mesma forma que associamos ao produto cartesiano a *construção de funções* ($\langle h, g \rangle$) podemos associar à união disjunta a chamada *alternativa* (ou *escolha*) de funções, que se designa por $[\, h, g \,]$, para $h : A \longrightarrow D$ e $g : B \longrightarrow D$ e tem o significado seguinte:

$$\begin{array}{rcl} [\, h, g \,] & : & (A+B) \longrightarrow D \\ [\, h, g \,](x) & \stackrel{\mathrm{def}}{=} & \begin{cases} x = i_1(a) & \Rightarrow \quad h(a) \\ x = i_2(b) & \Rightarrow \quad g(b) \end{cases} \end{array} \tag{1.30}$$

Tal como aconteceu para o produto Cartesiano, também neste caso um diagrama nos ajuda a encontrar as propriedades básicas das construções funcionais associadas à união disjunta:

$$\begin{tikzcd} A \arrow[r, "i_1"] \arrow[d, "f"'] & A+B \arrow[d, "f+g"'] & B \arrow[l, "i_2"'] \arrow[d, "g"] \\ C \arrow[r, "i_1"] \arrow[dr, "m"'] & C+D \arrow[d, "{[\, m, n \,]}"'] & D \arrow[l, "i_2"'] \arrow[dl, "n"] \\ & E & \end{tikzcd}$$

Repare-se que este diagrama é "semelhante" ao do produto Cartesiano, mas com todas as setas invertidas, projecções substituídas por injecções, "$+$" em lugar de "$\times$" e, finalmente, "$[\, \ldots, \ldots \,]$" em lugar de "$\langle \ldots, \ldots \rangle$" [16]. Ter-se-á:

$$[\, f, h \,] \circ i_1 = f \tag{1.31}$$

$$[\, f, h \,] \circ i_2 = h \tag{1.32}$$

[16] Dois diagramas nestas assim relacionados dizem-se *duais*. Naturalmente que, para preservarmos os mesmos símbolos em ambos os diagramas, o mesmo símbolo de função designa coisas diferentes em diagramas diferentes.

SETS	PASCAL	C/C++	**Descrição informal**
$A \times B$	`record` `P: A;` `S: B` `end;`	`struct {` `A first;` `B second;` `};`	'Records'
$A + B$	`record` `case` `tag: integer` `of x =` `1: (P:A);` `2: (S:B)` `end;`	`struct {` `int tag; /* 1,2 */` `union {` `A ifA;` `B ifB;` `} data;` `};`	'Records' variantes
B^A	`array[A] of B`	`B ...[A]`	'Arrays'
$A + 1$	`^A`	`A *...`	'Pointers'

Figura 1.5: SETS versus Linguagens de Programação.

$$(j + i) \circ i_1 = i_1 \circ j \tag{1.33}$$
$$(j + i) \circ i_2 = i_2 \circ i \tag{1.34}$$

assim como

$$[\, f, h\,] \circ (j + i) = [\, f \circ j, h \circ i\,] \tag{1.35}$$

e como

$$k \circ [\, f, h\,] = [\, k \circ f, k \circ h\,] \tag{1.36}$$

etc.

A união disjunta $A + B$ é extensível à "soma finitária" $A_1 + \cdots + A_n$, para $n \in I\!N$, também designável por $\sum_{j=1}^{n} A_j$, à qual estão associadas tantas injecções i_j quantos os termos envolvidos.

A Figura 1.5 apresenta uma analogia entre as construções primitivas da notação que estamos a definir e a notação para definição de estruturas de dados em linguagens de programação estruturadas, aqui exemplificadas em PASCAL e C.

Exercício 1.12 Demonstre ou refute as igualdades de funções que se seguem, relativas a construções funcionais atrás definidas:

$$(f \circ h) \times (g \circ i) = (f \times g) \circ (h \times i) \tag{1.37}$$
$$1_{A \times B} = 1_A \times 1_B \tag{1.38}$$
$$\langle f, g \rangle \circ h = \langle f \circ h, g \circ h \rangle \tag{1.39}$$
$$\pi_1 \circ \langle f, g \rangle = f \tag{1.40}$$
$$\pi_2 \circ \langle f, g \rangle = g \tag{1.41}$$
$$(f \circ h) + (g \circ i) = (f + g) \circ (h + i) \tag{1.42}$$
$$1_{A+B} = 1_A + 1_B \tag{1.43}$$
$$(j \times i) \circ \langle f, h \rangle = \langle j \circ f, i \circ h \rangle \tag{1.44}$$

□

Construções Derivadas

Com base nestas noções primitivas, podemos progredir para as seguintes construções derivadas:

- os *conjuntos* de elementos de A:

$$2^A \quad \cong \quad \{K \mid \ K \subseteq A\} \tag{1.45}$$

- as *relações binárias* de A para B:

$$2^{A \times B} \tag{1.46}$$

- as *funções parciais* de A para B:

$$A \rightharpoonup B = \bigcup_{K \subseteq A} B^K \tag{1.47}$$

- as *sequências* de elementos de A:

$$A^\star = \bigcup_{n \geq 0} A^{\overline{n}} \tag{1.48}$$

CONJUNTOS E RELAÇÕES

As construções (1.45) e (1.46) são conhecidas da teoria elementar de conjuntos, com os operadores habituais tais como a intersecção ($\cap$), reunião ($\cup$), *etc.*. O operador $card$ (*cardinal*) realiza o cálculo do número de elementos de um conjunto. Sempre que $card(x) = 1$ dizemos que x é um conjunto singular. E sempre que x é um conjunto singular, faz sentido escrever $the(x)$ como designação do único elemento que x contém. Ou seja,

$$the : 2^A \longrightarrow A \tag{1.49}$$

é um operador tal que $the(\{a\}) = a$.

Da mesma forma que de A derivamos 2^A, também de $f : A \longrightarrow B$ derivamos o operador 2^f, dito *imagem dada por* f, da forma seguinte:

$$\begin{array}{rcl} 2^f & : & 2^A \longrightarrow 2^B \\ 2^f(x) & \stackrel{\text{def}}{=} & \{f(a) \mid \ a \in x\} \end{array} \tag{1.50}$$

Ocasionalmente, poderemos escrever $f[x]$ em lugar de $2^f(x)$.

Sempre que r é uma relação em $2^{A\times B}$, r^{-1} designa a inversa de r, *i.é*, a relação em $2^{B\times A}$ definida por

$$r^{-1} = \{\langle b, a\rangle \mid \ \langle a, b\rangle \in r\} \tag{1.51}$$

E a composição de duas relações $r \in 2^{A\times B}$ e $s \in 2^{B\times C}$, designada por $r \circ s$, é a relação

$$r \circ s \stackrel{\text{def}}{=} \{\langle \pi_1(p), \pi_2(q)\rangle \mid \ p \in r \wedge q \in s \wedge \pi_2(p) = \pi_1(q)\} \tag{1.52}$$

No caso geral não temos apenas relações binárias ($2^{A\times B}$) mas sim relações n-árias ($2^{A_1\times\ldots\times A_n}$). Estas são um modelo matemático conveniente para a vulgar apresentação tabular da informação. Por exemplo, à tabela, ou quadro:

A_1	A_2	A_3
a	b	c
a	d	e
f	d	g

(1.53)

corresponde o objecto matemático

$$r \quad = \quad \{\langle a, b, c\rangle, \langle a, d, e\rangle, \langle f, d, g\rangle\}$$

Para relações n-árias será conveniente retermos as seguintes definições dos habituais operadores relacionais de *projecção/selecção*:

$$\begin{aligned} proj \quad &: \quad \overline{n} \times 2^{A_1\times\ldots\times A_n} \longrightarrow \bigcup_{i=1}^{n} 2^{A_i} \\ proj(i, t) &\stackrel{\text{def}}{=} \{\pi_i(r) \mid \ r \in t\} \\ &= \pi_i[t] \\ &= 2^{\pi_i}(t) \end{aligned} \tag{1.54}$$

e

$$\begin{aligned} sel \quad &: \quad (\sum_{i=1}^{n} A_i) \times 2^{A_1\times\ldots\times A_n} \longrightarrow 2^{A_1\times\ldots\times A_n} \\ sel(\langle i, a\rangle, t) &\stackrel{\text{def}}{=} \{r \in t \mid \ a = \pi_i(r)\} \end{aligned} \tag{1.55}$$

Exercício 1.13 Demonstre ou refute as igualdades seguintes relativas a construtores que se referiram acima:

$$2^{(f\circ g)} \quad = \quad 2^f \circ 2^g \tag{1.56}$$

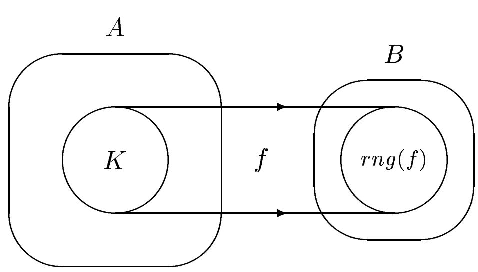


Figura 1.6: Uma função parcial finita $f \in A \rightharpoonup B$

$$
\begin{aligned}
2^f(\emptyset) &= \emptyset && (1.57)\\
2^f(x \cup y) &= 2^f(x) \cup 2^f(y) && (1.58)\\
2^f(x \cap y) &= 2^f(x) \cap 2^f(y) && (1.59)\\
(r \circ s)^{-1} &= s^{-1} \circ r^{-1} && (1.60)\\
r \circ (s \cup t) &= (r \circ s) \cup (r \circ t) && (1.61)\\
r \circ \emptyset &= \emptyset && (1.62)
\end{aligned}
$$

□

FUNÇÕES PARCIAIS FINITAS

Consideremos agora a construção (1.47). Se $f \in A \rightharpoonup B$, então f é uma aplicação $f : K \to B$ para algum $K \subseteq A$, *cf.* Figura 1.6. Ou seja, se $a \in K$, então $f(a) \in B$. E se $a \in A - K$? Que dizer de $f(a)$? Na verdade, a função f está *indefinida* para esses valores. Por isso chamamos *parciais* às funções em $A \rightharpoonup B$, pois não estão *totalmente* definidas sobre A. Dizemos que K é o *domínio* da função f acima,

$$K = dom(f) \tag{1.63}$$

e que f só está definida sobre esse seu domínio. O *contra-domínio* de f será, então, o subconjunto de B definido por

$$rng(f) = \{f(k) \mid \; k \in dom(f)\} \tag{1.64}$$

Como vimos atrás, dada uma qualquer aplicação $f : A \to B$ e $S \subseteq A$, $2^f(S)$ designa a *imagem* de S dada por f. Para funções parciais $f : A \rightharpoonup B$ será preciso garantir que $S \subseteq dom(f)$ e ter-se-á, trivialmente,

$$2^f(dom(f)) \quad = \quad rng(f)$$

Note-se ainda que, se duas funções parciais f e g são *coerentes*, então a sua união $f \cup g$ está também definida de acordo com as definições seguintes:

Definição 1.5 (Funções Coerentes) *Duas funções* $f : A \rightharpoonup B$ *e* $g : C \rightharpoonup D$ *dizem-se coerentes sempre que*

$$\forall a \in dom(f) \cap dom(g) : f(a) = g(a)$$

Nitidamente, se $dom(f) \cap dom(g) = \emptyset$*, então* f *e* g *são coerentes.* □

Definição 1.6 (União de Funções) *Se duas funções* $f : A \rightharpoonup B$ *e* $g : C \rightharpoonup D$ *são coerentes, então é possível definir a sua* união*, que é a função*

$$\begin{array}{rcl} f \cup g & : & A \cup C \rightharpoonup B \cup D \\ f \cup g(a) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcl} f(a) & \Leftarrow & a \in dom(f) \\ g(a) & \Leftarrow & a \in dom(g) \end{array} \right. \end{array}$$

□

Para quaisquer f e g (*e.g.* não-coerentes) é costume definir uma generalização da união, a *sobreposição* $f \dagger g$, como se segue:

$$f \dagger g(a) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lcl} g(a) & \Leftarrow & a \in dom(g) \\ f(a) & \Leftarrow & a \in (dom(f) - dom(g)) \end{array} \right. \tag{1.65}$$

Vê-se pois, que, para argumentos onde f e g entram em conflito, os valores de g sobrepoem-se aos de f. No limite, uma função parcial f poderá estar totalmente indefinida, ou seja, $dom(f) = \emptyset$; só existe uma função f nestas condições — a aplicação vazia

$$(\quad)$$

— que se costuma também representar por [ ].

Exercício 1.14 Demonstre a validade dos factos (A.31) a (A.43), que constam do apêndice A, e que se referem aos operadores de funções parciais finitas que se acabam de referir.
□

As funções parciais podem também ser definidas por extensão, *e.g.*

$$\begin{pmatrix} a & b & c \\ 1 & 2 & 3 \end{pmatrix}$$

ou por compreensão, *e.g.*

$$\begin{pmatrix} i \\ i+1 \end{pmatrix}_{i \in n}$$

Dois operadores úteis sobre funções ϕ em $A \rightharpoonup B$ são os de restrição, um dito de *restrição positiva*,

$$\phi \mid S \stackrel{\mathrm{def}}{=} \begin{pmatrix} a \\ \phi(a) \end{pmatrix}_{a \in dom(\phi) \cap S} \tag{1.66}$$

e outro dito de *restrição negativa*,

$$\phi \setminus S \stackrel{\mathrm{def}}{=} \begin{pmatrix} a \\ \phi(a) \end{pmatrix}_{a \in dom(\phi) - S} \tag{1.67}$$

Da mesma forma que de dois conjuntos A e B derivamos a construção $A \rightharpoonup B$, também, dadas duas aplicações $f : A \longrightarrow C$ e $g : B \longrightarrow D$, podemos ainda definir a construção $f \rightharpoonup g$, como se segue:

$$\begin{array}{rcl} f \rightharpoonup g & : & (A \rightharpoonup B) \longrightarrow (C \rightharpoonup D) \\ (f \rightharpoonup g)(\sigma) & \stackrel{\mathrm{def}}{=} & \begin{pmatrix} f(a) \\ g(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)} \end{array} \tag{1.68}$$

A única restrição é que f seja injectiva, por forma a garantir que $f \rightharpoonup g$ seja uma função e não uma relação [17]. Aceitaremos $A \rightharpoonup g$ e $f \rightharpoonup B$ como abreviaturas de, respectivamente, $1_A \rightharpoonup g$ e $f \rightharpoonup 1_B$ [18]. Logo

$$(A \rightharpoonup g)(\sigma) \stackrel{\mathrm{def}}{=} \begin{pmatrix} a \\ g(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)} \tag{1.69}$$

e

$$(f \rightharpoonup B)(\sigma) \stackrel{\mathrm{def}}{=} \begin{pmatrix} f(a) \\ \sigma(a) \end{pmatrix}_{a \in dom(\sigma)} \tag{1.70}$$

Exercício 1.15 Demonstre a validade dos factos (A.44) a (A.64) — ver apêndice A — sobre os operadores de restrição definidos por (1.66) e (1.67).
□

[17] Mas veja-se o Exercício 1.51 a este respeito.

[18] Toda esta notação e suas construções e "abreviaturas" encontra a sua explicação na noção de *functor* entre duas categorias que a secção 8.6 trará, a seu tempo, à consideração do leitor.

Exercício 1.16 Mostre que a coerência entre duas funções f e g (definição 1.5) pode ser definida pela relação

$$f \Delta g \stackrel{\mathrm{def}}{=} f \mid dom(g) = g \mid dom(f) \tag{1.71}$$

Será Δ uma relação transitiva? Justifique.

□

Exercício 1.17 Demonstre ou refute as seguintes igualdades, onde letras maiúsculas designam espécies de dados e minúsculas designam funções:

$$\begin{aligned}
A \rightharpoonup (f \circ g) &= (A \rightharpoonup f) \circ (A \rightharpoonup g) & (1.72)\\
dom \circ (f \rightharpoonup A) &= 2^f \circ dom & (1.73)\\
rng \circ (A \rightharpoonup f) &= 2^f \circ rng & (1.74)\\
dom \circ (A \rightharpoonup f) &= dom & (1.75)\\
rng \circ (i \rightharpoonup B) &= rng & (1.76)\\
(A \rightharpoonup f)(\sigma \mid S) &= ((A \rightharpoonup f)(\sigma)) \mid S & (1.77)\\
(A \rightharpoonup f)(\sigma_1 \cup \sigma_2) &= (A \rightharpoonup f)(\sigma_1) \cup (A \rightharpoonup f)(\sigma_2) & (1.78)\\
(f \rightharpoonup D) \circ (A \rightharpoonup g) &= f \rightharpoonup g & (1.79)\\
(A \rightharpoonup f)(\sigma) \dagger (A \rightharpoonup f)(\tau) &= (A \rightharpoonup f)(\sigma \dagger \tau) & (1.80)\\
A \rightharpoonup (1_B) &= 1_{A \rightharpoonup B} & (1.81)
\end{aligned}$$

□

SEQUÊNCIAS FINITAS

A construção (1.48) permite-nos ver sequências como sendo aplicações de um segmento inicial de $I\!N$ num dado conjunto A. Por exemplo, se $A = \{0, 1\}$, então a sequência

$$<0, 0, 1, 0> \in A^{\star}$$

"é", no fundo, a aplicação $s \in A^4$ seguinte:

$$s = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Um caso particular de sequência é a sequência vazia $<> \in A^0$, ie.

$$<> = (\quad)$$

As sequências acima foram descritas *por extensão*, à semelhança do que é habitual fazer com conjuntos, *e.g.* $\{0, 0, 1, 0\}$. Mas é óbvio que $<0, 0, 1, 0> \neq \{0, 0, 1, 0\} = \{0, 1\}$. Da mesma maneira que um conjunto pode ser descrito *por compreensão*, eg.

$$\{f(x) \mid \; x \in A \wedge p(x)\}$$

onde $p : A \rightarrow \{V, F\}$, também uma sequência o pode, usando-se a notação

$$<f(x) \mid x \leftarrow s \wedge p(x)> \tag{1.82}$$

onde $s \in A^\star$ é uma sequência. A expressão (1.82) designa a transformação por f da sub-sequência de s cujos elementos satisfazem p. A abreviatura $<f(x) \mid x \leftarrow s>$ pode usar-se sempre que $p(x) = V$, para todo o elemento de x de s. Note-se que, como s é uma aplicação, podemos concluir que

$$<f(x) \mid x \leftarrow s> = f \circ s \tag{1.83}$$

onde "$\circ$" designa a *composição de aplicações*. Mais ainda, podemos igualar (1.83) a $f^*(s)$, onde f^*, para $f : A \longrightarrow B$, é a construção que estende A^* a aplicações:

$$\begin{array}{rcl} f^* & : & A^* \longrightarrow B^* \\ f^*(s) & \stackrel{\text{def}}{=} & <f(a) \mid a \leftarrow s> \end{array} \tag{1.84}$$

Alguns operadores sobre sequências são úteis, como por exemplo:

1. $length : A^\star \longrightarrow I\!N_0$ que dá o comprimento de uma sequência; se $s \in A^n$ então $length(s) = n$.

2. $head : A^\star \longrightarrow A$ que dá a cabeça de uma sequência $s \in A^n$ desde que $n \geq 1$; é claro que $head(s) = s(1)$.

3. $tail : A^\star \longrightarrow A^\star$ que dá o resto de uma sequência $s \in A^n$ desde que $n \geq 1$; teremos que

$$tail(s) = \left(\begin{array}{cccccc} 1 & 2 & \ldots & i & \ldots & n-1 \\ s(2) & s(3) & \ldots & s(i+1) & \ldots & s(n) \end{array} \right)$$

4. $cons : A \times A^\star \longrightarrow A^\star$ que coloca um dado $a \in A$ à cabeça de uma sequência $s \in A^n$; assim, $cons(a, s) \in A^{n+1}$ e

$$cons(a, s) = \left(\begin{array}{cccccc} 1 & 2 & \ldots & i & \ldots & n+1 \\ a & s(1) & \ldots & s(i-1) & \ldots & s(n) \end{array} \right) \tag{1.85}$$

5. $elems : A^\star \rightarrow 2^A$ que dá o conjunto de todos os elementos presentes numa sequência s, ou seja

$$elems(s) = \{s(i) \mid \ 1 \leq i \leq length(s)\}$$

Destas construções resultam propriedades como as que se listam no apêndice A, secção A.4, e que serão úteis mais tarde.

Prioridades

Para simplificar as expressões em $Sets$ por diminuição do número de parênteses, convenciona-se a seguinte prioridade dos operadores:

$\star$
$\_^{-}$
$\times$
$\rightharpoonup$
$+$

Assim, por exemplo, a expressão

$$X \rightharpoonup A^2 \times B^\star + C$$

abrevia a expressão

$$(X \rightharpoonup ((A^2) \times (B^\star))) + C$$

etc.

1.3.2 Semântica por Modelos

Como acima antecipamos, o cálculo do significado de uma frase gerada por uma assinatura Σ far-se-á, nesta técnica semântica, recorrendo à teoria "ingénua" dos conjuntos finitos. Antes de mais, torna-se necessária a seguinte definição.

Definição 1.7 (Σ-álgebra) *Seja $\Sigma : \Omega \to S^\star \times S$ uma dada assinatura. Diremos que o par $\mathcal{A} = (\mathcal{A}_S, \mathcal{A}_\Omega)$ é uma Σ-álgebra desde que:*

1. *A_S seja uma aplicação de S em conjuntos finitos de* Sets*;*
2. *A_Ω seja uma aplicação de Ω em funções (em* Sets*) de conjuntos para conjuntos, e*
   (a) *para cada $\sigma \in \Omega$ tal que $\Sigma(\sigma) = \langle <s_1, \ldots, s_n>, s\rangle$, se verifica que a função em* Sets *correspondente, $\mathcal{A}_\Omega(\sigma)$, "respeita a funcionalidade de σ", quer dizer:*

$$\mathcal{A}_\Omega(\sigma) : \mathcal{A}_S(s_1) \times \mathcal{A}_S(s_2) \times \ldots \times \mathcal{A}_S(s_n) \to \mathcal{A}_S(s) \qquad (1.86)$$

ou seja, verifica-se o diagrama que se segue:

$$\begin{array}{ccccccccc} \sigma & : & s_1 & \times & \ldots & \times & s_n & \to & s \\ \downarrow & & \downarrow & & & & \downarrow & & \downarrow \\ \mathcal{A}(\sigma) & : & \mathcal{A}(s_1) & \times & \ldots & \times & \mathcal{A}(s_n) & \to & \mathcal{A}(s) \end{array}$$

Sempre que não resultar confusão, omitiremos os índices Ω *e* S *em* $\mathcal{A}_\Omega$ *e* $\mathcal{A}_S$*, respectivamente. Por exemplo, a expressão* (1.86) *pode escrever-se, sem ambiguidade,*

$$\mathcal{A}(\sigma) : \mathcal{A}(s_1) \times \ldots \times \mathcal{A}(s_n) \to \mathcal{A}(s)$$

pois sabemos que $\sigma \in \Omega$ *e* $s_1 \in S$*,* $s_2 \in S, \ldots$ etc.. $\square$

É imediato apercebermo-nos que uma dada Σ-álgebra $\mathcal{A}$ *dá significado* a Σ, na medida em que:

- $\mathcal{A}_S(s)$ indica, para toda a espécie $s \in S$, qual o conjunto de objectos a cuja espécie se deu o nome "s". Por exemplo, ao declararmos

$$\mathcal{A}_S(Quantia) = \mathbb{N}_0$$

  (*cf.* Σ_{SGIB} acima) estamos a dizer que o "significado" de uma *Quantia* é o número que representa o seu valor numa dada unidade monetária.

- $\mathcal{A}_\Omega(\sigma)$ indica a função "significado" atribuída a um determinado símbolo de operação. Por exemplo, se tivermos $\mathcal{A}_S(Bool) = \{0, 1\}$, podemos definir

$$\mathcal{A}_\Omega(\neg) = \begin{array}{c|c} 0 & 1 \\ \hline\hline 1 & 0 \end{array}$$

  para indicar que "$\neg$" é o símbolo escolhido para designar a operação de *complementação* entre booleanos. Podíamos ter escolhido "neg" ou "not" em lugar de "$\neg$". Assim como podíamos ter arbitrado

$$\mathcal{A}_\Omega(\neg) = \begin{array}{c|c} 0 & 1 \\ \hline\hline 0 & 0 \end{array}$$

  mas neste caso o significado dado a $\neg$ é contra-intuitivo, pois não coincide com a "carga semântica" de $\neg$. Relembra-se que a única restrição formalmente imposta a $\mathcal{A}_\Omega(\neg)$ é que, sendo

$$\neg : Bool \to Bool$$

  então $\mathcal{A}_\Omega(\neg)$ terá que ter a funcionalidade:

$$\mathcal{A}_\Omega(\neg) : \{0, 1\} \to \{0, 1\}$$

  cf. a expressão (1.86).

Sempre que, para darmos significado a uma dada assinatura Σ, definimos uma dada Σ-álgebra $\mathcal{A}$, dizemos que damos um *modelo* de Σ. Por isso esta técnica de semântica algébrica se designa, normalmente, por semântica *por modelos* ('model-oriented') [19].

Vejamos um exemplo, voltando à assinatura Σ_{SGIB} (1.4) construída anteriormente. Por inspecção, é fácil verificar que a seguinte definição de um modelo $\mathcal{A}$ é de facto uma Σ_{SGIB}-álgebra, qualquer que seja o preenchimento das reticências:

$$
\begin{array}{rcl}
\mathsf{A}_S(Quantia) & \stackrel{\mathrm{def}}{=} & I\!N_0 \\
\mathsf{A}_S(Titular) & \stackrel{\mathrm{def}}{=} & \ldots \\
\mathsf{A}_S(IdConta) & \stackrel{\mathrm{def}}{=} & \ldots \\
\mathsf{A}_S(Sistema) & \stackrel{\mathrm{def}}{=} & \mathcal{A}(IdConta) \rightharpoonup (2^{\mathcal{A}(Titular)} \times \mathcal{A}(Quantia)) \\
\mathsf{A}_\Omega(inic) & \stackrel{\mathrm{def}}{=} & \left(\quad \right) \\
\mathsf{A}_\Omega(abertura) & \stackrel{\mathrm{def}}{=} & \lambda(k,t,\sigma).\left\{ \begin{array}{ccl} k \in dom(\sigma) & \Rightarrow & \sigma \\ \neg(k \in dom(\sigma)) & \Rightarrow & \sigma \cup \left(\begin{array}{c} k \\ \langle \{t\}, 0 \rangle \end{array} \right) \end{array} \right. \\
\mathsf{A}_\Omega(fecho) & \stackrel{\mathrm{def}}{=} & \lambda(k,\sigma).\sigma \setminus \{k\} \\
\mathsf{A}_\Omega(\textit{depósito}) & \stackrel{\mathrm{def}}{=} & \lambda(k,q,\sigma). \\
& & \left\{ \begin{array}{ccll} k \in dom(\sigma) & \Rightarrow & let & x = \sigma(k) \\ & & & t_k = \pi_1(x) \\ & & & q_k = \pi_2(x) \\ & & in & \sigma \dagger \left(\begin{array}{c} k \\ \langle t_k, q_k + q \rangle \end{array} \right) \\ \neg(k \in dom(\sigma)) & \Rightarrow & \sigma & \end{array} \right. \\
\mathsf{A}_\Omega(\textit{co-titular}) & \stackrel{\mathrm{def}}{=} & \lambda(k,t,\sigma). \\
& & \left\{ \begin{array}{ccll} k \in dom(\sigma) & \Rightarrow & let & x = \sigma(k) \\ & & & t_k = \pi_1(x) \\ & & & q_k = \pi_2(x) \\ & & in & \sigma \dagger \left(\begin{array}{c} k \\ \langle t_k \cup \{t\}, q_k \rangle \end{array} \right) \\ \neg(k \in dom(\sigma)) & \Rightarrow & \sigma & \end{array} \right. \\
\mathsf{A}_\Omega(levantamento) & \stackrel{\mathrm{def}}{=} & \lambda(k,q,\sigma). \\
& & \left\{ \begin{array}{ccll} k \in dom(\sigma) & \Rightarrow & let & x = \sigma(k) \\ & & & t_k = \pi_1(x) \\ & & & q_k = \pi_2(x) \\ & & in & \left\{ \begin{array}{ccl} q \leq q_k & \Rightarrow & \sigma \dagger \left(\begin{array}{c} k \\ \langle t_k, q_k - q \rangle \end{array} \right) \\ q > q_k & \Rightarrow & \sigma \end{array} \right. \\ \neg(k \in dom(\sigma)) & \Rightarrow & \sigma & \end{array} \right. \\
\mathsf{A}_\Omega(\textit{balanço}) & \stackrel{\mathrm{def}}{=} & \lambda(k,\sigma).\left\{ \begin{array}{ccl} k \in dom(\sigma) & \Rightarrow & \pi_2(\sigma(k)) \\ \neg(k \in dom(\sigma)) & \Rightarrow & 0 \end{array} \right.
\end{array}
$$

[19] A designação "semântica denotacional" será também utilizada, ver mais à frente a Secção 1.3.4.

Exercício 1.18 Relativamente à Σ_{SGIB}-álgebra $\mathcal{A}$ acima,

1. Explique, por paravras suas, a semântica de cada operação.

2. Repare que *balanço* não distingue uma conta inexistente de uma conta existente com saldo 0. Reveja a álgebra $\mathcal{A}$ por forma a tal acontecer.

□

Exercício 1.19 Construa um modelo semântico — *i.é* uma álgebra $\mathcal{A}$ — para a assinatura Σ_G obtida no Exercício 1.4.

□

Exercício 1.20 Uma tabela relacional é um conjunto de tuplos, sendo um tuplo uma atribuição de valores a atributos. Podem existir atributos omissos, por exemplo o atributo C no segundo tuplo desta tabela,

A	B	C
$a1$	$b1$	$c1$
$a1$	$b2$	
$a2$	$b1$	$c3$
$a2$	$b1$	$c1$
$a1$		$c1$
$a3$		$c1$
	$b1$	$c2$

(1.87)

e outros.

Pretende-se especificar o problema da geração de histogramas relativos aos atributos de uma dada tabela. Por exemplo, dada a tabela 1.87, têm-se os seguinte histogramas para os atributos A, B e C, respectivamente:

Especifique um modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ para a seguinte assinatura,

$$\Sigma = \begin{cases} valores : AtrId \times Tabela \rightarrow Valores \\ atributos : Tabela \rightarrow AtrIds \\ contagem : AtrId \times Valor \times Tabela \rightarrow Nat \\ histograma : AtrId \times Tabela \rightarrow Histograma \end{cases}$$

que pretende exibir a funcionalidade que se pretende e de acordo com a semântica informal seguinte:

$valores(a,t)$ — deverá retornar o conjunto de todos os valores que ocorrem na coluna a da tabela t, onde a designa o identificador do respectivo atributo;

$atributos(t)$ — deverá retornar o conjunto de todos os atributos que ocorrem em pelo menos um tuplo da tabela;

$contagem(a,v,t)$ — calcula o número de ocorrências de um dado valor v na coluna a de uma dada tabela t;

$histograma(a,t)$ — fornece o histograma do atributo a na tabela t.

□

1.3.3 Classes de Modelos e sua Interpretação

Para tornarmos explícita a assinatura Σ de que uma dada álgebra $\mathcal{A}$ é modelo, escreveremos sempre

$$\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$$

Os seguintes dois casos particulares de Σ-álgebras terão um interesse especial.

Definição 1.8 (Modelo Trivial) *Seja $\Sigma : \Omega \to S^\star \times S$ uma assinatura. À Σ-álgebra $\mathcal{T}$ tal que, para todo o $s \in S$*

$$\mathcal{T}_S(s) = \{1\}$$

(onde $1 \in I\!N$), e para todo o $\sigma : s_1 \times \ldots \times s_n \to s$ em Ω,

$$\begin{array}{rcl} \mathcal{T}_\Omega(\sigma) & : & \{1\}^n \to \{1\} \\ \langle x_1, \ldots, x_n \rangle & \leadsto & 1 \end{array}$$

dá-se o nome de modelo trivial de Σ. □

Definição 1.9 (Modelo Inicial) *Seja $\Sigma : \Omega \to S^\star \times S$ uma assinatura. Seja $\mathcal{W}$ a Σ-álgebra tal que*

$$\mathcal{W}_S(s) = W_{\Sigma,s}$$

para todo $s \in S$ e, para $\sigma \in \Omega$ tal que $\sigma : s_1 \times \ldots \times s_n \to s$,

$$\begin{array}{rcl} \mathcal{W}_\Omega(\sigma) & : & W_{\Sigma,s_1} \times \ldots \times W_{\Sigma,s_n} \to W_{\Sigma,s} \\ \langle t_1, \ldots, t_n \rangle & \leadsto & \sigma(t_1, \ldots, t_n) \end{array}$$

Facilmente se entende que cada operador de $\mathcal{W}$ é uma operação de construção sintática de termos a partir de sub-termos sintaticamente compatíveis. À álgebra $\mathcal{W}$ damos o nome de modelo inicial *de Σ, por razões que se entenderão futuramente.* $\square$

Duas Σ-álgebras podem estar relacionadas entre si pelo conceito seguinte:

Definição 1.10 (Σ-homomorfismo) *Sejam $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ e $\mathsf{B} : \Sigma \longrightarrow \mathbf{Set}$ duas Σ-álgebras, para $\Sigma : \Omega \to S^\star \times S$. Seja $h = (h_s)_{s \in S}$ uma família de funções em* Sets *com as seguintes propriedades:*

- *para cada $s \in S$,*
$$h_s : \mathcal{A}(s) \to \mathcal{B}(s)$$
- *para cada $\sigma \in \Omega$ tal que $\sigma : s_1 \times \ldots \times s_n \to s$, verifica-se que h "respeita" a funcionalidade de σ, ou seja*
$$h_s(\mathcal{A}(\sigma)(x_1, \ldots, x_n)) = \mathcal{B}(\sigma)(h_{s_1}(x_1), \ldots, h_{s_n}(x_n)) \tag{1.88}$$
cf. *o diagrama:*
$$\begin{array}{ccccccccc} \mathcal{A}(\sigma) & : & \mathcal{A}(s_1) & \times & \ldots & \times & \mathcal{A}(s_n) & \to & \mathcal{A}(s) \\ & & \downarrow h_{s_1} & & & & \downarrow h_{s_n} & & \downarrow h_s \\ \mathcal{B}(\sigma) & : & \mathcal{B}(s_1) & \times & \ldots & \times & \mathcal{B}(s_n) & \to & \mathcal{B}(s) \end{array}$$

Então h diz-se ser um homomorfismo *de $\mathcal{A}$ para $\mathcal{B}$. Escreveremos $h : \mathcal{A} \to \mathcal{B}$ para designar que h é um homomorfismo de $\mathcal{A}$ e $\mathcal{B}$.* $\square$

Definição 1.11 (Σ-interpretação) *Seja, na definição anterior, $\mathcal{A} = \mathcal{W}$. Então cada*
$$h_s : W_{\Sigma,s} \to \mathcal{B}(s)$$
converte termos *de espécie s em* valores *de $\mathcal{B}(s)$ e é tal que*
$$h_s(\sigma(x_1, \ldots, x_n)) = \mathcal{B}(\sigma)(h_{s_1}(x_1), \ldots, h_{s_n}(x_n))$$
Quer dizer, cada termo $\sigma(x_1, \ldots, x_n)$ é convertido num valor de $\mathcal{B}(s)$ de forma estrutural*, ou seja, combinando via $\mathcal{B}(\sigma)$ os valores $h_{s_i}(x_i)$ em que foram convertidos todos os seus subtermos t_i ($1 \leq i \leq n$).*

A este caso particular de homomorfismo damos o nome de interpretação [20] *de Σ-termos numa álgebra $\mathcal{B}$.* $\square$

[20] Por vezes também designado *regra de cálculo*, ou de *tradução*.

Sempre que, para todo o $s \in S$, a função h_s de um homomorfismo $h : \mathcal{A} \to \mathcal{B}$ é sobrejectiva (em $\mathcal{B}(s)$) dizemos que h é um *epimorfismo*; se, além disso, h_s for *injectiva*, então h recebe o nome de *isomorfismo*.

Vamos estar particularmente interessados na classe de Σ-modelos $\mathcal{B}$ tais que $h : \mathcal{W} \to \mathcal{B}$ é um epimorfismo. Nestes casos diremos que $\mathcal{B}$ é uma álgebra *gerada (indutivamente) por termos*, no sentido em que, para um qualquer valor $v \in \mathcal{B}(s)$ $(s \in S)$ existe — pelo argumento de sobrejectividade de $h_s : W_{\Sigma,s} \to \mathcal{B}(s)$ — um termo $t \in W_{\Sigma,s}$ tal que

$$v = h_s(t)$$

Um caso particular de álgebra gerada por termos é a própria álgebra dos termos $\mathcal{W}$.

Para álgebras geradas por termos existe um método muito útil de demonstração de factos, que recebe o nome de *indução algébrica*. Este método de indução é um caso particular do método de *indução estrutural* sobre uma ordem *bem-fundada* [21] e consiste em provar a validade de um facto f, qualquer [22]:

$$\forall v \in \mathcal{B} : f(v) \text{ é verdadeiro}$$

convertendo-o em

$$\forall t \in \mathcal{W} : f(h(t)) \text{ é verdadeiro} \tag{1.89}$$

onde $h : \mathcal{W} \to \mathcal{B}$ é um epimorfismo — que é único, aliás [23] — e usar a estratégia seguinte:

1. *Casos de Base*: $t = \sigma$.

   Provar o facto (1.89) para todo o $t = \sigma$, onde σ é uma Σ-constante;

2. *Salto Indutivo*: $t = \sigma(t_1, \ldots, t_n)$, para $\sigma \in \Omega$ tal que $\sigma : s_1 \times \ldots \times s_n \to s$, e $t_i \in \mathcal{W}(s_i)$, para todo o $1 \leq i \leq n$.

   (a) *Hipótese de Indução*: supor que $f(h_{s_i}(t_i))$ é verdadeiro para $1 \leq i \leq n$;

   (b) Demonstrar que a hipótese (2a) é suficiente para provar

   $$f(h_s(\sigma(t_1, \ldots, t_n)))$$

□

Vejamos um exemplo na demonstração do próximo teorema.

[21] Ver Exercício 1.22.

[22] f será, normalmente, uma família S-indexada de predicados, como se verá em exemplos mais adiante.

[23] Como se verá adiante no Teorema 1.1.

Teorema 1.1 (Unicidade de Σ-intrepretações) *Seja* $\mathsf{B} : \Sigma \longrightarrow \mathbf{Set}$ *uma* Σ-*álgebra. Então o homomorfismo*

$$h : \mathcal{W} \rightarrow \mathcal{B}$$

é único.

Demonstração: *Basta provar que outro qualquer homomorfismo* $h' : \mathcal{W} \rightarrow \mathcal{B}$ *coincide com* h*, ou seja que, para todo o* $s \in S$*, a função* h'_s *é a mesma função que* h_s*. Como tanto* h_s *como* h'_s *terão que estar definidos para todo o* $t \in W_{\Sigma,s}$*, basta provar que* $h_s(t) = h'_s(t)$*, o que se pode fazer usando indução estrutural:*

1. Casos de Base*:* $t = \sigma$ *para* $\sigma : \rightarrow s$*.*
   Teremos que, sabendo que h *e* h' *são* Σ*-homomorfismos,*

   $$h_s(\mathcal{W}(\sigma)) = h_s(\sigma) = \mathcal{B}(\sigma)$$

   $$h'_s(\mathcal{W}(\sigma)) = h'_s(\sigma) = \mathcal{B}(\sigma)$$

   pela equação (1.88). Logo $h_s(t) = h'_s(t)$ *nestes casos.*

2. Salto Indutivo*:* $t = \sigma(t_1, \ldots, t_n)$
   (a) Hipótese de indução*:* $\forall 1 \leq i \leq n : h_{s_i}(t_i) = h'_{s_i}(t_i)$
   (b) *Calculemos* $h_s(\sigma(t_1, \ldots, t_n))$*. Teremos, pela equação (1.88),*

   $$h_s(\sigma(t_1, \ldots, t_n)) = \underbrace{\mathcal{B}(\sigma)(h_{s_1}(t_1), \ldots, h_{s_n}(t_n))}_{A}$$

   De igual forma,

   $$h'_s(\sigma(t_1, \ldots, t_n)) = \underbrace{\mathcal{B}(\sigma)(h'_{s_1}(t_1), \ldots, h'_{s_n}(t_n))}_{B}$$

   Ora, pela hipótese de indução, A *é igual a* B*; logo, como queríamos,*

   $$h_s(\sigma(t_1, \ldots, t_n)) = h'_s(\sigma(t_1, \ldots, t_n))$$

Q.E.D.
□

Exercício 1.21 Indique em que condições é que o único homomorfismo $h : \mathcal{W} \rightarrow \mathcal{T}$, onde $\mathcal{T}$ é o modelo trivial (Definição 1.8) é um epimorfismo.
□

Exercício 1.22 Uma ordem $<$ sobre um conjunto A diz-se *bem-fundada* sse

$$\forall \emptyset \subset C \subseteq A : (\exists m \in C : <_m \cap C = \emptyset)$$

onde $<_m$ designa o conjunto

$$<_m = \{a \in A \mid a < m\}$$

Sobre uma estrutura bem-fundada $(A; <)$ é possível demonstrar factos

$$\forall a \in A : p(a)$$

usando o seguinte método, dito de *indução estrutural*:

1. *Casos de Base*: provar $p(a)$ para todos os $a \in A$ tais que $<_a = \emptyset$.
2. *Salto Indutivo*: Seja $a \in A$ tal que $<_a \supset \emptyset$.

   (a) *Hipótese de indução*:
   $$\forall x < a : p(x)$$

   (b) *Passo*:
   $$(\forall x < a : p(x)) \Rightarrow p(a)$$

□

Neste contexto,

1. mostre que, em $I\!N$, a ordem $x \preceq y$ definida por
   $$x < y \textit{ e } x \textit{ é divisor inteiro de } y$$
   é bem-fundada, e caracterize o respectivo método de indução estrutural;
2. caracterize a ordem bem-fundada sobre $\mathcal{W}$ que está por detrás do método de indução algébrica.

□

O Teorema 1.1 é muito relevante pois permite-nos associar a cada Σ-álgebra $\mathcal{A}$ o único homomorfismo que traduz Σ-termos em valores de $\mathcal{A}$. Designaremos esse homomorfismo por $h_{\mathcal{A}}$ ou, quando daí não resultar ambiguidade, simplesmente por $\mathcal{A}$. Neste contexto, sempre que t é um Σ-termo, $h_{\mathcal{A}}(t)$ ou $\mathcal{A}(t)$ designarão a interpretação de t em $\mathcal{A}$.

Exercício 1.23 Relativamente ao modelo $\mathcal{A}$ definido na secção 1.3.2, calcule as seguintes interpretações:

$$\begin{array}{rl} i) & \mathcal{A}(init) \\ ii) & \mathcal{A}(abertura(i,t,init)) \\ iii) & \mathcal{A}(balanço(i,(abertura(i,t,init)))) \\ iv) & \mathcal{A}(fecho(i,abertura(i,t,init))) \end{array}$$

onde $i \in W_{\Sigma,IdConta}$ e $t \in W_{\Sigma,Titular}$.

□

Pode perfeitamente acontecer que dois termos distintos t_1 e t_2 possam ser tais que $\mathcal{A}(t_1) = \mathcal{A}(t_2)$, ou seja, tenham a mesma interpretação em $\mathcal{A}$. Como $\mathcal{A}$ (encarada como homomorfismo) dá significado a todos os Σ-termos, então diremos que t_1 e t_2 têm, em $\mathcal{A}$, o mesmo significado. Por exemplo, sejam

$$\begin{array}{rcl} \mathcal{A}(M) & = & I\!\!N \\ \mathcal{A}(\epsilon) & = & 1 \\ \mathcal{A}(\theta) & = & \lambda(x,y).x \times y \end{array}$$

as cláusulas que definem o monóide $\mathcal{A} = (I\!\!N; \times, 1)$ dos naturais sob a operação de multiplicação. Dados os termos

$$\begin{array}{rcl} t_1 & = & \epsilon \\ t_2 & = & \epsilon\theta\epsilon \end{array}$$

teremos

$$\mathcal{A}(t_1) = \mathcal{A}(\epsilon) = 1$$

e

$$\begin{array}{rcl} \mathcal{A}(t_2) & = & \mathcal{A}(\epsilon\theta\epsilon) \\ & = & \mathcal{A}(\theta)(\mathcal{A}(\epsilon), \mathcal{A}(\epsilon)) \\ & = & \mathcal{A}(\epsilon) \times \mathcal{A}(\epsilon) \\ & = & 1 \times 1 \\ & = & 1 \end{array}$$

Logo, os termos ϵ e $\epsilon\theta\epsilon$ têm o mesmo significado.

"Ter o mesmo significado" é uma *relação de equivalência* sobre W_Σ, que se pode definir por:

$$t \cong_{\mathcal{A}} t' \stackrel{\text{def}}{=} \mathcal{A}(t) = \mathcal{A}(t') \tag{1.90}$$

Portanto, $\cong_{\mathcal{A}}$ é simplesmente a relação-núcleo ('kernel') associada ao homomorfismo $h_{\mathcal{A}} : \mathcal{W} \rightarrow \mathcal{A}$ [24]. Essa relação é, aliás, uma congruência, pois é compatível com todos os operadores de Σ; porque estamos a trabalhar com álgebras heterógeneas, só podemos entender as frases anteriores se for estendido o conceito de congruência através das definições que se seguem.

Definição 1.12 (Σ-compatibilidade) *Seja $\Sigma : \Omega \rightarrow S^\star \times S$ uma assinatura, e* $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ *uma Σ-álgebra. Uma família de relações binárias sobre $\mathcal{A}$,* i.é

$$\varphi = (\varphi_s)_{s \in S}$$

[24] Isto é, $\cong_{\mathcal{A}}$ designa a mesma coisa que $K/h_{\mathcal{A}}$.

tal que $\varphi_s \subseteq \mathcal{A}(s) \times \mathcal{A}(s)$, diz-se Σ-compatível se φ for compatível com todos os Σ-operadores $\sigma : s_1 \times \ldots s_n \to s$, i.é *dados $a_i, a'_i \in \mathcal{A}(s_i)$, para $1 \leq i \leq n$, então*

$$(\forall 1 \leq i \leq n : a_i \varphi_{s_i} a'_i) \Rightarrow \mathcal{A}(\sigma)(a_1, \ldots, a_n) \varphi_s \mathcal{A}(\sigma)(a'_1, \ldots, a'_n)$$

□

A noção de compatibilidade de uma relação (de ordem, de equivalência, *etc.*) face a um operador traduz a ideia intuitiva de que "coisas relacionadas entre si, operadas pelas mesmas funções, dão resultados também relacionados entre si". É uma noção primitiva em Matemática, que torna possível técnicas de raciocínio estruturado a que nos habituamos já sem pensar nelas. Por exemplo, o método de redução para a resolução de sistemas de inequações, *e.g.*

$$\begin{array}{rcl} x+y & \leq & 2x \\ -y & \leq & 37 \\ \hline x & \leq & 2x+37 \end{array}$$

só é válido porque a *soma* $(+)$ é compatível com a relação *menor do que* $(\leq)$.

Os seguintes casos particulares de famílias de relações $(\varphi_s)_{s \in S}$ são sempre compatíveis com qualquer assinatura $\Sigma : \Omega \to S^\star \times S$:

- a relação *vazia* (*i.é* tal que $\varphi_s = \emptyset$ para todo o s);
- a relação *universal* (*i.é* tal que $\varphi_s = \mathcal{A}(s)^2$ para todo o s);
- a relação *identidade* (*i.é* tal que φ_s é a igualdade em $\mathcal{A}(s)$).

Os dois últimos casos são já Σ-congruências, de acordo com a definição que se segue:

Definição 1.13 (Σ-congruência) *Seja $\varphi = (\varphi_s)_{s \in S}$ uma família (de relações binárias) sobre uma álgebra $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$, compatível com a assinatura $\Sigma : \Omega \to S^\star \times S$. Se qualquer φ_s for uma relação de equivalência, então φ diz-se ser uma Σ-congruência.*
□

É uma Σ-congruência sobre W_Σ a relação-núcleo

$$\cong_{\mathcal{A}} = (\cong_{\mathcal{A},s})_{s \in S}$$

do homomorfismo $h_{\mathcal{A}} : \mathcal{W} \to \mathcal{A}$, *i.é* a relação que já foi definida pela expressão (1.90). Esta relação vai permitir-nos pensar na divisão de $\mathcal{W}$ em classes de congruência induzidas por $\mathcal{A}$. De facto, se uma relação de equivalência φ sobre um conjunto A conduz ao quociente A/φ,

$$A/\varphi = \{[a] \mid a \in A\}$$

onde

$$[a] = \{b \in A \mid a\varphi b\}$$

é uma classe de equivalência, também uma Σ-congruência $\cong$ sobre uma Σ-álgebra $\mathcal{A}$ conduz à álgebra-quociente $\mathcal{A}/\cong$, de acordo com a definição que se segue:

Definição 1.14 (Álgebra Quociente) *Seja $\Sigma : \Omega \to S^\star \times S$ uma assinatura, $\mathcal{A}$ uma Σ-álgebra e $\cong$ uma Σ-congruência sobre $\mathcal{A}$. Chamamos* álgebra quociente *$\mathcal{A}/\cong$ à Σ-álgebra definida como se segue:*

1. *para cada $s \in S$,*

$$\mathcal{A}/\cong(s) \ = \ \mathcal{A}(s)/\cong_s$$

2. *para cada $\sigma \in \Omega$,*
   (a) *se $\sigma :\to s$ é uma constante, então*

$$\mathcal{A}/\cong(\sigma) = [\mathcal{A}(\sigma)]$$

   (b) *se $\sigma : s_1 \times \ldots \times s_n \to s$, então, para $a_i \in \mathcal{A}(s_i)$ $(1 \leq i \leq n)$,*

$$\mathcal{A}/\cong(\sigma)([a_1], \ldots, [a_n]) = [\mathcal{A}(\sigma)(a_1, \ldots, a_n)]$$

□

Portanto, os "valores" manipulados por uma álgebra quociente $\mathcal{A}/\cong$ são classes da equivalência $\cong$. Quer dizer, $\mathcal{A}/\cong$ é mais *abstracta* do que $\mathcal{A}$ exactamente na medida em que não reconhece a diferença entre dois quaisquer valores $a \neq b$ de $\mathcal{A}$ tais que $[a] = [b]$.

Seja agora $\mathcal{C}(\Sigma)$ o conjunto de todas as álgebras quociente $\mathcal{W}/\cong$ definíveis por uma qualquer Σ-congruência $\cong$ sobre termos da linguagem Σ. Pelo que atrás se disse, pertence a $\mathcal{C}(\Sigma)$ o quociente de $\mathcal{W}$ induzido pela definição de um qualquer modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$, *i.é* $\mathcal{W}/\cong_{\mathcal{A}}$. Por outras palavras, a cada Σ-modelo corresponde um elemento de $\mathcal{C}(\Sigma)$. Assim, $\mathcal{C}(\Sigma)$ pode ser considerado o conjunto de todas as semânticas possíveis para a linguagem Σ.

Na verdade, $\mathcal{C}(\Sigma)$ é um instrumento muito útil para raciocinarmos sobre semântica(s). Vejamos como: sobre $\mathcal{C}(\Sigma)$ é possível definir a seguinte ordem parcial $\sqsubseteq$:

$$\mathcal{W}/\cong_1 \sqsubseteq \mathcal{W}/\cong_2 \;\; sse \;\; \cong_1 \subseteq \cong_2 \tag{1.91}$$

onde $\subseteq$ designa inclusão entre relações (conjuntos). A ordem $\sqsubseteq$ permite-nos comparar semânticas entre si: se $\cong_1 \subseteq \cong_2$, então é porque

$$t \cong_1 t' \Rightarrow t \cong_2 t'$$

ou seja, tudo o que é equivalente em $\cong_1$ é-o também em $\cong_2$. Mas o inverso pode não ser verdadeiro. Logo, as classes de equivalência de $\cong_2$ são "maiores" do que as de $\cong_1$, ou seja, as primeiras obtêm-se por "aglutinação" das segundas.

Esta perspectiva motiva a designação de ordem de "granularidade semântica" vulgarmente atribuída a $\sqsubseteq$. Sempre que dois modelos $\mathcal{A}$ e $\mathcal{B}$ são tais que

$$\mathcal{W}/\cong_{\mathcal{A}} \sqsubseteq \mathcal{W}/\cong_{\mathcal{B}}$$

então dizemos que a semântica de $\mathcal{A}$ é "mais fina" (ou "tem grão mais fino") do que $\mathcal{B}$, ou que esta é "mais grosseira" do que a primeira. Semânticas mais grosseiras tornam as coisas mais equivalentes entre si; semânticas mais finas distinguem mais as coisas entre si.

Estas duas direcções semânticas têm limites, contudo. Não é possível distinguir as coisas mais do que considerá-las todas diferentes entre si. Ora esta é a semântica do próprio $\mathcal{W}$, a que atrás se chamou *inicial*, correspondente à igualdade *literal*, *i.é* sintática, entre os termos. Na direcção oposta, não é possível tornar as coisas mais equivalentes do que considerá-las todas equivalentes entre si. Ora esta semântica, a que corresponde a congruência universal, é a que está associada ao modelo trivial $\mathcal{T}$, pois

$$\forall t, t' : \mathcal{T}(t) = \mathcal{T}(t') = 1$$

ou seja,

$$\forall t, t' : t \cong_{\mathcal{T}} t'$$

Em resumo, qualquer Σ-congruência $\cong$ está entre os limites,

$$\cong_{\mathcal{W}} \;\subseteq\; \cong \;\subseteq\; \cong_{\mathcal{T}} \tag{1.92}$$

ou seja, a ordem $\sqsubseteq$ é limitada universalmente, tanto superior como inferiormente. Mais do que isto, temos o resultado que se segue.

Teorema 1.2 *A estrutura $(\mathcal{C}(\Sigma); \sqsubseteq)$, onde $\sqsubseteq$ é a ordem sobre quocientes de $\mathcal{W}$ definida pela equação(1.91), é um recticulado completo.*

Demonstração: *Da definição da ordem $\sqsubseteq$ (equação (1.91)) resulta que o estudo da classe $\mathcal{C}(\Sigma)$ de quocientes de $\mathcal{W}$ ordenada por $\sqsubseteq$ se reduz ao estudo do conjunto de todas as congruências sobre W_Σ ordenadas pela inclusão ($\subseteq$). Esta é uma ordem parcial (reflexiva, transitiva e antissimétrica) com* máximo *($\cong_{\mathcal{T}}$) e* mínimo *($\cong_{\mathcal{W}}$), cf. equação (1.92). Seja F uma família de Σ-congruências. O maior dos minorantes de qualquer elemento de F, $\bigwedge F$, existe e é dado pela conjunção das suas congruências:*

$$t(\bigwedge F)t' \stackrel{\text{def}}{=} \forall \cong \in F : t \cong t' \tag{1.93}$$

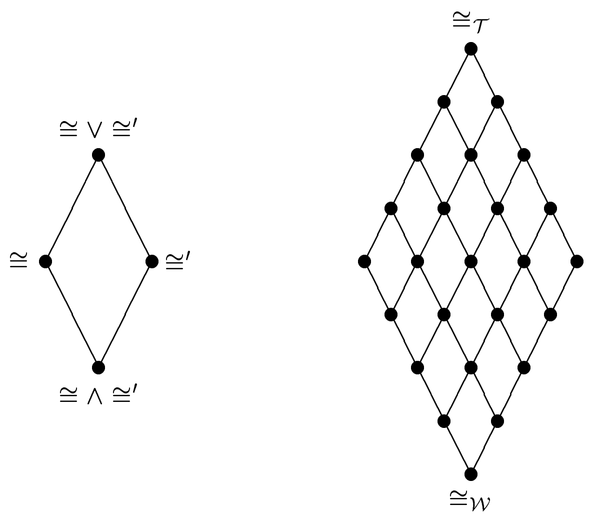


Figura 1.7: Reticulado de Σ-congruências.

O menor dos majorantes de qualquer elemento de F também existe, e define-se à custa de (1.93):

$$\bigvee F = \bigwedge \{\supseteq_{\cong} \mid \ \cong \in F\}$$

onde $\supseteq_{\cong}$ designa o conjunto de todos os majorantes de $\cong$. Logo, temos um reticulado completo, cujas operações 'meet' e 'join' se definem como é habitual:

$$\cong \vee \cong' \ \stackrel{\text{def}}{=} \ \bigvee\{\cong, \cong'\}$$

$$\cong \wedge \cong' \ \stackrel{\text{def}}{=} \ \bigcap\{\cong, \cong'\}$$

$\square$

Este resultado vem ilustrado na Figura 1.7 e é essencial à teoria algébrica que fundamenta a técnica de especificação formal que vem sendo exposta. A partir de agora é inequívoco o significado matemático da *semântica* que atribuimos a uma dada sintaxe através de um *modelo*, dos limites dessas semânticas e seu relacionamento entre si. Se estivermos a trabalhar com modelos *gerados por termos*, temos ainda que:

Teorema 1.3 *Se um modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ é gerado por termos, então o quociente $\mathcal{W}/\cong_{\mathcal{A}}$ é isomorfo de $\mathcal{A}$.*

Demonstração: *Este teorema é a versão do* Teorema do Homomorfismo *(ou da* Decomposição de Funções*) a nível de álgebras heterogéneas.*
□

Exercício 1.24 Seja $\Sigma : \Omega \to S^\star \times S$ a assinatura do Exercício 1.8.

1. Seja $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ o modelo seguinte:

$$\begin{array}{lcl} \mathcal{A}(s) & = & \{V, F\} \\ \mathcal{A}(r) & = & \mathcal{A}(s) \cup \{\perp\} \\ \mathcal{A}(0) & = & F \\ \mathcal{A}(1) & = & V \\ \mathcal{A}(2) & = & \perp \\ \mathcal{A}(\sigma) & = & \lambda(x, y).x \wedge y \end{array}$$

   Represente o homomorfismo $h_{\mathcal{A}}$.

2. Seja $\mathsf{B} : \Sigma \longrightarrow \mathbf{Set}$ outro modelo:

$$\begin{array}{lcl} \mathcal{B}(s) & = & I\!N_0 \\ \mathcal{B}(r) & = & \{x + jy \mid \ x, y \in I\!N_0\} \\ \mathcal{B}(0) & = & 0 \\ \mathcal{B}(1) & = & 1 \\ \mathcal{B}(2) & = & j \\ \mathcal{B}(\sigma) & = & \lambda(x, y).x + jy \end{array}$$

   Será $\mathcal{A}$ um epimorfismo? Descreva a congruência $\cong_{\mathcal{A}}$ e compare-a com $\cong_{\mathcal{B}}$.

□

Como duas álgebras isomorfas são *abstractamente idênticas*, então podemos afixar a cada nó $\mathcal{W}/\cong$ do reticulado $(\mathcal{C}(\Sigma); \sqsubseteq)$ todas as Σ-álgebras (modelos) que lhe são isomorfas. Ou seja, podemos dividir o reticulado $(\mathcal{C}(\Sigma); \sqsubseteq)$ pela relação de Σ-isomorfismo.

Finalmente:

Teorema 1.4 *Se* $\mathsf{A}, \mathsf{B} : \Sigma \longrightarrow \mathbf{Set}$ *são dois modelos tais que existe um* Σ*-homomorfismo* $h : \mathcal{A} \to \mathcal{B}$*, então* $\mathcal{W}/\cong_{\mathcal{A}} \sqsubseteq \mathcal{W}/\cong_{\mathcal{B}}$*.*

Demonstração: *A composição* $h \circ h_{\mathcal{A}}$ *é um* Σ*-homomorfismo de* $\mathcal{W}$ *para* $\mathcal{B}$*, que terá que coincidir com* $h_{\mathcal{B}}$ *pelo Teorema 1.1. Sejam* t *e* t' *dois termos tais que* $h_{\mathcal{A}}(t) = h_{\mathcal{A}}(t')$*. Então*

$$h(h_{\mathcal{A}}(t)) = h(h_{\mathcal{A}}(t'))$$

ou seja,

$$h_{\mathcal{B}}(t) = h_{\mathcal{B}}(t')$$

Logo $\cong_{\mathcal{A}} \subseteq \cong_{\mathcal{B}}$*, como queríamos.* Q.E.D.
□

Σ	$\mathcal{A}$	$\mathcal{B}$
$Bool$	2	2
$Item$	A	A
$Base$	2^A	$\mathbb{N} \rightharpoonup 2^A$
$inic$	$\emptyset$	$\left(\ \right)$
$insere$	$\lambda(b,i).b \cup \{i\}$	$\lambda(b,i).\ \ let\ \ n = H(i)$ $\quad s = \begin{cases} b(n) & \Leftarrow \quad n \in dom(b) \\ \emptyset & \Leftarrow \quad n \notin dom(b) \end{cases}$ $\quad in\ \ b \dagger \begin{pmatrix} n \\ s \cup \{i\} \end{pmatrix}$
$consta?$	$\lambda(b,i).i \in b$	$\lambda(b,i).\ \ let\ \ n = H(i)$ $\quad in\ \begin{cases} (i \in b(n)) & \Leftarrow \quad n \in dom(b) \\ false & \Leftarrow \quad n \notin dom(b) \end{cases}$
inv-$Base$	$\lambda b.TRUE$	$\lambda b.\forall n \in dom(b) : (\emptyset \subset b(n) \wedge \forall i \in b(n) : H(i) = n)$

Figura 1.8: Dois modelos $\mathcal{A}$ e $\mathcal{B}$ de assinatura para *base de dados* elementar

Este último resultado mostra-nos que a granularidade semântica entre dois modelos pode ser caracterizada directamente pela definição de um homomorfismo entre eles.

Exercício 1.25 (Aplicação do teorema anterior)

1. Mostre que a ordem $\sqsubseteq$ é reflexiva (Sugestão: mostre que a identidade é um Σ-homomorfismo).
2. Mostre que a ordem $\sqsubseteq$ é transitiva (Sugestão: mostre que a composição de dois Σ-homomorfismos é ainda um Σ-homomorfismo).

□

Exercício 1.26 Considere a seguinte assinatura que especifica as operações elementares de uma *base de dados* (muito simplificada!),

$$\Sigma = \begin{cases} inic & : & \rightarrow Base \\ insere & : & Base \times Item \rightarrow Base \\ consta? & : & Base \times Item \rightarrow Bool \\ \text{inv-}Base & : & Base \rightarrow Bool \end{cases}$$

bem como os dois Σ-modelos $\mathcal{A}$ e $\mathcal{B}$ que se esquematizam na Figura 1.8, assumindo-se em $\mathcal{B}$ uma função de 'hashing' $H : A \longrightarrow \mathbb{N}$.

Verifique se a seguinte família de funções é um Σ-homomorfismo $h : \mathcal{B} \longrightarrow \mathcal{A}$:

$$\begin{pmatrix} h_{Bool} & \stackrel{\text{def}}{=} & \lambda b.b \\ h_{Item} & \stackrel{\text{def}}{=} & \lambda i.i \\ h_{Base} & \stackrel{\text{def}}{=} & \lambda b. \bigcup_{n \in dom(b)} b(n) \end{pmatrix}$$

□

1.3.4 Introdução à Semântica Denotacional

Dedicou-se alguma atenção neste capítulo a mostrar como a especificaçaão matemática de um objecto da vida real *gera* uma *linguagem abstracta* — que nos permite "falar dele" — e um *modelo* semântico dessa linguagem — que nos permite dizer o que o mesmo objecto "significa".

Mostrou-se na Secção 1.2.2 que o objecto a especificar pode ser, ele próprio, uma *linguagem de programação*. É agora propósito nosso mostrar que, se a especificação do nível sintático de uma linguagem de programação pode ser, à partida, mais simples — pois uma sua descrição em BNF é muitas vezes um dado do problema — a sua especificação semântica não é isenta de dificuldades, mesmo quando se trata de linguagens de progamação elementares. É assim que o 'moto' de Scott,

"... extend BNF to semantics,"

gerou uma disciplina própria — a *Semântica Denotacional* — que se dedica aos problemas técnicos que são levantados quando se pretende explicar formalmente o significado do acto de programar uma máquina [25].

O nosso ponto de partida será a gramática G cuja caracterização algébrica, a nível sintático, conduziu à assinatura Σ_G (1.6), para a qual pretendemos construir agora um modelo semântico

$$\mathcal{A} : \Sigma_G \to Sets$$

Começamos pela escolha dos domínios semânticos a associar às espécies de Σ_G, *i.é* não-terminais de G:

$$\begin{aligned} \mathcal{A}(\langle \mathit{Nato} \rangle) &\cong Range \\ \mathcal{A}(\langle \mathit{Var} \rangle) &\cong 2 \\ \mathcal{A}(\langle \mathit{Cmd} \rangle) &\cong Memory \to Memory \end{aligned} \tag{1.94}$$

onde

$$\begin{aligned} Range &\cong \{i - 1 \mid i \in 2^{32}\} \\ Memory &\cong 2 \rightharpoonup Range \end{aligned}$$

o que merece os comentários seguintes:

- por $\langle \mathit{Nato} \rangle$ queremos "significar" os números naturais "de 32 bits";
- como há apenas duas variáveis `x` e `y`, são apenas dois os endereços respectivos;

[25] A *Semântica Denotacional* não tem logrado uma aceitação sufucientemente ampla na camada dos pragmáticos da "arte" devido à sua complexidade teórica, *e.g.* o facto de os seus domínios matemáticos ultrapassarem muitas vezes a classe dos $Sets$, ver Secção 1.5.

- para semântica de um comando escolhemos uma transformação de uma configuração de variáveis ($Memory$) para outra; de facto, a única coisa que a linguagem *pode fazer* é *variar* os valores das suas variáveis; de reparar que a função parcial finita $2 \rightharpoonup Range$ permite modelar variáveis indefinidas.

Para tornar mais sugestiva a definição dos operadores semânticos — relembremos que existirá um por cada produção de G — vamos usar a respectiva sintaxe concreta envolvida nos parênteses de "dupla parede" $[\![\ldots]\!]$ que são tradicionais em Semântica Denotacional. Por exemplo, em lugar de escrevermos

$$\mathcal{A}(ciclo) \stackrel{\mathrm{def}}{=} \lambda(n,c).\ldots n \ldots c \ldots$$

escreveremos, simplesmente,

$$[\![\texttt{while}\, n\, \texttt{do}\, c]\!] \stackrel{\mathrm{def}}{=} \ldots n \ldots c \ldots$$

uma vez que o identificador $\mathcal{A}$ do modelo a construir está subentendido em todo o exercício.

Comecemos pelas constantes da espécie $\langle$*Var*$\rangle$:

$$\begin{aligned} [\![\texttt{x}]\!] &\stackrel{\mathrm{def}}{=} 1 \\ [\![\texttt{y}]\!] &\stackrel{\mathrm{def}}{=} 2 \end{aligned}$$

e pelos construtores da espécie $\langle$*Nato*$\rangle$:

$$[\![\texttt{0}]\!] \stackrel{\mathrm{def}}{=} 0 \qquad (1.95)$$

$$[\![\texttt{suc}(n)]\!] \stackrel{\mathrm{def}}{=} rem(\frac{[\![n]\!] + 1}{2^{32}}) \qquad (1.96)$$

$$[\![valor(v)]\!] \stackrel{\mathrm{def}}{=} \ldots$$

A definição (1.96) prevê o caso em que há 'overflow' em $Range$. Paramos na última definição para explicarmos a razão pela qual não podemos prescindir do respectivo operador $valor$ que "injecta" variáveis em números naturais. Se tivéssemos escrito apenas $[\![v]\!]$ essa informação perder-se-ia e o símbolo v seria ambíguo: no lado esquerdo da definição v significaria um número natural e no lado direito da mesma definição significaria uma variável.

Mas outras dificuldades aqui se levantam e são as seguintes: como saber o valor de uma variável sem consultar a memória da máquina? qual é esse valor se a variável estiver indefinida? Teremos que, de certa forma, parametrizar a semântica de um número natural pelo estado da memória da máquina; se esse número for uma constante, ignora-se a memória; se for o valor de uma variável, consulta-se o respectivo endereço na memória [26]. Antes de mais, teremos que sofisticar (1.94) para

$$\langle \textit{Nato} \rangle \quad \cong \quad Memory \rightarrow Range \tag{1.97}$$

E faremos um tratamento simplista do problema da indefinição de variáveis, decidindo que seja 0 o valor de uma variável indefinida:

$$[\![valor(v)]\!] \quad \stackrel{\text{def}}{=} \quad \lambda\sigma. \left\{ \begin{array}{lcl} [\![v]\!] \notin dom(\sigma) & \Rightarrow & 0 \\ [\![v]\!] \in dom(\sigma) & \Rightarrow & \sigma([\![v]\!]) \end{array} \right.$$

Antes de passarmos aos construtores de comandos ($\langle \textit{Cmd} \rangle$), teremos que adaptar as semânticas de `0` (1.95) e `suc(n)` (1.96) ao novo domínio semântico para $\langle \textit{Nato} \rangle$ (1.97) — bastará considerar a constante `0` como a *função constante*

$$[\![\texttt{0}]\!] \quad \stackrel{\text{def}}{=} \quad \lambda\sigma.0$$

e redefinir

$$[\![\texttt{suc(}n\texttt{)}]\!] \quad \stackrel{\text{def}}{=} \quad \lambda\sigma.rem(\frac{[\![n]\!](\sigma)+1}{2^{32}})$$

Por exemplo, qual a semântica de `suc(x)` quando a memória regista

$$\sigma_1 \quad = \quad \left(\begin{array}{cc} 1 & 2 \\ 10 & 20 \end{array} \right) \; ? \tag{1.98}$$

Teremos

$$\begin{array}{rcl} [\![\texttt{suc(}x\texttt{)}]\!](\sigma_1) & = & [\![valor(x)]\!](\sigma_1)+1 \\ & = & (\left\{ \begin{array}{lcl} [\![x]\!] \notin dom(\sigma_1) & \Rightarrow & 0 \\ [\![x]\!] \in dom(\sigma_1) & \Rightarrow & \sigma_1([\![x]\!]) \end{array} \right.)+1 \\ & = & (\left\{ \begin{array}{lcl} 1 \notin \{1,2\} & \Rightarrow & 0 \\ 1 \in \{1,2\} & \Rightarrow & \sigma_1(1) \end{array} \right.)+1 \\ & = & \sigma_1(1)+1 \\ & = & 10+1 \\ & = & 11 \end{array} \tag{1.99}$$

Passemos então ao não-terminal (*i.é* espécie) $\langle \textit{Cmd} \rangle$. De imediato escrevemos

$$\begin{array}{rcl} [\![\texttt{skip}]\!] & \stackrel{\text{def}}{=} & \lambda\sigma.\sigma \\ [\![c\texttt{;}d]\!] & \stackrel{\text{def}}{=} & [\![d]\!] \circ [\![c]\!] \end{array}$$

[26] Esta complicação é típica das chamadas *linguagens imperativas* que correm sobre a chamada arquitectura de *von Newman* baseada em memória estática. A grande maioria das linguagens comerciais (*e.g.* C, PASCAL *etc.*) são imperativas.

querendo significar que `skip` "não faz nada" (deixa a memória como estava) e que a composição sequencial de comandos permite ao segundo comando (d) modificar a memória que lhe foi devolvida pela execução do primeiro comando (c). Quanto à atribuição, teremos

$$[\![v\,\texttt{:=}\,n]\!] \stackrel{\text{def}}{=} \lambda\sigma.\sigma \dagger \left(\begin{array}{c} [\![v]\!] \\ [\![n]\!](\sigma) \end{array} \right)$$

Por exemplo, qual o efeito de executarmos `x := suc(x)` sobre σ_1 (1.98)? Aproveitando (1.99), teremos:

$$\begin{aligned} [\![\texttt{x := suc(x)}]\!](\sigma_1) &= \sigma_1 \dagger \left(\begin{array}{c} [\![\texttt{x}]\!] \\ [\![\texttt{suc(x)}]\!](\sigma_1) \end{array} \right) \\ &= \sigma_1 \dagger \left(\begin{array}{c} 1 \\ 11 \end{array} \right) \\ &= \left(\begin{array}{cc} 1 & 2 \\ 11 & 20 \end{array} \right) \end{aligned}$$

i.é o valor da variável `x` foi incrementado.

Quanto à composição condicional de comandos definimos a semântica que se segue:

$$[\![\texttt{if } n \texttt{ then } c \texttt{ else } d]\!] = \lambda\sigma. \left\{ \begin{array}{lcl} [\![n]\!](\sigma) > 0 & \Rightarrow & [\![c]\!](\sigma) \\ [\![n]\!](\sigma) = 0 & \Rightarrow & [\![d]\!](\sigma) \end{array} \right.$$

interpretando 0 como $falso$ e $n > 0$ como $verdadeiro$.

Resta-nos o comando iterativo

$$[\![\texttt{while } n \texttt{ do } c]\!] \stackrel{\text{def}}{=} \lambda\sigma.\ldots$$

A intuição que temos sobre "ciclos" convida-nos à definição (recursiva):

$$\begin{aligned} &[\![\texttt{while } n \texttt{ do } c]\!] \stackrel{\text{def}}{=} \\ &\quad \lambda\sigma. \left\{ \begin{array}{lcl} [\![n]\!](\sigma) = 0 & \Rightarrow & \sigma \\ [\![n]\!](\sigma) > 0 & \Rightarrow & [\![\texttt{while } n \texttt{ do } c]\!]([\![c]\!](\sigma)) \end{array} \right. \end{aligned} \tag{1.100}$$

Quer dizer, se a variável de controlo n é falsa, saimos do ciclo sem nada fazer. No caso contrário, executamos c e voltamos ao ciclo, agora sobre a memória devolvida por c.

Por muito intuitiva que seja a definição (1.100), as dificuldades que nos levanta não são desprezáveis. Repare-se que basta que n seja uma constante ou que c não modifique adequadamente a variável de controlo do ciclo, para que a execução de `while` n `do` c não termine. E qual a semântica *formal* de um comando que não termina?

É para podermos vir a responder a este tipo de questões que passaremos, no Capítulo 2, a estudar a semântica matemática da recursividade.

Exercício 1.27 Mostre que

$$[\![\texttt{skip}]\!] = [\![\texttt{x := x}]\!] = [\![\texttt{while 0 do } c]\!]$$

□

Exercício 1.28 Suponha que, para fugirmos aos problemas que acabamos de levantar àcerca da definição (1.100), resolvemos dar a semântica “de ciclo ‘for’ ” a `while` n `do` c, *i.é* qualquer coisa como: *“lê-se o valor de* n *antes de se iniciar o ciclo e executa-se* c n*-vezes”*.

Escreva formalmente essa semântica alternativa a (1.100).

□

Exercício 1.29 Por simplicidade, a definição sintática e semântica da linguagem que foi assunto desta secção só prevê a manipulação de duas variáveis `x` e `y`.

- Generalize essa linguagem à manipulação de um número arbitrário de variáveis identificadas por identificadores alfanuméricos.
- Estenda a mesma linguagem por forma a poder declarar variáveis inteiras ou ‘array’s unidimensionais de inteiros e a realizar o seu endereçamento e atribuição, *e.g.*

```
      ⋮
var   a:array[10];
      ⋮
      a[3] := 0; x := a[4];
      ⋮
```

Aborde <u>informalmente</u> o impacto desta extensão à gramática no correspondente modelo semântico estudado neste curso.

□

1.4 Exercícios

Exercício 1.30 Considere duas assinaturas Σ_1 e Σ_2 tal que S_1 é um subconjunto de S_2 e, para todo o $s \in S_1$ e $w \in S_1^*$, Σ_2 contem pelo menos tantos operadores de funcionalidade $w \to s$ quanto Σ_1. Mostre que toda a Σ_2-álgebra “é” uma Σ_1-álgebra e interprete o resultado.

□

Exercício 1.31 Conhece com certeza, da literatura em sistemas de informação, a noção de *vista* ('view') de uma base de dados, pela qual se fornece a um utilizador apenas uma "perspectiva" da base, omitindo-se-lhe a outra parte. Por exemplo, no exemplo $SGIB$ (sistema de gestão de uma instituição bancária) — *cf.* (1.4) — podemos querer criar uma vista que permita ver o saldo de cada conta sem se poder saber quem são os respectivos titulares.

A noção de Σ-homomorfismo permite especificar esta noção de *vista* de forma simples. Seja $\mathcal{A}$ o Σ_{SGIB}-modelo que foi dado nas aulas e seja $\mathcal{B}$ um outro Σ_{SGIB}-modelo tal que existe o seguinte homomorfismo h, ou "vista", de $\mathcal{A}$ para $\mathcal{B}$:

$$\begin{aligned} h_{Sistema} &\stackrel{\mathrm{def}}{=} \lambda\sigma.\begin{pmatrix} k \\ \pi_2(\sigma(k)) \end{pmatrix}_{k\in dom(\sigma)} \\ h_{Titular} &\stackrel{\mathrm{def}}{=} \lambda x.x \\ h_{Quantia} &\stackrel{\mathrm{def}}{=} \lambda x.x \\ h_{IdConta} &\stackrel{\mathrm{def}}{=} \lambda x.x \end{aligned}$$

Infira uma Σ_{SGIB}-álgebra $\mathcal{B}$ a partir de $\mathcal{A}$ e de h, justificando o seu raciocínio.

□

Exercício 1.32 Considere o seguinte modelo que especifica, a um elevado nível de abstracção, a estrutura de páginas de um serviço de informação em hipertexto tipo WWW ('World Wide Web') sobre a INTERNET, onde Ref (endereço de cada página) e $Text$ (unidade textual de informação) são espécies que a este nível de abstracção não interessa detalhar:

$$\begin{aligned} WWW &\cong Ref \rightharpoonup URL \quad \textit{/* URL='Universal Resource Location' */} \\ URL &\cong (Text + Ref)^* \end{aligned}$$

Especifique o operador seguinte sobre WWW que deverá devolver os endereços de todas as páginas que em σ se referem a r:

$$\begin{aligned} refsTo &: WWW \times Ref \longrightarrow 2^{Ref} \\ refsTo(\sigma, r) &\stackrel{\mathrm{def}}{=} \ldots \end{aligned}$$

□

Exercício 1.33 Suponha que M e S designam os conjuntos, finitos e disjuntos, dos números dos alunos inscritos a uma disciplina e provenientes de dois cursos da Universidade do Minho. Suponha que $itoa : I\!N \longrightarrow Str$ e $strcat : Str \times Str \longrightarrow Str$ são funções matemáticas cuja manipulação de naturais e 'strings' corresponde à semântica das mesmas em C. Sejam ainda dadas as seguintes funções auxiliares:

$$\begin{aligned} s(x) &\stackrel{\mathrm{def}}{=} strcat(\texttt{"s"}, strcat(x, \texttt{"@ci.uminho.pt"})) \\ m(x) &\stackrel{\mathrm{def}}{=} strcat(\texttt{"m"}, strcat(x, \texttt{"@ci.uminho.pt"})) \end{aligned}$$

Estará correctamente escrita a seguinte expressão, em SETS:

$$(itoa \rightharpoonup (s \circ itoa))(1_S) \cup (itoa \rightharpoonup (m \circ itoa))(1_M) \quad ?$$

Na afirmativa, tipifique-a e descreva o seu significado informal por (breves!) palavras suas.
□

Exercício 1.34 Suponha que alguém especifica um *plano de estudos* de uma licenciatura como sendo um conjunto de disciplinas,

$$PlanoEstudos \quad \cong \quad 2^{Disciplina}$$

onde uma disciplina ou é obrigatória ou opcional,

$$Disciplina \quad \cong \quad Obrigat + Opcional$$

sendo uma disciplina obrigatória caracterizada por uma série de atributos,

$$\begin{array}{rclcr} Obrigat & \cong & A : 5 & \times & \textit{/*ano do curso */} \\ & & AC : ACien & \times & \textit{/*área científica */} \\ & & N : Nome & \times & \textit{/*nome */} \\ & & R : 3 & \times & \textit{/*regime (1.º sem., 2.º sem., anual) */} \\ & & E : Esc & & \textit{/*escolaridade,}\ \text{etc. */} \end{array}$$

e uma disciplina opcional caracterizada como se segue,

$$\begin{array}{rclcr} Opcional & \cong & A : 5 & \times & \textit{/*ano do curso */} \\ & & Nr : I\!N & \times & \textit{/*número da opção */} \\ & & R : 3 & \times & \textit{/*regime (1.º sem., 2.º sem., anual) */} \\ & & E : Esc & \times & \textit{/*escolaridade,}\ \text{etc. */} \\ & & O : 2^{Opcao} & & \textit{/*opções oferecidas */} \end{array}$$

onde cada opção se caracteriza por:

$$\begin{array}{rclcr} Opcao & \cong & AC : ACien & \times & \textit{/*área científica */} \\ & & N : Nome & & \textit{/*nome */} \end{array}$$

Especifique funções sobre $PlanoEstudos$ capazes de calcular:

1. Os nomes de todas as disciplinas efectivamente leccionadas.
2. O conjunto das áreas científicas exclusivamente opcionais.

□

Exercício 1.35 Prove as igualdades (A.70), (A.76), (A.80) e (A.85).
□

Exercício 1.36 Verifique a validade da seguinte propriedade sobre os operadores de *restrição* de uma função finita $\phi \in A \rightharpoonup B$:

$$\phi \mid \overline{S} = \phi \setminus S$$

em que $\overline{S}$ designa o complementar de S relativo a A.
□

Exercício 1.37 Relativamente aos diagramas funcionais em SETS que se seguem,

$$
\begin{tikzcd}
A \arrow[d, "f"'] & \mathsf{F}(A) \arrow[r, "dom_A"] \arrow[d, "\mathsf{F}(f)"'] & \mathsf{G}(A) \arrow[d, "\mathsf{G}(f)"] \\
B & \mathsf{F}(B) \arrow[r, "dom_B"'] & \mathsf{G}(B)
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[d, "f"'] & C \rightharpoonup A \arrow[r, "x_A"] \arrow[d, "C \rightharpoonup f"'] & 2^A \arrow[d, "2^f"] \\
B & C \rightharpoonup B \arrow[r, "x_B"'] & 2^B
\end{tikzcd}
$$

identifique as estruturas F e G, bem como o operador x, por forma a estes diagramas ilustrarem duas propriedades válidas da notação SETS. Justifique a sua resposta.

□

Exercício 1.38 Uma noção "intermédia" entre conjuntos (2^A) e sequências (A^*) é a de *multi-conjunto*, *i.é* um conjunto em que qualquer elemento $a \in A$ pode ocorrer mais do que uma vez:

$$MSet(A) \quad \cong \quad A \rightharpoonup I\!N \tag{1.101}$$

Assim, o multiconjunto vazio é representado pela função finita vazia $\begin{pmatrix} \ \end{pmatrix}$, e a união de multiconjuntos corresponde à adição de multiciplidades, *i.é* se x e y são multiconjuntos em $MSet(A)$, a sua união, designada $x \oplus y$, será a função

$$
\begin{array}{rcll}
x \oplus y & \stackrel{\text{def}}{=} & & x \setminus dom(y) \\
& & \cup & y \setminus dom(x) \\
& & \cup & \begin{pmatrix} a \\ x(a) + y(a) \end{pmatrix}_{a \in dom(x) \cap dom(y)}
\end{array}
\tag{1.102}
$$

1. Mostre que $\oplus$ pode ser alternativamente definida como se segue:

$$x \oplus y \quad \stackrel{\text{def}}{=} \quad x \dagger y \dagger \begin{pmatrix} a \\ x(a) + y(a) \end{pmatrix}_{a \in dom(x|(dom(y)))} \tag{1.103}$$

2. Complete a definição do seguinte operador sobre $MSet(A)$:

$$
\begin{array}{rcl}
\otimes & : & I\!N \times MSet(A) \to MSet(A) \\
n \otimes \sigma & \stackrel{\text{def}}{=} & \ldots\ldots
\end{array}
$$
/ a multiplicidade de cada elemento do multi-conjunto σ é multiplicada n vezes */*

e prove a seguinte propriedade sobre esse operador:

$$n \otimes (m \otimes \sigma) \quad = \quad (n \times m) \otimes \sigma \tag{1.104}$$

□

Exercício 1.39 Na sequência do Exercício 1.38, especifique a operação de *intersecção* de multi-conjuntos.

□

Exercício 1.40 Num 'fuzzy set', ou "conjunto difuso", os seus elementos pertencem-lhe com um certo grau de incerteza, entre 100% (grau de pertença máxima) e 0% (não-pertença). Seja então A um conjunto finito não-vazio e defina-se o espaço dos 'fuzzy subsets' de A da forma seguinte:

$$FSet(A) \quad \cong \quad A \rightharpoonup 100$$

1. Complete a definição dos seguintes operadores sobre $FSet(A)$:

$$\begin{array}{rcl} empty & : & \to FSet(A) \\ empty & : & \stackrel{\rm def}{=} \ldots\ldots \quad \textit{/*o 'fuzzy set' com toda a certeza vazio */} \\ \\ \epsilon & : & FSet(A) \times A \to \{0\} \cup 100 \\ \epsilon(fs, a) & \stackrel{\rm def}{=} & \ldots\ldots \quad \textit{/*o grau de pertença de } a \textit{ em } s \textit{ */} \\ \\ assured & : & FSet(A) \times 100 \to 2^A \\ assured(fs, i) & \stackrel{\rm def}{=} & \ldots\ldots \quad \textit{/*os elementos de fs garantidos acima da incerteza } i \textit{ */} \end{array}$$

2. Partindo do aforismo "toda a relação é um conjunto", definiremos o espaço das *relações difusas* de A para B como o espaço de todos os *sub-conjuntos difusos* de $A \times B$, *i.é*:

$$FRel(A, B) \cong (A \times B) \rightharpoonup 100$$

Defina a composição $\rho \circ \tau$ de duas relações difusas $\rho \in FRel(A, B)$ e $\tau \in FRel(B, C)$.

3. Uma relação binária ρ diz-se transitiva sse, para todo o a, b, c, se tem

$$a\rho b \wedge b\rho c \quad \Rightarrow \quad a\rho c$$

Generalize a noção de transitividade a relações difusas.

□

Exercício 1.41 Considere o seguinte texto que abrevia e simplifica os requisitos informais colocados por uma empresa comercial a uma empresa informática que a primeira contratou para lhe produzir determinado 'software':

> *A empresa* XYZ LDA. *consta de 3 estabelecimentos ou lojas de comércio, prevendo-se que este número venha a aumentar no futuro.*
>
> *A informação associada a cada loja consta de (a) o 'stock' de artigos armazenados ou expostos para venda (cada artigo tem um código próprio); (b) as folhas de vendas.*
>
> *Existe uma folha de venda por cada dia útil. Cada folha de venda regista as facturas emitidas nas vendas daquele dia. Cada factura indica os artigos vendidos e em que quantidade. As facturas são numeradas e emitidas sequencialmente. Pode haver facturas anuladas.*
>
> *O sistema a desenvolver deverá prever pelo menos as seguintes operações:*
>
> 1. *Emissão de factura — seu registo na folha de vendas e simultânea descarga, no 'stock', dos artigos vendidos.*
> 2. *Aquisição de mais 'stock' para uma dada loja.*
> 3. *Transferência de segmentos de 'stock' de loja para loja.*
>
> □

1. Construa uma assinatura Σ capaz de captar as espécies e os operadores pretendidos nos requisitos acima.

2. Construa o correspondente modelo, isto é, a Σ-álgebra que, na sua opinião, melhor capta a semântica dos mesmos. Não esqueça eventuais invariantes ou pré-condições.
   Sugestão: interprete o seguinte esboço e tome-o, se desejar, como ponto de partida:

$$\left\{\begin{array}{rcl} System & \cong & IdL \rightharpoonup Info \\ Info & \cong & Stock \times Sales \\ Stock & \cong & IdA \rightharpoonup I\!N \\ Sales & \cong & InvNr \rightharpoonup (Date \times (IdA \rightharpoonup I\!N)) \\ InvNr & \cong & I\!N \text{ /*invoice number */} \end{array}\right. \tag{1.105}$$

   Baseie a funcionalidade do seu modelo em operações que conhece sobre multi-conjuntos.

$\square$

Exercício 1.42 Considere o seguinte fragmento de gramática de contexto livre para expressões aritméticas:

$$\begin{array}{rcl} G & = & \langle NT, T, \langle \mathit{Exp}\rangle, P\rangle \\ NT & = & \{\langle \mathit{Exp}\rangle, \langle \mathit{Sinal}\rangle, \langle \mathit{Termo}\rangle, \langle \mathit{Factor}\rangle, \langle \mathit{MulOp}\rangle, \langle \mathit{AdiOp}\rangle\} \\ T & = & \{\, \texttt{+}\, ,\, \texttt{-}\, ,\, \texttt{*}\, ,\, \texttt{/}\, ,\, \texttt{(}\, ,\, \texttt{)}\, ,\, \texttt{id}\, ,\, \texttt{num}\,\} \\ P & = & \left\{\begin{array}{rcl} \langle \mathit{Exp}\rangle & ::= & \langle \mathit{Sinal}\rangle\langle \mathit{Termo}\rangle(\langle \mathit{AdiOp}\rangle\langle \mathit{Termo}\rangle)^* \\ \langle \mathit{Sinal}\rangle & ::= & \epsilon \mid \texttt{+} \mid \texttt{-} \\ \langle \mathit{Termo}\rangle & ::= & \langle \mathit{Factor}\rangle(\langle \mathit{MulOp}\rangle\langle \mathit{Factor}\rangle)^* \\ \langle \mathit{Factor}\rangle & ::= & \texttt{id} \mid \texttt{num} \mid \texttt{(}\ \langle \mathit{Exp}\rangle\ \texttt{)} \\ \langle \mathit{AdiOp}\rangle & ::= & \texttt{+} \mid \texttt{-} \\ \langle \mathit{MulOp}\rangle & ::= & \texttt{*} \mid \texttt{/} \end{array}\right. \end{array}$$

1. Construa a correspondente assinatura algébrica Σ_G.
2. Calcule o Σ_G-termo correspondente à seguinte frase da linguagem:

```
- id + num * ( num - id )
```

$\square$

Exercício 1.43 Em LISP existe um único suporte para representação de informação, designado por *expressão-S*, de acordo com a sintaxe expressa por (1.12). Construa uma assinatura Σ para essa sintaxe, e indique o Σ-termo que representa a seguinte expressão-S:

```
( if ( equal x 0 ) 0 1)
```

$\square$

Exercício 1.44 Considere a seguinte gramática para um fragmento da linguagem SQL ('Structured Query Language'), o conhecido padrão fixado internacionalmente pela ANSI/ISO para interpelação de bases de dados:

$$
\begin{array}{rcl}
G & = & \langle NT, T, \langle \mathit{SQL} \rangle, P \rangle \\
NT & = & \{ \langle \mathit{SQL} \rangle, \langle \mathit{Instruções} \rangle, \langle \mathit{Expressão} \rangle, \langle \mathit{Filtro} \rangle, \langle \mathit{Atributos} \rangle, \langle \mathit{Relação} \rangle, \langle \mathit{Atr} \rangle, \langle \mathit{Condição} \rangle \} \\
T & = & \{ \texttt{;}\ ,\ \texttt{.}\ ,\ \texttt{SCHEMA}\ ,\ \texttt{END}\ ,\ \texttt{:=}\ ,\ \texttt{SELECT}\ ,\ \texttt{FROM}\ ,\ \texttt{WHERE}\ ,\ \texttt{TO}\ ,\ \texttt{STR}\ \} \\
P & = & \left\{ \begin{array}{rcl}
\langle \mathit{SQL} \rangle & ::= & \langle \mathit{Instruções} \rangle \\
\langle \mathit{Instruções} \rangle & ::= & \langle \mathit{Expressão} \rangle\ \texttt{;}\ \langle \mathit{Instruções} \rangle | \\
 & & \texttt{SCHEMA}\ \langle \mathit{Relação} \rangle\ \texttt{;}\ \langle \mathit{Instruções} \rangle | \\
 & & \texttt{END .} \\
\langle \mathit{Expressão} \rangle & ::= & \langle \mathit{Relação} \rangle\ \texttt{:=}\ \langle \mathit{Expressão} \rangle | \\
 & & \texttt{SELECT}\ \langle \mathit{Atributos} \rangle\ \texttt{FROM}\ \langle \mathit{Relação} \rangle \langle \mathit{Filtro} \rangle \\
\langle\ \mathit{Filtro}\ \rangle & ::= & \epsilon\ |\ \texttt{WHERE}\ \langle \mathit{Condição} \rangle \\
\langle\ \mathit{Atributos}\ \rangle & ::= & \langle \mathit{Atr} \rangle^{+} \\
\langle\ \mathit{Relação}\ \rangle & ::= & \texttt{STR} \\
\langle\ \mathit{Condição}\ \rangle & ::= & \texttt{STR} \\
\langle\ \mathit{Atr}\ \rangle & ::= & \texttt{STR}
\end{array} \right.
\end{array}
$$

1. Especifique uma assinatura Σ_G que descreva G algebricamente.
2. Desenhe sob a forma de árvore o Σ_G-termo que representa a seguinte frase da linguagem:

   ```
   SCHEMA Alunos ;
   Positivas := SELECT Nome, Nr FROM Alunos WHERE Class > 9;
   END.
   ```

   omitindo os detalhes do nível léxico (*cf.* `STR`).

□

Exercício 1.45 Suponha que, em resposta ao exercício 1.43, dois leitores diferentes derivaram, a partir da sintaxe dada, as assinaturas seguintes:

$$
\Sigma_1 = \left\{ \begin{array}{l}
atom2sexp : Atom \to Sexp \\
list2sexp : List \to Sexp \\
nil :\to List \\
cons : Sexp \times List \to List
\end{array} \right.
$$

$$
\Sigma_2 = \left\{ \begin{array}{l}
join : S \times L \to L \\
inj : L \to S \\
lift : A \to S \\
empty :\to L
\end{array} \right.
$$

Qual destas assinaturas escolheria para algebrizar ⟨Expressão-S⟩? Mas… por que é que não escolheu a outra? Como caracteriza formalmente a relação que existe entre ambas as assinaturas?
□

Exercício 1.46 Dadas duas gramáticas independentes de contexto G e G', sejam L_G e $L_{G'}$ as linguagens geradas por G e G', respectivamente, e Σ_G e $\Sigma_{G'}$ duas assinaturas algébricas que representam G e G', respectivamente.

Sabendo que G e G' não têm quaisquer símbolos não-terminais em comum, determine a operação gramatical $\varphi(G, G')$ que realiza a união das duas linguagens, *i.é* tal que

$$L_{\varphi(G,G')} = L_G \cup L_{G'}$$

assim como a correspondente assinatura algébrica $\Sigma_{\varphi(G,G')}$.
□

Exercício 1.47 Comente, com base no que estudou neste capítulo, a seguinte frase (célebre):

"A sintaxe é uma álgebra e a semântica é um quociente seu."

□

Exercício 1.48 Considere o seguinte fragmento de um modelo em que se pretende especificar o sistema de informação associado ao cálculo de classificações de alunos de uma dada disciplina:

$$\begin{array}{rcl}
Sistema & \cong & Inscritos \times Grupos \times Notas \\
Inscritos & \cong & Nr \rightharpoonup Nome \times Curso \\
Grupos & \cong & Nr \rightharpoonup NrGrupo \\
Notas & \cong & Teoricas \times Praticas \\
Teoricas & \cong & Teste \times Exame \times Recurso \\
Teste & \cong & Nr \rightharpoonup 20 \\
Exame & \cong & Nr \rightharpoonup 20 \\
Recurso & \cong & Nr \rightharpoonup 20 \\
Praticas & \cong & NrGrupo \rightharpoonup 20
\end{array}$$

1. Em relação à seguinte função sobre $Sistema$,

$$f(\sigma) \stackrel{\text{def}}{=} dom(\pi_1(\sigma)) - \bigcup_{i \in 3} dom(\pi_i(\pi_1(\pi_3(\sigma))))$$

caracterize a sua funcionalidade e descreva, por (breves) palavras suas, o seu significado informal.

2. Enriqueça o modelo com as funções seguintes:
   (a) função que identifica os alunos com nota prática positiva;
   (b) função que calcula a nota final de um aluno, sabendo que:
      - o peso do trabalho prático na nota final é de 40%;
      - a nota teórica mínima é 9;
      - o trabalho prático é obrigatório, *i.é*, sem informação prática positiva o aluno reprova;
      - só pode ser feita melhoria de nota no exame de $Recurso$;
   (c) função que calcula o histograma das notas finais dos alunos que não reprovaram.

NB: Para simplificar a sua especificação, trabalhe com atritmética inteira.

□

Exercício 1.49 Seja

$$
\begin{array}{rcl}
ProcNet & \cong & PId \rightharpoonup Proc \\
Proc & \cong & Q \times (I+1) \times Behaviour \times Output \\
Behaviour & \cong & (Q \times I) \rightharpoonup Q \\
Output & \cong & 2^{Pid} \times ((Q \times I) \rightharpoonup I)
\end{array}
$$

um modelo simplificado para *redes de processos* comunicantes determinísticos ($ProcNet$), em que cada processo ($Proc$) é identificado univocamente (PId) e é descrito pela informação seguinte:

- o seu estado actual (Q);
- o (próximo) item de informação pendente no seu 'input' para processamento, se for caso disso ($I + 1$);
- o seu comportamento ($Behaviour$), definido como uma tabela de transição de estados;
- o conjunto de processos (2^{PId}) para os quais se debita o 'output' que, quando existe, é dado pela tabela anexa que associa 'outputs' às respectivas transições.

1. Especifique o invariante sobre $ProcNet$ que determina que se um processo tem transições com 'output' então existe pelo menos um processo conhecido que é seu receptor.
2. Especifique uma função $ready$ que indique, para uma dada rede $r \in ProcNet$, os identificadores de todos os processos que são elegíveis para executar em r, de acordo com as seguintes condições de elegibilidade de processo para execução: (a) o processo tem 'input' pendente; (b) esse 'input' é aceitável no estado em que o processo se encontra (vide tabela de transição de estados); (c) se essa transição produz 'output' — todos os respectivos processos receptores estão sem 'input'.

□

Exercício 1.50 No contexto do Exercício 1.2, considere a seguinte especificação do modelo $SGIB$ (sistema de gestão de uma instituição bancária), agora acrescentado por forma a poder registar transacções de depósito e levantamento por cheque:

$$
\begin{array}{rclll}
SGIB & \cong & IdConta \rightharpoonup Status & & \\
Status & \cong & H : 2^{Titular} & \times & \\
 & & B : \mathbb{Z} & \times & \textit{/*saldo anterior */} \\
 & & W : NrCheq \rightharpoonup Cheque & \times & \textit{/*levantamentos (cheques) */} \\
 & & C : 2^{Credito} & & \textit{/*depósitos */} \\
Cheque & \cong & D : Data & \times & \textit{/*data do cheque */} \\
 & & C : \mathbb{N}_0 & & \textit{/*quantia levantada */} \\
Credito & \cong & D : Data & \times & \textit{/*data do depósito */} \\
 & & B : Balcao & \times & \textit{/*balcão do depósito */} \\
 & & C : \mathbb{N}_0 & & \textit{/*quantia depositada */} \\
Data & \cong & \mathbb{N} & &
\end{array}
$$

Suponha atómicas (*e.g.* alfanuméricas) as espécies não definidas e repare que, para simplificar, a espécie $Data$ é modelada por naturais (*e.g.* 19960909 modela "9 de Setembro de 1996"). Repare ainda que o saldo anterior da conta é um inteiro (negativo se o saldo estiver "a descoberto").

Especifique a função $clear : SGIB \times IdConta \times Data \longrightarrow SGIB$ que apaga de uma conta todos os movimentos (crédito ou débito) anteriores a uma dada data, actualizando convenientemente o saldo anterior. (**Sugestão:** defina as funções auxiliares que lhe parecerem convenientes.)

□

Exercício 1.51 Recorde (1.70) e mostre que, para $(f \rightharpoonup B)(\sigma)$ ser uma construção bem definida, não é necessário que f seja bijectiva, bastando que se verifique o facto seguinte, entre f e σ:

$$\forall a, a' \in dom(\sigma) : f(a) = f(a') \Rightarrow \sigma(a) = \sigma(a')$$

□

1.5 Notas Bibliográficas

Neste capítulo apresentaram-se os fundamentos da chamada *especificação construtiva* (ou *orientada a modelos*) de *Tipos Abstractos de Dados* (TADs).

A abstracção dos dados é um dos principais resultados da investigação em Ciências da Computação das duas últimas décadas. Existe uma vasta literatura sobre o assunto, que parte dos trabalhos de Guttag [GH78] e do grupo ADJ [GTW78], e inclui livros de texto como [Bp82], [EM85], [Jon80, Jon86] e [Hen88]. A abordagem algébrica 'standard' à semântica denotacional de gramáticas de contexto livre pode encontrar-se em [GTWA77]. Uma caracterização categorial dos conceitos de assinatura (categoria), modelo (functor) *etc.* pode ser encontrada em [Wan79]. A referência [AKM81] é um bom livro de texto para rever os conceitos matemáticos requeridos neste capítulo.

As referências [Jon80, Jon86] dedicam particular atenção à *especificação construtiva*. [Hen88] ilustra com muita clareza a construção de modelos semânticos para linguagens, ainda que use apenas assinaturas homogéneas.

A semântica denotacional de linguagens de programação foi abordada neste capítulo de forma intencionalmente simplista. A infinitude típica dos domínios semânticos por ela requeridos leva-nos de *conjuntos* para os chamados *domínios de Scott* desenvolvidos nos trabalhos (hoje "clássicos") de Scott & Strachey, da escola de Oxford. O leitor interessado na teoria destes domínios poderá consultar, por exemplo, as referências [Sto81, Gor79].

Capítulo 2

Especificação Recursiva

2.1 Introdução

Pelo que já se viu, a definição de um dado modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ assume a forma de um par de sistemas de definições, um para as espécies,

$$\left\{\begin{array}{lcl} \mathcal{A}(s_1) & = & \ldots \\ \mathcal{A}(s_2) & = & \ldots \\ & \vdots & \\ \mathcal{A}(s_n) & = & \ldots \end{array}\right. \tag{2.1}$$

e outro para os operadores:

$$\left\{\begin{array}{lcl} \mathcal{A}(\sigma_1) & = & \lambda \ldots \\ & \vdots & \\ \mathcal{A}(\sigma_m) & = & \lambda \ldots \end{array}\right. \tag{2.2}$$

As definições presentes nestes sistemas podem ser mutuamente dependentes, ou seja, os seus lados direitos — as reticências "$\ldots$" em (2.1) e (2.2) — podem referir-se a lados esquerdos. Por exemplo, a cláusula

$$\mathcal{A}(Sistema) = \mathcal{A}(IdConta) \rightharpoonup 2^{\mathcal{A}(Titular)} \times \mathcal{A}(Quantia) \tag{2.3}$$

que consta da Σ_{SGIB}-álgebra da secção 1.3.2, está nessas condições. Também não repugna que um operador possa usar, na sua definição, a definição de outros operadores, *e.g.*

$$\mathcal{A}(\textit{média}) = \lambda(x, y).\mathcal{A}(div)(\mathcal{A}(soma)(x, y), 2) \tag{2.4}$$

para uma assinatura contendo os operadores *média*, *soma*, *etc.*

Vejamos agora uma consequência importante da dependência mútua entre as definições de um sistema. Antes de mais, comecemos por introduzir uma simplificação de notação: sempre que fôr claro o modelo em que estamos a trabalhar — o que acontece sempre que estamos a considerar apenas um — omitiremos a sua designação, *i.é* escreveremos x em lugar de $\mathcal{A}(x)$, onde x é uma qualquer entidade sintática (espécie ou operador).

As definições de espécies e operadores tornam-se assim consideravelmente mais legíveis. Por exemplo, a expressão (2.3) reduz-se a

$$Sistema = IdConta \rightharpoonup 2^{Titular} \times Quantia$$

e a (2.4) reduz-se a

$$\textit{média} = \lambda(x, y).div(soma(x, y), 2)$$

com esta simplificação.

Consideremos agora a definição seguinte:

$$List = \{NIL\} \cup X \times List \tag{2.5}$$

i.é a simplificação da cláusula

$$\mathcal{A}(List) = \{NIL\} \cup \mathcal{A}(X) \times \mathcal{A}(List)$$

de um modelo ($\mathcal{A}$) para a espécie $List$. Este modelo para *listas* é bem conhecido: uma lista ou "não é nada" (NIL), ou é constituída pelo emparelhamento da sua cabeça com a sua cauda. Por exemplo, a lista $<x_1, x_2, x_3>$ vem neste modelo representada pelo aninhamento de pares $\langle x_1, \langle x_2, \langle x_3, NIL\rangle\rangle\rangle$. $List$ vem pois dependente, em (2.5), de X — a espécie dos seus elementos — o que faz sentido.

Contudo, de (2.5) depreendemos também que $List$ é dependente dela própria! Esta dependência "circular" fará sentido? É o que veremos a seguir.

2.2 Sobre as Definições Recursivas

Definição 2.1 (Sistemas de Definições Recursivas) *Seja dado um sistema de definições (mutuamente dependentes) das variáveis X_1 a X_n:*

$$\left\{\begin{array}{lcl} X_1 & = & \ldots \\ X_2 & = & \ldots \\ & \vdots & \\ X_n & = & \ldots \end{array}\right.$$

A relação de dependência $\delta \subseteq \{X_1, \ldots, X_n\}^2$ *entre definições constrói-se como se segue:*

$$X \, \delta Y \Leftrightarrow \text{Y ocorre no lado direito da definição de X}$$

Seja δ^+ *o fecho transitivo de* δ*. Sempre que existe uma variável* X *tal que* $X \, \delta X$ *ou* $X \, \delta^+ X$*, dizemos que essa variável está definida* recursivamente — *recursividade* directa *ou* indirecta*, respectivamente.*

Um sistema admitindo uma ou mais variáveis directa ou indirectamente recursivas diz-se um sistema mutuamente recursivo.
$\square$

As definições recursivas são habituais em Matemática. Por exemplo, as duas cláusulas que definem a função factorial $! : \mathbb{N}_0 \to \mathbb{N}$,

$$\left\{ \begin{array}{rcl} 0! & = & 1 \\ (n+1)! & = & (n+1) \times n! \end{array} \right.$$

induzem o cálculo recursivo dessa função. A nível de conjuntos, a definição do fecho transitivo de uma relação R,

$$R^+ = R \cup R \circ R^+ \tag{2.6}$$

é recursiva. O mesmo acontece com a definição do conjunto W_Σ de todos os Σ-termos gerados por uma assinatura Σ, que é um sistema recursivo de forma

$$(W_{\Sigma,s} = \ldots)_{s \in S}$$

relembrar a equação (1.10).

Mas a recursividade é um fenómeno ainda mais vulgar do que parece à primeira vista. Por exemplo, a equação

$$x = 3 + \frac{x}{2} \tag{2.7}$$

pode ser considerada uma definição recursiva (em $\mathbb{R}$) do número 6 (*i.é* a sua solução).

Este último exemplo é sugestivo, pois mostra-nos que podemos considerar a definição recursiva de um objecto como uma *equação* cuja “solução” é esse objecto. Ou seja, um sistema de definições recursivas “é” um sistema de equações, tal como estas são entendidas vulgarmente em Matemática.

Contudo, esta perspectiva levanta alguns problemas. Por exemplo, a seguinte definição recursiva (*i.é* equação):

$$x = \frac{x^2 + 3}{4}$$

admite as soluções 1 e 3. O que é que estamos aqui a definir, recursivamente? O 1 ou o 3?

Se quisermos um exemplo “patológico”, a equação

$$x = x$$

define qualquer objecto, pois tem uma infinidade de soluções. Há também a situação oposta: em $I\!N$, a equação

$$x = x + 1$$

não tem soluções, *i.é* temos outra situação anormal — uma definição recursiva que nada define!

Em resumo, temos que caracterizar com cuidado o contexto algébrico em que escrevemos definições recursivas, para garantir que as mesmas tenham (pelo menos) uma solução. No caso de existir mais do que uma solução, precisamos de um critério para escolher uma dessas soluções (a “melhor”, se existir).

Voltemos à equação (2.7), e procuremos a sua solução, *i.é* resolvámo-la em ordem a x:

$$\begin{aligned} x = 3 + x/2 \quad &\equiv \quad x + (-x/2) = (3 + x/2) + (-x/2) && (2.8)\\ &\equiv \quad x \times (1 + (-1/2)) = 3 + (x/2) + (-x/2)) && (2.9)\\ &\equiv \quad x \times 1/2 = 3 && (2.10)\\ &\equiv \quad x = 3 \times 2 && (2.11)\\ &\equiv \quad x = 6 \end{aligned}$$

Esta resolução, que chegou de facto à solução $x = 6$, foi possível graças a determinadas propriedades válidas no universo de definição da equação. Por exemplo, o primeiro passo (2.8) assume a compatibilidade da soma em relação à igualdade, *i.é* a propriedade

$$\begin{array}{rcl} a & = & b \\ c & = & d \\ \hline a + c & = & b + d \end{array}$$

Em (2.9) assume-se a distributividade do produto pela soma

$$a \times (b + b) = a \times b + a \times c$$

e a associatividade desta última. Em (2.10) ocorre uma importante simplificação: a adição de dois inversos aditivos resulta no elemento neutro da soma. O mesmo acontece em (2.11), agora em relação a inversos multiplicativos.

Em suma, as duas estruturas de *grupo*, a aditiva $\langle \mathbb{R}; +, 0, - \rangle$ e a multiplicativa $\langle \mathbb{R}; \times, 1, ^{-1} \rangle$, permitem a técnica de *cancelamento*, *e.g.*

$$x + b = c \ \equiv \ x = c - b$$

que nos permite, com tanta facilidade, passar “coisas” de um lado para o outro de uma equação.

Tentemos agora o mesmo tipo de resolução para a equação

$$x = R \cup R \circ x \tag{2.12}$$

em $\langle 2^{P \times P}; \cup, \emptyset \rangle$, que é exactamente (2.6) renomeando R^+ para x. Tal como na resolução acima, a ideia é "vermo-nos livres" de $R \circ x$ no segundo membro de (2.12), passando esse termo para o primeiro membro. Ora se, por um lado, a união $\cup$ é associativa e admite $\emptyset$ como elemento neutro, por outro lado apenas $\emptyset$ se tem a si próprio como inverso, sendo falso que

$$\forall R \subseteq P \times P : (\exists R' \subseteq P \times P : R \cup R' = \emptyset)$$

Portanto, a "habitual" técnica de resolução de equações não é aplicável à equação $x = R \cup R \circ x$, pelo facto de estar definida num universo algébrico não suficientemente rico em propriedades.

Será então *impossível* resolver $x = R \cup R \circ x$, em ordem a x? A simples substituição de x, pelo seu valor,

$$\begin{array}{rcl} x & \equiv & R \cup R \circ x \\ & \equiv & R \cup R \circ (R \cup R \circ x) \\ & \equiv & R \cup R^2 \cup R^2 \circ x \end{array}$$

e assim sucessivamente,

$$\begin{array}{cl} \equiv & R \cup R^2 \cup R^2 \circ (R \cup R \circ x) \\ \equiv & R \cup R^2 \cup R^3 \cup R^3 \circ x \\ \equiv & \ldots \end{array}$$

apesar de ineficaz — pois permanece sempre um termo em x no segundo membro — aponta-nos a estratégia a seguir, que se baseia nas definições e teoremas que se seguem.

Definição 2.2 (Ponto-Fixo) *Seja $f : A \rightarrow A$ uma função. Qualquer $a_0 \in A$ tal que*

$$a_0 = f(a_0)$$

designa-se um ponto-fixo *de f.*
□

Por exemplo, dada a função

$$\begin{array}{rcl} f : [0, 10] & \rightarrow & [0, 10] \\ x & \leadsto & 10 - x \end{array}$$

fácilmente se vê que 5 é ponto-fixo de f, pois $f(5) = 10 - 5 = 5$.

Também se constata fácilmente que uma solução x_0 de uma equação qualquer

$$x = f(x)$$

é um ponto-fixo de f, pois é tal que $x_0 = f(x_0)$. Por exemplo, a função f implícita em (2.7) é $f(x) = 3 + \frac{x}{2}$, e de facto $f(6) = 6$.

Definição 2.3 (Função Crescente) *Seja $f : A \to B$ uma função, e $\leq_A$ e $\leq_B$ duas ordens parciais definidas sobre A e B, respectivamente. A função f diz-se-á* crescente*, ou* monótona*, se*

$$\forall a, a' \in A : a \leq_A a' \Rightarrow f(a) \leq_B f(a')$$

□

Teorema 2.1 (Pontos-fixos em Reticulados) *[Tarski 1955] Seja*

- $\mho = \langle A; \leq \rangle$ *um* reticulado completo*,*
- *$f : A \to A$ uma função crescente com respeito a $\leq$;*
- *P o conjunto de todos os pontos-fixos de f,* i.é

$$P = \{a \in A \mid \; f(a) = a\}$$

Então

- *P é não-vazio e $\langle P; \leq \rangle$ é um (sub)reticulado completo.*
- *Em particular, o maior de todos os pontos-fixos ($\bigvee P$) e o menor ($\bigwedge P$) são dados por:*

$$\bigvee P = \bigvee \{x \mid \; f(x) \geq x\} \tag{2.13}$$

$$\bigwedge P = \bigwedge \{x \mid \; f(x) \leq x\} \tag{2.14}$$

Demonstração[1]*: Seja*

$$u = \bigvee \{x \mid \; f(x) \geq x\} \tag{2.15}$$

cuja existência é garantida pela completude do reticulado subjacente. É então claro que, para todo o x tal que $f(x) \geq x$, $x \leq u$. Sendo f crescente, teremos

$$f(x) \leq f(u) \tag{2.16}$$

[1] Este teorema, a base da teoria da *recursividade* essencial à Informática teórica, é conhecido por Teorema de Knaster-Tarski, por derivar de um resultado prévio de Knaster(1928). Esta demonstração, de 1939, só foi publicada em 1955.

e, por transitividade de $\leq$,

$$x \leq f(u) \tag{2.17}$$

Logo $f(u)$ é um majorante; como u é o menor dos majorantes, temos

$$u \leq f(u) \tag{2.18}$$

e, por monotonia de f,

$$f(u) \leq f(f(u))$$

ou seja, $f(u)$ pertence ao conjunto $\{x \mid\ f(x) \geq x\}$; consequentemente,

$$f(u) \leq u \tag{2.19}$$

por (2.15). As fórmulas (2.18,2.19) implicam que u é um ponto-fixo de f, i.é *o maior ('join') de todos os pontos-fixos de f:*

$$\bigvee P = \bigvee\{x \mid\ f(x) \geq x\} \in P \tag{2.20}$$

o que completa a demonstração de (2.13). Na demonstração de (2.14) considera-se o reticulado dual, $\mho' = \langle A; \geq \rangle$, que é também completo e onde f é ainda crescente. O 'meet' em $\mho$ corresponde ao 'join' em $\mho'$; aplicando aqui (2.20), concluimos

$$\bigwedge P = \bigwedge\{x \mid\ f(x) \leq x\} \in P$$

Falta-nos agora demonstrar que $\langle P; \leq \rangle$ é reticulado completo. Seja

$$[a, b] = \{x \in A \mid\ a \leq x \leq b\}$$

a designação de um intervalo em A, e $\top = \bigvee A$. Seja $Y \subseteq P$. O sistema $\langle [\bigvee Y, \top]; \leq \rangle$ é um reticulado completo. Logo, os pontos-fixos de f, restringida ao intervalo $[\bigvee Y, \top]$, têm um g.l.b. $v \in P$, cf. *equação (2.17). Ora $v \sqsupseteq y$ para qualquer $y \in Y$, pois $v \sqsupseteq \bigvee Y$. Logo v é o $l.u.b.$ de Y no sistema $\langle P; \leq \rangle$. O $g.l.b.$ de Y neste sistema obtém-se por raciocínio dual.* Q.E.D.
□

Exercício 2.1 Seja dada a aplicação real de variável real

$$\begin{array}{rcl} f : [0, 10] & \rightarrow & [0, 10] \\ x & \leadsto & 3 + \frac{x}{2} \end{array}$$

implícita na definição recursiva (2.7).

1. Mostre que o intervalo $[0, 10]$, ordenado pela relação $x \leq y$ em $\mathbb{R}$, é um reticulado completo.
2. Mostre que f é crescente em relação a $\leq$.
3. Aplique o teorema de Tarski para mostrar que o sub-reticulado das soluções da definição (2.7) é o conjunto singular $\{6\}$.

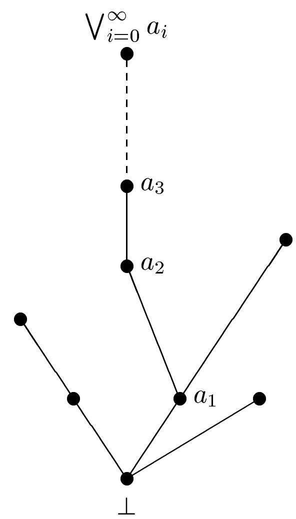


Figura 2.1: Limite de uma cadeia ascendente numa *c.p.o*.

□

O teorema anterior deixa em aberto duas questões de natureza prática:

- não sendo P, em geral, um conjunto singular, como dar um significado "único" a uma equação $x = f(x)$?
- como "mecanizar" o cálculo desse significado?

É o que vamos ver de seguida.

Definição 2.4 (C.p.o.) *Seja $\leq$ uma ordem parcial sobre A tal que*

- $\exists \perp \in A : (\forall a \in A : \perp \leq a)$ i.é $\langle A; \leq \rangle$ *é limitada inferiormente.*
- *toda a cadeia ascendente $a_0 \leq a_1 \leq \ldots \leq a_i \leq a_{i+1} \leq \ldots$ de elementos de A tem um limite superior em A,* i.é

$$\bigvee_{i=0}^{\infty} a_i \in A$$

cf. *Figura 2.1*.

Dizemos então que A é um conjunto c.p.o *('complete partially-ordered set'), ou simplesmente, uma* c.p.o *('complete partial order').* □

Exercício 2.2 Mostrar que $A \rightharpoonup B$, ordenado por $\leq$ tal que

$$f \leq g \stackrel{\text{def}}{=} dom(f) \subseteq dom(g) \wedge \forall a \in dom(f) : f(a) = g(a)$$

é um conjunto *c.p.o*.
□

Definição 2.5 (Função Contínua) *Sejam A e B dois conjuntos* c.p.o. *e $f : A \rightarrow B$ uma função entre eles. f dir-se-á* contínua *sse*

$$f(\bigvee_{i=0}^{\infty} a_i) = \bigvee_{i=0}^{\infty} f(a_i)$$

□

Exercício 2.3 Mostre que

1. toda a função contínua é crescente;
2. a função identidade $f(x) = x$ é crescente e contínua;
3. toda a função constante $f(x) = k$ é crescente e contínua.

□

O teorema seguinte fornece-nos um "algorítmo" para calcularmos o menor dos pontos-fixos de uma definição recursiva sobre *c.p.o.*s.

Teorema 2.2 (Primeiro Teorema da Recursividade) *[Kleene 1952] Seja*

$$(A; \leq)$$

uma c.p.o. *e $f : A \rightarrow A$ uma função contínua. Então o menor dos pontos-fixos de f, designado μf, existe e é determinado por*

$$\mu f = \bigvee_{i=0}^{\infty} f^i(\perp)$$

Demonstração*: Se f é contínua é também crescente. Logo $(f^i(\perp))_{i=0}^{\infty}$ é uma cadeia ascendente e tem um $l.u.b.$ que designaremos por μf. Vamos agora mostrar que μf é o menor ponto-fixo de f, em dois passos:*

1. *μf é ponto-fixo de f: Sendo f contínua, teremos*

$$\begin{array}{rcl} f(\mu f) & = & f(\bigvee_{i=0}^{\infty} f^i(\perp)) \\ & = & \bigvee_{i=0}^{\infty} f(f^i(\perp) \\ & = & \bigvee_{i=0}^{\infty} f^{i+1}(\perp) \\ & = & \bigvee_{i=1}^{\infty} f^i(\perp) \\ & = & \mu f \end{array}$$

   Logo $\mu f = f(\mu f)$.

2. *$\mu f \leq y$ para qualquer outro ponto-fixo y de f:*

   (a) *primeiro, mostraremos que $f^i(\perp) \leq y$ para todo o $i \geq 0$, por indução sobre i:*
   Para $i = 0$,
   $$f^0(\perp) = \perp \leq y$$
   Para $i > 0$, temos $f^{i-1}(\perp) \leq y$ por hipótese de indução; então
   $$\begin{array}{rcl} f^i(\perp) & = & f(f^{i-1}(\perp)) \\ & \leq & f(y) \\ & = & y \end{array}$$
   pois f é crescente e y é ponto-fixo de f.

   (b) *Sendo $f^i(\perp) \leq y$ para $i \geq 0$, temos que y é majorante da cadeia $(f^i(\perp))_{i=0}^{\infty}$; mas sendo μf o $l.u.b.$ dessa cadeia, teremos que $\mu f \leq y$, como queríamos.*

□

Exercício 2.4 Seja

$$\begin{array}{rcl} f : [0,10] & \rightarrow & [0,10] \\ x & \rightsquigarrow & 3 + \frac{x}{2} \end{array}$$

a função real de variável real implícita na definição recursiva (2.7). Considere o intervalo [0,10] ordenado pela relação $x \leq y$ em $\mathbb{R}$, com limite universal inferior $\perp = 0$, e $\bigvee = Max$, onde $Max : 2^{\mathbb{R}} \rightarrow \mathbb{R}$ é a função que calcula o máximo de um conjunto de reais.

1. Mostre que f é contínua.
2. Mostre que a solução da equação (2.7) pelo Teorema de Kleene é dada por

$$\mu f = lim_{i\rightarrow\infty}\ 3 \times (2 - \frac{1}{2^{i-1}}) = 6$$

□

As resoluções dos Exercícios 2.1 e 2.4 mostram-nos duas maneiras de resolver a equação $x = 3 + \frac{x}{2}$ que são mais complicadas do que a resolução habitual, pois esses exercícios ignoram propriedades que são válidas sobre $\mathbb{R}$. No entanto, em muitos outros casos essas maneiras são as únicas disponíveis, como já vimos. É o caso da equação (2.12), que vamos agora resolver pelo Teorema de Kleene, em $\langle 2^{P\times P}; \subseteq, \emptyset\rangle$. Teremos então:

$$\begin{array}{rcl} f : 2^{P\times P} & \to & 2^{P\times P} \\ x & \leadsto & R \cup R \circ x \end{array} \tag{2.21}$$

e

$$\begin{array}{rcl} f^0(\emptyset) & = & \emptyset \\ f^1(\emptyset) & = & f(f^0(\emptyset)) \\ & = & f(\emptyset) \\ & = & R \cup R \circ \emptyset \\ & = & R \\ f^2(\emptyset) & = & R \cup R \circ R \\ & = & R \cup R^2 \\ & \vdots & \\ f^i(\emptyset) & = & \bigcup_{j=1}^{i} R^j \end{array}$$

Logo

$$\begin{array}{rcl} \mu f & = & \bigcup_{i=0}^{\infty} f^i(\emptyset) \\ & = & \bigcup_{i=0}^{\infty}(\bigcup_{j=1}^{i} R^j) \\ & = & \bigcup_{j=1}^{\infty} R^j \\ & = & R^+ \textit{(fecho transitivo de } R\textit{)} \end{array}$$

Omitimos, por razões de economia de exposição, duas demonstrações importantes: que $\langle 2^{P\times P}; \subseteq, \emptyset\rangle$ é uma *c.p.o.* e que a aplicação f da definição (2.21) é contínua — o que decorre das propriedades dos conjuntos e fica dado como exercício.

Em resumo: dada a garantia da sua existência em *c.p.o.*s e o carácter "algorítmico" do seu cálculo via Teorema de Kleene, é vulgar considerar o menor dos pontos-fixos (μf) como sendo o "significado" 'standard' de uma definição recursiva

$$x = f(x)$$

O que não impede que haja outros pontos-fixos, é claro.

Exercício 2.5 Verificar se a relação universal $P \times P \in 2^{P\times P}$ é o maior de todos os pontos-fixos da equação $x = R \cup R \circ x$.
□

Note-se agora que o teste de continuidade de uma função pode ser largamente simplificado pelo teorema seguinte.

Teorema 2.3 (Continuidade de Funções) *Seja $f(x)$ uma expressão em x definida pela composição de funções crescentes e da variável x. Então f é contínua.*

Demonstração*: Faz-se por indução estrutural sobre a álgebra dos termos onde f está definida. Assim, $f(x)$ ou é x (identidade) ou é uma constante k, ou é da forma $\sigma(f_1(x), \ldots, f_n(x))$ onde σ é crescente e cada $f_i(x)$ é uma expressão em x envolvendo funções crescentes. Nos dois primeiros casos, $f(x)$ é contínua,* cf. *Exercício 2.3. No terceiro caso (geral), mostraremos*

1. *que f é crescente;*
2. *que $\bigvee_{i=0}^{\infty} f(x_i) \leq f(\bigvee_{i=0}^{\infty} x_i)$;*
3. *o facto (2) na direcção oposta($\geq$ em lugar de $\leq$)*

A conjunção dos resultados (2) e (3) garante, por antissimetria de $\leq$, o resultado que queremos provar.

1. *Seja $z \leq y$; então, se cada $f_i(x)$ é contínua por hipótese de indução, também é crescente; logo $\forall 1 \leq i \leq n : f_i(z) \leq f_i(y)$. Então, porque σ é crescente em todos os seus argumentos, temos*

$$\sigma(f_1(z), \ldots, f_n(z)) \leq \sigma(f_1(y), \ldots, f_n(y))$$

i.é

$$f(z) \leq f(y)$$

Logo f é crescente.

2. *Como $x_i \leq \bigvee_{i=0}^{\infty} x_i$ para qualquer $i \geq 0$, então, pelo resultado anterior, $f(x_i) \leq f(\bigvee_{i=0}^{\infty} x_i)$ para qualquer $i \geq 0$. Logo, no limite ($i \to \infty$), $\bigvee_{i=0}^{\infty} f(x_i) \leq f(\bigvee_{i=0}^{\infty} x_i)$.*
3. *A hipótese de indução aqui garante-nos que $f_j(\bigvee_{i=0}^{\infty} x_i) \leq \bigvee_{i=0}^{\infty} f_j(x_i)$ para $1 \leq j \leq n$; logo,*

$$\begin{array}{rcl} f(\bigvee_{i=0}^{\infty}(x_i)) & = & \sigma(f_1(\bigvee_{i=0}^{\infty} x_i), \ldots, f_n(\bigvee_{i=0}^{\infty} x_i)) \\ & \leq & \sigma(\bigvee_{i=0}^{\infty} f_1(x_i), \ldots, \bigvee_{i=0}^{\infty} f_n(x_i)) \end{array} \tag{2.22}$$

Notemos agora que, para cada $1 \leq j \leq n$, vai existir um $i_j \geq 0$ no qual se atinge o limite da cadeia correspondente, ou seja,

$$k \geq i_j \Rightarrow (\bigvee_{i=o}^{\infty} f_j(x_i) = f_j(x_k))$$

Seja i_0 o máximo de $i_1, \ldots, i_n$. Então, para cada $1 \leq j \leq n$,

$$\bigvee_{i=o}^{\infty} f_j(x_i) = f_j(x_{i_0})$$

Assim, podemos reescrever (2.22) em

$$\begin{array}{rcl} \sigma(f_1(x_{i_0}),\ldots,f_n(x_{i_0})) & = & f(x_{i_0}) \\ & \leq & \bigvee_{i=0}^{\infty} f(x_i) \end{array}$$

Combinando resultados, temos então, por transitividade de $\leq$,

$$f(\bigvee_{i=0}^{\infty} x_i) \leq \bigvee_{i=0}^{\infty} f(x_i)$$

ou seja, f é contínua.

Q.E.D.
□

Assim, por exemplo, a continuidade de $f(x) = R \cup R \circ x$ resumir-se-á a demonstrar que a união $\cup$ e $g(x) = R \circ x$, para qualquer $R \subseteq P \times P$ são funções crescentes em relação à inclusão de conjuntos ($\subseteq$).

2.3 Modelos Semânticos Recursivos

2.3.1 Motivação

Após esta exposição sobre resultados genéricos da teoria da recursividade, cumpre-nos aplicá-los ao nosso contexto inicial — a definição recursiva dos modelos de espécies em *Sets*. Mesmo sem investigarmos alguma estrutura ordenada em *Sets*, podemos já avaliar modelos "patológicos" como

$$\mathcal{A}(s) = \mathcal{A}(s)$$

(*i.é* $s = s$) que "não definem nada", ou seja, $\mathcal{A}(s) = \emptyset$; de facto, este é o menor dos pontos-fixos da definição acima, para $\perp = \emptyset$ em *Sets*. Qual será a ordem $\leq$ a impôr a *Sets*? Peguemos na equação (2.5) de novo,

$$List = \{NIL\} \cup X \times List \tag{2.23}$$

e pensemos agora no (meta) significado do sinal "=" que aí ocorre. De que tipo de "igualdade" se trata? Por exemplo, quando escrevemos

$$Quantia = \mathbb{N}_0 \tag{2.24}$$

queremos de facto dizer que *Quantia* é o nome que damos a $\mathbb{N}_0$ no modelo? Ou queremos dizer que a toda a quantia q corresponde um número natural que representa o número de unidades, de uma determinada moeda padrão, (*e.g.* escudos) que q contém?

Parece mais razoável esta segunda interpretação. De igual modo, a todo o número natural n corresponde uma quantia, que é a quantia cujo valor é n numa determinada moeda padrão. Assim sendo, estamos a dizer que *Quantia* é *isomorfa* de $\mathbb{N}_0$, ou seja, devemos reescrever (2.24) para

$$Quantia \cong \mathbb{N}_0$$

Note-se que a substituição, num modelo $\mathcal{A}$, de um conjunto por um seu *isomorfo* é "pacífica" em termos semânticos, porquanto não altera a congruência $\cong_{\mathcal{A}}$. De facto, dois modelos isomorfos $\mathcal{A} \cong \mathcal{B}$ são semânticamente indistinguíveis. Portanto, podemos reescrever (2.23) para

$$List \cong \{NIL\} \cup X \times List \tag{2.25}$$

obtendo uma definição mais geral para a mesma semântica de *List*. Fiquemos, assim, com a ideia geral de que, num modelo em *Sets*, não "vale a pena" distiguirmos modelos isomorfos, ou seja, sempre que declaramos, para $s \in S$,

$$\mathcal{A}(s) = e$$

estamos a dizer que o portador de s na álgebra $\mathcal{A}$ é isomorfo do conjunto que a expressão e designa em *Sets*.

Notemos agora que, em *Sets*,

$$A \cap B = \emptyset \Rightarrow A \cup B \cong A + B \tag{2.26}$$

Logo, também podemos escrever

$$List \cong \{NIL\} + X \times List \tag{2.27}$$

em lugar de (2.25), pois $NIL \notin X \times List$ por não ser um par-ordenado. Mais ainda, todos os conjuntos singulares são isomorfos entre si:

$$\{NIL\} \cong \{\emptyset\} \cong \{1\} \cong \ldots \cong 1 \tag{2.28}$$

podendo ser abstractamente designados pelo objecto ("canónico") 1; assim, (2.27) reescreve para

$$List \cong 1 + X \times List \tag{2.29}$$

Vamos agora mostrar que X^* — *cf.* definição (1.48) — é uma solução, *i.é* ponto-fixo de (2.29). Queremos portanto provar que

$$X^* \cong 1 + X \times X^* \tag{2.30}$$

Ora

$$\begin{array}{rcl} X^* & = & \bigcup_{i \geq 0} X^i \\ & \cong & \sum_{i \geq 0} X^i \end{array} \tag{2.31}$$

por uma aplicação de (2.26) iterada a i-argumentos X^i, que são todos disjuntos, pois temos que

$$A \neq B \Rightarrow X^A \cap X^B = \emptyset \tag{2.32}$$

(leia-se: duas aplicações com igual codomínio X mas domínios diferentes A e B nunca podem ser a mesma aplicação). Substituindo X^* no lado direito de (2.30) via (2.31) teremos:

$$X^* \cong 1 + X \times (\sum_{i \geq 0} X^i)$$

que reescreve para

$$X^* \cong 1 + \sum_{i \geq 0} X \times X^i \tag{2.33}$$

iterando a i-argumentos a seguinte lei distributiva

$$A \times (B + C) \cong A \times B + A \times C \tag{2.34}$$

válida em *Sets*. Somos agora tentados a escrever X^{i+1} em lugar de $X \times X^i$, em (2.33); de facto, a função *cons* atrás estudada (1.85) estabelece o isomorfismo

$$X \times X^i \cong X^{i+1} \tag{2.35}$$

da esquerda para a direita. Fazendo uma mudança de variável, $j = i + 1$, reescrevemos (2.33) para

$$X^* \cong 1 + \sum_{j \geq 1} X^j$$

Finalmente, 1 pode ser substituído por X^0, já que

$$X^0 \cong 1 \tag{2.36}$$

(leia-se: o conjunto de todas as funções de domínio vazio e contradomínio X, é singular — contém apenas a função totalmente indefinida); portanto,

$$\begin{array}{rcl} X^* & \cong & X^0 + \sum_{j \geq 1} X^j \\ & \cong & \sum_{j \geq 0} X^j \\ & \cong & \bigcup_{j \geq 0} X^j \\ & = & X^* \end{array}$$

ou seja, X^* é ponto-fixo de $List \cong 1 + X \times List$, como queríamos.

Será o menor dos pontos-fixos? Antes de respondermos, vamos reflectir sobre o raciocínio acima, analisando em mais detalhe a álgebra que lhe está subjacente.

2.3.2 Introdução ao Cálculo de Isomorfismo em $Sets$

As propriedades (2.26), (2.32), (2.34), (2.35) e (2.36) são apenas uma amostra da álgebra que nos permite, em $Sets$, raciocinar sobre estruturas de dados isomorfas. De facto, não é difícil mostrar (*e.g.* por argumentos de cardinalidades ou estabelecimento explícito de bijecções) que $\times$ e $+$ têm em $Sets$ as propriedades de um *semi-anel comutativo*, a menos de isomorfismo ($\cong$) [2]:

$$
\begin{aligned}
A \times B &\cong B \times A && (2.37)\\
A \times (B \times C) &\cong (A \times B) \times C && (2.38)\\
A \times 1 &\cong A && (2.39)\\
A + B &\cong B + A && (2.40)\\
A + (B + C) &\cong (A + B) + C && (2.41)\\
A + 0 &\cong A && (2.42)\\
A \times 0 &\cong 0 && (2.43)\\
A \times (B + C) &\cong (A \times B) + (A \times C) && (2.44)
\end{aligned}
$$

É assim que — tal como vimos na secção 1.3.1 — faz sentido escrevermos produtos finitos,

$$A_1 \times \ldots \times A_n \;\text{ i.é }\; \Pi_{i=1}^{n} A_i \qquad (2.45)$$

assim como uniões disjuntas finitas,

$$A_1 + \cdots + A_n \;\text{ i.é }\; \sum_{i=1}^{n} A_i$$

e tem-se que

$$\underbrace{A \times \ldots \times A}_{n} \cong A^n \qquad (2.46)$$

[2] Semi-anel e não anel porque não há inversos aditivos, isto é, $+, 0$ não formam um grupo (formam um monóide abeliano). Aliás, $\times, 1$ também formam um monóide abeliano.

Recomenda-se ao leitor que acompanhe o estudo destas leis com uma análise informal do seu significado prático à luz da correspondência da figura 1.5, da página 25. Por exemplo, a associatividade do produto (2.38) converte-se na seguinte equivalência óbvia entre estruturas de dados `record`:

```
record                                record
    F: A;                                 F: record
    S: record                                    F: A;
           F: B;    equivalente a                S: B
           S: C;                              end;
        end                               S: C;
end;                                  end;
```

em PASCAL.

$$\underbrace{A + \ldots + A}_{n} \quad \cong \quad n \times A \tag{2.47}$$

como se esperava. E, é claro, os operadores $+$ e $\times$ são compatíveis com $\cong$, ou seja, se $A \cong B$ e $C \cong D$, então

$$A \times C \cong B \times D$$

e

$$A + C \cong B + D$$

o que nos permite substituir isomorfos por isomorfos, estruturalmente, *cf.* por exemplo o salto de (2.27) para (2.29).

Exercício 2.6

1. Demonstre a validade, em *Sets*, das leis acima citadas, isto é, estabeleça bijecções entre os membros de cada lei, ou use argumentos sobre as suas cardinalidades.

2. Quais dessas leis justificam o resultado

$$1 + 1 \quad \cong \quad 2 \ \ ? \tag{2.48}$$

□

2.3.3 Pontos-Fixos em $Sets$

Vimos na secção anterior que a estrutura de *Sets* é bastante rica em propriedades que nos permitem raciocínios como o que fizemos na secção 2.3.1. Contudo, reparamos que não existem inversos aditivos nem multiplicativos. Por isso não podemos "resolver" (2.29) em ordem a $List$, obtendo X^*; em vez disso, limitámo-nos a conjecturar que X^* podia ser uma solução e provámo-lo de seguida.

Uma maneira de descobrir X^* como solução seria calculá-la via Teorema de Kleene, se isso fôr possível. Mesmo antes de investigarmos uma topologia que seja adequada em *Sets* para o efeito, construamos a correspondente cadeia ascendente de Kleene, a partir de 0 (*i.é* $\emptyset$):

$$\left.\begin{array}{rcl} f^0(0) & = & 0 \\ f^1(0) & = & 1 + X \times 0 \\ & \cong & 1 + 0 \\ & \cong & 1 \\ f^2(0) & = & 1 + X \times 1 \\ & \cong & 1 + X \\ f^3(0) & = & 1 + X \times (1 + X) \\ & \cong & 1 + X + X^2 \\ & \vdots & \\ f^i(0) & = & \sum_{j \geq 0}^{i-1} X^j \\ & \vdots & \end{array}\right\} \tag{2.49}$$

que parece tender, de facto, para $\sum_{j \geq 0} X^j \cong X^*$.

Em que sentido é a cadeia (2.49) uma cadeia ascendente limitada superiormente? Aliás, qual é a ordem (*c.p.o.*) subjacente?

É imediata a constatação de que uma tal ordem terá que comparar conjuntos em *Sets* "a menos de um isomorfismo", já que a *igualdade* entre conjuntos é, em *Sets*, uma relação de equivalência *fina* demais. Portanto, não nos serve como ordem ($\leq$) de Kleene a simples relação de inclusão entre conjuntos ($\subseteq$), já que, por exemplo, pretendemos que

$$1 \leq 1 + X \tag{2.50}$$

— *cf.* $f^1(0) \leq f^2(0)$ em (2.49) — seja verdadeiro, e temos que

$$1 \subseteq 1 + X$$

é falso.

A ordem que nos serve será, pois, a de "subconjunto a menos de um isomorfismo", ou seja

$$\begin{array}{rcl} X \leq Y & \stackrel{\mathrm{def}}{=} & \exists S \subseteq Y : X \cong S \\ & \Leftrightarrow & \exists h : X \to Y : h \textit{ é injectiva} \end{array} \tag{2.51}$$

Reparemos agora que esta ordem satisfaz (2.50) de facto: podemos escolher para h a injecção de 1 na união disjunta $1 + X$, isto é:

$$\begin{array}{rcl} i_1 : 1 & \to & 1 + X \\ x & \rightsquigarrow & (1, x) \end{array}$$

cf. (1.16).

Genericamente, para todo o A, B verifica-se

$$\begin{array}{rcl} A & \leq & A+B \\ B & \leq & A+B \end{array}$$

através das duas injecções i_A e i_B associadas ao co-produto $A+B$. Assim, é fácil verificar que a cadeia (2.49) é de facto ascendente em relação à ordem (2.51), pois é tal que

$$f^i(0) \cong f^{i-1}(0) + X^{i-1}$$

para $i > 0$, ou seja

$$f^{i-1}(0) \leq f^i(0)$$

Notemos, contudo, que $\leq$ não é uma ordem parcial, mas sim uma *pré-ordem*, já que não verifica a propriedade de antissimetria:

$$X \leq Y \wedge Y \leq X \not\Rightarrow X = Y$$

O que se verifica é

$$X \leq Y \wedge Y \leq X \Rightarrow X \cong Y$$

ou seja, o fecho simétrico da pré-ordem $\leq$ é a relação de isomorfismo em *Sets*. Isto significa que teremos que abordar o cálculo de limites de Kleene, $\bigvee_{i=0}^{\infty} f^i$, não exactamente em *Sets*, mas sim no quociente $Sets/\cong$. Mas esse estudo aprofundado fica, de momento, por fazer, já que merece uma abordagem categorial suficientemente geral.

Exercício 2.7 Discuta formalmente a validade da afirmação seguinte, em $Sets$:

- 0 é o menor dos pontos-fixos da definição

$$List \cong X \times List$$

□

Exercício 2.8 Serão as construções básicas da notação $Sets$ (produto, co-produto e exponenciação) monótonas com respeito à ordem $\leq$ (2.51)? Justifique adequadamente.
□

2.3.4 Funcionalidade de Modelos Recursivos

O leitor com certeza reparou que grande parte do nosso esforço na construção de especificações recursivas se centrou na definição de modelos possivelmente recursivos para as espécies, ignorando até agora os modelos para os operadores.

Nesta secção vamos justificar essa atitude mostrando que os modelos de operadores devem ser construídos após terem sido definidos os modelos das espécies envolvidas. Mais do que isso, uma boa parte dessa construção é induzida pelas estruturas escolhidas para modelar as espécies.

Vejamos como: vamos supor que uma assinatura $\Sigma : \Omega \to S^\star \times S$ admite duas espécies s e r e admite um operador

$$f : s \to r$$

Vamos supor ainda que já se escolheram, num determinado modelo $\mathcal{A}$, as estruturas

$$\begin{array}{rcl} \mathcal{A}(s) & \cong & A \times B \\ \mathcal{A}(r) & \cong & C \end{array}$$

para modelos de s e r em $\mathcal{A}$, sendo A, B, C conjuntos finitos em *Sets*. Queremos agora construir o modelo

$$\mathcal{A}(f) : A \times B \to C \tag{2.52}$$

de f em $\mathcal{A}$.

É evidente que $\mathcal{A}(f)$ vai depender da semântica que tivermos em mente para f. Mas, teremos liberdade absoluta quanto ao que escrevermos sobre $\mathcal{A}(f)$? A primeira limitação é a que resulta de (2.52), ou seja

$$\forall x \in A \times B : f(x) \in C$$

(voltamos a simplificar $\mathcal{A}(f)$ em f). Mas $x \in A \times B$ significa que $x = (a, b)$ para algum $a \in A$ e $b \in B$, *cf.* (1.15). Isto tem a ver com a construção (categorial) do próprio produto $A \times B$, dada pelo diagrama (1.18), como vimos.

Somos assim induzidos a esboçar

$$\begin{array}{rll} f(x) \stackrel{\mathrm{def}}{=} & let & a = \pi_1(x) \\ & & b = \pi_2(x) \\ & in & \ldots a \ldots b \ldots \end{array} \tag{2.53}$$

onde "$\ldots a \ldots b \ldots$" é uma qualquer expressão envolvendo as variáveis a e b, com resultado em C, e π_1 e π_2 são as duas selecções canónicas associadas ao produto cartesiano — relembrar (1.18). Referir-nos-emos a (2.53) como sendo a forma "canónica" — ou 'kit' — para escrevermos funções sobre produtos cartesianos [3].

[3] Este 'kit' estende-se naturalmente ao produto finito de n conjuntos $\Pi_{i=1}^n A_i$, *i.é* $A_1 \times \ldots \times A_n$, envolvendo n selecções π_1 a π_n.

Vejamos agora o que acontece se o modelo de s em $\mathcal{A}$ for

$$\mathcal{A}(s) \cong A + B \tag{2.54}$$

cf. (1.16). De acordo com esta equação, se $x \in A + B$ então $x = (1, a)$ ou $x = (2, b)$, para $a \in A$ e $b \in B$. Relembremos que $A + B$ é a união *disjunta* de A e B, quer dizer, valores $y \in A \cap B$ que se confundiriam em $A \cup B$ (união normal) não se confundem em $A + B$ pois são "injectados" em $A + B$ após prévia etiquetagem: com 1, se vêm de A, ou 2 se vêm de B — relembrar as injecções i_1, i_2 em (1.27,1.28).

Será que volta a existir algum 'kit' para a definição de $\mathcal{A}(f)$, induzido agora pela união-disjunta (2.54)? Esse 'kit' deverá reflectir o acto de teste correspondente a x ser um argumento que foi injectado de A ou foi injectado de B,

$$f(x) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lcl} x = i_1(a) & \Rightarrow & \ldots a \ldots \\ x = i_2(b) & \Rightarrow & \ldots b \ldots \end{array} \right. \tag{2.55}$$

— recordar (1.29) e (1.30), por exemplo. Aqui "$\ldots a \ldots$" é uma qualquer expressão em $a \in A$ com valor em C, e "$\ldots b \ldots$" é uma qualquer expressão em $b \in B$ também com valor em C [4].

Uma versão muito vulgarizada do esquema (2.55) é a seguinte,

$$f(x) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lcl} \text{is-}A(x) & \Rightarrow & \ldots x \ldots \\ \text{is-}B(x) & \Rightarrow & \ldots x \ldots \end{array} \right. \tag{2.56}$$

que — com algum abuso de linguagem — ignora deliberadamente as injecções i_1 e i_2, substituindo a sua manipulação pelos (meta) predicados is-A e is-B, que decidem se um dado x "é de A" ou "é de B". Um caso particular de união-disjunta é a forma $1 + A$, relembrar (2.29). Neste caso, é também vulgar substituir os meta-predicados is-$1(x)$ e is-$A(x)$ por

$$\left\{ \begin{array}{lcl} x = NIL & \Rightarrow & \ldots \\ x \neq NIL & \Rightarrow & \ldots \end{array} \right. \tag{2.57}$$

neste contexto, *cf.* (2.28) [5].

Notemos agora que 'kits' como (2.53) ou (2.55) podem ser compostos estruturalmente, com potencial simplificação do processo de escrita de especificações funcionais. Por exemplo, para

$$f : 1 + X \times List \to C \tag{2.58}$$

[4] Este 'kit' estende-se naturalmente ao somatório de n conjuntos $\sum_{i=1}^{n} A_i$, *i.é* $A_1 + \cdots + A_n$, envolvendo n injecções i_1 a i_n.

[5] Esta notação é habitual em metodologias como VDM ('Vienna Development Method') *cf.* secção de **Notas Bibliográficas** deste capítulo.

— relembrar (2.29) — escreveremos a composição

$$f(x) \stackrel{\rm def}{=} \begin{cases} x = NIL & \Rightarrow \quad \ldots x \ldots \\ x \neq NIL & \Rightarrow \quad \begin{array}{ll} let & a = \pi_1(x) \\ & b = \pi_2(x) \\ in & \ldots a \ldots b \ldots \end{array} \end{cases} \tag{2.59}$$

que é guiada pela estrutura da expressão $1 + X \times List$ (união disjunta primeiro e depois produto, no segundo argumento da união).

Vejamos agora em que medida esta técnica (estrutural) de especificação funcional é aplicável na generalidade dos casos. Por exemplo, vamos agora tentar construir, estruturadamente, a especificação do operador

$$\begin{array}{rcl} elems : List & \to & 2^X \\ elems(x) & \stackrel{\rm def}{=} & \ldots \end{array}$$

que deverá saber calcular o número de elementos de uma lista $l \in List$, para $List$ definido por (2.29), *i.é* [6]

$$List \cong 1 + X \times List$$

Somos naturalmente tentados a usar o esquema (2.59), registando de imediato que o conjunto dos elementos de uma lista $x = NIL$ é vazio:

$$elems(x) \stackrel{\rm def}{=} \begin{cases} x = NIL & \Rightarrow \quad \emptyset \\ x \neq NIL & \Rightarrow \quad \begin{array}{ll} let & a = \pi_1(x) \\ & b = \pi_2(x) \\ in & \ldots a \ldots b \ldots \end{array} \end{cases}$$

Reparemos agora que $b \in List$, o que significa que $elems(b)$ é uma expressão matematicamente correcta, cujo significado é "os elementos da lista que se obtém de x retirando-lhe o seu primeiro elemento". Ocorre-nos assim a pormenorização seguinte,

$$\ldots a \ldots b \ldots \to \ldots a \ldots elems(b) \ldots$$

E só agora chega a oportunidade de sermos verdadeiramente criativos: para obtermos $elems(x)$ verificamos que basta "juntar a a $elems(b)$":

$$\begin{array}{rcl} \ldots a \ldots b \ldots & \to & \ldots a \ldots elems(b) \ldots \\ & \to & \{a\} \cup elems(b) \end{array}$$

obtendo

$$elems(x) \stackrel{\rm def}{=} \begin{cases} x = NIL & \Rightarrow \quad \emptyset \\ x \neq NIL & \Rightarrow \quad \begin{array}{ll} let & a = \pi_1(x) \\ & b = \pi_2(x) \\ in & \{a\} \cup elems(b) \end{array} \end{cases}$$

[6] Relembrar o mesmo operador atrás definido ao nível de X^*, *cf.* secção 1.3.1.

ou ainda,

$$elems(x) \stackrel{\text{def}}{=} \begin{cases} x = NIL & \Rightarrow \emptyset \\ x \neq NIL & \Rightarrow \{\pi_1(x)\} \cup elems(\pi_2(x)) \end{cases} \tag{2.60}$$

eliminando a cláusula 'let' por simples substituição.

Vamos agora repetir a estratégia do exercício anterior para o operador

$$\begin{array}{rcl} length : List & \rightarrow & I\!N_0 \\ length(x) & \stackrel{\text{def}}{=} & \ldots \end{array}$$

que queremos que calcule o número de elementos de uma lista x para o mesmo modelo $List$. É imediato obtermos, a partir da experiência anterior,

$$length(x) \stackrel{\text{def}}{=} \begin{cases} x = NIL & \Rightarrow 0 \\ x \neq NIL & \Rightarrow \begin{array}{ll} let & a = \pi_1(x) \\ & b = \pi_2(x) \\ in & \ldots a \ldots length(b) \ldots \end{array} \end{cases} \tag{2.61}$$

onde também começamos por registar que o comprimento de $x = NIL$ é 0.

Notemos agora que a cabeça de x, $a = \pi_1(x)$, é irrelevante para o nosso resultado, já que a única consequência da sua existência é o incremento, em uma unidade, do comprimento do resto de x. Assim, instanciaremos

$$\ldots a \ldots length(b) \ldots \rightarrow 1 + length(b)$$

neste caso. Feitas as substituições e simplificações, derivamos de (2.61) a definição:

$$length(x) \stackrel{\text{def}}{=} \begin{cases} x = NIL & \Rightarrow 0 \\ x \neq NIL & \Rightarrow 1 + length(\pi_2(x)) \end{cases} \tag{2.62}$$

Em síntese, podemos estabelecer o seguinte 'kit' — esquema mais geral — para toda e qualquer operação f de $List$ para um outro domínio C,

$$f(x) = \begin{cases} x = NIL & \Rightarrow \ldots x \ldots \\ x \neq NIL & \Rightarrow \ldots \pi_1(x) \ldots f(\pi_2(x)) \ldots \end{cases} \tag{2.63}$$

onde os contextos com reticências têm a mesma funcionalidade ($List \rightarrow C$) que f. Com isto queremos dizer que cada instância de f deverá ser uma simplificação deste esquema-base.

Mas há subtilezas a considerar nessas simplificações. Por exemplo, seja $f = head$, o operador que queremos que dê a "cabeça" de uma lista em $List$. Teremos, a partir de (2.63):

$$head(x) = \begin{cases} x = NIL & \Rightarrow \ldots x \ldots \\ x \neq NIL & \Rightarrow \pi_1(x) \end{cases} \tag{2.64}$$

Quer dizer, $head$ não chega a ser uma função recursiva. Mas, como completar o contexto $\ldots x \ldots$ para $x = NIL$? A nossa intuição "diz-nos" que a expressão $head(NIL)$ não tem significado, já que uma lista vazia não tem quaisquer elementos, muito menos uma "cabeça" (primeiro elemento). O mesmo tipo de problema ocorre na definição do operador $tail$, complementar de $head$,

$$tail(x) = \begin{cases} x = NIL & \Rightarrow & \ldots ?? \ldots \\ x \neq NIL & \Rightarrow & \pi_2(x) \end{cases}$$

que nos dá o *resto* da lista x.

2.3.5 Operadores Parciais

Os operadores $head$ e $tail$ acima ilustram uma dificuldade muito relevante no contexto da definição de operadores numa especificação (modelo): podem existir valores concretos de um argumento x (pelo menos) de um operador $f(\ldots, x, \ldots)$ para os quais não faz sentido aplicar esse operador. Isto é, se x_0 é um desses valores, então $f(\ldots, x_0, \ldots)$ é uma expressão em *Sets* sem significado matemático.

Este tipo de situações ocorre com frequência em Matemática e designa-se vulgarmente por *parcialidade*. Assim, diz-se *parcial* todo o operador $f : A \rightarrow B$ que não aceita a *totalidade* dos valores do seu domínio A. Por exemplo, o operador

$$\sqrt{x} : \mathbb{R} \rightarrow \mathbb{R} \tag{2.65}$$

não é *total* em $\mathbb{R} \rightarrow \mathbb{R}$, pois se $x < 0$ então $\sqrt{x}$ não é um número real. Já se tivéssemos definido

$$\sqrt{x} : \mathbb{R} \rightarrow \mathbb{C}$$

em lugar de (2.65), teríamos um operador total.

Já anteriormente contactamos com operadores parciais. Por exemplo, vimos que $the(x)$ (1.49) só está definido para conjuntos singulares x.

Há duas maneiras "clássicas" para resolver o problema da parcialidade de um operador. Segundo a primeira, omitem-se pura e simplesmente as cláusulas correspondentes a argumentos semânticamente "inconvenientes". Pegando em (2.64), por exemplo, reduziremos a definição de $head(x)$ a

$$head(x) \stackrel{\mathrm{def}}{=} \{x \neq NIL \Rightarrow \pi_1(x) \tag{2.66}$$

e a de $tail(x)$ a

$$tail(x) \stackrel{\mathrm{def}}{=} \{x \neq NIL \Rightarrow \pi_2(x) \tag{2.67}$$

Neste caso, a parcialidade é explicitamente assumida, designando-se o teste

$$x \neq NIL$$

a *pré-condição* de $tail$, que é exactamente a mesma pré-condição de $head$. Um operador parcial toma assim a forma genérica,

$$\begin{array}{rcl} f : A & \rightarrow & B \\ f(x) & \stackrel{\mathrm{def}}{=} & \{ \; pre_f(x) \Rightarrow \ldots x \ldots \end{array} \tag{2.68}$$

para f unário (sem perda de generalidade), onde $pre_f : A \rightarrow 2$ é um operador ou expressão booleana.

Esta estratégia para a parcialidade é a mais simples, à primeira vista, mas obriga-nos a um cuidado especial. Sempre que f é envolvido numa expressão

$$\varphi(\ldots, f(x), \ldots) \tag{2.69}$$

é preciso ter o cuidado de garantir que $f(x)$ está em condições de ser calculado, *i.é.*, que x é argumento válido de x. Poderá mesmo ser necessário escrever, em lugar de (2.69),

$$pre_\varphi(x) \Rightarrow \varphi(\ldots, f(x), \ldots)$$

sendo $pre_\varphi(x)$ tal que

$$pre_\varphi(x) \Rightarrow pre_f(x)$$

cf. (2.68). Quer dizer, um operador parcial só pode ser empregue em contextos que garantam a sua pré-condição. Nesta abordagem, um operador f que é total pode ser visto como um operador parcial cuja pré-condição é universalmente verdadeira, *i.é*,

$$\forall x \in A : pre_f(x) = V$$

A segunda abordagem canónica à parcialidade toma uma opção inversa: em lugar de restringir o domínio de uma função $f : A \rightarrow B$, alarga-lhe o contradomínio B introduzindo um valor "indefinido" $\perp$ ao qual correspondem todos os valores $f(x)$ para $x \in A$ tal que $pre_f(x) = F$. Quer dizer, define-se a seguinte "versão total" de f:

$$\begin{array}{rcl} f_\perp : A & \rightarrow & B \cup \{\perp\} \\ f_\perp(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{rcl} pre_f(x) & \Rightarrow & f(x) \\ \neg pre_f(x) & \Rightarrow & \perp \end{array} \right. \end{array}$$

As duas abordagens têm igual capacidade expressiva. Contudo, a segunda leva-nos para o domínio dos modelos *ordenados*, que só será estudado no Capítulo 3. Por isso mesmo ficaremos, de momento, com a primeira, salvaguardados os cuidados que se expuseram quanto à validação de pré-condições.

2.3.6 Esquematologia Funcional Básica

Pelo que atrás se expôs, existem fundamentalmente as seguintes primitivas em *Sets* para construção de modelos de espécies:

- o produto cartesiano ($A \times B$)
- a união disjunta ($A + B$)
- a exponenciação (A^B), de que resultam
  - as partes de um conjunto (2^A)
  - as funções parciais ($A \rightharpoonup B$)
  - as sequências (A^*)

cf. secção 1.3.1, às quais se acrescentou neste capítulo a *recursividade*. Muitas expressões em $Sets$, literalmente diferentes, acabam por ser equivalentes na medida em que são isomorfas. Por exemplo, já vimos que

$$List \cong X^*$$

para a definição recursiva $List \cong 1 + X \times List$ (assumindo a menor solução desta equação). Assim, todos os esquemas de especificação que tivermos para operadores sobre $List$ têm um "parente" isomorfo em X^*, e vice-versa. Por exemplo, não será difícil constatar que o esquema (2.63) se reescreve, para X^*, como se segue,

$$\begin{array}{rcl} f : X^* & \to & C \\ f(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcl} x = <> & \Rightarrow & \ldots x \ldots \\ x \neq <> & \Rightarrow & \ldots head(x) \ldots f(tail(x)) \ldots \end{array} \right. \end{array} \tag{2.70}$$

cf. também (2.66) e (2.67).

Ora, em (2.70) temos um esquema funcional recursivo sobre um domínio X^* que não consideramos recursivo à partida. Será que (2.70) é passível de "desrecursivação" ou simplificação? Vejamos como instâncias suas o são, de facto.

Manipulação de Sequências

Uma vasta classe de operadores sobre sequências

$$f : X^* \to C$$

é tal que $C = Y^*$ e o esquema (2.70) é instanciado como se segue:

$$f(x) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lcl} x = <> & \Rightarrow & <> \\ x \neq <> & \Rightarrow & cons(\varphi(head(x)), f(tail(x))) \end{array} \right. \tag{2.71}$$

para $\varphi : X \to Y$. Reparemos primeiro que (2.71) é uma definição "bem-instanciada", ou seja "bem-tipificada": para $x = <>$, temos $f(x) = <>$ e de facto $<> \in Y^*$, logo $f(x) \in Y^*$; para $x \neq <>$, teremos

$$\begin{array}{rcl} x \in X^* & \Rightarrow & tail(x) \in X^* \\ & \Rightarrow & f(tail(x)) \in Y^* \end{array} \tag{2.72}$$

pois $f : X^* \to Y^*$; também temos que

$$\begin{array}{rcl} x \in X^* & \Rightarrow & head(x) \in X \\ & \Rightarrow & \varphi(head(x)) \in Y \end{array} \tag{2.73}$$

pois $\varphi : X \to Y$. Como $cons : Y \times Y^* \to Y^*$, teremos que, obtidos os resultados (2.72) e (2.73),

$$cons(\varphi(head(x)), f(tail(x))) \in Y^*$$

como queríamos.

Em lugar de (2.71) é vulgar escrever-se a definição "não recursiva"

$$f(x) \stackrel{\text{def}}{=} <\varphi(a) \mid a \leftarrow x>$$

ou ainda a composição

$$\varphi \circ x$$

que já foi apresentada informalmente na secção 1.3.1. A este tipo de operadores sobre sequências dá-se vulgarmente a designação de *filtro*, designação filiada na terminologia UNIX sobre processamento de "streams" (*cf.* AWK, LEX, YACC *etc.*).

Um operador de filtragem mais geral obtém-se de (2.71) tomando a decisão adicional de só "passar para a saída" os valores $\varphi(a)$ tais que $p(a) = V$ para um dado predicado de filtragem $p : X \to 2$:

$$f(x) \stackrel{\text{def}}{=} \left\{ \begin{array}{lcll} x = <> & \Rightarrow & <> & \\ x \neq <> & \Rightarrow & let & h = head(x) \\ & & & t = tail(x) \\ & & in & \left\{ \begin{array}{rcl} p(h) & \Rightarrow & cons(\varphi(h), f(t)) \\ \neg p(h) & \Rightarrow & f(t) \end{array} \right. \end{array} \right. \tag{2.74}$$

Uma abreviatura conveniente de (2.74) é

$$f(x) \stackrel{\text{def}}{=} <\varphi(a) \mid a \leftarrow x \wedge p(a)>$$

cf. a expressão (1.82) já apresentada anteriormente.

Reparemos finalmente que, se um operador envolve mais do que um argumento que seja uma sequência, os esquemas que atrás foram descritos serão aplicáveis em relação a um desses argumentos, como se pode ver pelo Exercício 2.9.

Exercício 2.9 Adapte o esquema (2.70) à síntese do operador binário sobre listas de concatenação, completando:

$$x \frown y \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lll} x = <> & \Rightarrow & y \\ x \neq <> & \Rightarrow & \ldots head(x) \ldots tail(x) \frown y \ldots \end{array} \right.$$

e verifique se $\frown$ satisfaz as propriedades (A.67) a (A.69) do apêndice A.

□

Exercício 2.10 É dada a seguinte definição de uma ordem sobre sequências:

$$\begin{array}{lcl} \sqsubseteq & : & A^\star \times A^\star \longrightarrow 2 \\ l \sqsubseteq l' & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{rll} l = < > & \Rightarrow & V \\ l \neq < > \wedge l' = < > & \Rightarrow & F \\ l \neq < > \wedge l' \neq < > & \Rightarrow & (\left\{ \begin{array}{lll} l(1) = l'(1) & \Rightarrow & tail(l) \\ l(1) \neq l'(1) & \Rightarrow & l \end{array} \right.) \sqsubseteq tail(l') \end{array} \right. \end{array}$$

1. Fazendo $A = I\!N$, dê exemplos de três sequências l_1, l_2, l_3 tal que:

$$l_1 \sqsubseteq l_2$$

$$l_1 \not\sqsubseteq l_3$$

2. Indique, mediante a apresentação de contra-exemplos, quais dos seguintes factos *não* são verdadeiros, onde $l, l' \in A^\star$:

$$\begin{array}{rcl} \sqsubseteq & \text{é} & \text{uma relação de equivalência} \\ \sqsubseteq & \text{é} & \text{uma ordem total} \\ l \sqsubseteq l' & \Rightarrow & length(l) \leq length(l') \\ \sqsubseteq & \text{é} & \text{uma ordem bem-fundada} \end{array}$$

□

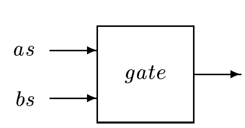
Exercício 2.11 Sendo dadas as cláusulas,

$$\begin{array}{rcl} Stream & \cong & A^\star \\ BoolStream & \cong & 2^\star \end{array}$$

complete a seguinte definição explícita

$$\begin{array}{rcl} gate & : & Stream \times BoolStream \longrightarrow Stream \\ gate(as, bs) & \stackrel{\mathrm{def}}{=} & \left\{ \; length(as) \leq length(bs) \quad \Rightarrow \quad \ldots \right. \end{array}$$

de uma função



que consome dois 'streams', deixando passar para a saída apenas os elementos do primeiro 'stream' que são "permitidos" pelo correspondente valor booleano do segundo 'stream'. Exemplos:

$$\begin{array}{rcl} gate(<a,b,c>, <V,F,V,V>) & = & <a,c> \\ gate(<a,b,a>, <F,F,F>) & = & < > \end{array}$$

Indique, através da apresentação de contra-exemplos, quais dos seguintes factos *não* são verdadeiros, para todos os $bs, bs' \in BoolStream$ e $as \in Stream$ ($\sqsubseteq$ é a relação do Exercício 2.10 e $\frown$ é a função definida no Exercício 2.9):

$$\begin{array}{rcl} gate(bs,bs) & = & bs \\ gate(as,bs) & \sqsubseteq & as \\ F \in elems(bs) & \Rightarrow & gate(as,bs) = < > \\ gate(gate(as,bs),bs') & = & gate(as, bs \frown bs') \end{array}$$

□

Manipulação de Conjuntos

Vamos agora supor que o conjunto de partida da função f da equação (2.70) é 2^X em lugar de X^*, *i.é*

$$f : 2^X \to C \tag{2.75}$$

e procuremos encontrar um esquema funcional genérico — um 'kit' — para f.

Reparemos que, sendo X finito, qualquer $x \subseteq X$ (*i.é* $x \in 2^X$) também o é. Logo, é possível pensar num esquema para (2.75) análogo a (2.70), em que à sequência vazia $<> \in X^*$ faça corresponder agora o conjunto vazio $\emptyset \in 2^X$:

$$\begin{array}{rcl} f : 2^X & \to & C \\ f(x) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} x = \emptyset & \Rightarrow & \ldots \\ x \neq \emptyset & \Rightarrow & \ldots \end{array} \right. \end{array}$$

Contudo, operadores tipo $head$ e $tail$ não fazem qualquer sentido em 2^X, já que não existe qualquer ordem de acesso num conjunto $x \subseteq X$. Mas não é difícil imaginar qual o tipo de decomposição a definir sobre 2^X correspondente à decomposição $head/tail$ sobre sequências em X^* — se $head$ faz a escolha perfeitamente determinada do primeiro elemento de uma sequência em X^*, em 2^X essa escolha é livre, *i.é* basta escolher indeterminadamente qualquer elemento:

$$f(x) \stackrel{\text{def}}{=} \left\{ \begin{array}{lcl} x = \emptyset & \Rightarrow & \ldots \\ x \neq \emptyset & \Rightarrow & \begin{array}{ll} let & e \in x \\ in & \ldots \end{array} \end{array} \right.$$

A construção análoga a $tail(s)$, cálculo do "resto" de uma sequência s, será então o "resto"

$$x - \{e\}$$

de um conjunto x ao qual se retirou um elemento e. Em suma, temos o esquema:

$$\begin{array}{rcl} f : 2^X & \rightarrow & C \\ f(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcll} x = \emptyset & \Rightarrow & \ldots & \\ x \neq \emptyset & \Rightarrow & let & e \in x \\ & & in & \ldots e \ldots f(x - \{e\}) \ldots \end{array} \right. \end{array} \tag{2.76}$$

Vejamos, a título de exemplo, duas instâncias completas de (2.76). A primeira é uma função que conta o número de elementos de um conjunto,

$$\begin{array}{rcl} card : 2^X & \rightarrow & I\!N_0 \\ card(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcll} x = \emptyset & \Rightarrow & 0 & \\ x \neq \emptyset & \Rightarrow & let & e \in x \\ & & in & 1 + card(x - \{e\}) \end{array} \right. \end{array}$$

i.é $card(x) = |x|$ (*cf.* $length$ (2.62), o operador análogo para sequências). A segunda será uma função que se pretende que "sequencialize" um conjunto, *i.é* que o transforme numa sequência sem repetições de acordo com uma ordem arbitrária:

$$\begin{array}{rcl} list : 2^X & \rightarrow & X^* \\ list(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcll} x = \emptyset & \Rightarrow & <> & \\ x \neq \emptyset & \Rightarrow & let & e \in x \\ & & in & cons(e, list(x - \{e\})) \end{array} \right. \end{array}$$

Esta função é, de certo modo, a "inversa" de

$$\begin{array}{rcl} elems : X^* & \rightarrow & 2^X \\ elems(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcl} x = <> & \Rightarrow & \emptyset \\ x \neq <> & \Rightarrow & \{head(x)\} \cup elems(tail(x)) \end{array} \right. \end{array}$$

cf. (2.60), na medida em que

$$\forall x \in 2^X : elems(list(x)) = x \tag{2.77}$$

Por analogia com X^*, é possível definir a seguinte instância do esquema funcional (2.76) semelhante a (2.74):

$$\begin{array}{rcl} f_\varphi : 2^X & \rightarrow & 2^Y \\ f_\varphi(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcll} x = \emptyset & \Rightarrow & \emptyset & \\ x \neq \emptyset & \Rightarrow & let & e \in x \\ & & & a = \left\{ \begin{array}{lcl} p(e) & \Rightarrow & \{\varphi(e)\} \\ \neg p(e) & \Rightarrow & \emptyset \end{array} \right. \\ & & in & a \cup f_\varphi(x - \{e\}) \end{array} \right. \end{array} \tag{2.78}$$

para qualquer $\varphi : X \rightarrow Y$.

Não é difícil identificar em (2.78) a vulgar fórmula de “abstracção de Zermelo-Frænkl”, *i.é*

$$f_\varphi(x) = \{\varphi(e) \mid e \in x \wedge p(e)\} \tag{2.79}$$

Se não houver predicado de filtragem (p), a equação (2.79) reduz-se à forma ainda mais simples e habitual,

$$\{\varphi(e) \mid e \in x\}$$

Outra configuração muito habitual do esquema (2.76) é a que se segue,

$$\begin{array}{rcl} f_\varphi : 2^X & \rightarrow & Y \\ f_\varphi(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcll} x = \emptyset & \Rightarrow & \epsilon & \\ x \neq \emptyset & \Rightarrow & let & e \in x \\ & & in & \varphi(e)\theta f_\varphi(x - \{e\}) \end{array} \right. \end{array} \tag{2.80}$$

onde $\varphi : X \rightarrow Y$ e a estrutura $(Y; \theta, \epsilon)$ constitui um monóide abeliano (*i.é* θ é associativa e comutativa e ϵ é seu elemento neutro).

A (2.80) dá-se vulgarmente o nome de *esquema de redução (de x) por θ*, fazendo sentido escrever

$$f_\varphi(x) \stackrel{\mathrm{def}}{=} \Theta_{e \in x}\varphi(e) \tag{2.81}$$

(onde Θ generaliza θ a mais do que dois argumentos) como simplificação de (2.80). Por exemplo, instanciando

$$\left\{ \begin{array}{ccl} X & \rightarrow & \mathbb{Z}_0 \\ Y & \rightarrow & \mathbb{N}_0 \\ \theta & \rightarrow & + \\ \epsilon & \rightarrow & 0 \\ \varphi & \rightarrow & \lambda n.n^2 \end{array} \right.$$

obtemos, de (2.81), a função

$$f(x) \stackrel{\mathrm{def}}{=} \sum_{e \in x} e^2$$

que calcula o somatório dos quadrados de todos os números inteiros contidos num conjunto x.

Interessa agora reter duas instâncias relevantes de (2.81), para duas substituições típicas:

$$\left\{ \begin{array}{ccc} Y & \rightarrow & 2 \\ \theta & \rightarrow & \wedge \\ \epsilon & \rightarrow & V \end{array} \right. \quad , \quad \left\{ \begin{array}{ccc} Y & \rightarrow & 2 \\ \theta & \rightarrow & \vee \\ \epsilon & \rightarrow & F \end{array} \right.$$

obtendo-se

$$\textstyle\bigwedge_{e \in x} \varphi(e) \quad , \quad \bigvee_{e \in x} \varphi(e) \tag{2.82}$$

É imediato ver aqui as duas formas de *quantificação* (respectivamente *universal* e *existencial*) de um predicado $\varphi : X \to 2$ sobre um conjunto finito x, *i.é*

$$\forall e \in x : \varphi(e) \quad , \quad \exists e \in x : \varphi(e) \tag{2.83}$$

Convenhamos que muito se simplificará um texto de uma especificação em que se escreva qualquer uma das expressões de (2.83) em lugar da correspondente função recursiva ao nível de (2.80)!

Para terminar este sumário de esquemas funcionais sobre conjuntos, convem referir ainda as duas instâncias de (2.81) que se obtêm para

$$\left\{ \begin{array}{ccc} Y & \to & 2^A \\ \theta & \to & \cup \\ \epsilon & \to & \emptyset \end{array} \right. \quad , \quad \left\{ \begin{array}{ccc} Y & \to & 2^A \\ \theta & \to & \cap \\ \epsilon & \to & A \end{array} \right. \tag{2.84}$$

podendo agora considerar-se $\varphi : X \to 2^A$ uma família de conjuntos, *i.é*

$$\varphi = (\varphi_e)_{e \in x}$$

Somos conduzidos então aos esquemas

$$\bigcup_{e \in x} \varphi_e \quad , \quad \bigcap_{e \in x} \varphi_e \tag{2.85}$$

que designaremos por *união (intersecção) generalizadas*, já que generalizam a n-argumentos a união ($A \cup B$) a intersecção ($A \cap B$) de conjuntos.

Vejamos finalmente uma ilustração do emprego múltiplo, estrutural destes esquemas. Seja dado o seguinte modelo em $Sets$ para *árvores genealógicas*:

$$\begin{array}{rcll} AG & \cong & Ind + Fam & \textit{/*AG = árvore genealógica */} \\ & & & \textit{/*Ind = indivíduo */} \\ & & & \textit{/*Fam = família */} \\ Fam & \cong & Ind \times & \textit{/*cabeça de casal */} \\ & & Ind \times & \textit{/*o outro cônjuge */} \\ & & 2^{AG} & \textit{/*a descendência do casal */} \end{array} \tag{2.86}$$

Qual será o 'layout' mais genérico de uma função

$$f : AG \to C$$

que tenha AG como conjunto de partida? É imediato esboçarmos:

$$f(ag) \stackrel{\text{def}}{=} \left\{ \begin{array}{lcll} \text{is-}Ind(ag) & \Rightarrow & \ldots & \\ \text{is-}Fam(ag) & \Rightarrow & let & i = \pi_1(ag) \\ & & & j = \pi_2(ag) \\ & & & d = \pi_3(ag) \\ & & in & \ldots i \ldots j \ldots d \ldots f \ldots \end{array} \right. \tag{2.87}$$

guiados pelas estruturas $+$ e $\times$ em $Sets$. Mas reparemos que AG ocorre em expoente (2^{AG}) em (2.86). Isto faz-nos pensar numa função auxiliar,

$$aux : 2^{AG} \to \ldots$$

que siga o esquema (2.76), tomando aqui o f da (2.87) o lugar de φ. Passaremos a ter um sistema de duas funções mutuamente recursivas,

$$\begin{array}{lcl} f(ag) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} \text{is-}Ind(ag) & \Rightarrow & \ldots \\ \text{is-}Fam(ag) & \Rightarrow & \ldots \pi_1(ag) \ldots \pi_2(ag) \ldots aux(\pi_3(ag)) \ldots \end{array} \right. \\ aux(d) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} d = \emptyset & \Rightarrow & \ldots \\ d \neq \emptyset & \Rightarrow & let \quad e \in d \\ & & in \quad \ldots f(e) \ldots aux(d - \{e\}) \ldots \end{array} \right. \end{array} \tag{2.88}$$

passível das simplificações atrás enunciadas, conforme os casos.

Suponhamos que, a título de exemplo, o operador $f(ag)$ a modelar é $todos(ag)$, que se pretende que calcule o conjunto de “todos” os indivíduos presentes na árvore genealógica ag. Não é difícil completar (2.88) para este caso:

$$\begin{array}{lcl} todos : AG & \to & 2^{Ind} \\ todos(ag) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} \text{is-}Ind(ag) & \Rightarrow & \{ag\} \\ \text{is-}Fam(ag) & \Rightarrow & \left\{ \begin{array}{l} \pi_1(ag), \\ \pi_2(ag) \end{array} \right\} \cup aux(\pi_3(ag)) \end{array} \right. \end{array} \tag{2.89}$$

para

$$\begin{array}{lcl} aux & : & 2^{AG} \to 2^{Ind} \\ aux(d) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} d = \emptyset & \Rightarrow & \emptyset \\ d \neq \emptyset & \Rightarrow & let \quad e \in d \\ & & in \quad todos(e) \cup aux(d - \{e\}) \end{array} \right. \end{array} \tag{2.90}$$

É nítido que (2.90) corresponde à primeira expressão de (2.85) gerada pela primeira instanciação de (2.84). Portanto, é possível “eliminar” aux obtendo a seguinte simplificação de (2.89):

$$todos(ag) \stackrel{\text{def}}{=} \left\{ \begin{array}{lcl} \text{is-}Ind(ag) & \Rightarrow & \{ag\} \\ \text{is-}Fam(ag) & \Rightarrow & \{\pi_1(ag), \pi_2(ag)\} \cup \bigcup_{e \in \pi_3(ag)} todos(e) \end{array} \right.$$

Exercício 2.12 Construa estruturalmente a definição dos seguintes predicados sobre AG (2.86):

$$\begin{array}{rcl} \textit{semRepetição} & : & AG \to 2 \\ \textit{biSexuada} & : & AG \to 2 \end{array}$$

que deverão garantir, respectivamente:

- que nenhum indivíduo vem a ser descendente de si próprio;
- que os dois cônjuges de cada casal da árvore têm sexos diferentes (assuma aqui definida uma função $sexo : Ind \to \{M, F\}$).

□

Manipulação de Funções Parciais Finitas

Pretendemos agora estudar esquemas funcionais para operadores sobre funções finitas (parciais), *i.é* com funcionalidade da forma

$$f : (A \rightharpoonup B) \longrightarrow C$$

Relembremos a definição

$$A \rightharpoonup B = \bigcup_{K \subseteq A} B^K$$

onde

$$B^K = \{\phi \mid \ \phi : K \to B\}$$

cf. (1.16), isto é, $A \rightharpoonup B$ é o conjunto de todas as funções parciais finitas cujo domínio está contido em A e cujo contra-domínio está contido em B. Como já vimos, cada uma dessas funções pode ser considerada um conjunto de pares ordenados $(a, b) \in A \times B$, *i.é*

$$\phi \in A \rightharpoonup B \quad \Rightarrow \quad \phi \in 2^{A \times B} \tag{2.91}$$

É imediato então aplicarmos a funções finitas todos os esquemas desenvolvidos para conjuntos. Pegando no esquema básico (2.76) obteremos

$$\begin{array}{lcl} f & : & (A \rightharpoonup B) \to C \\ f(\phi) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lll} \phi = () & \Rightarrow & \ldots \\ \phi \neq () & \Rightarrow & let \quad (a, b) \in \phi \\ & & in \quad \ldots a \ldots b \ldots f(\phi - \{(a, b)\}) \ldots \end{array} \right. \end{array} \tag{2.92}$$

Sendo ϕ uma função, a escolha $(a, b) \in \phi$ fica igualmente determinada pela escolha $a \in dom(\phi)$, ficando b acessível via $b = \phi(a)$. Somos conduzidos a uma versão "mais elegante" de (2.92)

$$f(\phi) \stackrel{\text{def}}{=} \left\{ \begin{array}{lll} \phi = () & \Rightarrow & \ldots \\ \phi \neq () & \Rightarrow & let \quad a \in dom(\phi) \\ & & in \quad \ldots a \ldots \phi(a) \ldots f(\phi \setminus \{a\}) \ldots \end{array} \right. \tag{2.93}$$

onde ocorre o operador de *restrição* sobre funções ϕ em $A \rightharpoonup B$ — relembrar (1.67).

A notação que se emprega para as várias instâncias de (2.93) põe em evidência o carácter *aplicativo* de uma função finita. Por exemplo, escrever-se-á, para $\phi \in A \rightharpoonup B$,

$$\left(\begin{array}{c} a \\ \phi(a) \end{array} \right)_{a \in X}$$

em lugar de

$$\{(a, \phi(a)) \in A \times B \mid \; a \in X\}$$

Vejamos finalmente uma ilustração do emprego estrutural de esquemas associados a funções finitas. Seja dado o seguinte modelo em $Sets$ de um sistema hierárquico de ficheiros (tipo UNIX ou MS/DOS):

$$\begin{array}{rclll} FS & \cong & Id \rightharpoonup (File + Dir) & \text{/*}\textit{FS = `file system'}\text{ */} & \\ & & & \text{/*}\textit{Id = identificador}\text{ */} & \\ & & & \text{/*}\textit{File = ficheiro}\text{ */} & (2.94) \\ Dir & \cong & FS & \text{/*}\textit{Dir = directoria,}\text{ i.é */} & \\ & & & \text{/*}\quad\textit{sub-file system}\text{ */} & \end{array}$$

Pretendemos saber se um dado identificador $i \in Id$ ocorre algures num sistema de ficheiros $fs \in FS$ como nome de um ficheiro e/ou directoria. Uma ideia é calcular o valor de

$$i \in todosId(fs)$$

quer dizer, o problema é transferido para a definição de um operador

$$todosId : FS \to 2^{Id}$$

que calcule o conjunto de todos esses identificadores.

Começando pelo esquema (2.93), teremos

$$\begin{array}{rcl} todosId & : & FS \to 2^{Id} \\ todosId(fs) & \stackrel{\rm def}{=} & \left\{ \begin{array}{lllll} fs = () & \Rightarrow & \emptyset & & \\ fs \neq () & \Rightarrow & let & i \in dom(fs) & \\ & & & x = fs(i) & \\ & & in & \quad \ldots i \ldots x \ldots & \cup \\ & & & \quad \ldots todosId(fs \setminus \{i\}) \ldots & \end{array} \right. \end{array} \tag{2.95}$$

Aplicando a (2.95) o equivalente a (2.80,2.81), teremos

$$todosId(fs) \stackrel{\rm def}{=} \bigcup_{i \in dom(fs)} \begin{array}{ll} let & x = fs(i) \\ in & \ldots i \ldots x \ldots \end{array}$$

Analisando agora a variável x, que pertence à união $File + Dir$, teremos

$$todosId(fs) \stackrel{\text{def}}{=} \bigcup_{i \in dom(fs)} \begin{array}{ll} let & x = fs(i) \\ in & \left\{ \begin{array}{lll} \text{is-}File(x) & \Rightarrow & \{i\} \\ \text{is-}Dir(x) & \Rightarrow & \ldots i \ldots x \ldots \end{array} \right. \end{array}$$

Como $Dir \cong FS$, teremos finalmente

$$todosId(fs) \stackrel{\text{def}}{=} \bigcup_{i \in dom(fs)} \begin{array}{ll} let & x = fs(i) \\ in & \left\{ \begin{array}{lll} \text{is-}File(x) & \Rightarrow & \{i\} \\ \text{is-}Dir(x) & \Rightarrow & \{i\} \cup todosId(x) \end{array} \right. \end{array} \tag{2.96}$$

Exercício 2.13 Acrescente a (2.94) a seguinte cláusula:

$$Path \cong Id^* \quad \begin{array}{l} \text{/*}\textit{Path = trajectória que selecciona}\text{ */} \\ \text{/*}\quad \textit{uma directoria ou ficheiro}\text{ */} \end{array} \tag{2.97}$$

Aplique a estratégia que acima se ilustrou com a função $todosId$ à definição de três novos operadores sobre FS:

$$\begin{array}{rcl} files & : & FS \rightarrow 2^{Path} \\ dirs & : & FS \rightarrow 2^{Path} \\ mkdir & : & FS \times Id \times Path \rightarrow FS \end{array}$$

que deverão ser tais que:

- $files(fs)$ é o conjunto de todas as trajectórias de ficheiros em fs;
- $dirs(fs)$ é o conjunto de todas as trajectórias de directorias em fs;
- $mkdir(fs, i, p)$ é o resultado de se acrescentar a fs uma nova directoria com nome i na trajectória p.

□

Exercício 2.14 Mostre que, para $fs \in FS$, a função f definida por

$$f(fs) \stackrel{\text{def}}{=} dom(fs) \cup \bigcup_{x \in rng(fs) \wedge \text{is-}Dir(x)} f(x)$$

é equivalente à função $todosId$.
Sugestão: baseie-se no seguinte resultado:

$$\bigcup_{a \in A} (\left\{ \begin{array}{lll} p(a) & \Rightarrow & f(a) \\ \neg(p(a)) & \Rightarrow & \emptyset \end{array} \right.) = \bigcup_{a \in A \wedge p(a)} f(a)$$

□

Exercício 2.15 Complete o seguinte esboço de uma função sobre FS que especifica a semântica do comando `rmdir p`, onde `p` é uma 'path':

$rmdir : FS \times Path \longrightarrow FS$

$$rmdir(fs, p) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lll} length(p) = 0 & \Rightarrow & fs \\ length(p) = 1 & \Rightarrow & \begin{array}{ll} let & i = head(p) \\ in & \left\{ \begin{array}{lll} i \notin dom(fs) & \Rightarrow & fs \\ i \in dom(fs) & \Rightarrow & \left\{ \begin{array}{lll} \text{is-}File(fs(i)) & \Rightarrow & fs \\ \text{is-}Dir(fs(i)) & \Rightarrow & \ldots \end{array} \right. \end{array} \right. \end{array} \\ length(p) > 1 & \Rightarrow & \ldots \end{array} \right.$$

□

2.3.7 Recursividade Polinomial

Tem interesse a generalização da esquematologia funcional que se tem vindo a estudar a definições arbitrariamente recursivas de tipos de dados da forma

$$X \cong \mathcal{F}(X) \tag{2.98}$$

que generalize, por exemplo, definições como a de $List$ (2.29). Exemplos de outras estruturas conhecidas, com padrão de recursividade semelhante a $List$ são, por exemplo,

$$X \cong 1 + A \times X^2 \quad \text{/*}\textit{árvores binárias sobre } A \text{ */} \tag{2.99}$$
$$X \cong 1 + A \times X^\star \quad \text{/*}\textit{árvores generalizadas sobre } A \text{ */} \tag{2.100}$$
$$X \cong V + A \times X^\star \quad \text{/*}\textit{“frases” ou termos sobre } V \textit{ e } A \text{ */} \tag{2.101}$$

etc.

Tal generalização obrigar-nos-á, antes de mais, a continuar a reflexão iniciada na secção 2.3.2 sobre a notação ($Sets$) que vimos utilizando neste texto, investigando agora propriedades da exponenciação. Já se viu que $\times$ e $+$ têm em $Sets$ as propriedades de um *semi-anel comutativo*, a menos de isomorfismo ($\cong$). Reparemos agora que, quanto à exponenciação, temos os factos seguintes:

$$A^0 \cong 1 \tag{2.102}$$
$$A^1 \cong A \tag{2.103}$$
$$(B \times C)^A \cong B^A \times C^A \tag{2.104}$$
$$C^{A \times B} \cong (C^B)^A \tag{2.105}$$
$$A^{B+C} \cong A^B \times A^C \tag{2.106}$$
$$1^A \cong 1 \tag{2.107}$$

Tem-se ainda que

$$A \rightharpoonup B \quad \cong \quad (B+1)^A \tag{2.108}$$

e que, por

$$A \neq B \quad \Rightarrow \quad X^A \cap X^B = \emptyset \tag{2.109}$$

e (2.26), se tem

$$A^\star \quad \cong \quad \sum_{n \geq 0} A^n \tag{2.110}$$

$$A \rightharpoonup B \quad \cong \quad \sum_{K \subseteq A} B^K \tag{2.111}$$

Se interpretarmos, em (2.108), $1 \cong \{\perp\}$, onde $\perp$ representa o valor especial de "indefinição" [7], então a equação (2.108) pode ser encarada como a transformação canónica que converte funções parciais em funções totais através da bijecção

$$i : (A \rightharpoonup B) \to (B+1)^A$$

tal que

$$i(\phi)(a) \stackrel{\text{def}}{=} \begin{cases} \phi(a) & \Leftarrow a \in dom(\phi) \\ \perp & \Leftarrow a \notin dom(\phi) \end{cases} \tag{2.112}$$

Para terminar este *cálculo de isomorfismo* — que adiante se verá que é muito útil para raciocinar sobre especificações — temos ainda,

$$A^B \cong A^X \times A^{B-X} \quad \Leftarrow \quad X \subseteq B \tag{2.113}$$

$$A^n \quad \cong \quad A \times A^{n-1} \tag{2.114}$$

$$n \neq m \quad \Rightarrow \quad X^n \cap X^m = \emptyset \tag{2.115}$$

onde as leis (2.114) e (2.115) podem ser encaradas como instâncias de (2.113) e (2.109), respectivamente [8].

Definição 2.6 *Designaremos por* polinomialmente recursiva *toda a definição da forma (2.98) tal que*

$$\mathcal{F}(X) \quad \cong \quad \sum_{i=0}^{n} C_i \times X^i \tag{2.116}$$

para $C_0 \neq 0$.

[7] Ver mais à frente a secção 3.4.2.

[8] Notar ainda que (2.114) parafraseia (2.35).

É fácil de ver que (2.29) e (2.99) são definições recursivas polinomiais. Recorrendo a (2.110) vê-se que também (2.100) e (2.101) são polinomiais. Vemos assim que estruturas aparentemente não polinomiais — por não serem enunciadas directamente segundo o padrão (2.116) — o são indirectamente através do cálculo de isomorfismo. Quando existe, designa-se por *forma canónica* essa versão polinomial de uma estrutura em $Sets$.

Um resultado sugestivo para converter modelos de dados na sua forma canónica é a fórmula do próprio *binómio de Newton*,

$$(A+B)^n \cong \sum_{p=0}^{n} {}^{n}C_p \times A^{n-p} \times B^p \tag{2.117}$$

Por exemplo, considere-se o seguinte modelo de *esquemas genealógicos* (*cf.* 'pedigrees'):

$$\begin{array}{rrll} GenDia \cong & A: & Ind \times & \textit{/*dados sobre indivíduo */} \\ & F: & (GenDia+1) \times & \textit{/*genealogia do seu pai */} \\ & & & \textit{/*(se for conhecida) */} \\ & M: & (GenDia+1) \times & \textit{/*genealogia de sua mãe */} \\ & & & \textit{/*(se for conhecida) */} \end{array} \tag{2.118}$$

cujo padrão abstracto é

$$\mathcal{F}(X) \quad = \quad I \times (X+1)^2$$

Ter-se-á então, aplicando o binómio de Newton,

$$\begin{array}{rcl} \mathcal{F}(X) & = & I \times (X+1)^2 \\ & \cong & I \times (X^2 + 2 \times X + 1) \\ & \cong & I \times X^2 + 2 \times I \times X + I \end{array}$$

que, literalmente, significa: *"sobre um indivíduo I ou ambos o pai e mãe são conhecidos (X^2), ou um de ou o pai ou a mãe é conhecido ($2 \times X$), ou tanto o pai como a mãe são desconhecidos (1)"*.

Exercício 2.16 Parta do facto (2.108) para mostrar que, em $Sets$,

$$\begin{array}{rcl} 1 \rightharpoonup B & \cong & B+1 \\ A \rightharpoonup 0 & \cong & 1 \\ 0 \rightharpoonup B & \cong & 1 \end{array}$$

□

Exercício 2.17

1. Demonstre ou refute os factos seguintes, em $Sets$,

$$\begin{aligned}(A \rightharpoonup (B \rightharpoonup C)) &\cong (A \times B) \rightharpoonup C && (2.119)\\ 2^{(A^B)} &\cong (2^A)^B && (2.120)\\ 0^A &\cong 0 && (2.121)\\ 0^* &\cong 1 && (2.122)\\ 1^* &\cong I\!N && (2.123)\end{aligned}$$

   para A, B e C quaisquer.

2. Com base na alínea anterior, mostre que toda a assinatura homogénea (*cf.* Definição 1.3) se pode representar por um *alfabeto graduado*, $\Omega \to I\!N_0$.

□

Exercício 2.18 Discuta formalmente a validade da afirmação seguinte, em $Sets$:

- 0 não é ponto-fixo da definição

$$Sexp \cong Atom + Sexp^*$$

□

Exercício 2.19 Uma *árvore de decisão* é uma estrutura com base na qual se pode representar e navegar sobre conhecimento organizado hierarquicamente em termos de perguntas e respostas. Em cada nó da árvore é feita uma pergunta ($What$) e está disponível um *menú* de respostas possíveis (em $Answer$). A principal acção sobre a árvore é a de *resposta*, *i.é* a escolha de uma opção do menú, que selecciona a sub-árvore respectiva. O percurso termina com a decisão do sistema sempre que é atingido uma folha da árvore.

Considere os seguinte modelo recursivo para *árvores de decisão*:

$$DecTree \cong What \times (Answer \rightharpoonup DecTree) \tag{2.124}$$

que instancia (2.98) com o seguinte padrão recursivo abstracto

$$\mathcal{F}(X) \quad\cong\quad A \times (B \rightharpoonup X) \tag{2.125}$$

1. Complete a definição seguinte de alguns operadores sobre $DecTree$:

$$\begin{aligned} menu &: DecTree \longrightarrow 2^{Answer}\\ menu(dt) &\stackrel{\rm def}{=} dom(\pi_2(dt))\end{aligned}$$

$$\begin{aligned} decide &: DecTree \times Answer \longrightarrow DecTree\\ decide(dt, a) &\stackrel{\rm def}{=} \begin{cases} a \in menu(dt) & \Rightarrow \quad \ldots\\ a \notin menu(dt) & \Rightarrow \quad dt\end{cases}\end{aligned}$$

$$\begin{aligned} ask &: DecTree \longrightarrow What\\ ask(dt) &\stackrel{\rm def}{=} \ldots \text{ /*qual a pergunta corrente? */}\end{aligned}$$

$$\begin{aligned} end? &: DecTree \longrightarrow 2\\ end?(dt) &\stackrel{\rm def}{=} \ldots \text{ /*terminou a árvore? */}\end{aligned}$$

2. Calcule a forma polinomial canónica de $DecTree$ (2.124).

□

Esquema Algorítmico Polinomial

Se expandirmos (2.116), obtendo

$$C_0 + C_1 \times X + C_2 \times X^2 + \cdots + C_i \times X^i + \cdots + C_n \times X^n$$

e se nos lembrarmos (*cf.* Secção 2.3.4) que a união-disjunta conduz a esquemas algorítmicos por casos, então é intuitivo conjecturarmos que qualquer função

$$f \quad : \quad X \to C$$

que faça o 'browsing' de X deve ser uma alternativa de tantos casos quantos os termos $C_i \times X^i$ cujo coeficiente $C_i \neq 0$, pois $0 \times X^i \cong 0$. Mais ainda, deverá haver tantas invocações recursivas de f por caso quanto o expoente i do termo respectivo. Logo, o termo C_0 corresponde ao *caso de paragem* de f, com nenhuma invocação recursiva. Daí impormos sempre que $C_0 \neq 0$, sem o que f não terminaria nunca.

Em suma, teremos [9]:

$$\begin{array}{lll} f & : & X \longrightarrow C \\ f(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcll} x = \langle 1, c_0 \rangle & \Rightarrow & \ldots c_0 \ldots & \\ x = \langle 2, \langle c_1, x_1 \rangle\rangle & \Rightarrow & \ldots c_1 \ldots f(x_1) \ldots & \\ & \vdots & & \\ x = \langle i+1, \langle c_i, x_1, \ldots, x_i \rangle\rangle & \Rightarrow & \ldots c_i \ldots f(x_1) \ldots f(x_i) \ldots & (C_i \neq 0) \\ & \vdots & & \end{array} \right. \end{array}$$

Vejamos, como exemplo de aplicação deste esquema, a especificação de uma função

$$belems \quad : \quad X \longrightarrow 2^A$$

[9] Para não sobrecarregar a notação escreveremos aqui e em situações semelhantes $\langle c_i, x_1, \ldots, x_i \rangle$ em lugar de

$$\langle c_i, \left(\begin{array}{ccc} 1 & \ldots & i \\ x_1 & \ldots & x_i \end{array} \right) \rangle$$

pois $C_i \times X^i \cong C_i \times \underbrace{X \times \ldots \times X}_{i \textit{ vezes}}$.

('binary elems') que desempenhe, a nível de árvores binárias (2.99), a mesma tarefa que $elems$ a nível de listas lineares — a de coleccionar todos os elementos de A presentes em nós de uma árvore [10]. Porque (2.99) é do 2.º grau, com $C_1 = 0$, teremos apenas dois casos — o de paragem, para árvores vazias e o recursivo para nós com 2 sub-árvores:

$$\begin{array}{rcl} belems & : & X \longrightarrow 2^A \\ belems(x) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} x = \langle 1, NIL \rangle & \Rightarrow & \ldots \\ x = \langle 2, \langle a, x_1, x_2 \rangle \rangle & \Rightarrow & \ldots belems(x_1) \ldots belems(x_2) \ldots \end{array} \right. \end{array}$$

Finalmente, a concretização das reticências é óbvia:

$$\begin{array}{rcl} belems & : & X \longrightarrow 2^A \\ belems(x) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} x = \langle 1, NIL \rangle & \Rightarrow & \{\} \\ x = \langle 2, \langle a, x_1, x_2 \rangle \rangle & \Rightarrow & \{a\} \cup belems(x_1) \cup belems(x_2) \end{array} \right. \end{array}$$

Exercício 2.20 Relembre o seguinte modelo formal AB para *árvores binárias de procura*,

$$\begin{array}{rcl} AB & = & X \\ X & \cong & 1 + A \times X^2 \end{array}$$

— *cf.* definição (2.99) — supondo agora em A definida uma ordem total $\leq$.

1. Especifique a função que calcula o *peso* de uma árvore binária em AB.
2. Escreva a definição formal dos predicados $equilibrada$ e $ordenada$ que testam se uma árvore está balanceada e ordenada segundo $\leq$.

□

Conversão de Gramáticas em Estruturas Polinomiais

Recorde da secção 1.2.2 a técnica de conversão de gramáticas independentes de contexto em assinaturas algébricas.

Existe um método alternativo de representação de tais gramáticas em sistemas de definições polinomiais em SETS que tem a vantagem de ser passível de raciocínio e de, em particular, caracterizar a noção de equivalência abstracta entre gramáticas. O método é o seguinte:

> De uma dada gramática independente de contexto $G = \langle N, T, S, P \rangle$, expressa em BNF, gerar um sistema de equações em SETS de acordo com os passos seguintes:

[10] Ao "grau" do polinómio que especifica uma estrutura de dados polinomial, a terminologia tradicional associa a *aridade* ou nível de *linearidade* da estrutura. Neste contexto, podemos considerar árvores binárias como sendo *listas bi-lineares*, listas lineares como sendo *árvores unárias etc.*

- começa-se por eliminar todas as ocorrências de símbolos terminais nas produções em P;
- a cada produção do BNF corresponde uma equação em SETS;
- a cada símbolo não-terminal $\langle A\rangle$ em N corresponde a variável A em SETS;
- usa-se a seguinte tabela de conversão de BNF em SETS:

BNF		SETS
$\alpha \mid \beta$	$\mapsto$	$\alpha + \beta$
$\alpha\, \beta$	$\mapsto$	$\alpha \times \beta$
α^*	$\mapsto$	α^*
α^+	$\mapsto$	$\alpha \times \alpha^*$
ϵ	$\mapsto$	1

Por exemplo, é fácil ver que o seguinte BNF (recordar Exercício 1.6)

$$\begin{array}{lcl} \langle A\rangle & ::= & \mathtt{a}\ \langle B\rangle \mid \mathtt{b}\ \langle A\rangle\ \mathtt{a}\ \langle B\rangle^* \\ \langle B\rangle & ::= & \mathtt{a\ b} \mid \mathtt{a}\ \langle A\rangle^+\ \mathtt{c} \end{array}$$

se converte no seguinte sistema, em SETS:

$$\begin{array}{lcl} A & \cong & B + A \times B^* \\ B & \cong & 1 + A \times A^* \end{array}$$

Já o seguinte fragmento de gramática de contexto livre para expressões de aritmética inteira,

$$\begin{array}{rcl} G & = & \langle NT, T, \langle Exp\rangle, P\rangle \\ NT & = & \{\langle Exp\rangle, \langle Int\rangle, \langle BinOp\rangle\} \\ T & = & \{\ \mathtt{+}\ ,\ \mathtt{-}\ ,\ \mathtt{*}\ ,\ \mathtt{/}\ ,\ \mathtt{(}\ ,\ \mathtt{)}\ , \ldots,\ \mathtt{-2}\ ,\ \mathtt{-1}\ ,\ \mathtt{0}\ ,\ \mathtt{1}\ ,\ \mathtt{2}\ , \ldots\} \\ P & = & \left\{ \begin{array}{rcl} \langle Exp\rangle & ::= & \langle Int\rangle \mid \mathtt{(}\ \langle Exp\rangle\ \mathtt{)} \mid \langle Exp\rangle\langle BinOp\rangle\langle Exp\rangle \\ \langle Int\rangle & ::= & \ldots\ \mathtt{-2} \mid \mathtt{-1} \mid \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \ldots \\ \langle BinOp\rangle & ::= & \mathtt{+} \mid \mathtt{-} \mid \mathtt{*} \mid \mathtt{/} \end{array} \right. \end{array}$$

é convertível no seguinte sistema de equações, recorrendo a este método e ao cálculo de isomorfismos de SETS:

$$\left\{ \begin{array}{rcl} Exp & \cong & Int + Exp + 4 \times Exp^2 \\ Int & \cong & \mathbb{Z}_0 \end{array} \right. \tag{2.126}$$

isto é, numa só equação polinomial do 2.º grau:

$$Exp \cong \mathbb{Z}_0 + Exp + 4 \times Exp^2 \tag{2.127}$$

Duas gramáticas dir-se-ão “equivalentes” ou “isomorfas” se puderem ser reduzidas a sistemas de equações nas mesmas variáveis (a menos de renomeação) que sejam isomorfas uma a uma em $Sets$.

Por exemplo, é possível usar esta técnica de conversão e as leis do cálculo de isomorfismo de SETS para validar as duas transformações para resolver conflitos LL(1) em gramáticas independentes de contexto que se apresentaram no quadro do Exercício 1.7.

No caso da primeira dessas transformações, é imediato transformar G' em

$$\left\{ \begin{array}{lcl} A & \cong & \alpha \times A' \\ A' & \cong & \beta + \gamma \end{array} \right. \leftrightarrow \quad A \quad \cong \quad \alpha \times (\beta + \gamma) \quad \leftrightarrow \quad A \quad \cong \quad \alpha \times \beta + \alpha \times \gamma$$

que é exactamente o resultado da conversão do respectivo fragmento G.

No caso da segunda, ter-se-á, após a conversão de G [11]

$$\begin{array}{lcl} A & \cong & \beta \times A' \\ & \updownarrow & \\ A & \cong & \beta \times (\alpha \times A' + 1) \\ & \updownarrow & \\ A & \cong & \beta \times \alpha \times A' + \beta \times 1 \\ & \updownarrow & \\ A & \cong & \underbrace{\beta \times A'}_{A} \times \alpha + \beta \\ & \updownarrow & \\ A & \cong & A \times \alpha + \beta \end{array}$$

o que coincide de facto com o resultado da conversão do correspondente fragmento G.

Exercício 2.21

1. Aplique o processo de conversão acima descrito às seguintes produções de uma dada gramática:

$$\begin{array}{lcl} \langle B \rangle & ::= & \mathtt{a} \mid \mathtt{a}\ \langle A \rangle^{*}\ \mathtt{b}\ \langle A \rangle\ \mathtt{c} \\ \langle A \rangle & ::= & \mathtt{a}\ \langle B \rangle^{*}\ \mathtt{a}\ \langle A \rangle\ \mathtt{a} \mid \langle B \rangle\ \mathtt{a} \end{array}$$

2. Verifique se esta gramática e a do Exercício 1.6, acima transcrita, são isomorfas.

□

[11] É fácil justificar cada passo com as respectivas leis de SETS.

Exercício 2.22 Mostre que, em SETS, a resolução em ordem a X do sistema de equações

$$\begin{cases} X & \cong & Y \times B \\ Y & \cong & 1 + A \times Y \end{cases}$$

conduz a

$$X \quad \cong \quad B + A \times X$$

para quaisquer A e B.

□

Exercício 2.23 Aplique o processo que conhece para a conversão de BNF em SETS às seguintes produções de uma gramática de contexto livre, bem conhecida, para expressões aritméticas simples:

$$\begin{array}{rcl} G & = & \langle NT, T, \langle E\rangle, P\rangle \\ NT & = & \{\langle E\rangle\} \\ T & = & \{\, \texttt{+}\,,\ \texttt{*}\,,\ \texttt{(}\,,\ \texttt{)}\,,\ \texttt{id}\,,\ \texttt{num}\,\} \\ P & = & \left\{\ \langle E\rangle \quad ::= \quad \langle E\rangle \texttt{ + } \langle E\rangle \mid \langle E\rangle \texttt{ * } \langle E\rangle \mid \texttt{ (} \langle E\rangle \texttt{) } \mid \texttt{ id } \mid \texttt{ num } \ \right\} \end{array}$$

Recorrendo ao cálculo de isomorfismo, mostre ainda que a gramática acima é convertível numa equação polinomial do 2.º grau em SETS. Finalmente, enuncie o método de indução polinomial que está associado à equação que obteve.

□

Exercício 2.24 Considere o seguinte desafio: representar expressões aritméticas de $\langle Exp\rangle$ acima num tipo de dados em PASCAL que não recorra a 'records' variantes. Com base em (2.126), mostre que a seguinte proposta de um colega seu,

```
type Exp = record
    L: ^Node;
    I: Integer
   end;
Node = record
    P: ^BinOps;
    N: ^Node;
    end;
BinOps = record
      O: BinOp;
      E: Exp
      end;
BinOp = (TIMES,DIV,PLUS,DIFF);
```

está correcta, onde `Integer` representa o não-terminal $\langle Int\rangle$. **Sugestão**: "inverta" o PASCAL dado, simplifique-o, e recorra ao resultado provado no Exercício 2.22.

□

2.4 Modelos com Invariantes

Relembremos o Exercício 2.12 em que foram definidos, sobre o modelo AG para árvores genealógicas (2.86), dois predicados — *semRepetição* e *biSexuada*. Reparemos que estes dois predicados registam a "ordem natural das coisas" quanto a árvores genealógicas pois, biologicamente, não só a procriação entre indivíduos é bi-sexuada como também nenhum indivíduo pode ser descendente de si próprio.

Portanto, o modelo de AG aproxima-se da realidade quando lhe associamos tais predicados, e talvez alguns mais. Se definirmos um novo predicado que combine os anteriores,

$$\begin{array}{rcl} \textit{inv-AG} & : & AG \to 2 \\ \textit{inv-AG}(ag) & \stackrel{\mathrm{def}}{=} & \textit{semRepetição}(ag) \wedge \textit{biSexuada}(ag) \end{array}$$

temos nele registada uma propriedade característica, desejável ou "invariante" sobre árvores genealógicas. Quer dizer, o respectivo modelo acaba por vir a ser um subconjunto de AG — aquele que contém apenas as árvores que verificam o invariante *inv-AG*,

$$\{ag \in AG \mid \textit{inv-AG}(ag) = V\}$$

— e não todo o espaço matemático disponibilizado pela definição (2.86).

O conceito de invariante é essencial à aplicação da especificação formal a problemas práticos. Raramente a realidade a especificar é tão "regular" que se possa construir um seu modelo matemático "perfeito", *i.é* sem condicionamentos extra. Por exemplo, pensemos numa espécie de dados tão "simples" de especificar como a vulgar $Data$,

$$Data \cong 31 \times 12 \times I\!N \tag{2.128}$$

onde, como habitualmente, 31 e 12 designam os respectivos segmentos iniciais de $I\!N$, isto é, se $(d, m, a) \in Data$, então $1 \leq d \leq 31$ (dia do mês) e $1 \leq m \leq 12$ (mês). Ora, toda a gente sabe que o modelo cartesiano apresentado em (2.128) é demasiado lato, pois nem todos os meses admitem 31 dias, e existe mesmo um deles ($m = 2$) cujo número de dias depende do ano em questão. Aliás, o problema é bem mais complicado, como se pode avaliar pelo exercício que se segue.

Exercício 2.25 Com base nos seguintes textos [12]:

> "**Do Ano e Sua Divisão** — (...) Júlio César instituiu o ano, de que hoje usamos, de 365 dias e 6 horas, a qual quantidade não é exacta, pois vemos claramente adiantar-se o tempo; (...) a Santa Madre Igreja usa do ano que instituiu Júlio César, tomando em cada ano as 6 horas, que formam um dia inteiro em cada quatro anos, chamando-se bissexto a esse ano, a que se acrescenta um dia (...).
>
> **Da Reforma do Calendário** — Tendo-se observado, que desde a celebração do concílio de Niceia, em 325, até ao ano de 1582, se haviam antecipado os equinócios 10 dias do

[12] In *Lunário de Prognóstico Perpétuo, para Todos os Reinos e Províncias*, por Jerónimo Cortez, Valenciano (séc.XVIII), re-edição Lello & Irmão, 1910.

> assento fixo em que os colocara Dionísio Romano; (...) mandou o papa Gregório XIII proceder à reforma do Calendário, em virtude da qual se determinou: 1.º que no mês de Outubro de 1582 se suprimissem 10 dias, contando 4 no dia de S.Francisco, e 15 no seguinte; 2.º que em cada 400 anos se suprimissem 3 dias, principiando de 1700, 1800, 1900, 2100, 2200, 2300, 2500, *etc.* (que por isso não são bissextos), para diminuir o excesso do ano sinodal ao civil, e os equinócios ficarem imóveis a 21 de Março e 23 de Setembro (...)."

complete as definições dos seguintes predicados, onde $Data$ foi definida em (2.128).

$$\begin{array}{lcl} dataOk & : & Data \to 2 \\ bissexto & : & I\!N \to 2 \end{array}$$

□

Nos exemplos anteriores, vimos razões de ordem biológica ou cosmológica para a introdução de invariantes em modelos. Mas a razão mais vulgar para a sua ocorrência é de ordem *normativa*, isto é, deve-se a regulamentos, normas ou leis impostas aos funcionamento das instituições. Por exemplo, se o regulamento de uma escola impõe, quanto à passagem de ano dos seus alunos, que um aluno não pode ter mais do que o equivalente a duas cadeiras anuais em atrazo, então teremos esse "invariante" a afectar todas as fichas que registem a informação escolar dos alunos. E será conveniente provar que nenhuma transação prevista numa eventual aplicação de gestão académica "ilegaliza" o sistema, *i.é* conduz à existência de fichas que contrariem essa cláusula (e eventualmente outras!).

Assim, a presença de invariantes num modelo representa um "ónus" adicional ao seu desenvolvimento, já que se torna necessário demonstrar que os operadores nele presentes *satisfazem*, ou *respeitam* esses invariantes. Mais concretamente, sempre que um invariante

$$\textit{inv-s} : \mathcal{A}(s) \to 2$$

afecta uma espécie $s \in S$ num modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$, então, para todo o Σ-operador

$$\sigma : s_1 \times \ldots \times s_n \to s$$

e todos os termos $t_i \in W_{\Sigma,s_i}$ $(1 \leq i \leq n)$, será preciso provar que, em $\mathcal{A}$,

$$\textit{inv-s}(\sigma(t_1, \ldots, t_n)) = V \tag{2.129}$$

Sendo σ um operador com semântica parcial, (2.129) assume a forma de uma implicação,

$$pre_\sigma(t_1, \ldots, t_n) \Rightarrow \textit{inv-s}(\sigma(t_1, \ldots, t_n)) \tag{2.130}$$

da qual (2.129) não é mais que um caso particular, sendo a condição $pre_\sigma(t_1, \ldots, t_n)$ universalmente verdadeira. Se alguma das espécies-argumento s_i estiver afectada ela própria por um invariante *inv-*s_i, esse facto pode ser útil na demonstração de (2.130), reforçando o antecedente da sua implicação e assumindo a forma:

$$pre_\sigma(t_1,\ldots,t_n) \wedge (\forall 1 \leq i \leq n : \textit{inv-}s_i(t_i)) \Rightarrow \textit{inv-}s(\sigma(t_1,\ldots,t_n)) \quad (2.131)$$

Repare-se que o conjunto dos invariantes *inv-s*, para todo o $s \in S$, é uma família S-indexada de predicados. Assim, (2.131) é não é mais do que o salto indutivo da demonstração, pelo método de *indução algébrica*, que $\mathcal{A}$ satisfaz essa família de predicados. Mas vejamos, antes de mais, um exemplo.

2.4.1 Prova de Invariantes: Um Exemplo

Neste exemplo pretende-se especificar um sistema não-hierárquico de ficheiros organizados em rede,

$$FSR \quad\cong\quad Id \rightharpoonup File \times 2^{LnkId \times Id}$$

i.é associáveis entre si por *ligações* ($LnkId$) entre os seus identificadores (Id), por exemplo as ligações entre ficheiros que a seguinte tabela regista:

Id	$LnkId$	Id
`x.c`	`includes`	`x.h`
`x.tex`	`manual-of`	`x.c`
`x.c`	`source-of`	`x`
`x1.h`	`includes`	`stdio.h`
$\vdots$	$\vdots$	$\vdots$

É conveniente afixar a FSR um invariante que impeça falsas associações:

$$\text{inv-}FSR \quad \stackrel{\text{def}}{=} \quad \lambda\sigma.\forall k \in dom(\sigma) : links(\sigma,k) \subseteq dom(\sigma) \quad (2.132)$$

onde

$$links(\sigma,k) \quad \stackrel{\text{def}}{=} \quad \{ \; k \in dom(\sigma) \quad \Rightarrow \quad \pi_2[\pi_2(\sigma(k))] \quad (2.133)$$

Acrescentam-se agora os operadores:

$$deps(\sigma,k) \quad \stackrel{\text{def}}{=} \quad \{j \in dom(\sigma) \mid \; k \in links(\sigma,j)\} \quad (2.134)$$

$$rem(\sigma,k) \quad \stackrel{\text{def}}{=} \quad \{ \; deps(\sigma,k) \subseteq \{k\} \quad \Rightarrow \quad \sigma \setminus \{k\} \quad (2.135)$$

o primeiro para interrogação quanto a dependências e o segundo para a remoção de ficheiros. Em suma, tem-se a assinatura:

$$\Sigma = \begin{cases} deps : FSR \times Id \to Ids \\ rem : FSR \times Id \to FSR \\ \text{inv-}FSR : FSR \to Bool \\ links : FSR \times Id \to Ids \end{cases}$$

(e no modelo: $Ids \cong 2^{Id}$).

Vamos de seguida aplicar (2.131) para demonstração da preservação do invariante inv-FSR por parte do operador rem.

Facto a provar:

$$\text{inv-}FSR(\sigma) \wedge \underbrace{deps(\sigma,k) \subseteq \{k\}}_{A} \quad \Rightarrow \quad \underbrace{\text{inv-}FSR(\sigma \setminus \{k\})}_{B} \tag{2.136}$$

Demonstração: Por casos.

Caso 1 — $k \notin dom(\sigma)$:

$$\begin{array}{ccc} \text{inv-}FSR(\sigma) \wedge deps(\sigma,k) \subseteq \{k\} & \Rightarrow & \text{inv-}FSR(\sigma \setminus \{k\}) \\ & \updownarrow & \qquad (2.137) \\ \text{inv-}FSR(\sigma) \wedge deps(\sigma,k) \subseteq \{k\} & \Rightarrow & \text{inv-}FSR(\sigma) \\ & \updownarrow & \qquad (2.138) \\ & V & \end{array}$$

□

Caso 2 — $k \in dom(\sigma)$: Como $\{k\} = \{j \in dom(\sigma) \mid j = k\}$, temos

$$\begin{array}{lc} A = & deps(\sigma,k) \subseteq \{k\} \\ & \updownarrow \qquad (2.139) \\ & \{j \in dom(\sigma) \mid k \in links(\sigma,j)\} \subseteq \{j \in dom(\sigma) \mid j = k\} \\ & \updownarrow \qquad (2.140) \\ & \forall j \in dom(\sigma) : k \in links(\sigma,j) \Rightarrow j = k \\ & \updownarrow \qquad (2.141) \\ & \forall j \in dom(\sigma) : j \neq k \Rightarrow \neg k \in links(\sigma,j) \end{array}$$

e

$$\begin{array}{c} \text{inv-}FSR(\sigma) \wedge A \\ \updownarrow \qquad (2.142) \\ \forall j \in dom(\sigma) : (links(\sigma,j) \subseteq dom(\sigma)) \wedge (j \neq k \Rightarrow \neg k \in links(\sigma,j)) \\ \downarrow \qquad (2.143) \\ \forall j \in dom(\sigma) - \{k\} : (links(\sigma,j) \subseteq dom(\sigma)) \wedge (V \Rightarrow \neg k \in links(\sigma,j)) \\ \updownarrow \qquad (2.144) \\ \forall j \in dom(\sigma) - \{k\} : (links(\sigma,j) \subseteq dom(\sigma)) \wedge \neg k \in links(\sigma,j) \\ \updownarrow \qquad (2.145) \\ \forall j \in dom(\sigma) - \{k\} : links(\sigma,j) \subseteq dom(\sigma) \wedge \{k\} \cap links(\sigma,j) = \emptyset \\ \updownarrow \qquad (2.146) \\ \forall j \in dom(\sigma) - \{k\} : links(\sigma,j) \subseteq dom(\sigma) \wedge links(\sigma,j) \subseteq \overline{\{k\}} \\ \updownarrow \qquad (2.147) \\ \forall j \in dom(\sigma) - \{k\} : links(\sigma,j) \subseteq (dom(\sigma) - \{k\}) \end{array}$$

$$\Updownarrow \tag{2.148}$$
$$\forall j \in dom(\sigma \setminus \{k\}) : links(\sigma, j) \subseteq dom(\sigma \setminus \{k\})$$
$$\Updownarrow \tag{2.149}$$
$$\text{inv-}FSR(\sigma \setminus \{k\}) \qquad = B$$

Logo a implicação (2.136) verifica-se. □

Justificação passo a passo:

(2.137) — leis (A.14) e (A.53) do apêndice A

(2.138) — lei (A.1) do mesmo apêndice

(2.139) — por substituição

(2.140) — leis (A.10) e (A.13)

(2.141) — lei (A.2)

(2.142) — por substituição e lei (A.5)

(2.143) — lei (A.6)

(2.144) — lei (A.3)

(2.145) — lei (A.14)

(2.146) — lei (A.15)

(2.147) — leis (A.16) e (A.17)

(2.148) — lei (A.54)

(2.149) — por instanciação ('folding' [13]) de (2.136) — substitua-se σ por $\sigma \setminus \{k\}$

□

Em suma, a prova acabou por se desenrolar segundo dois casos, ambos de natureza dedutiva. Será sempre assim na generalidade dos casos? Por que razão se conduziu a prova dessa forma? Na secção seguinte estudar-se-ão em detalhe técnicas de prova de invariantes ajustadas a cada modelo semântico.

Exercício 2.26 Considere a seguinte função recursiva

$$\begin{array}{rcl} f & : & I\!N_0 \longrightarrow I\!N_0 \\ f(k) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} k = 0 & \Rightarrow & 0 \\ k > 0 & \Rightarrow & impar(k) + f(k-1) \end{array} \right. \end{array}$$

onde

$$\begin{array}{rcl} impar & : & I\!N \longrightarrow I\!N \\ impar(j) & \stackrel{\text{def}}{=} & 2j - 1 \end{array}$$

Mostre, por indução sobre $I\!N_0$, que f calcula o quadrado de k, *i.é* que $f(k) = k^2$.

□

[13] Consultar a propósito a secção B.4 em apêndice.

2.4.2 Métodos (Indutivos) de Prova

Tradicionalmente, na matemática, os métodos de prova dividem-se em *dedutivos* e *indutivos*. O exercício 2.26 ilustra um dos métodos de indução dita *primitiva*, *i.é* sobre $I\!N_0$.

Mas $I\!N_0$ é apenas um dos domínios de dados arbitrariamente complexos que podemos construir em $Sets$. Será que métodos como a indução primitiva são suficientes para raciocinar sobre essas estruturas arbitrariamente complexas? A resposta é negativa. Mas a verdadeira questão é: assim como a indução primitiva se *ajusta* de forma particular aos raciocínios sobre $I\!N_0$, não existirá uma forma estrutural de associar a cada estrutura em $Sets$ o "seu" método de prova (indutiva)?

Mostraremos de imediato que não é difícil associar um método de prova às principais construções de $Sets$ — $\times$, $+$, *, $\rightharpoonup$, *etc.* — para definição de modelos. Seja e uma expressão envolvendo esses operadores, e seja $\varphi : e \rightarrow 2$ um predicado em $Sets$. Suponhamos que queremos provar a validade de

$$\forall x \in e : \varphi(x) \tag{2.150}$$

Para $e = X \times Y$ a prova desdobrar-se-á trivialmente sobre X e Y, simultâneamente. Para $e = X + Y$ a prova desdobra-se também sobre X e Y, mas agora por casos, conforme is-$X(x)$ ou is-$Y(x)$ é verdadeiro.

Nos restantes casos, será preciso recorrer a métodos de indução estrutural baseados em ordens bem-fundadas (*cf.* Exercício 1.22).

Conjuntos

Para $e = 2^X$, a ordem bem-fundada mais óbvia é a *relação de cobertura* do reticulado $(2^X; \subseteq)$, *i.é* a relação $<$ tal que

$$x' < x \Leftrightarrow (\exists e \in x : x' = x - \{e\})$$

da qual $\emptyset$ é limite universal inferior e, portanto, o único caso de base do método. Assim, para provar (2.150) neste contexto, *i.é* provar

$$\forall x \in 2^X : \varphi(x)$$

usar-se-á o método seguinte:

1. *Caso de Base*: provar $\varphi(\emptyset)$.
2. *Salto Indutivo*: seja $x \in 2^X$ tal que $x \supset \emptyset$.
   (a) *Hipótese de indução*:

$$\forall e \in x : \varphi(x - \{e\})$$

(b) *Passo*:

$$\forall e \in x : \varphi(x - \{e\}) \Rightarrow \varphi(x)$$

Exercício 2.27 Use o método de indução estrutural sobre conjuntos para verificar a validade de (2.77).

□

Exercício 2.28 Considere a seguinte assinatura para manipulação de grafos acíclicos:

$$\Sigma = \begin{cases} init : \to Graph \\ allNodes : Graph \to Nodes \\ sucs : Node \times Graph \to Nodes \\ addArc : Node \times Node \times Graph \to Graph \\ acyclic : Graph \to Bool \end{cases}$$

sobre a qual se construiu o modelo

$$\mathcal{A}(\Sigma) = \begin{cases} Graph \cong 2^{Node \times Node} \\ Nodes \cong 2^{Node} \\ Bool \cong 2 \\ init \stackrel{\text{def}}{=} \emptyset \\ allNodes \stackrel{\text{def}}{=} \lambda(g).\pi_1[g] \cup \pi_2[g] \\ sucs \stackrel{\text{def}}{=} \lambda(a,g).\bigcup_{t \in g \wedge \pi_1(t)=a} \{\pi_2(t)\} \cup sucs(\pi_2(t), g) \\ addArc \stackrel{\text{def}}{=} \lambda(a, a', g).\left\{ \begin{array}{lll} a \notin sucs(a', g) & \Rightarrow & g \cup \{\langle a, a' \rangle\} \end{array} \right. \\ acyclic \stackrel{\text{def}}{=} \lambda(g).\forall a \in allNodes(g) : a \notin sucs(a, g) \end{cases}$$

Mostre que o operador $addArc$ <u>não</u> preserva o invariante $acyclic$.

□

Sequências

Para $e = X^*$, a ordem bem-fundada que "encaixa" no esquema funcional associado a este caso é

$$s' < s \Leftrightarrow s' = tail(s) \tag{2.151}$$

para $s, s' \in X^*$. A ordem (2.151) é também uma relação de cobertura, a da *c.p.o* $(X^*; \leq)$ para $s' \leq s$ entendida como "s' é sufixo de s", cujo limite universal inferior é a sequência vazia $<>$. Em resumo, para provar

$$\forall x \in X^* : \varphi(x)$$

temos o método seguinte:

1. *Caso de Base*: provar $\varphi(<>)$.

2. *Salto Indutivo*: seja $x \in X^*$ tal que $x \neq <>$.

(a) *Hipótese de indução*:

$$\varphi(tail(x)) = V$$

(b) *Passo*:

$$\varphi(tail(x)) \Rightarrow \varphi(x)$$

Funções Finitas

Finalmente, para provarmos

$$\forall x \in X \rightharpoonup Y : \varphi(x)$$

— instância de (2.150) para funções finitas — podemos, simplesmente, fazer indução estrutural sobre o domínio $dom(x) \in 2^X$.

Indução Polinomial

Usar-se-á para (2.116) o método indutivo seguinte:

1. *Caso de Base*: $(i = 0)$ provar $\varphi(\langle 1, c_0 \rangle)$ para qualquer $c_0 \in C_0$.
2. *Salto Indutivo*: $(i > 0)$ para cada $0 < i \leq n$ tal que $C_i \neq 0$, seja $c_i \in C_i$ e $x_1, \ldots, x_j, \ldots, x_i \in X$:

   (a) *Hipótese de indução*:

   $$\bigwedge_{j=1}^{i} \varphi(x_j)$$

   (b) *Passo*:

   $$\bigwedge_{j=1}^{i} \varphi(x_j) \quad \Rightarrow \quad \varphi(\langle i+1, \langle c_i, x_1, \ldots, x_i \rangle \rangle)$$

Exercício 2.29

1. Enuncie o método de indução polinomial associado a $Exp \cong \mathsf{F}(Exp)$ dada por (2.127).
2. Sobre o respectivo esquema algorítmico polinomial, especifique a função

$$\begin{array}{rcl} eval & : & Exp \longrightarrow \mathbb{Z}_0 \\ eval(e) & \stackrel{\text{def}}{=} & \ldots \end{array}$$

/* *calcula o valor em $\mathbb{Z}_0$ da expressão e; o terminal " / " deverá ser interpretado como o operador de divisão inteira* */

□

Exercício 2.30 É dada a seguinte definição de uma função para inversão de sequências:

$$\begin{array}{lcl} invl & : & X^\star \longrightarrow X^\star \\ invl(l) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} l = <> & \Rightarrow & <> \\ l \neq <> & \Rightarrow & invl(tail(l)) \frown <head(l)> \end{array} \right. \end{array}$$

onde $\frown$ é a função definida no Exercício 2.9. Mostre que $invl$ e $\frown$ comutam entre si da forma seguinte:

$$invl(a \frown b) \quad = \quad invl(b) \frown invl(a)$$

Sugestão: faça indução sobre a (sequências) e tome em consideração as leis (A.83) e (A.84).

□

Exercício 2.31 Na sequência do Exercício 2.30 verifique (*e.g.* por indução) se a especificação de $invl$ satisfaz a seguinte propriedade:

$$\forall s \in X^\star : elems(invl(s)) = elems(s)$$

□

Exercício 2.32 Considere a seguinte função que especifica um tipo de inserção em lista de naturais: estratégia de

$$\begin{array}{lcl} ins & : & I\!N^* \times I\!N \longrightarrow I\!N^* \\ ins(l,a) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} l = <> & \Rightarrow & <a> \\ l \neq <> & \Rightarrow & \begin{array}{ll} let & h = head(l) \\ & t = tail(l) \\ in & \left\{ \begin{array}{lcl} a = h & \Rightarrow & l \\ a < h & \Rightarrow & cons(a,l) \\ a > h & \Rightarrow & cons(h, ins(t,a)) \end{array} \right. \end{array} \end{array} \right. \end{array}$$

1. Demonstre ou refute se ins preserva o invariante sobre listas que se segue:

$$\phi(l) \stackrel{\text{def}}{=} length(l) = card(elems(l))$$

2. Qual o significado informal deste invariante? É capaz de especificar um outro invariante que ins (também) satisfaça? Justifique informalmente.

□

Exercício 2.33 No contexto do Exercício 2.32, demonstre que ins preserva o invariante sobre listas que se segue:

$$\varphi(l) \stackrel{\text{def}}{=} \forall i,j \leq length(l) : i < j \Rightarrow l(i) < l(j)$$

□

Exercício 2.34 Demonstre por indução em $X^\star$ o facto seguinte:

$$\forall l \in X^\star : invl(invl(l)) = l$$

onde $\frown$ e $invl$ são as funçõees referidas no Exercício 2.30.
□

Exercício 2.35 Na sequência do Exercício 2.11, verifique a validade de:

$$length(gate(bs, bs)) \quad \leq \quad length(bs)$$

□

Exercício 2.36 Na sequência do Exercício 2.10,

1. demonstre que a relação binária $\sqsubseteq$ é reflexiva;
2. verifique se $\langle A^\star, \sqsubseteq\rangle$ é uma ordem com limite universal inferior.

□

Exercício 2.37 Suponha que, em (2.99), $A = K \times D$, onde K é um domínio não-vazio de *chaves* de acesso a D.

Especifique neste contexto um invariante que impeça a mesma chave $k \in K$ de se repetir em posições diferentes da mesma árvore [14].
□

Exercício 2.38 Considere o seguinte modelo em $Sets$ de uma *árvore-B*

$$\begin{array}{rcl} BT & \cong & 1 + Block \\ Block & \cong & BT \times (A \times BT)^\star \end{array}$$

onde, sobre o seu tipo de elementos A se considera definida uma ordem total $<$.

1. Verifique se a estrutura BT é polinomial. **Sugestão**: Relembre (2.31).

[14] O leitor deverá identificar aqui as chamadas *árvores binárias de procura* ('binary search trees') de [Wir76].

2. Complete a seguinte definição da função

$$
\begin{array}{lcl}
collect & : & BT \longrightarrow 2^A \\
collect(t) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcll}
t = NIL & \Rightarrow & \emptyset & \\
t \neq NIL & \Rightarrow & let & t_0 = \pi_1(t) \\
 & & & l = \pi_2(t) \\
 & & & m = length(l) \\
 & & in & collect(t_0) \cup \ldots
\end{array} \right.
\end{array}
$$

que deverá colectar o conjunto de todos os elementos presentes numa dada árvore-B.

3. Construa a definição de um invariante sobre BT, que garanta as cláusulas estruturais que são conhecidas sobre *árvores-B*, a saber:

   - todos os blocos, excepto a raiz, têm $n \leq m \leq 2n$ elementos, onde $n \geq 1$ é a ordem da árvore em questão;
   - sendo $\boxed{t_0 \mid a_1 \mid t_1 \mid a_2 \mid t_2 \mid \ldots \mid a_m \mid t_m}$ um bloco a qualquer nível da árvore, se a ocorre em t_i ($0 \leq i \leq m$) então $a_i < a < a_{i+1}$ (mas atenção quando $i \in \{0, m\}$!).

   a partir do esboço seguinte:

$$
\text{inv-}BT(t) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lcll}
t = NIL & \Rightarrow & V & \\
t \neq NIL & \Rightarrow & let & t_0 = \pi_1(t) \\
 & & & l = \pi_2(t) \\
 & & & m = length(l) \\
 & & & a_1 = \pi_1(l(1)) \\
 & & & t_m = \pi_2(l(m)) \\
 & & in & \ldots m \ldots \wedge \\
 & & & \forall a \in collect(t_0) : a < a_1 \wedge \\
 & & & \forall a \in collect(t_m) : \ldots \wedge \\
 & & & \ldots
\end{array} \right.
$$

□

Exercício 2.39 No contexto do exercício 2.20, especifique a operação

$$insert : AB \times A \longrightarrow AB$$

que insere elementos de As numa árvore binária ordenada. Use o método indutivo associado a esta estrutura para demonstrar que a sua proposta para $insert$ preserva o respectivo (sub)invariante $ordenada$:

$$\forall x \in AB, a \in A : ordenada(x) \Rightarrow ordenada(insert(x, a))$$

□

Exercício 2.40 Estude o esquema indutivo associado à definição recursiva (2.101) de termos com variáveis.

□

Exercício 2.41 Na sequência do Exercício 2.20, relembre a conhecida técnica de representação de *árvores binárias de procura* sob a forma de *'arrays' com marcas*, *i.é* a que representa a árvore

$$t = (12, (6, (3, (), ()), (9, (), ())), ())$$

pelo 'array'

$$a = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|}\hline 12 & 6 & \Box & 3 & 9 & & & \Box & \Box & \Box & \Box \\ \hline\end{array}$$

Considerando o seguinte modelo para *'arrays' com marcas*,

$$\begin{array}{rcl} AM & = & I\!N \rightharpoonup A \cup \{\Box\} \\ & \cong & I\!N \rightharpoonup A + 1 \end{array}$$

onde se supõe, para simplificar a notação, que $\Box \notin A$,

1. dada a seguinte especificação da função que representa ABs em AMs:

$$\begin{array}{rcl} rep & : & AB \longrightarrow AM \\ rep(t) & \stackrel{\rm def}{=} & aux(t, 1) \\ & onde & \end{array}$$

$$aux(t,i) \stackrel{\rm def}{=} \begin{cases} t = NIL & \Rightarrow \quad \begin{pmatrix} i \\ \Box \end{pmatrix} \\ t \neq NIL & \Rightarrow \quad \begin{array}{ll} let & a = \pi_1(t) \\ & st = \pi_2(t) \\ & t_e = st(1) \\ & t_d = st(2) \\ in & \begin{pmatrix} i \\ a \end{pmatrix} \quad \cup \\ & aux(t_e, 2i) \quad \cup \\ & aux(t_d, 2i+1) \end{array} \end{cases}$$

   use indução polinomial sobre AB para mostrar que essa função satisfaz o invariante seguinte:

$$\begin{array}{rcl} invAM & : & AM \longrightarrow 2 \\ invAM(a) & \stackrel{\rm def}{=} & \forall n \in dom(a) : a(n) \neq \Box \Rightarrow 2n \in dom(a) \wedge 2n+1 \in dom(a) \end{array}$$

   i.é, que

$$\forall t \in AB : invAM(rep(t))$$

2. indique — justificando — quais dos seguintes predicados escolheria para exprimir mais propriedades desejáveis (*i.é*, invariantes) para qualquer *'array' com marcas* $a \in AM$:

$$\forall n \in dom(a) : 2n \in dom(a) \wedge 2n+1 \in dom(a) \tag{2.152}$$

$$\forall n \in I\!N : n \notin dom(a) \Rightarrow \frac{n}{2} \notin dom(a) \vee \frac{n}{2} - 1 \notin dom(a) \tag{2.153}$$

$$\forall n \in dom(a) : a(n) \neq \Box \Rightarrow \{2n, 2n+1\} \subseteq dom(a) \tag{2.154}$$

nenhum dos anteriores (2.155)

$\Box$

2.5 Comparação e Classificação de Modelos

Finalizaremos este capítulo com algumas questões que naturalmente se levantam quando estamos a especificar um problema que se nos pôs de novo e que são as seguintes: *será o problema de facto "novo"? será que não se pode "re-utilizar" na nova especificação alguma experiência já adquirida em especificações do passado?*.

Normalmente, é com a sua intuição e experiência que o especificador tenta "responder" a estas questões. E o problema coloca-se mais, de certo modo, ao nível do desenvolvimento de uma solução — ou implementação — do que a nível da sua especificação. Contudo, a questão de fundo que aqui se levanta — o problema da *comparação* de especificações — é muito relevante. Vejamos, ainda que de forma sumária, como o cálculo de isomorfismo em $Sets$ permite uma estratégia inequívoca e rigorosa para abordar essa questão de fundo.

Veremos o seguinte exemplo. Seja $B \cong 2$ na definição abstracta de a árvore de decisão que atrás se deu em (2.125). Podemos então fazer o seguinte $\cong$-raciocínio:

$$\begin{array}{rcl} A \times (B \rightharpoonup X) & \cong & A \times (2 \rightharpoonup X) \\ & \cong & A \times (X+1)^2 \\ & \cong & A \times (X+1) \times (X+1) \end{array}$$

— *cf.* (2.108) and (2.46) — ou seja, de (2.125) obtemos:

$$X \cong A \times (X+1) \times (X+1) \tag{2.156}$$

— nada mais do que o padrão recursivo de *esquemas genealógicos* ou 'pedigrees', *cf.* (2.118).

É óbvio que (2.156) e (2.118) são o mesmo modelo de dados, a menos de uma renomeação de símbolos mais sugestiva e da introdução explícita de selectores no produto cartesiano. Conclui-se que *esquemas genealógicos* "são" casos particulares de *árvores de decisão*, quer dizer, se se tiver disponível uma 'package' que implemente estas, então essa 'package' pode ser simplesmente *re-utilizada* para implementar aqueles. É claro que a comparação plena entre os dois modelos deve também examinar a sua estrutura algorítmica [15]. Mas o objectivo deste exemplo — e dos exercícios 2.64, 2.65 e 2.66 que constam da secção 2.6 — é aqui tão somente o de mostrar a imediata utilização dos isomorfismos de $Sets$ na *comparação* de especificações de tipos de dados.

[15] Por exemplo, tomar decisões em $DecTree$ corresponde exactamente a ascender informação genealógica em $GenDia$, para um dado *menu* (conjunto de "respostas" disponíveis) em cada nível de "decisão": $2 \cong \{pai, \textit{mãe}\}$.

2.6 Exercícios

Exercício 2.42 Calcule o menor dos pontos fixos da equação

$$x = R \cup x^{-1}$$

onde $R \subseteq P \times P$ é uma relação em $P \supset \emptyset$, e r^{-1} designa a relação definida por (1.51). Coincidirá o resultado do seu cálculo com o maior dos pontos-fixos da mesma equação? Justifique.

□

Exercício 2.43 Calcule o menor dos pontos fixos da equação

$$x = R \cup x^{\Delta}$$

onde $R \subseteq P \times P$ é uma relação em $P \supset \emptyset$, e r^{Δ} designa a relação

$$r^{\Delta} = \{\langle a, a\rangle \mid\ a \in \pi_1[r] \cup \pi_2[r]\}$$

□

Exercício 2.44 Considere a definição recursiva $x = f(x)$ que é dada pela equação seguinte

$$x = 1_P \cup R \cup x^{-1} \cup R \circ x$$

em $2^{P \times P}$ (relações binárias sobre um conjunto P não vazio), onde $R \subseteq P \times P$, 1_P designa a relação $\{\langle p, p\rangle \mid\ p \in P\}$ e $x^{-1} = \{\langle q, p\rangle \mid\ (p, q) \in x\}$.

Calcule μf para $R = \emptyset$ e mostre que, para qualquer R, $P \times P$ é o maior de todos os pontos-fixos de f.

□

Exercício 2.45 Pretendendo-se determinar a menor solução da equação

$$x = R \cup x^2$$

onde $R \subseteq P \times P$ é uma relação em $P \supset \emptyset$, decidiu-se aplicar-lhe o Teorema de Kleene, e assim calcular μf para $f(x) = R \cup x^2$.

1. Mostre que f é, de facto, uma função contínua.
2. Complete o referido processo de cálculo, justificando os passos já dados:

$$\begin{array}{rcl}
f^0(\emptyset) & = & \emptyset \\
f^1(\emptyset) & = & R \\
f^2(\emptyset) & = & R \cup R^2 \\
f^3(\emptyset) & = & R \cup (R \cup R^2)^2 \\
 & = & R \cup ((R \cup R^2) \circ (R \cup R^2)) \\
 & = & R \cup ((R \cup R^2) \circ R) \cup ((R \cup R^2) \circ R^2) \\
 & = & R \cup R^2 \cup R^3 \cup R^3 \cup R^4 \\
 & = & R \cup R^2 \cup R^3 \cup R^4 \\
 & \vdots &
\end{array}$$

□

Exercício 2.46 Comente informalmente a seguinte afirmação:

> *O conjunto de todos os pontos fixos de $x = R \cup R \circ x$ é exactamente constituído por todas as relações r que obedecem às seguintes propriedades: (a) r é transitiva; (b) r inclui R.*

□

Exercício 2.47 Discuta formalmente a validade da seguinte afirmação:

> *Existe pelo menos uma relação $R \subseteq P \times P$ tal que o seu fecho simétrico, dado pela equação $x = R \cup x^{-1}$, coincide com o seu fecho transitivo, dado pela equação $y = R \cup R \circ y$.*

□

Exercício 2.48 Calcule, com base na álgebra dos SETS, a menor solução da seguinte equação, para A e B não-vazios:

$$X \quad \cong \quad A \times X + (X \rightharpoonup 0) + B \times X$$

□

Exercício 2.49 Pretendendo provar o isomorfismo

$$A^\star \times I\!N \cong A^\star \times A^\star \tag{2.157}$$

em $Sets$, alguém propôs a seguinte função como bijecção, com origem em $A^\star \times A^\star$:

$$pack(l, l') \quad \stackrel{\mathrm{def}}{=} \quad \langle l \frown l', length(l) + 1 \rangle$$

Será (2.157) um isomorfismo válido em $Sets$? Será $pack$ de facto uma bijecção? Justifique informalmente.

□

Exercício 2.50 Mostre, através de um contra-exemplo, que o isomorfismo

$$(A^\star)^2 \quad \cong \quad 2^\star \times A^\star$$

é, em geral, falso.

□

Exercício 2.51 Considere a seguinte função recursiva, escrita em notação SETS,

$$g \quad \stackrel{\mathrm{def}}{=} \quad 1 + f \times g$$

onde se assume a existência da função $f : A \longrightarrow B$.

1. Recorrendo a (1.19) e (1.29), re-escreva g em notação funcional clássica e tipifique-a, isto é, complete as reticências em

$$\begin{array}{rcl} g & : & \cdots \longrightarrow \cdots \\ g(x) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcl} \cdots & \Rightarrow & \cdots \\ \cdots & \Rightarrow & \cdots \end{array} \right. \end{array}$$

2. Faça $f = card$. Qual o significado de g para este caso? Justifique informal mas adequadamente.
3. Encontra alguma relação entre g e $f^\star$ (1.84)? Na afirmativa, caracterize-a adequadamente.

□

Exercício 2.52 Recorde o fragmento de gramática de contexto livre para expressões aritméticas que é assunto do Exercício 1.42. Recorra ao método que estudou na secção 2.3.7 para converter a gramática acima num sistema de equações em SETS, resolvendo-o em ordem a Exp.

□

Exercício 2.53 Considere a seguinte definição de uma função sobre sequências:

$$subl : A^\star \times I\!N \times I\!N_0 \to A^\star$$

$$subl(l,i,n) \stackrel{\mathrm{def}}{=}$$

$$\left\{ \begin{array}{lcl} n = 0 \vee l = <> & \Rightarrow & <> \\ n > 0 \wedge l \neq <> & \Rightarrow & \begin{array}{ll} let & h = head(l) \\ & t = tail(l) \\ in & \left\{ \begin{array}{lcl} i = 1 & \Rightarrow & <h> \frown subl(t,i,n-1) \\ i > 1 & \Rightarrow & subl(t,i-1,n) \end{array} \right. \end{array} \end{array} \right.$$

1. Comece por preencher o quadro

i	n	$subl(l,i,n)$
2	2	
3	3	
4	3	

para $l = <2,3,1,1,2>$. Descreva, então, por palavras suas, o propósito e a utilidade da função $subl$.

2. Mostre (*e.g.* por indução em $A^\star$) que o seguinte facto se verifica:

$$subl(x,1,length(x)) \quad = \quad x \tag{2.158}$$

□

Exercício 2.54 Relembre do Exercício 2.53 a função $subl$ que extrai de uma sequência l a sua sub-sequência de n elementos (ou de tantos quantos fôr possível extrair) que começa na i-esima posição.

1. Demonstre (por indução em $A^\star$) ou refute (por contra-exemplo) os seguintes factos sobre $subl$:

$$f^\star(subl(l,i,n)) = subl(f^\star(l),i,n) \tag{2.159}$$
$$subl(l, length(l)+1, n) = <> \tag{2.160}$$

   onde $f^\star(l)$ é a construção (1.84).

2. Interprete por palavras suas e através de exemplos de aplicação os factos verificados.

□

Exercício 2.55 Na sequência do Exercício 2.53,

1. Mostre (*e.g.* por apresentação de um contra-exemplo) que o seguinte facto (suposto universalmente quantificado) não se verifica:

$$length(subl(l,i,n)) = n \tag{2.161}$$

2. Mostre (*e.g.* por indução em $A^\star$) que o seguinte facto (suposto universalmente quantificado) se verifica:

$$subl(l,i,n+m) = subl(l,i,n) \frown subl(l,i+n,m) \tag{2.162}$$

□

Exercício 2.56 Uma das operações mais primitivas de um pacote de 'data mining' é aquela que permite aglutinar ou sumariar dados numéricos de acordo com critérios vários de classificação. Por exemplo, sejam dados três quadros,

Mês	**Total** ($\times 10^6$ $00)
Janeiro	21
Fevereiro	10
Março	9
Abril	15
Maio	9
Junho	9
Julho	10
Agosto	12
Setembro	7
Outubro	5
Novembro	3
Dezembro	30

(a) — Vendas

Mês	**Estação**
Janeiro	Inverno
Fevereiro	Inverno
Março	Primavera
Abril	Primavera
Maio	Primavera
Junho	Verão
Julho	Verão
Agosto	Verão
Setembro	Outono
Outubro	Outono
Novembro	Outono
Dezembro	Inverno

(b) — Critério "Estação"

Mês	**Vendedor**
Janeiro	A
Fevereiro	A
Março	B
Abril	B
Maio	C
Junho	C
Julho	A
Agosto	A
Setembro	B
Outubro	B
Novembro	C
Dezembro	C

(c) — Critério "Vendedor"

em que: (a) corresponde à relação de vendas, por mês, de uma dada firma comercial; (b) corresponde ao critério que classifica meses em estações do ano; e (c) corresponde ao critério que relaciona meses com os 3 vendedores da firma.

As duas tabelas que se seguem correspondem à aglutinação de (a) seguindo, respectivamente, o critério (b) e o critério (c):

Estação	**Total** ($\times 10^6$ \$00)
Inverno	61
Primavera	33
Verão	31
Outono	15

(d) — Critério "Estação"

Vendedor	**Total** ($\times 10^6$ \$00)
A	53
B	36
C	51

(e) — Critério "Vendedor"

Repare que as tabelas (a), (d) e (e) são multiconjuntos de tipo genérico

$$MSet(A) \quad \cong \quad A \rightharpoonup I\!N$$

— *cf.* Exercício 1.38 — e as tabelas (b) e (c) são funções de classificação de tipo genérico

$$A \rightharpoonup B$$

1. Especifique formalmente a função $msetTot : MSet(A) \longrightarrow I\!N_0$ que calcula o total das multiplicidades de um multiconjunto.
2. Especifique formalmente a função

$$\begin{array}{rcl} msetAgl & : & MSet(A) \times (A \rightharpoonup B) \longrightarrow MSet(B) \\ msetAgl(ms, f) & \stackrel{\rm def}{=} & \ldots \end{array}$$

   que realiza a operação de aglutinação que acima se ilustrou.
3. Demonstre ou refute o facto seguinte sobre essa função:

$$msetAgl(ms, ms) \quad = \quad id$$

   onde id representa a função identidade no conjunto relevante (qual?).

□

Exercício 2.57 Tendo em consideração o seguinte fragmento de um modelo que conhece,

$$\begin{array}{rcl} Sistema & \stackrel{\rm def}{=} & IdConta \rightharpoonup (2^{Titular} \times Quantia) \\ Quantia & \stackrel{\rm def}{=} & I\!N \end{array}$$

referente à especificação de um sistema de gestão de contas bancário muito simplificado, responda às seguintes questões:

1. Mostre, através do contra-exemplo sugerido no quadro seguinte,

$\sigma =$

$IdConta$	$2^{Titular}$	$Quantia$
k_1	$\{t_1, t_2\}$	10
k_2	$\{t_3\}$	10

   que não está bem especificada a seguinte função que pretende calcular o capital total depositado num dado sistema bancário:

$$\begin{array}{rcl} total & : & Sistema \longrightarrow Quantia \\ total(\sigma) & \stackrel{\rm def}{=} & \sum rng(\left(\begin{array}{c} k \\ \pi_2(\sigma(k)) \end{array} \right)_{k \in dom(\sigma)}) \end{array}$$

   Re-especifique $total$ de forma adequada.

2. Especifique uma função através da qual se possa calcular o conjunto de todos os titulares de contas de um dado banco:

$$todosTit \quad : \quad Sistema \longrightarrow 2^{Titular}$$

□

Exercício 2.58 Recorde o fragmento de uma especificação de um *plano de estudos* de uma licenciatura apresentado no Exercício 1.34. Especifique um invariante sobre $PlanoEstudos$ capaz de garantir as seguintes propriedades:

1. Não há "buracos" no plano de estudos, *i.é*, se existe o ano $n > 1$ do curso, então existe também o ano $n - 1$, *etc.*
2. O número de disciplinas por semestre não pode exceder 5.

□

Exercício 2.59 No contexto do Exercício 1.38 suponha que, para os requisitos do problema do Exercício 1.3, se decidiu modelar a base de dados de produção de equipamento ($Equip$) com multiconjuntos:

$$\begin{array}{rcl} Equip & = & \#Equip \rightharpoonup (\#Unid \rightharpoonup I\!N) \\ \#Unid & = & \#Comp + \#Equip \end{array} \tag{2.163}$$

onde *#Equip (código de equipamento)* e *#Comp (código de componente)* são tipos primitivos.

1. Após uma análise cuidadosa de (2.163), decerto verificará ser necessário fixar um invariante ϕ sobre $Equip$. Comece por escrevê-lo por palavras suas que depois deverá traduzir formalmente na definição de um predicado.
2. Suponha ainda que a mesma equipa já esboçou a especificação da função "explosão em componentes" como se segue:

$$\begin{array}{rcl} explode & : & \#Equip \times Equip \longrightarrow Comps \\ explode(e, \sigma) & \stackrel{\rm def}{=} & \begin{array}{ll} let & b = \sigma(e) \\ in & \bigoplus_{u \in dom(b)} \cdots \end{array} \end{array}$$

   onde ocorre a iteração do operador união de dois multiconjuntos (1.102) e

$$Comps = \#Comp \rightharpoonup I\!N$$

   Complete a definição de $explode$.

□

Exercício 2.60 Considere o seguinte fragmento (aliás incompleto) de um programa em C para gestão de um determinado tipo de tabelas de 'hashing':

```
#define    TAM   100
.
.
.
.
struct inf
{
  med      largura;
  med      altura;
};

struct arvbin
{
  int codigo;
  struct inf    inform;
  struct arvbin *dir;
  struct arvbin *esq;
};

typedef  struct inf INF;
typedef  struct arvbin *AB;

AB hash[TAM];
```

1. Converta para modelo de dados em $Sets$ o tipo da variável `hash`, à luz da analogia que estudou entre as construções primitivas de $Sets$ e a notação para definição de estruturas de dados em linguagens de programação estruturadas.

2. Sabendo que a função de 'hashing' opera sobre o campo `codigo` de `arvbin`, especifique o seguinte invariante a afixar a esse modelo de dados: *para cada índice do 'array' de 'hashing', os códigos que ocorrem em nós da respectiva árvore de colisão têm todos o mesmo valor de 'hashing', que é exactamente o índice respectivo.*

□

Exercício 2.61 No contexto do exercício 1.32, complete a especificação do invariante

$$\text{inv-}WWW(\sigma) \quad \stackrel{\text{def}}{=} \quad \forall k \in dom(\sigma) : \ldots$$

que deverá garantir que nenhuma página WWW contém uma referência para uma página WWW que não existe.

□

Exercício 2.62 Um orçamento é uma associação de custos a ítens a realizar, podendo certos ítens desdobrar-se em sub-ítens e assim sucessivamente. Por exemplo, o orçamento de uma obra de construção civil poderia ter o aspecto do quadro seguinte,

Obra de betão armado			*10000*
Paredes e rebocos			*5000*
Carpintaria	*Aberturas*	*Janelas e portas*	*3000*
		Portas interiores	*2000*
	Pavimentos		*2000*
Louças sanitárias etc.			*1000*
Pavimentos			*2000*

Suponha que alguém definiu o seguinte modelo matemático recursivo para orçamentos,

$$\begin{array}{rcl} Orc & \cong & Item \rightharpoonup (I\!N + SubOrc) \;\; \text{/*}\textit{Orc = 'orçamento'}\text{ */} \\ SubOrc & \cong & Orc \;\; \text{/* }\textit{SubOrc = sub-orçamento}\text{ */} \end{array}$$

onde $Item$ é uma entidade atómica para identificação de ítens orçamentais.

1. Complete a seguinte especificação de um operador sobre Orc que calcula o valor total de um orçamento:

$$\begin{array}{rcl} total & : & Orc \longrightarrow I\!N_0 \\ total(\sigma) & \stackrel{\mathrm{def}}{=} & \ldots \end{array}$$

2. Pretendem-se ainda especificar as seguintes operações sobre Orc:

$$\textit{rúbricas} \;:\; Orc \longrightarrow 2^{(Item^*)} \tag{2.164}$$

/* rúbricas(σ)) *devolve o conjunto de todas rúbricas do orçamento associadas a custos, por exemplo,* <Carpintaria, Aberturas, Portas interiores>, $<Pavimentos>$, etc. */

$$cats \;:\; Orc \longrightarrow 2^{(Item^*)} \tag{2.165}$$

/* $cats(\sigma)$) *devolve o conjunto de todas as categorias orçamentais, por exemplo,* <Carpintaria, Aberturas> */

$$novaSubcat \;:\; Orc \times Item \times Item^* \longrightarrow Orc \tag{2.166}$$

/* $novaSubcat(\sigma, i, c)$) *cria* i *como subcategoria orçamental de c, por exemplo,* $i =$ Móveis de Cozinha *e* $c =$ <Carpintaria> */

Verifica-se que se pode prescindir da sua especificação se forem "reutilizadas" funções semelhantes já especificadas num problema que é "isomorfo" ao presente. Identifique esse problema e as funções em causa.

□

Exercício 2.63 A sintaxe que se segue pretende formalizar a informação contida num ficheiro de configuração de 'email' (tipicamente, o ficheiro `.mailrc` em UNIX):

$$\begin{array}{rcl} Mailrc & \cong & Alias \rightharpoonup 2^{(Email + Alias)} \\ Email & \cong & UsrName \times Domain \\ UsrName & \cong & Str \\ Domain & \cong & Str \\ Alias & \cong & Str \end{array}$$

Por exemplo, o endereço `jno@di.uminho.pt` é representado pelo elemento $\langle$`"jno"`, `"di.uminho.pt"`$\rangle$ de $Email$. $Mailrc$ exprime a estrutura de derivação de 'aliases' em endereços ou outros 'aliases'.

1. Especifique formalmente a função,

$$\begin{array}{rcl} expand & : & Mailrc \times 2^{Alias} \longrightarrow 2^{Email} \\ expand(\sigma, s) & \stackrel{\mathrm{def}}{=} & \ldots \end{array}$$

   que deverá calcular a expansão total, em endereços, de um dado conjunto de 'aliases'.

2. Complete os seguintes esboços de especificação formal de mais duas funções sobre $Mailrc$:

$$\begin{array}{rcl} whoGetsMe & : & Mailrc \times Email \longrightarrow 2^{Alias} \\ whoGetsMe(\sigma, e) & \stackrel{\mathrm{def}}{=} & \ldots \text{ /* calcula todos os 'aliases' que atingem um dado endereço */} \end{array}$$

$$\begin{array}{rcl} usrsByDomain & : & Mailrc \longrightarrow \ldots\ldots \\ usrsByDomain(\sigma) & \stackrel{\mathrm{def}}{=} & \ldots \text{ /* calcula o mapa que atribui, a cada domínio presente em } \sigma \text{, o respectivo conjunto de utilizadores conhecidos */} \end{array}$$

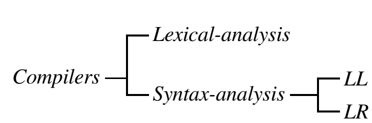
□

Exercício 2.64 Pretende-se especificar um sistema, encomendado por uma *biblioteca*, para arquivo de livros e sua classificação por *assuntos*. Os assuntos deverão estar organizados hierarquicamente numa *taxonomia*. A cada livro deve poder ser associado o conjunto dos assuntos pelos quais o livro pode ser pesquisado.

Suponha que uma equipa de projectistas chegou já ao seguinte modelo de dados para o sistema:

$$\begin{array}{rcl} System & \cong & Base \times Tax \\ Base & \cong & Key \rightharpoonup Book \times Subjects \\ Tax & \cong & Subject \rightharpoonup Tax \\ Key & \cong & \ldots \\ Book & \cong & \ldots \\ Subjects & \cong & 2^{Subject} \\ Subject & \cong & \ldots \end{array}$$

onde, por exemplo, a taxonomia



será modelada pela função finita, em Tax,

$$\left(\begin{array}{c} \textit{Compilers} \\ \left(\begin{array}{cc} \textit{Lexical-analysis} & \textit{Syntax-analysis} \\ (\ ) & \left(\begin{array}{cc} LL & LR \\ (\ ) & (\ ) \end{array} \right) \end{array} \right) \end{array} \right)$$

1. Construa a definição do invariante sobre Tax:
$$\begin{array}{rcl} \mathit{noSubRepeated} & : & Tax \longrightarrow 2 \\ \mathit{noSubRepeated}(t) & \stackrel{\mathrm{def}}{=} & \ldots \mathit{allSubjs}(\ldots)\ldots \end{array}$$
que deverá garantir que "nenhum assunto aparece repetido na taxonomia" e onde *allSubjs* é a função:
$$\begin{array}{rcl} \mathit{allSubjs} & : & Tax \longrightarrow Subjects \\ \mathit{allSubjs}(t) & \stackrel{\mathrm{def}}{=} & \mathit{dom}(t) \cup \bigcup_{t' \in rng(t)} allSubjs(t') \end{array}$$
que colecciona todos os assuntos presentes na taxonomia t.
2. Complete a seguinte especificação de uma função que dá todos os sub-assuntos de um dado assunto:
$$\begin{array}{rcl} \mathit{subSubs} & : & Subject \times Tax \longrightarrow Subjects \\ \mathit{subSubs}(a,t) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcl} t = (\quad) & \Rightarrow & \emptyset \\ t \neq (\quad) & \Rightarrow & \left\{ \begin{array}{lcl} a \in \mathit{dom}(t) & \Rightarrow & allSubjs(t(a)) \\ a \notin \mathit{dom}(t) & \Rightarrow & \ldots \end{array} \right. \end{array} \right. \end{array}$$
3. Complete a seguinte especificação de uma função que dá as cotas de todos os livros que versam determinado assunto (**NB**: se um livro versa o assunto x que é sub-assunto de y, então também versa y):
$$\begin{array}{rcl} allBooksBySubject & : & Subject \times System \longrightarrow 2^{Key} \\ allBooksBySubject(a,\sigma) & \stackrel{\mathrm{def}}{=} & \begin{array}{ll} let & db = \pi_1(\sigma) \\ & t = \pi_2(\sigma) \\ in & \ldots \end{array} \end{array}$$
4. Mostre (usando as leis de isomorfismo em $Sets$) que Tax é um caso particular de FS (2.94). **Sugestão**: repare que todo o sistema hierárquico de ficheiros *sem quaisquer ficheiros* "é uma taxonomia de directorias".

□

Exercício 2.65 Reconheça o padrão recursivo que obtém de (2.125) instanciando $A = 1$ e tire conclusões no contexto do exercício anterior.

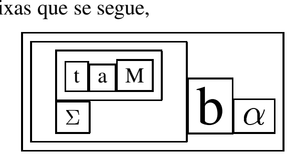
Que se especifica nesse padrão quando se força ainda $B = 1$?

□

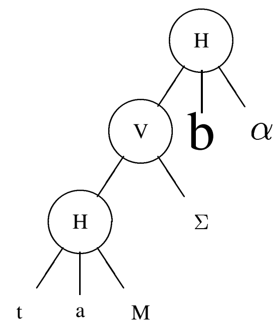
Exercício 2.66 O processamento de texto a nível da *tipografia electrónica* não trabalha ao nível da linha/página de texto mas sim com *caixas tipográficas*, que se compoem umas com as outras para obter efeitos gráficos arbitrariamente complexos. Por exemplo, o fragmento de texto que se segue,

$$\begin{array}{l}\text{taM}\\ \Sigma\end{array}\ \text{b}\alpha$$

corresponde à estruturação de caixas que se segue,



i.é ao seguinte padrão de composição:



onde “H” e “V” designam, respectivamente, composição *na horizontal* e composição *na vertical* [16].
Considere a seguinte especificação recursiva de *caixa tipográfica* (Box):

$$\begin{array}{rcl} Box & \cong & AtomicBox \quad + \\ & & VerticalBox \quad + \\ & & HorizontalBox \\ VerticalBox & \cong & Box^\star \\ HorizontalBox & \cong & Box^\star \\ AtomicBox & \cong & height : I\!N_0 \\ & & width : I\!N_0 \\ & & contents : \ldots \end{array}$$

em que se omitem os detalhes sobre o conteúdo de cada caixa elementar ($contents$).

1. Complete a especificação seguinte de uma função que calcula a altura de uma dada caixa:

$$\begin{array}{rcl} h & : & Box \longrightarrow I\!N_0 \\ h(b) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{rcl} \text{is-}AtomicBox(b) & \Rightarrow & height(b) \\ \text{is-}VerticalBox(b) & \Rightarrow & \ldots \\ \text{is-}HorizontalBox(b) & \Rightarrow & \ldots \end{array} \right. \end{array}$$

2. O raciocínio seguinte sobre Box,

$$\begin{array}{rcl} Box & \cong & AtomicBox + VerticalBox + HorizontalBox \\ & \cong & AtomicBox + Box^\star + Box^\star \\ & \cong & AtomicBox + 2 \times (Box^\star) \end{array}$$

permite-nos verificar que Box é, afinal, um caso particular de um tipo de dados abordado neste texto. Identifique-o, justificando.

3. Descreva o método de indução associado a Box que lhe permitirá provar predicados da forma

$$\forall b \in Box : \varphi(b)$$

sobre a estrutura Box.

[16] Assim, uma *página* de texto é uma caixa que se obtém por composição na vertical de tantas caixas quantas as suas linhas; por sua vez, uma *linha* de texto é uma caixa que se obtém por composição na horizontal de tantas caixas quantas os seus caracteres, etc.

□

Exercício 2.67 (Especificação de Redes Semânticas) Considere a descrição do conceito de *rede semântica hierárquica* que se segue:

- Uma *rede semântica hierárquica* é uma forma de base de conhecimento organizada numa rede cujos nodos são ocupados por *objectos*. Todo o objecto é identificado unìvocamente por um *identificador de objecto*.
- Um objecto pode ser a descrição de uma *classe* de "coisas" (*e.g.* as classes *Indivíduo*, *Solteiro*, *Edifício etc.*) ou a de uma sua *instância*. Uma *instância* é, pois, uma concretização de uma classe. Por exemplo, o *Complexo Pedagógico de Gualtar* é uma instância da classe *Edifício*. Notar, ainda, que uma instância pode concretizar mais do que uma classe. Por exemplo, o referido Complexo Pedagógico é não só instância da classe *Edifício*, mas também da classe *Objecto Inanimado*.
- Entre classes estabelece-se uma relação de *sub-/super-classe*. Por exemplo, *Edifício* é (simultaneamente) sub-classe das classes *Imóvel* e *Prédio Urbano*. Esta relação de *sub-classe* é transitiva e bem-fundada, e tem como limite universal superior a "maior" de todas as classes — o *Universo* — que é, portanto, a única classe que não é subclasse de nenhuma outra classe.
- A informação que a rede associa a uma dada classe contém não só a indicação das suas super-classes, mas também o conjunto de todos os seus *atributos* característicos. Por exemplo, *Nome*, *Data de Nascimento*, *Pai*, *Mãe etc.* são atributos que fazem sentido na classe *Indivíduo*. Uma classe tem acessíveis não só os atributos que nela se declaram, mas também todos aqueles que ela "herda" das suas super-classes.
- Uma instância concretiza valores para os atributos que lhe são conferidos pelas suas super-classes. Um valor de um atributo pode ser uma constante (*e.g.* a data "1990.04.05" é um valor possível para *Data*) ou uma referência a outro objecto. Por exemplo, o identificador do pai de um dado indivíduo pode ocorrer como valor do seu atributo *Pai*. É claro que este atributo é da classe *Indivíduo*.

Suponha que se pretende especificar a sintaxe (Σ) e um modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ para redes semânticas, tendo já sido identificadas as espécies

$$S = \left\{ \begin{array}{rl} \textit{RS} & \textit{/*rede semântica */} \\ \textit{Oid} & \textit{/*identificador de objecto */} \\ \textit{Oids} & \textit{/*plural de } \text{Oid */} \\ \textit{Objecto} & \textit{/*objecto */} \\ \textit{Classe} & \textit{/*classe */} \\ \textit{Instância} & \textit{/*instância */} \\ \textit{Aid} & \textit{/*identificador de atributo */} \\ \textit{Valores} & \textit{/*valores de atributos */} \\ \textit{Bool} & \textit{/*valores booleanos */} \end{array} \right.$$

e os operadores

$$\begin{array}{rcl} \textit{inicialização} & : & \to RS \\ \textit{Universo} & : & \to Oid \end{array}$$

1. Construa um modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ para as entidades sintáticas acima identificadas.
2. Acrescente a Σ os operadores ou espécies que, na sua opinião, faltam para descrever os requisitos acima (desenhe um diagrama-ADJ que os ilustre).

3. Provavelmente sentiu necessidade de introduzir um operador

$$\mathit{inv} : RS \to Bool$$

que pretende que seja *invariante* de RS. Identifique quais as cláusulas que introduziria em $\mathcal{A}(\mathit{inv})$, e justifique a sua necessidade.

4. Com base na alínea anterior, apresente a definição formal de $\mathcal{A}(\mathit{inv})$.

5. Defina operadores para manuseamento da informação contida em RS. Prove que a propriedade $\mathcal{A}(\mathit{inv})$ não é destruída por aqueles que têm RS como co-aridade.

□

Exercício 2.68 Muitos predicados sobre funções parciais finitas $\sigma \in A \rightharpoonup B$ são da forma

$$\forall i \in dom(\sigma) : \phi(\sigma(i)) \tag{2.167}$$

onde $\phi : B \to 2$ é um predicado sobre B.

Seja então $A \rightharpoonup \phi$ aceite como abreviatura da quantificação (2.167), para A finito. Mais formalmente, $A \rightharpoonup \phi$ é o predicado sobre $A \rightharpoonup B$ definido por:

$$A \rightharpoonup \phi \quad \stackrel{\text{def}}{=} \quad \lambda\sigma.\forall i \in dom(\sigma) : \phi(\sigma(i))$$

para ϕ um predicado válido sobre B.

Sendo a construção (2.167) rica em propriedades muito úteis em prova de invariantes, demonstre ou refute as seguintes, para $S \subseteq A$:

$$A \rightharpoonup \phi(\sigma_1 \cup \sigma_2) \quad \Leftrightarrow \quad (A \rightharpoonup \phi(\sigma_1)) \wedge (A \rightharpoonup \phi(\sigma_2)) \tag{2.168}$$

$$A \rightharpoonup \phi(\sigma) \quad \Rightarrow \quad A \rightharpoonup \phi(\sigma \setminus S) \tag{2.169}$$

$$(A \rightharpoonup \phi) \wedge (A \rightharpoonup \varphi) \quad \Leftrightarrow \quad A \rightharpoonup (\phi \wedge \varphi) \tag{2.170}$$

$$A \rightharpoonup (\neg\phi) \quad \Leftrightarrow \quad \neg(A \rightharpoonup \phi) \tag{2.171}$$

Sugestão: relembre o facto seguinte, para X finito:

$$\forall x \in X : p(x) \quad \Leftrightarrow \quad \bigwedge_{x \in X} p(x)$$

□

Exercício 2.69 Na sequência do exercício 2.68, demonstre a seguinte propriedade da construção em jogo:

$$(A \rightharpoonup \phi(\sigma_1)) \wedge (A \rightharpoonup \phi(\sigma_2)) \quad \Rightarrow \quad A \rightharpoonup \phi(\sigma_1 \dagger \sigma_2) \tag{2.172}$$

□

Exercício 2.70 $ActPlan$ é o nome de uma aplicação que se pretende especificar para escalonar tarefas e fazer a gestão de recursos necessários à sua execução. Cada tarefa ou actividade é identificada por um código ($\#Act$), ao qual se associa a sua descrição textual ($Description$), a sua duração ($Span$, medida em *e.g.* semanas ou dias), quais os recursos que envolve ($Resources$) e a indicação de quais as tarefas que deverão estar terminadas antes que a tarefa possa arrancar:

$$\begin{array}{rcll} ActPlan & \cong & \#Act \rightharpoonup Description \times Span \times Resources \times Dependences & \\ Description & \cong & STR & \text{/*descrição da actividade */} \\ Span & \cong & I\!N & \text{/*duração da actividade */} \\ Resources & \cong & \#Res \rightharpoonup I\!N & \text{/* tipo e quantidade de recursos necessários} \\ & & & \text{para realizar uma actividade */} \\ Dependences & \cong & 2^{\#Act} & \text{/*actividades precedentes */} \end{array}$$

Um escalonamento, definido por

$$Schedule \cong \#Act \rightharpoonup I\!N_0$$

é uma indicação, para cada tarefa, do instante (medido em *e.g.* semanas ou dias) em que está previsto a tarefa arrancar.

Especifique o operador

$$bestSchedule \quad : \quad ActPlan \rightarrow Schedule$$

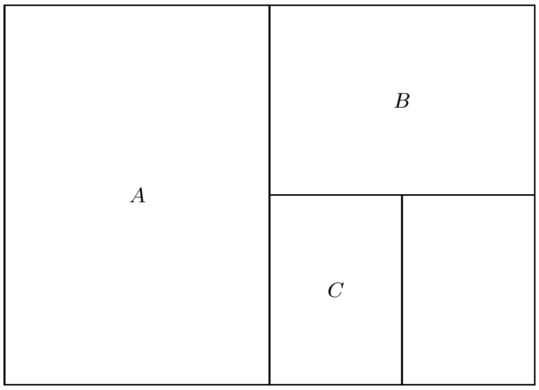
que deverá, consultando um mapa de actividades, construir o melhor escalonamento das suas actividades. Por *melhor* entende-se aquele em que as tarefas são escalonadas o mais cedo possível, sem violar quaisquer dependências entre si.

Sugestão: Se vir nisso conveniência, acrescente um invariante à estrutura $ActPlan$. Pode definir funções auxiliares.

□

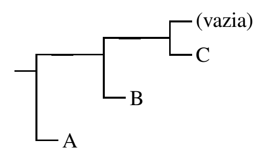
Exercício 2.71 Pretende-se que o transporte de embalagens num supermercado se faça em paletes previamente arrumadas por um robot. Este deverá usar um algoritmo de arrumação por partição binária da base da palete ($100cm \times 141cm$). Para optimizar a arrumação, tanto as embalagens como as paletes são normalizadas, isto é, as suas bases são rectângulos cujo comprimento é $\sqrt{2}$ maior que a largura [17].

A figura que se segue mostra a arrumação de três caixas A, B e C numa palete, estando ainda disponível um espaço de arrumação que poderá receber uma embalagem com as dimensões de C ou ser dividido em dois sub-espaços para arrumar duas embalagens mais pequenas e assim sucessivamente:



[17] Trata-se de um requisito que, além de útil para a partição binária do espaço, permite identificar as dimensões de uma embalagem apenas pela largura. Para simplificar, não nos procuparemos com a altura de embalagens e paletes.

ou seja, temos a seguinte "árvore de arrumação":



A partição binária divide sempre um espaço disponível ao meio, segundo o comprimento.

Suponha que, para especificar este problema, se definiram já as seguintes estruturas de informação:

$$
\begin{array}{rcll}
Space & \cong & 1 + Box + Space^2 & \text{/* um espaço na palete ou está vazio, ou contém uma caixa ou está dividido em 2 sub-espaços */} \\
Box & \cong & Id \times Width & \text{/* identificação da embalagem e sua largura de base */} \\
Width & \cong & I\!N_0 &
\end{array}
$$

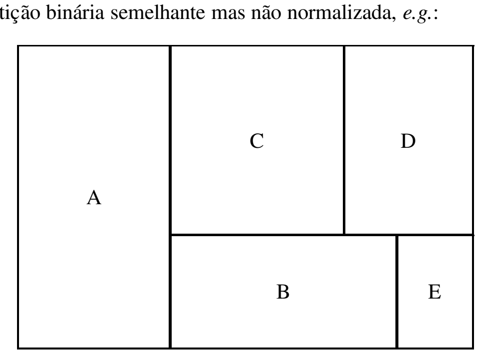
1. Mostre que a definição de $Space$ é polinomialmente recursiva e construa o respectivo esquema algorítmico.

2. Especifique os seguintes operadores sobre $Space$, onde $Ids = 2^{Id}$:

$$
\begin{array}{rcll}
whichBoxes & : & Space \to Ids & \text{/* deverá dar como resultado todos os identificadores das embalagens já colocadas na palete */} \\
freeSpace & : & Space \to Width & \text{/* deverá dar como resultado a largura do maior (sub-)espaço ainda livre na palete */}
\end{array}
$$

3. Acha que o robot optimizará a arrumação da palete inserindo embalagens por ordem crescente de largura? Ou decrescente? Justifique informalmente.

4. Generalize esta especificação de estrutura de dados a paletes de dimensões $l \times h$ quaisquer, sujeitas a uma partição binária semelhante mas não normalizada, *e.g.*:



explicando as suas alterações. Defina ainda o respectivo esquema de indução polinomial.

□

Exercício 2.72 Numa linguagem de especificação formal de edifícios, entende-se por divisão ('room') o seguinte:

$$Room \cong \mathbb{N}_0 \times \mathbb{N}_0 \times (WallO \rightharpoonup Open)$$

Assim, para além da largura e do comprimento da divisão, especificam-se as suas aberturas (portas e janelas) da forma seguinte:

$$\begin{array}{rcl} WallO & \cong & Bw + Tw + Lw + Rw \\ Bw & \cong & \mathbb{N}_0 \\ Tw & \cong & \mathbb{N}_0 \\ Lw & \cong & \mathbb{N}_0 \\ Rw & \cong & \mathbb{N}_0 \end{array}$$

tal que $o \in WallO$ designa o 'offset' de uma abertura na respectiva parede (Tw= '*top* wall', Bw= '*bottom* wall', Lw= '*left* wall' e Rw= '*right* wall)', contado a partir da origem que é o canto inferior esquerdo do rectângulo que a divisão define, e

$$\begin{array}{rcl} Open & \cong & Wind + Door \\ Wind & \cong & \mathbb{N} \\ Door & \cong & \mathbb{N} \end{array}$$

fixa a largura de uma abertura.

Especifique um invariante inv-$Room$ capaz de garantir que:

- em nenhuma parede duas ou mais aberturas se sobrepõem;
- nenhuma abertura de nenhuma parede se "rasga" para além da extensão dessa parede.

□

Exercício 2.73 No contexto do exercício 2.71, suponha que o algoritmo de arrumação por partição binária da base da palete é feito segundo duas direcções: partição na horizontal ou na vertical. Cada partição subdivide a correspondente área rectangular em dois sub-rectângulos, a partir da indicação de um 'offset', *i.é*, de uma distância medida a partir da esquerda num "corte" vertical, ou medida a partir da base num "corte" horizontal.

Para isso adoptou-se a seguinte especificação da estrutura de dados que suporta a descrição do conteúdo de uma palete (para simplificar, ignora-se a dimensão em altura):

$$Palete \cong Height \times Width \times Partition$$ /* *uma palete tem um dado comprimento, uma dada largura, registando-se de seguida o seu conteúdo* */

$$Partition \cong 1 + Box + (VOffset + HOffset) \times Partition^2$$ /* *se uma partição não está vazia, ou contém uma caixa ou é de novo partida binariamente na vertical ou na horizontal* */

$$
\begin{array}{rcll}
Box & \cong & Id \times Height \times Width & \text{/* identificação da embalagem e suas dimensões */} \\
VOffset & \cong & I\!N_0 & \\
HOffset & \cong & I\!N_0 & \\
Width & \cong & I\!N_0 & \\
Height & \cong & I\!N_0 &
\end{array}
$$

Especifique um predicado

$$
\begin{array}{rcl}
\text{inv-}Palete & : & Palete \longrightarrow 2 \\
\text{inv-}Palete(\langle h, w, p \rangle) & \stackrel{\rm def}{=} & \ldots
\end{array}
$$

capaz de garantir as seguintes propriedades que se pretendem invariantes:

a) Todo o 'offset' é válido, *i.é*, não ultrapassa as dimensões do rectângulo que está a ser partido.

b) Toda a caixa cabe na partição em que foi colocada.

□

Exercício 2.74 Codifique em PASCAL ou C (à sua escolha) a estrutura de dados $Palete$ que é assunto do exercício 2.73, com base na analogia do quadro da figura 1.5 entre as construções básicas da notação SETS e correspondentes construções sintáticas dessas linguagens de programação. Sugere-se que o cálculo de isomorfismo em SETS seja utilizado previamente na simplificação dessa estrutura.
□

Exercício 2.75 É vulgar em documentos científicos (monografias, livros, artigos, *etc.*) existir um apêndice de bibliografia contendo a descrição de todos os documentos que foram citados ao longo do texto principal.

Um índice remissivo de autores é mais um apêndice ao texto principal indicando, por ordem alfabética de nomes de autores referidos, para cada nome de autor a lista ordenada das páginas em que vem citadas publicações suas, por exemplo:

ARBIB — 10,11

GOGUEN — 28

HOROWITZ — 2,3,15,16,19

JONES — 3,7,28

JOURDAN — 11,12,29

MANES — 10,11

SAHNI — 2,3,15,16,19

SPIVEY — 3,7

WIRTH — 2,3

No ambiente LaTeXpara preparação de texto a geração de bibliografias pode ser feita automaticamente a partir de uma base de dados bibliográfica que associa a cada documento uma *chave de citação* que o identifica univocamente. Já a geração de índices remissivos de autores não está prevista na instalação LaTeXde base.

Pretende-se especificar formalmente o processo de geração de um tal índice, que deverá ter a estrutura:

$$Index \cong (Author \times Page^{\star})^{\star}$$

e ser gerado a partir de duas estruturas de informação que podem ser obtidas em LaTeXsem dificuldade:

$$Key \rightharpoonup Author^{*}$$

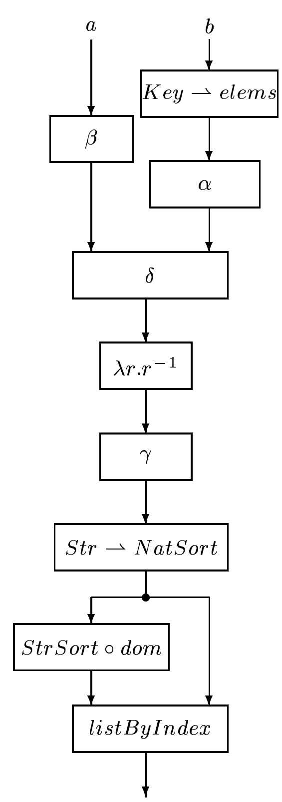
— indicando, para cada chave de citação, a lista dos seus autores — e

$$2^{Page \times 2^{Key}}$$

— indicando páginas e conjuntos de chaves de citação que ocorrem nessas páginas. Tem-se ainda $Page \cong \mathbb{N}$ e $Author \cong Str$, onde Str ('string') é uma espécie primitiva.

1. Complete (justificando) o seguinte diagrama que especifica a função $mkAuthorIndex$,

$mkAuthorIndex(a, b) \quad \stackrel{\mathrm{def}}{=}$



preenchendo α, β, δ e γ com funções das álgebras $2^{A\times B}$ e $A \rightharpoonup B$, adequadamente seleccionadas do repertório seguinte:

$$
\begin{array}{rcl}
merge & : & 2^{A\times 2^B} \longrightarrow 2^{A\times B} \\
merge(r) & \stackrel{\mathrm{def}}{=} & \bigcup_{p\in r} \{\langle \pi_1(p), b\rangle \mid\ b \in \pi_2(p)\} \\
 & & \\
discollect & : & A \rightharpoonup 2^B \longrightarrow 2^{A\times B} \\
discollect(\sigma) & \stackrel{\mathrm{def}}{=} & \bigcup_{a\in dom(\sigma)} \{\langle a, b\rangle \mid\ b \in \sigma(a)\} \\
 & & \\
collect & : & 2^{A\times B} \longrightarrow A \rightharpoonup 2^B \\
collect(r) & \stackrel{\mathrm{def}}{=} & \left(\begin{array}{c} a \\ \{\pi_2(q) \mid\ q \in r \wedge a = \pi_1(q)\} \end{array} \right)_{a\in 2^{\pi_1(r)}}
\end{array}
$$

$$
\begin{array}{rcl}
\circ & : & 2^{A\times B} \times 2^{B\times C} \longrightarrow 2^{A\times C} \\
r \circ s & \stackrel{\mathrm{def}}{=} & \{\langle \pi_1(p), \pi_2(q)\rangle \mid\ p \in r \wedge q \in s \wedge \pi_2(p) = \pi_1(q)\}
\end{array}
$$

2. Sabendo que $NatSort$ e $StrSort$ são óbvias funções de ordenação (infira a respectiva assinatura), especifique a função genérica $listByIndex$ que deverá encarar a sequência que lhe é passada como primeiro argumento como índice para listar a função finita que é o seu outro argumento. (**NB:** atente que o resultado da função é da espécie $Index$.)

3. Formule e demonstre as seguintes propriedades (axiomas) de $mkAuthorIndex$:

   (a) Se a base de dados bibliográfica fôr vazia o índice de autores será também necessariamente vazio.

   (b) Citar uma chave de citação que não conste da base de dados bibliográfica nada acrescenta ao índice gerado.

□

Exercício 2.76 Recorde do Exercício 2.75 a função $listByIndex$.

1. Demonstre ou refute a seguinte igualdade sobre essa função, que se supõe universalmente quantificada nas variáveis livres:

$$listByIndex(l, \left(\quad\right)) \quad = \quad listByIndex(<>, \sigma)$$

   (**Sugestão**: evite recorrer a métodos de prova indutiva.)

2. Poderá ser $listByIndex$ uma função sobrejectiva? Justifique.

3. Suponha que

$$Students \quad \cong \quad Nr \rightharpoonup Name \times Course$$

   é a estrutura de informação que regista, para cada número de aluno desta disciplina, o seu nome e o curso a que pertence (suponha $Nr \cong I\!N$, $Name \cong Str$ e $Course \cong Str$). Suponha ainda que

$$Marks \quad \cong \quad Nr \rightharpoonup Mark$$

é a estrutura de informação que regista, para cada número de aluno que entregou o exame da 1.ª chamada, a respectiva classificação (onde $Mark \cong \mathbb{N}$, por exemplo a nota 10.23 é representada pelo natural 1023).

Especifique formalmente a função

$$\begin{array}{rcl} listStudents & : & Students \times Marks \longrightarrow (Nr \times Name \times Course \times Mark)^\star \\ listStudents(\sigma, \mu) & \stackrel{\rm def}{=} & \ldots \end{array}$$

que deverá gerar a pauta do exame da 1.ª chamada, ordenada por ordem crescente de números de aluno, onde cada linha da pauta indica, para cada aluno, o seu número, nome, curso e nota. (**Sugestão**: recorra a $listByIndex$ e a outras funções auxiliares que conhece para resolver esta alínea.)

4. Repita a alínea anterior por forma a $listStudents$ gerar a pauta ordenada alfabeticamente por nomes de alunos. (**Sugestão**: a mesma da alínea anterior.)

□

Exercício 2.77 No contexto do Exercício 2.75:

1. Especifique a função

$$\begin{array}{rcl} NatSort & : & 2^{\mathbb{N}} \longrightarrow \mathbb{N}^\star \\ NatSort(s) & \stackrel{\rm def}{=} & \ldots \end{array}$$

que aí se refere e que deverá listar, por ordem crescente, o conjunto s que lhe é passado como argumento.

2. Verifique (*e.g.* por indução) se a especificação de $NatSort$ que acaba de propôr satisfaz a seguinte propriedade:

$$\forall s \in 2^{\mathbb{N}} : elems(NatSort(s)) = s$$

3. Escreva um predicado que formalize a seguinte propriedade que $mkAuthorIndex$ deverá satizfazer:

> *A repetição de uma mesma chave de citação dentro da mesma página de texto não altera o índice gerado.*

Acha que o modelo proposto para $mkAuthorIndex$ no Exercício 2.75 vai satisfazer esta propriedade? Responda informal e sumariamente.

□

Exercício 2.78 Considere a seguinte especificação de uma função em SETS baseada na função $subl$ do Exercício 2.53:

$$\begin{array}{rcll} split(l, n) & \stackrel{\rm def}{=} & let & c = length(l) \\ & & & i = div(c, n) \\ & & & m = \left\{ \begin{array}{rcl} rem(c,n) = 0 & \Rightarrow & i \\ \neg(rem(c,n) = 0) & \Rightarrow & i+1 \end{array} \right. \\ & & in & <subl(l, (j-1) \times n + 1, n) \mid j \leftarrow inseq(m)> \end{array}$$

onde div e rem são as conhecidas funções de aritmética inteira e $inseq$ é a função

$$inseq(i) \stackrel{\rm def}{=} \left\{ \begin{array}{rcl} i = 0 & \Rightarrow & < > \\ \neg(i = 0) & \Rightarrow & inseq(n-1) \frown <i> \end{array} \right.$$

1. Após calcular $split(<2,5,4>,2)$, tipifique $split$, *i.é* preencha as reticências em

$$split \quad : \quad \cdots \longrightarrow \cdots$$

2. Explique por palavras suas o significado das funções $inseq$ e $split$.

3. Mostre (*e.g.* por apresentação de um contra-exemplo) que o seguinte facto não se verifica:, em geral:

$$elems(length^{\star}(split(l,n))) \quad = \quad \{n\}$$

□

Exercício 2.79 Seja $Term$ a espécie de dados "termo com variáveis" que a seguir se define:

$$\begin{array}{rcl} Term & \cong & Var + Exp \\ Exp & \cong & Op \times Term^{*} \end{array}$$

onde as espécies Var (símbolo de variável) e Op (símbolo de operador) são atómicas. Complete a definição seguinte que testa se dois termos t e t' em $Term$ são *unificáveis*, no sentido da inferência por *resolução* usada em PROLOG:

$$unif(t,t') \quad \stackrel{\rm def}{=} \quad \left\{ \begin{array}{lcl} \text{is-}Var(t) \vee \text{is-}Var(t') & \Rightarrow & V \\ \text{is-}Exp(t) \wedge \text{is-}Exp(t') & \Rightarrow & \ldots \end{array} \right.$$

NB: $unif$ deverá seguir o famoso *algoritmo de unificação* de Robinson (1965), segundo o qual dois termos são unificáveis se e só se

- um deles for uma variável

- ambos forem expressões com o mesmo operador (e a mesma aridade, portanto) cujos subtermos sejam, por sua vez, unificáveis dois a dois.

□

Exercício 2.80 CAMILA é uma linguagem (interpretada) destinada à prototipagem rápida de especificações formais previamente formuladas em SETS. Em CAMILA é possível definir módulos (ficheiros com a extensão `.cam`) que incluem, através de uma cláusula `#include` semelhante à do C, outros módulos CAMILA. Por exemplo, um módulo `mset.cam` (para manipulação de multiconjuntos) pode ter que invocar, para poder correr, o módulo `ffun.cam` (para manipulação de funções finitas).

A 'shell' de interpretação de módulos CAMILA é um interpretador de LISP evoluído, que se designa por `xmetoo`. A interpretação de um qualquer módulo `x.cam` pressupõe a sua compilação prévia para `xmetoo`, originando o ficheiro `x.met`. Sempre que tal compilação encontra uma cláusula da forma `#include y.cam`, processa-se `y.cam`, gerando-se `y.met`, e assim sucessivamente.

É fácil de ver que a interdependência repetitiva entre módulos CAMILA torna a interpretação desnecessariamente lenta (um mesmo módulo pode ser compilado/carregado mais do que uma vez) e a sua interdependência cíclica (que, em teoria, é normal) pode ter consequências perfeitamente indesejáveis.

Pretendendo-se uma pequena aplicação que racionalize este processo de inclusão, suponha que dispõe já de uma pequena aplicação que inspecciona o sistema de ficheiros e dele infere uma tabela cuja estrutura formal é a seguinte:

$$\begin{array}{rcl} CamilaFiles & \cong & 2^{Name \times Date \times (Met + Cam)} \\ Met & \cong & 1 \\ Cam & \cong & 2^{Name} \\ Name & \cong & Str \\ Date & \cong & \mathbb{N} \end{array}$$

Esta tabela indica os nomes dos ficheiros `.met` ou `.cam` que se encontraram, a data da sua última alteração e, no caso de um ficheiro `.cam`, os nomes dos ficheiros `.cam` que aquele inclui.

1. Especifique formalmente a função,

$$\begin{array}{rcl} traInclude & : & CamilaFiles \times 2^{Name} \longrightarrow 2^{Name} \\ traInclude(\sigma, s) & \stackrel{\rm def}{=} & \ldots \end{array}$$

   que deverá calcular os nomes de todos os ficheiros `.cam` que devem ser (transitivamente) carregados quando se pretende carregar aqueles cujos nomes estão em S. (**Importante**: $traInclude$ deve conseguir tratar inclusão cíclica entre ficheiros.)

2. Sempre que a data de um ficheiro `.met` é posterior a data do correspondente `.cam`, torna-se desnecessária a compilação prévia deste. Assumindo disponível a função $traInclude$, especifique uma função capaz de separar os módulos que têm de ser compilados e carregados daqueles para os quais basta carregar o correspodente código `.met`, por este estar actualizado.

3. Suponha que em lugar da estrutura de informação $CamilaFiles$ que acima se caracterizou, alguém propõe a seguinte estrutura formal para registar a mesma informação:

$$CamilaFiles' \cong 2^{Date \times Name} \times 2^{Name \times Date \times 2^{Name}}$$

   Será $CamilaFiles'$ isomorfa a $CamilaFiles$? Justifique formalmente.

□

Exercício 2.81 Considere o seguinte fragmento da especificação formal de uma *folha de cálculo* (muito simplificada):

$$\begin{array}{rcll} FCALC & \cong & Coord \rightharpoonup Cell & \\ Coord & \cong & L : \mathbb{N} \;\times\; C : \mathbb{N} & \text{/*L-linha; C-coluna */} \\ Cell & \cong & \mathbb{Z} + Coord & \\ Range & \cong & UL : Coord \;\times\; LR : Coord & \text{/* sub-área rectangular da folha, definida por duas coordenadas: os seus cantos superior esquerdo e inferior direito, respectivamente */} \end{array}$$

1. Especifique as funções seguintes sobre $FCALC$, seguindo a informação dada em comentário:

$$
\begin{array}{rcl}
init & : & \longrightarrow FCALC \\
init & \stackrel{\mathrm{def}}{=} & \ldots \textit{ /*inicialização da folha de cálculo */} \\
 & & \\
wrCell & : & Coord \times Cell \times FCALC \longrightarrow FCALC \\
wrCell(xy, c, \sigma) & \stackrel{\mathrm{def}}{=} & \ldots \textit{ /* na folha } \sigma \textit{ inscrever o valor c na célula cujas coordenadas são dadas} \\
 & & \quad \textit{por } xy \textit{ */} \\
ranMove & : & Range \times Coord \times FCALC \longrightarrow FCALC \\
ranMove(r, xy, \sigma) & \stackrel{\mathrm{def}}{=} & \ldots \textit{ /* na folha } \sigma \textit{ mover a área } r \textit{ para uma nova área cuja coordenada} \\
 & & \quad \textit{superior esquerda é dada por c */}
\end{array}
$$

2. Se, em lugar de $ranMove$, tiver que especificar a operação $ranCopy$ que, em lugar de mover, **copia** uma área de um local para outro, que alterações teria que fazer à sua especificação de $ranCopy$? Responda informalmente.
3. Especifique um invariante sobre $FCALC$ que garanta que a folha não tem células que se referem a células indefinidas.
4. Verifique se $wrCell$ satisfaz o invariante que definiu na alínea anterior. Na negativa, re-especifique essa função e esquematize o respectivo argumento de preservação.

□

2.7 Notas Bibliográficas

A construção de modelos matemáticos do mundo físico tem longas tradições na Física e nas Ciências da Engenharia. A especificação formal de 'software' orientada a modelos (eventualmente recursivos, como se viu neste capítulo) segue essa tradição e tem sido objecto de intensa investigação nas duas últimas décadas, tendo sido já propostas algumas notações padrão (vulg. *linguagens de especificação*) como META-IV [18] [Jon80, BJ82, Jon86] e Z [Spi89].

No capítulo que aqui termina procurou-se sistematizar, na concisa "linguagem categorial" de $Sets$, as técnicas básicas associadas à modelação da realidade usando domínios recursivos. Para um estudo aprofundado de alguns resultados abreviados nesta sistematização ver [MA86], referência a não deixar de consultar no tocante à aplicação da teoria das categorias à ciência da programação.

Os livros de Manna [Man74], Stoy [Sto81] e Bakker [Bak80] são referências padrão para o estudo da teoria "topológica" da computação recursiva, originária dos trabalhos de Scott & Strachey.

Ao organizar a concepção de modelos em torno de unidades estruturais,

[18] Vulg. VDM, de 'Vienna Development Method'.

uma construção de $Sets$
+
um esquema funcional de 'browsing' ('kit')
+
um esquema indutivo como método de prova

a abordagem que se apresentou procura transpor para o plano da especificação formal algumas das principais intuições da *Programação Estruturada* dos anos setenta. Em particular, a de que as estruturas de dados ocupam um papel principal na concepção dos programas, como se pode sentir ao longo da leitura do conhecido e paradigmático livro de Niklaus Wirth (*cf. Algorithms + Data Structures = Programs* [Wir76], p.xii e p.169):

> "An outstanding contribution to bring order into the bewildering variety of terminology and concepts on data structures was made by Hoare through his "Notes on Data Structuring" [19]. It made clear that (...) the structure and choice of algorithms often strongly depend on the structure of the underlying data. (...)
>
> Yet, this book starts with a chapter on data structure for two reasons. First, one has the intuitive feeling that data precede algorithms: you must have some objects before you can perform operations on them. Second, (...) this book assumes that the reader is familiar with the basic notions of computing programming.
>
> (...) Indeed, if the analogy between program structures and data structures is to be extended, the purely recursive data structure could well be placed at the level corresponding with the procedure, whereas the introduction of pointers is comparable to the use of **goto** statements. (...) The parallel development of corresponding program and data structures is shown in concise form in Table 4.1."

Construction Pattern	Program Statement	Data Type
Atomic element	Assignment	Scalar type
Enumeration	Compound statement	Record type
Repetition by a known factor	For statement	Array type
Choice	Conditional statement	Variant record, type union
Repetition by an unknown factor	While or repeat statement	Sequence or file type
Recursion	Procedure statement	Recursive data type
General "graph"	Go to statement	Structure linked by pointers

3 **Table 4.1** Correspondences of Program and Data Structures.

[19] *Structuring Programming*, pp.83-174.

Capítulo 3

Semântica Axiomática

3.1 Introdução

Pretendemos estudar neste capítulo alternativas à técnica de definição da semântica de uma determinada linguagem por apresentação de modelos (semânticos), que estudamos nos capítulos anteriores.

Vejamos primeiro qual a motivação para tal estudo. Suponhamos definida uma assinatura $\Sigma : \Omega \rightarrow S^\star \times S$ e um seu modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$, tal como anteriormente. Reparemos que faz sentido "testarmos" $\mathcal{A}$ no sentido de se inspeccionar se este modelo "satisfaz" determinada propriedade que, porventura, nos ocorreu caracterizar. Por exemplo, o texto seguinte regista uma propriedade concreta sobre o problema $SGIB$ (relembrar Σ_{SGIB} do Capítulo 1 e o Σ_{SGIB}-modelo da secção 1.3.2):

> "Fica tudo na mesma se levantarmos, de uma conta, exactamente a mesma quantia que nela acabamos de depositar".

Esta propriedade pode ser formalizada em termos da linguagem gerada a partir Σ_{SGIB}, escrevendo:

$$\left.\begin{array}{rcl} s' & = & \mathit{depósito}(ic, q, s) \\ s'' & = & \mathit{levantamento}(ic, q, s') \end{array}\right\} \Rightarrow s' = s''$$

para qualquer quantia q, número de conta ic e configurações s, s', s'' do sistema bancário. Simplificando, obtemos

$$levantamento(ic, q, \mathit{depósito}(ic, q, s)) = s \tag{3.1}$$

Outra propriedade poderia ser

$$\mathit{balanço}(ic, s) \geq \mathit{qmínima} \tag{3.2}$$

onde a constante *qmínima* $:\to Quantia$ designa a quantia mínima que deve estar depositada numa qualquer conta ic de um tema bancário s. Mais ainda, algo estará errado no modelo que se propôs na secção 1.3.2 se a seguinte propriedade se não verificar:

$$\textit{balanço}(ic, \textit{depósito}(ic, q, s)) = q + \textit{balanço}(ic, s) \tag{3.3}$$

onde $+ : Quantia \times Quantia \to Quantia$ designa a adição de quantias no sistema. Esta propriedade exprime o desejado efeito do operador de *depósito* — o aumento em q unidades monetárias do balanço que a conta em questão exibia antes desse depósito.

Dos exemplos acima — (3.1, 3.2 e 3.3) — infere-se que as propriedades de uma especificação $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ se podem exprimir, matematicamente, sob a forma de relações entre pares de Σ-termos: $t = t'$, $t \leq t'$ *etc.* Adiante-se já que t e t' não são exactamente Σ-termos, mas sim uma extensão sua, na medida em que neles ocorrem símbolos especiais — as *variáveis* (*cf.* ic, s, q *etc.*) — como "abreviaturas" de quaisquer sub-termos que se possam porventura encaixar na correspondente posição do termo em que ocorrem.

A formalização do conceito de termo com variáveis virá mais à frente. Para já, o que interessa reter é a ideia de que a própria Σ-linguagem (W_Σ) parece ser suficiente para exprimir propriedades válidas sobre um dado modelo $\mathcal{A}$. A verificação de validade (ou não) de uma uma determinada propriedade $t = t'$ em $\mathcal{A}$ pode, então, materializar-se no cálculo de

$$\mathcal{A}(t) = \mathcal{A}(t')$$

Se $\mathcal{A}(t)$ tiver o mesmo valor que $\mathcal{A}(t')$, então a propriedade $t = t'$ *verifica-se* em $\mathcal{A}$; caso contrário, não se verifica. É claro que o cálculo de $\mathcal{A}(t)$, que abrevia a interpretação $h_{\mathcal{A}}(t)$, *cf.* Teorema 1.1, tem que contar agora com o cálculo de valores de variáveis, como veremos [1].

Reparemos que, para modelos elaborados e extensos, é muito útil registar as suas propriedades mais significativas. Essas propriedades podem ser suficientes para raciocinar sobre cada modelo, evitando assim o explícito processo de cálculo de frases da Σ-linguagem.

Ora é esta a principal motivação para a técnica de especificação dita *axiomática*: se é (ou, pelo menos, parece ser) possível registar o conhecimento sobre um dado modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ sob a forma de uma colecção de Σ-propriedades (axiomas), para quê preocuparmo-nos com $\mathcal{A}$? Não bastará, ignorando qualquer modelo, limitar a especificação da semântica de Σ ao registo cuidadoso das suas propriedades consideradas *nucleares*? É esta técnica que passaremos a estudar, com rigor, neste capítulo.

[1] Ver mais à frente a Definição 3.1.

3.2 Especificação Axiomática

A definição que se segue é uma extensão da Definição 1.4.

Definição 3.1 (Σ-termo com variáveis) *Seja* $\Sigma : \Omega \to S^\star \times S$ *uma assinatura e* V *um conjunto de* símbolos de variáveis *tal que* $V \cap \Omega = \emptyset$*. Seja ainda dada uma aplicação*

$$X : V \to S$$

que associa a cada símbolo de variável o seu "tipo" (espécie em S*). Então é possível construir a assinatura*

$$\Sigma(X) = \Sigma \cup \left(\begin{array}{c} x \\ (<>, X(x)) \end{array} \right)_{x \in V}$$

A qualquer termo $t \in W_{\Sigma(X)}$ *daremos o nome de* Σ-termo com variáveis em X, *ou* $\Sigma(X)$-termo*. A álgebra inicial de todos os* $\Sigma(X)$*-termos designar-se-á por* $\mathcal{W}(X)$*,* cf. *Definição 1.9.* $\square$

Infere-se desta definição que as *variáveis* têm o estatuto de constantes "especiais", cujo significado vai tornar-se claro a seguir.

Definição 3.2 (Instanciação de Variáveis) *Seja* $\Sigma(X)$ *a assinatura construída sobre* $\Sigma : \Omega \to S^\star \times S$ *e* $X : V \to S$*, tal como foi descrito acima. Seja* $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ *um* Σ*-modelo. Chama-se* instanciação *das variáveis de* X *em* $\mathcal{A}$ *a qualquer aplicação* ρ

$$\rho : V \to |\mathcal{A}|$$

— onde $|\mathcal{A}|$ *designa*

$$|\mathcal{A}| = \bigcup_{s \in S} \mathcal{A}(s)$$

— tal que

$$\rho(x) \in \mathcal{A}(X(x))$$

O conjunto de todas as aplicações ρ *nestas circunstâncias será designado por* $V_{\mathcal{A}}$
$\square$

Uma constante é, pois, um símbolo com "valor variável", valor esse determinado por uma dada instanciação $\rho \in V_{\mathcal{A}}$ para um determinado Σ-modelo $\mathcal{A}$. Estamos agora em condições de poder estender a Definição 1.11 no sentido de admitir termos com variáveis.

Definição 3.3 ($\Sigma(X)$-interpretação) *Seja $\Sigma(X)$ a assinatura construída como nas definições anteriores, e seja* $\mathsf{B} : \Sigma \longrightarrow \mathbf{Set}$ *um Σ-modelo. Seja $\rho \in V_{\mathcal{B}}$ uma instanciação de X-variáveis em $\mathcal{B}$.*

A $\Sigma(X)$-interpretação induzida por ρ, designada por $h^{\rho}_{\mathcal{B}}$ ou, simplesmente, por $\mathcal{B}^{\rho}$, é o $\Sigma(X)$-homomorfismo

$$h^{\rho}_{\mathcal{B}} = (h^{\rho}_{\mathcal{B},s})_{s \in S}$$

definido por

$$\begin{array}{lcl} h^{\rho}_{\mathcal{B},s} & : & W_{\Sigma(X),s} \to \mathcal{B}(s) \\ h^{\rho}_{\mathcal{B},s}(t) & \stackrel{\text{def}}{=} & \begin{cases} \rho(t) & \Leftarrow \quad t \in V \\ \mathcal{B}(\sigma) & \Leftarrow \quad \sigma \in \Omega \wedge \pi_1(\Sigma(\sigma)) = <> \\ \mathcal{B}(\sigma)(h^{\rho}_{\mathcal{B},s_1}(t_1), \ldots, h^{\rho}_{\mathcal{B},s_n}(t_n)) & \Leftarrow \quad t = \sigma(t_1, \ldots, t_n) \wedge \\ & \qquad \forall 1 \leq i \leq n : t_i \in W_{\Sigma(X),s_i} \end{cases} \end{array}$$

$\square$

Note-se que uma $\Sigma(X)$-interpretação é exactamente a Σ-interpretação correspondente, excepto no que diz respeito a variáveis, cujo valor é calculado com base numa instanciação concreta. Assim, o Teorema 1.1 é ampliado como se segue.

Teorema 3.1 (Unicidade de $\Sigma(X)$-interpretações) *O homomorfismo $\mathcal{B}^{\rho}$ (*i.é *$h^{\rho}_{\mathcal{B}}$) definido na definição anterior é, para uma dada interpretação $\rho \in V_{\mathcal{B}}$, único.*

Demonstração*: É em tudo semelhante à do Teorema 1.1, desdobrando-se os casos de base no tratamento de constantes e variáveis. Como a instanciação de variáveis está fixada à partida, os homomorfismos h e h' também coincidem para eles.* $\square$

Definição 3.4 (Substituição) *No contexto da Definição 3.2, chama-se* substituição *a toda a instanciação ρ que faz corresponder a uma variável um termo com variáveis,* i.é $\rho \in V_{\mathcal{W}(X)}$. $\square$

O Teorema 3.1 permite-nos simplificar a notação, designando também por ρ a única interpretação $\mathcal{W}(X)^{\rho}$ que é induzida por uma dada substituição ρ. Chamaremos *interpretações de re-escrita* a esta classe de interpretações.

Vamos ilustrar os conceitos de *substituição* e *re-escrita*, que são muito importantes no contexto do que vai seguir-se. Sejam dados os termos com variáveis

$$\begin{array}{rcl} t & = & \mathit{depósito}(ic, q, s) \\ t' & = & \mathit{depósito}(ic, q', s') \end{array} \tag{3.4}$$

no contexto da assinatura $\Sigma_{SGIB}(X)$, de onde se inferem as atribuições

$$\begin{array}{lclcl} X(ic) & = & \textit{IdConta} & & \\ X(q) & = & X(q') & = & \textit{Quantia} \\ X(s) & = & X(s') & = & \textit{Sistema} \end{array}$$

Reparemos que é uma substituição a aplicação ρ seguinte:

$$\left\{ \begin{array}{lcl} \rho(ic) & = & ic \\ \rho(q) & = & q \\ \rho(q') & = & q' \\ \rho(s) & = & s \\ \rho(s') & = & \textit{depósito}(ic, q, s) \end{array} \right. \tag{3.5}$$

É possível, então, re-escrever qualquer termo de $\mathcal{W}(X)$ com base nesta substituição. Pegando, por exemplo, em t' acima (3.4), obteremos para $\rho(t')$

$$\rho(t') = \textit{depósito}(ic, q', \textit{depósito}(ic, q, s))$$

quer dizer, tudo se passou como se a variável s' fosse *substituída* pelo subtermo (de $\rho(t')$) $\textit{depósito}(ic, q, s)$.

Para exprimir mais comodamente processos de substituição (parcial) como este, é vulgar uma notação abreviada que indica apenas as variáveis que são substituídas, e os valores que as substituem; a notação é

$$[t'/x]t \tag{3.6}$$

querendo significar a interpretação (re-escrita) de t determinada pela substituição que coincide com a identidade excepto quanto à variável x, que é substituída por t'. É claro que (3.6) se generaliza facilmente para n-variáveis:

$$[t_1/x_1, \ldots, t_n/x_n]t$$

Assim, a substituição (3.5) poderá ser escrita, nesta notação, sob a forma de:

$$[\textit{depósito}(ic, q, s)/s']\textit{depósito}(ic, q', s')$$

Definição 3.5 (Alcance de um $\Sigma(X)$-termo) *Cada $\Sigma(X)$-termo t representa uma classe de Σ-termos, que são aqueles que se obtêm de t por substituição de variáveis suas por Σ-termos. Essa classe, designada* alcance de t*, representa-se por $[t]$ e é definida por*

$$[t] = \{\mathcal{W}^{\rho}(t) \mid \rho \in V_{\mathcal{W}}\}$$

□

A definição anterior é sugestiva porquanto nos permite utilizar $\Sigma(X)$-termos para representar objectos *genéricos*, *vagos* ou *abstractos*. Por exemplo, o termo $2k - 1$, na variável natural $k \in I\!N$, representa-nos a classe de todos os números *ímpares*, ou seja, o objecto genérico "número ímpar". Isto na medida em que o alcance $[2k - 1]$ é exactamente essa classe.

A caracterização de objectos abstractos, ou vagos, pode obter-se por um processo inverso da substituição, chamado *generalização* ou *abstracção*, e que consiste em substituir sub-termos por variáveis. Por exemplo, vamos supor dada uma assinatura Σ para descrição de objectos geométricos no plano cartesiano, envolvendo espécies como *Ponto*, *Raio*, *Círculo* e *Coord* e operadores como

$$\begin{array}{rcll} \mathit{ponto} & : & \mathit{Coord} \times \mathit{Coord} & \rightarrow & \mathit{Ponto} \\ \mathit{círculo} & : & \mathit{Ponto} \times \mathit{Raio} & \rightarrow & \mathit{Círculo} \end{array}$$

incluindo constantes como:

$$\begin{array}{rcl} \ldots, -1, 0, +1, +2, \ldots & :\rightarrow & \mathit{Coord} \\ 10, 20, 30, \ldots & :\rightarrow & \mathit{Raio} \end{array}$$

Assim, o termo

$$\mathit{círculo}(\mathit{ponto}(0,0), 10) \tag{3.7}$$

designa o círculo de raio 10 unidades, centrado na origem. Vamos agora generalizar (3.7). Se substituirmos a constante 10 por uma variável r tal que $X(r) =$ *Raio*, obtemos

$$\mathit{círculo}(\mathit{ponto}(0,0), r) \tag{3.8}$$

que é o $\Sigma(X)$-termo que nos representa qualquer círculo centrado na origem do plano cartesiano, *i.é* o objecto genérico "círculo centrado na origem". Se fizermos, para $X(y) =$ *Coord*, mais uma generalização sobre (3.8),

$$\mathit{círculo}(\mathit{ponto}(0,y), r)$$

temos agora a classe de "todos os círculos cujo centro se situa no eixo das abcissas". Finalmente, não é possível generalizar mais do que substituir a constante 0 por uma variável $X(x) =$ *Coord*, obtendo

$$\mathit{círculo}(\mathit{ponto}(x,y), r)$$

que, claramente, representa a classe de "todos os círculos", *i.é* o objecto abstracto "*círculo*" [2].

Exercício 3.1 Um caso particular de substituição $\rho \in V_{\mathcal{W}(X)}$ é aquela que substitui bijectivamente variáveis por variáveis, normalmente designada *re-nomeação*.

[2] Não é desinteressante relacionar formas puras como *círculo*, *esfera*, *cone etc.* com o conceito Platónico de *arquétipo*, *cf.* as **Notas Bibliográficas** deste capítulo.

Caracterize formalmente uma *re-nomeação* e mostre que não se altera o alcance de um termo quando este é sujeito a uma qualquer re-nomeação das suas variáveis.
□

Definição 3.6 (Σ-Axioma) *Sejam dados, tal como em definições anteriores,* $\Sigma : \Omega \to S^\star \times S$ *e* $X : V \to S$. *Um* $\Sigma(X)$-axioma *é um qualquer par*

$$(t, t') \in W_{\Sigma(X),s} \times W_{\Sigma(X),s}$$

para uma dada espécie $s \in S$.

O conjunto de todos os $\Sigma(X)$*-axiomas é, pois,*

$$A(\Sigma, X) \stackrel{\mathrm{def}}{=} \bigcup_{s \in S} (W_{\Sigma(X),s})^2 \qquad (3.9)$$

□

Reparemos que a definição de axioma nos garante, por construção, que ambos os termos envolvidos são do mesmo tipo, *i.é* os seus operadores mais externos (ou as suas variáveis externas) têm a mesma espécie de resultado. Por (contra) exemplo, os termos t da expressão (3.4) e o termo

$$t' = \mathit{qmínima}$$

não podem formar um axioma (t, t').

Finalmente, estamos em condições de poder definir a noção de *especificação axiomática*.

Definição 3.7 (Especificação Axiomática) *Seja dada uma assinatura* $\Sigma : \Omega \to S^\star \times S$ *e um conjunto de variáveis tipificadas por* $X : V \to S$. *Seja* $E = (E_s)_{s \in S}$ *uma família de* $\Sigma(X)$*-axiomas.*

Ao terno (Σ, X, E) *daremos o nome de* especificação axiomática. □

O que significa, então, apresentar uma especificação axiomática (Σ, X, E)? Nas secções seguintes veremos duas interpretações possíveis para esse significado.

Exercício 3.2 Considere uma interface axiomática $I = \langle \Sigma, X, E \rangle$ em que Σ admite os operadores

$$\begin{array}{ccl} + & : & nat \times nat \to nat \\ 1 & : & \to nat \end{array}$$

e E inclui o axioma

$$x + y \quad \equiv \quad y + x$$

Sejam $t = 1 + x$ e $t' = x + 1$ dois $\Sigma(X)$-termos. Indique, justificando formalmente, quais das seguintes afirmações estão correctas:

$$\text{– Os alcances } [t] \text{ e } [t'] \text{ são conjuntos idênticos.} \tag{3.10}$$

$$\text{– A intersecção de } [t] \text{ com } [t'] \text{ é o conjunto vazio de } \Sigma\text{-termos.} \tag{3.11}$$

$$\text{– } [t] \cap [t'] \text{ é um conjunto singular.} \tag{3.12}$$

□

3.3 Semântica Equacional

Esta perspectiva da especificação axiomática encara um conjunto E de axiomas como sendo um conjunto de *equações*, no sentido tradicional deste termo. Para tornar mais explícita esta perspectiva, escreve-se muitas vezes

$$t \equiv t'$$

em lugar de (t, t'), para qualquer $(t, t') \in E_s$ em E.

Vejamos como exprimir formalmente o (meta)significado de um conjunto E de equações. No Capítulo 1 estudamos a estrutura $(\mathcal{C}(\Sigma); \sqsubseteq)$ de quocientes sobre $\mathcal{W}$ na qual se pode representar uma qualquer semântica para a linguagem gerada por Σ. A cada quociente $\mathcal{W}/\cong$ em $(\mathcal{C}(\Sigma); \sqsubseteq)$ está associada uma Σ-congruência $\cong$. Vejamos agora como, a cada família E de equações corresponde uma Σ-congruência também. Por outras palavras, a apresentação de um conjunto E de equações pode ser considerado um método alternativo à construção de um Σ-modelo para se definir uma semântica sobre uma assinatura Σ.

Definição 3.8 (Congruência Equacional Mínima) *Seja (Σ, X, E) uma especificação axiomática. A* congruência equacional *(dita* mínima*) induzida por (Σ, X, E) é a menor família de relações*

$$\cong_E = (\cong_{E,s})_{s \in S}$$

tal que, para todo o $s \in S$,

$$\cong_{E,s} \subseteq W_{\Sigma,s} \times W_{\Sigma,s}$$

e que satisfaz, para $s \in S$:

$$(x, x') \in E_s \quad \Rightarrow \quad \{(\rho(x), \rho(x')) \mid \rho \in V_{\mathcal{W}}\} \subseteq \cong_{E,s} \tag{3.13}$$

$$t \in W_{\Sigma,s} \quad \Rightarrow \quad (t, t) \in \cong_{E,s} \tag{3.14}$$

$$(t, t') \in \cong_{E,s} \quad \Rightarrow \quad (t', t) \in \cong_{E,s} \tag{3.15}$$

$$\begin{array}{c} (t, t') \in \cong_{E,s} \\ \wedge \\ (t', t'') \in \cong_{E,s} \end{array} \quad \Rightarrow \quad (t, t'') \in \cong_{E,s} \tag{3.16}$$

Finalmente, para cada operador $\sigma : s_1 \times \ldots \times s_n \rightarrow s$ *e* $t_i, t'_i \in W_{\Sigma,s_i}$ *(*$1 \leq i \leq n$*),*

$$(t_i, t'_i) \in \cong_{E,s_i} \Rightarrow (\sigma(t_1, \ldots, t_n), \sigma(t'_1, \ldots, t'_n)) \in \cong_{E,s} \tag{3.17}$$

□

Teorema 3.2 *A congruência equacional* $\cong_E$ *induzida por uma especificação equacional* (Σ, X, E) *é, de facto, uma* Σ*-congruência.*

Demonstração*: É uma relação de equivalência pelas cláusulas de fecho reflexivo (3.14), simétrico (3.15) e transitivo (3.16) da Definição 3.8. A cláusula (3.17) garante a* Σ*-compatibilidade.* Q.E.D.

□

Exercício 3.3 Será que, numa especificação equacional, "mais axiomas implicam sempre mais semântica"? Quer dizer, sendo (Σ, X, E) e (Σ, X, E') duas especificações sobre a mesma assinatura $\Sigma : \Omega \rightarrow S^\star \times S$, demonstre ou refute o facto seguinte,

$$E \subset E' \Rightarrow \cong_E \subset \cong_{E'}$$

onde " $\subset$ " designa a inclusão (estrita) de famílias S-indexadas de conjuntos.

□

Vejamos um exemplo de especificação equacional, que se construirá sobre a assinatura Σ definida por (1.6) a partir da gramática dada pelas produções (1.5) que descrevem a sintaxe de uma pequena linguagem de comandos, de que se construiu um modelo (denotacional) na Secção 1.3.4. Seja $V = \{c, d, e, v, n, m\}$ e X tal que

$$\begin{array}{lclclcl} X(c) & = & X(d) & = & X(e) & = & <Cmd> \\ X(v) & & & & & = & <Var> \\ X(m) & = & X(n) & & & = & <Nato> \end{array} \tag{3.18}$$

Na seguinte apresentação do conjunto $E_{<Cmd>}$ de Σ-equações sobre X usar-se-á a sintaxe concreta de (1.5) em lugar da sintaxe abstracta de (1.6), por ser a primeira mais sugestiva. Comecemos pelos axiomas que relacionam a composição sequencial com ela própria, e a constante `skip`:

$$c; \mathtt{skip} \equiv c \tag{3.19}$$

$$\mathtt{skip}; c \equiv c \tag{3.20}$$

$$c; (d; e) \equiv (c; d); c \tag{3.21}$$

Quer dizer, damos à estrutura $\langle <Cmd>; ";", \mathtt{skip} \rangle$ a semântica de um monóide. Portanto, faz sentido estender o operador ";" a n-argumentos, obtendo um operador que podemos designar por

$$\overset{\bullet}{,}{}^{n}_{i=1} c_i$$

e que corresponde, nas vulgares linguagens de programação, à construção

$$\texttt{begin } c_1; c_2; \ldots; c_n \texttt{ end}$$

Que dizer sobre o comando de atribuição? O seguinte axioma

$$v := n; v := m \quad \equiv \quad v := m \tag{3.22}$$

equaciona a composição sequencial com a atribuição e quer, no fundo, indicar que a última atribuição a uma variável é "absorvente" em relação a uma que a preceda [3]. Quanto à composição condicional, não é difícil postular

$$\texttt{if 0 then } c \texttt{ else } d \quad \equiv \quad d \tag{3.23}$$

$$\texttt{if suc}(n) \texttt{ then } c \texttt{ else } d \quad \equiv \quad c \tag{3.24}$$

Quanto à composição iterativa, ocorre-nos imediatamente

$$\texttt{while 0 do } c \equiv \texttt{skip} \tag{3.25}$$

mas reparemos que não temos capacidade expressiva na linguagem para completar a equação

$$\texttt{while suc}(n) \texttt{ do } c \equiv \ldots \tag{3.26}$$

sobre um comando iterativo que não termina. Podemos então considerar em E os axiomas — equações — (3.19) a (3.25), ou seja:

$$E_{<Cmd>} = \left\{ \begin{array}{rcl} c; \texttt{skip} & \equiv & c \\ \texttt{skip}; c & \equiv & c \\ c; (d; e) & \equiv & (c; d); e \\ v := n; v := m & \equiv & v := m \\ \texttt{if 0 then } c \texttt{ else } d & \equiv & d \\ \texttt{if suc}(n) \texttt{ then } c \texttt{ else } d & \equiv & c \end{array} \right. \tag{3.27}$$

Então o triplo (Σ, X, E) — onde X é a tipificação de variáveis definida por (3.18) e E é uma família de equações que inclui (3.27) acima — é uma *especificação equacional* da semântica da pequena linguagem de comandos definida por (1.6).

Há várias questões que ocorrem sobre esta semântica:

- será "suficiente"? Quer dizer, teremos em E todos os axiomas que são básicos para a inferência de equivalências entre programas da linguagem?

- Será que é impossível completar a semântica que se pretendia na equação (3.26)? Será que num modelo essa semântica se exprime com facilidade?

[3] Notar que, na prática, este axioma traduz uma simplificação da realidade; de facto, apenas quando m é uma constante é que ele faz sentido — *cf.* o que pode acontecer quando m refere uma variável.

- Qual a relação entre a semântica equacional definida por E e a semântica denotacional construída na Secção 1.3.4?

Quanto à primeira questão, é difícil (em geral) ter a certeza que os axiomas básicos escolhidos são "suficientes", isto na medida em que sejam capazes de gerar todas as equivalências consideradas "desejáveis". Alguma experimentação à volta de E (3.27) mostraria "falta" de semântica: por exemplo, nada sabemos sobre o controlo da estrutura `if ...then ...else` quando este é feito por variáveis. Um axioma adicional,

$$v := n; \texttt{if}\ v\ \texttt{then}\ c\ \texttt{else}\ d \equiv v := n; \texttt{if}\ n\ \texttt{then}\ c\ \texttt{else}\ d \tag{3.28}$$

poderá ajudar alguma coisa, mas não resolverá todos os problemas quanto a esta estrutura de controlo [4]. Quanto à segunda questão, gostaríamos de poder atribuir uma semântica *indefinida* (parcial) ao comando iterativo da equação (3.26). Isso só será possível quando estudarmos a variante *inequacional* da semântica axiomática.

Finalmente, a secção que irá seguir-se tratará da relação que é levantada na terceira questão acima.

3.3.1 O Sistema Lógico-dedutivo DEQ(E)

Antes disso, porém, definiremos aqui o sistema lógico-dedutivo $DEQ(E)$ associado à interpretação equacional de um conjunto E de axiomas. Consta de 6 regras, cada uma da forma

$$\frac{Premissas}{\mathit{Conclusão}} \tag{3.29}$$

regras essas que espelham o processo da construção da congruência mínima equacional $\cong_E$:

R_1 *Reflexividade* — para todo o termo t,

$$\overline{t \equiv t}$$

R_2 *Simetria* —-

$$\frac{t \equiv t'}{t' \equiv t}$$

R_3 *Transitividade* —

$$\frac{t \equiv t', t' \equiv t''}{t \equiv t''}$$

[4] A possibilidade de se escreverem *axiomas condicionais*, que se estudará mais tarde, melhorará este estado de coisas.

R_4 *Substituição* (de “iguais por iguais”) — para cada operador $\sigma : s_1 \times \ldots \times s_n \to s$ e $t_i, t'_i \in W_{\Sigma,s_i}$ $(1 \leq i \leq n)$,

$$\frac{t_1 \equiv t'_1, \ldots, t_n \equiv t'_n}{\sigma(t_1, \ldots, t_n) \equiv \sigma(t'_1, \ldots, t'_n)}$$

R_5 *Instanciação* — para $\rho \in V_{\mathcal{W}}$,

$$\frac{t \equiv t'}{\mathcal{W}^\rho(t) \equiv \mathcal{W}^\rho(t')}$$

R_6 *Equações* — para cada equação $(t, t') \in E$

$$\frac{}{t \equiv t'}$$

cf. Definição 3.8.

Para aplicar a regra genérica (3.29) temos que ter já deduzidos todos os factos que são *Premissas*. Destes, a aplicação da regra deriva o facto que consta na *Conclusão*. Uma *dedução*, ou *prova*, é uma sequência de factos

$$t_1 \equiv t'_1, t_2 \equiv t'_2, \ldots, t_k \equiv t'_k, \ldots$$

tais que cada facto pode ser derivado por aplicação de qualquer das regras a factos que ocorrem anteriormente na sequência. Isto implica que o primeiro facto de uma *dedução* deve ser uma instância da regra R_1 ou da regra R_6, pois estas são as únicas regras com premissas vazias.

Escreveremos

$$\vdash_E t \equiv t'$$

sempre que exista uma dedução cujo último facto é $t \equiv t'$, e diremos que $t \equiv t'$ é um *teorema* de $DEQ(E)$. Quer dizer, $\vdash_E$ pode ser encarada como uma Σ-relação. Ou melhor, $\vdash_E$ é uma Σ-congruência sobre $\mathcal{W}(X)$, *cf.* regras R_1, R_2, R_3 e R_4 de $DEQ(E)$. É de esperar que $\vdash_E$, restringida a $\mathcal{W}$, coincida com $\cong_E$.

Para terminarmos a apresentação de $DEQ(E)$, refira-se que a melhor maneira de apresentar uma dedução é construir a respectiva *árvore de prova*, *i.é* estruturas como a seguir se ilustram:

$$R_3 \; \frac{R_6 \; \dfrac{}{c;\, \texttt{skip} \equiv c} \; , \; R_2 \; \dfrac{R_6 \; \dfrac{}{\texttt{skip}\,;c \equiv c}}{c \equiv \texttt{skip}\,;c}}{c;\, \texttt{skip} \equiv \texttt{skip}\,;c} \tag{3.30}$$

documentando os saltos dedutivos com as regras R_1-R_6 utilizadas em cada caso. Outro exemplo será

$$R_4 \quad \frac{R_1 \quad \dfrac{}{c \equiv c} \quad , \quad R_6 \quad \dfrac{}{\texttt{while}\ 0\ \texttt{do}\ c \equiv \texttt{skip}}}{c;\ \texttt{while}\ 0\ \texttt{do}\ c \equiv c;\ \texttt{skip}} \tag{3.31}$$

Juntando as árvores (3.31) com (3.30) poderíamos obter

$$R_3 \quad \frac{R_4 \quad \dfrac{\cdots}{c;\ \texttt{while}\ 0\ \texttt{do}\ c \equiv c;\ \texttt{skip}} \quad , \quad R_3 \quad \dfrac{\cdots}{c;\ \texttt{skip} \equiv \texttt{skip}\ ; c}}{c;\ \texttt{while}\ 0\ \texttt{do}\ c \equiv \texttt{skip}\ ; c}$$

e assim por diante.

Exercício 3.4 Seja Σ a seguinte assinatura:

$$\Sigma \quad = \quad \begin{cases} 0, 1 :\to b \\ f : a \to b \\ - : b \to b \\ +, \times : b \times b \to b \end{cases}$$

Seja E o conjunto dos seguintes Σ-axiomas:

$$y + y \equiv y \tag{3.32}$$
$$y \times y \equiv y \tag{3.33}$$
$$y + (-y) \equiv 1 \tag{3.34}$$
$$(-y) + y \equiv 1 \tag{3.35}$$
$$y \times (-y) \equiv 0 \tag{3.36}$$
$$(-y) \times y \equiv 0 \tag{3.37}$$

Mostre que a introdução de um novo axioma,

$$-f(x) \equiv f(x)$$

implica o facto $0 \equiv 1$.

□

Exercício 3.5 Suponha que à especificação equacional da semântica da pequena linguagem de comandos atrás estudada (*cf.* (3.27,3.28)) se acrescenta agora o axioma

$$v := n; v' := v \quad \equiv \quad v' := n; v := n$$

sobre o comando de atribuição. Serão todos os axiomas até agora compilados suficientes para caracterizar a semântica "intuitiva" que temos da linguagem? Justifique a sua resposta com base na construção de uma árvore de prova para o facto

```
x := 0; y := x;
x := suc(x);                 ≡  y := 0; x := suc(0);
if x then skip else skip
```

segundo o sistema lógico-dedutivo $DEQ(E)$.

□

3.3.2 Especificação Equacional versus Modelos

Definição 3.9 (Satisfação de Equações) *Seja* $\Sigma : \Omega \rightarrow S^\star \times S$ *uma assinatura,* (Σ, X, E) *uma especificação equacional e* $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ *um modelo. Diremos que* $\mathcal{A}$ satisfaz as equações *em* E*, ou que é um* modelo *de* E*, sempre que* $\cong_E \subseteq \cong_{\mathcal{A}}$*, quer dizer:*

$$t \cong_E t' \Rightarrow \mathcal{A}(t) = \mathcal{A}(t')$$

□

Portanto, se $\mathcal{A}$ está nas condições da definição anterior, então

$$\forall (t, t') \in E : (\forall \rho \in V_{\mathcal{A}} : \rho(t) = \rho(t'))$$

i.é todos os axiomas de E são satisfeitos em $\mathcal{A}$ (bem como todas as suas consequências equacionais). Mas pode haver mais "coisas verdadeiras" em $\mathcal{A}$ que não são consequências de E, pois, em geral,

$$\cong_{\mathcal{A}} \not\subseteq \cong_E$$

Relembremos aqui o facto (2.77) cuja demonstração, proposta no Exercício 2.27, corresponde à validação de um axioma (equação)

$$elems(list(x)) \equiv x$$

que relaciona dois operadores cujas funcionalidades podiam ser

$$\begin{array}{rcl} elems & : & Seqs \rightarrow Sets \\ list & : & Sets \rightarrow Seqs \end{array}$$

para um modelo $\mathcal{A}$ tal que

$$\begin{array}{rcl} \mathcal{A}(Sets) & \cong & 2^X \\ \mathcal{A}(Seqs) & \cong & X^* \end{array}$$

Definição 3.10 (Interface) *Sempre que* $I = (\Sigma, X, E)$ *é uma especificação equacional e* $\mathcal{A}$ *satisfaz* E*, diremos também que* I *é uma* interface *para* $\mathcal{A}$ *— ou que* $\mathcal{A}$ *tem* I *como interface — e escreveremos*

$$\mathcal{A} :\rightarrow I \tag{3.38}$$

notação que fará sentido mais tarde [5].

I *dir-se-á uma* interface completa *para* $\mathcal{A}$ *se também acontecer*

$$t \cong_{\mathcal{A}} t' \Rightarrow t \cong_E t'$$

i.é*, se* $\cong_{\mathcal{A}}$ *e* $\cong_E$ *coincidirem.*

$\mathcal{C}(I)$ *ou* $\mathcal{C}(E)$ *são designações que utilizaremos para a classe de todos os modelos* $\mathcal{A}$ *que satisfazem as equações de* $I = (\Sigma, X, E)$. □

[5] Ver Capítulo 4.

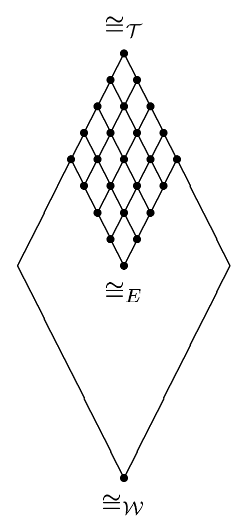


Figura 3.1: Sub-reticulado de congruências que satisfazem uma especificação equacional.

Exercício 3.6 Calcule a Σ-congruência $\cong_{\emptyset}$ associada à especificação equacional $(\Sigma, X, \emptyset)$. Mostre que $I = (\Sigma, X, \emptyset)$ é interface de qualquer modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$.
□

Teorema 3.3 *Todas as Σ-congruências satisfazendo os axiomas de uma especificação equacional (Σ, X, E) formam um sub-reticulado completo (do reticulado de* todas *as Σ-congruências) cujo limite universal mínimo é $\cong_E$,* cf. *Figura 3.1 (e relembrar a Figura 1.7).*

Demonstração*: Se uma dada Σ-congruência $\cong$ satisfaz E, então está nas condições de $\cong_{\mathcal{A}}$ da Definição 3.9. Logo, $\cong_E \subseteq \cong$ para todo o $\cong$ nessas condições,* i.é *$\cong_E$ é limite universal inferior. Òbviamente, $\cong_{\mathcal{T}}$ satisfaz os axiomas,* i.é *é limite universal superior.*

Se duas congruências $\cong_1$ e $\cong_2$ satisfazem os axiomas, então

$$\begin{array}{ccl} t \cong_E t' & \Rightarrow & t \cong_1 t' \\ t \cong_E t' & \Rightarrow & t \cong_2 t' \\ \hline t \cong_E t' & \Rightarrow & t \cong_1 t' \wedge t \cong_2 t' \\ & \Leftrightarrow & t(\cong_1 \cap \cong_2)t' \end{array}$$

quer dizer, a sua conjunção (g.l.b.) também satisfaz os axiomas. Este resultado pode ser iterado a $\bigcap S$, onde S é uma família finita de congruências satisfazendo *os axiomas.* $\bigcup S$ *constrói-se tal como no Teorema 1.2.* □

Reparemos nas consequências práticas da Definição 3.10 e do teorema anterior. Se $I = (\Sigma, X, E)$ é a interface de um modelo $\mathcal{A}$, então $\mathcal{A} \in \mathcal{C}(I)$, *i.é* a congruência $\cong_E$ é mais fina do que $\cong_{\mathcal{A}}$. Portanto, ao compararmos as semânticas respectivas, podemos tecer as seguintes considerações:

- $\cong_{\mathcal{A}}$ é a "verdadeira semântica" que queremos definir, *i.é* através de um modelo que se construiu, para uma dada Σ-linguagem, segundo o *método denotacional* clássico. Contudo, decidir a equivalência $t \cong_{\mathcal{A}} t'$ com base nos cálculos explícitos de $\mathcal{A}(t)$ e $\mathcal{A}(t')$ pode, em modelos volumosos, ser pouco prático. Isto justifica a apresentação de uma interface equacional $I = (\Sigma, X, E)$ que "esconda" a complexidade de $\mathcal{A}$ por detrás de um conjunto E de axiomas que agrupe propriedades úteis na prática sobre $\mathcal{A}$.

- $\cong_E$ tem, pois, "menos semântica" que $\cong_{\mathcal{A}}$, e muitos factos verdadeiros sobre $\mathcal{A}$ não serão dedutíveis de E; contudo, em muitos casos, a leitura de E pode poupar o analista que pensa "re-utilizar" $\mathcal{A}$, de fazer uma "visita" explícita a esse modelo.

No Capítulo 4 este "jogo" que é preciso fazer, na prática, entre modelos e as suas interfaces, ficará mais explícito no contexto da parametrização e re-utilização de especificações orientadas a modelos.

Exercício 3.7 Dada a assinatura

$$\Sigma = \left\{ \begin{array}{l} f : \rho \to \mu \\ u : \to \mu \\ b : \to \rho \\ a : \mu \times \mu \to \mu \\ d : \pi \times \rho \to \mu \\ e : \pi \times \rho \to \rho \end{array} \right.$$

1. Mostre que existe apenas uma Σ-álgebra $\mathcal{A}$ (descreva-a) cujos portadores não diferem de $\mathbb{Z}$ (conjunto de todos os números inteiros) e que admite no máximo os operadores $+$ (soma de dois inteiros), $-$ (simétrico de um inteiro) e 0 (zero).

2. Verifique se $\mathcal{A}$ é modelo da interface $\langle \Sigma, E \rangle$ onde E é o conjunto dos seguintes dois Σ-axiomas:

$$f(b) \equiv u \tag{3.39}$$
$$f(e(z, x)) \equiv a(d(z, x), f(x)) \tag{3.40}$$

□

Exercício 3.8 No contexto do Exercício 3.7, defina uma Σ-álgebra $\mathcal{A}$ tal que:

- $\mathcal{A}$ é modelo da interface $\langle \Sigma, E \rangle$ onde E é o conjunto dos Σ-axiomas (3.39) e (3.40).

- A semântica de f é a da função que calcula o comprimento de uma lista finita.

□

Exercício 3.9 Relembre as Σ-congruências $\cong_{\mathcal{A}}$ e $\cong_{\mathcal{B}}$ associadas aos respectivos modelos do Exercício 1.24. Seja E uma família de Σ-axiomas tal que

$$\begin{array}{rcl} E_s & = & \emptyset \\ E_r & = & \{\sigma(x,y) \equiv \sigma(y,x)\} \end{array}$$

- Compare a granularidade de $\cong_E$ com a de $\cong_{\mathcal{A}}$ e $\cong_{\mathcal{B}}$.
- Suponha agora que $E_s = \{0 \equiv 1\}$. Será que $\cong_E = \cong_{\mathcal{T}}$?

□

3.4 Semântica Inequacional

Em especificação axiomática, os axiomas que muitas vezes gostaríamos de poder escrever para captar a realidade assumem a forma de *inequações* $t \leq t'$ e não de equações $t \equiv t'$. Por exemplo, se estivessemos a especificar um programa de verificação de erros de ortografia ('spelling'), podíamos ter definido a espécie

$$\mathit{Vocabulário} \cong 2^{\mathit{Palavra}}$$

bem como o operador

$$\begin{array}{rcl} \mathit{inserePalavra} & : & \mathit{Vocabulário} \times \mathit{Palavra} \to \mathit{Vocabulário} \\ \mathit{inserePalavra}(v,p) & \stackrel{\mathrm{def}}{=} & v \cup \{p\} \end{array}$$

Com base na teoria de conjuntos, não será difícil demonstrar o facto

$$\forall v, p : v \subseteq \mathit{inserePalavra}(v,p) \tag{3.41}$$

Ora tal facto não pode ser registado numa interface equacional pois o símbolo $\subseteq$ não goza das propriedades de uma relação de equivalência mas sim das de uma ordem parcial. A necessária extensão do conceito de especificação equacional passa a ser estudada a seguir.

3.4.1 Modelos Parcialmente Ordenados

Seja Pos a classe categoria de todos os conjuntos finitos parcialmente ordenados cujos morfismos sejam apenas funções crescentes, *i.é* se $(A; \leq_A)$ e $(B; \leq_B)$ são dois desses conjuntos, uma função (morfismo)

$$f : A \longrightarrow B$$

estará em Pos se e só se, para todo o $a \in A$,

$$a \leq_A a' \Rightarrow f(a) \leq_B f(a') \tag{3.42}$$

Podemos então passar à definição seguinte.

Definição 3.11 (Modelo Parcialmente Ordenado) *Relembrar a definição de uma Σ-álgebra (Definição 1.7), onde $\Sigma : \Omega \to S^\star \times S$ é uma assinatura. Uma Σ-álgebra* ordenada*, ou* Σ-modelo parcialmente ordenado *é todo o functor*

$$\mathcal{A} : \Sigma \to Pos$$

ou seja:

- *para $s \in S$, $\mathcal{A}(s)$ é um conjunto parcialmente ordenado cuja ordem se designará por $\leq_{\mathcal{A}(s)}$;*
- *para cada operador $\sigma : s_1 \times \ldots \times s_n \to s$ em Σ, $\mathcal{A}(\sigma)$ é uma função*

$$\mathcal{A}(\sigma) : \mathcal{A}(s_1) \times \ldots \times \mathcal{A}(s_n) \to \mathcal{A}(s)$$

crescente nos seus n-argumentos, i.é*, para $a_i, a'_i \in \mathcal{A}(s_i), 1 \leq i \leq n$,*

$$(\forall 1 \leq i \leq n : a_i \leq_{\mathcal{A}(s_i)} a'_i) \Rightarrow \mathcal{A}(\sigma)(a_1, \ldots, a_n) \leq_{\mathcal{A}(s)} \mathcal{A}(\sigma)(a'_1, \ldots, a'_n)$$

Quer dizer, a família

$$\leq_{\mathcal{A}} = (\leq_{\mathcal{A}(s)})_{s \in S}$$

de ordem parciais é Σ-compatível. A um modelo $\mathcal{A} : \Sigma \to Pos$ dá-se também o nome de Σ_{po}-álgebra ou Σ_{po}-modelo. □

O conceito de Σ-*homomorfismo* será também ampliado, em concordância com a definição anterior:

Definição 3.12 (Σ_{po}-homomorfismo) *Sejam $\mathcal{A}, \mathcal{B} : \Sigma \to Pos$ dois Σ_{po}-modelos. Seja*

$$h = (h_s)_{s \in S}$$

uma família de funções crescentes em Pos, i.é

- *para $s \in S$,*

$$h_s : \mathcal{A}(s) \longrightarrow \mathcal{B}(s)$$

e

$$a \leq_{\mathcal{A}(s)} a' \Rightarrow h_s(a) \leq_{\mathcal{B}(s)} h_s(a')$$

- *a cláusula (1.88) verifica-se.*

Então, diremos que h é um Σ_{po}-homomorfismo. Esta definição é uma extensão da Definição 1.10. □

Σ_{po}-epimorfismos ou Σ_{po}-isomorfismos definem-se como Σ_{po}-homomorfismos sobrejectivos e bijectivos, respectivamente.

É imediato verificar-se que a noção de Σ_{po}-álgebra estende a de Σ-álgebra. De facto, toda a álgebra $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ pode ser encarada como uma Σ_{po}-álgebra $\mathcal{A}' : \Sigma \rightarrow Pos$ considerando que, para cada $s \in S$, $\leq_{\mathcal{A}'(s)}$ é a relação identidade em $\mathcal{A}(s)$:

$$a \leq_{\mathcal{A}'(s)} a' \textit{ sse } a = a'$$

Definição 3.13 (Σ_{po}-interpretação) *Define-se tal como na Definição 1.11. De facto, se*

$$h : \mathcal{W} \rightarrow \mathcal{A}$$

*é um homomorfismo, para $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$, é também um Σ_{po}-homomorfismo para $\mathcal{A} : \Sigma \rightarrow Pos$, já que a ordem parcial sobre W_Σ é a trivial (*i.é *a igualdade literal de termos).* □

Assim, as definições de instanciação de variáveis (*cf.* Definição 3.2), de $\Sigma(X)$-interpretação (*cf.* Definição 3.3), e o teorema da unicidade de $\Sigma(X)$-interpretações (*cf.* Teorema 3.1) estendem-se automaticamente a Σ_{po}-álgebras e Σ_{po}-homomorfismos.

3.4.2 Os Modelos "Planos" e a Recursividade

Vejamos uma classe muito importante de modelos ordenados. Seja $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$ um modelo não ordenado. Vamos, a partir dele, construir um outro modelo

$$\mathcal{A}_\perp : \Sigma \rightarrow Pos$$

— este ordenado — como se segue:

- para $s \in S$, seja $\mathcal{A}_\perp(s) = \mathcal{A}(s) \cup \{\perp_s\}$, para um valor $\perp_s$ qualquer não presente em $\mathcal{A}(s)$, sendo a ordem $\leq_{\mathcal{A}_\perp(s)}$ a vulgarmente designada *ordem parcial plana* ('flat c.p.o.'),

$$\forall a \in \mathcal{A}_\perp(s) : \perp_s \leq a \tag{3.43}$$

abreviando $\leq_{\mathcal{A}_\perp(s)}$ para $\leq$, para não sobrecarregar a notação, *cf.* Figura 3.2.

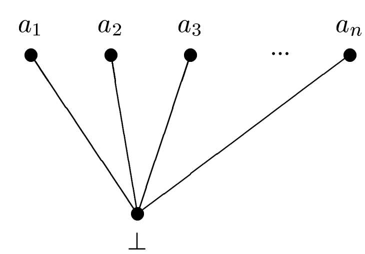


Figura 3.2: A ordem parcial plana sobre $\mathcal{A}_\perp(s)$.

- para cada Σ-operador $\sigma : s_1 \times \ldots \times s_n \to s$, a função correspondente em $\mathcal{A}_\perp$ *estende naturalmente* $\mathcal{A}(\sigma)$, *i.é*,

$$\mathcal{A}_\perp(\sigma)(a_1, \ldots, a_n) \stackrel{\mathrm{def}}{=} \begin{cases} \mathcal{A}(\sigma)(a_1, \ldots, a_n) & \Leftarrow \quad \forall 1 \leq i \leq n : a_i \neq \perp_{s_i} \\ \perp_s & \Leftarrow \quad \exists 1 \leq i \leq n : a_i = \perp_{s_i} \end{cases} \tag{3.44}$$

Exercício 3.10 Mostre que o modelo $\mathcal{A}_\perp$ construído a partir de $\mathcal{A}$ de acordo com (3.43) e (3.44) é, de facto, um modelo parcialmente ordenado.
□

De uma modo geral, toda a função n-ária f que, tal como (3.44), satisfaz a cláusula

$$f(a_1, \ldots, \perp, \ldots, a_n) = \perp$$

para $1 \leq i \leq n$, diz-se uma função *estrita*. A interpretação canónica para cada símbolo $\perp_s$ (3.43) é a de *indefinição*, *i.é*, $\perp_s$ representa o *valor indefinido* da espécie s. Este valor é muitas vezes artificialmente introduzido para "representar" o resultado de uma função que não termina.

Aliás, sempre que atrás escrevemos definições de funções com pré-condições, tipicamente

$$\mathcal{A}(\sigma) = \lambda x.\{pre(x) \Rightarrow \ldots$$

estávamos já implicitamente a trabalhar com a correspondente versão ordenada,

$$\mathcal{A}_\perp(\sigma) \quad = \quad \lambda x. \begin{cases} pre(x) & \Rightarrow \ldots \\ \neg pre(x) & \Rightarrow \perp_s \end{cases} \tag{3.45}$$

— *cf.* o que no Capítulo 2 se discutiu a propósito de $f_\perp$ (pág. 87).

Exercício 3.11 Estará correcto afirmar-se que todo o modelo $\mathcal{A}$ que só contém funções *estritas* é parcialmente ordenado? Justifique formalmente a sua resposta.
□

Os modelos planos constituem, portanto, uma ferramenta matemática adequada ao tratamento da indefinição de funções parciais. Como estas se podem ainda definir de forma arbitrariamente recursiva, convém particularizar aqui a respectiva abordagem por pontos fixos.

Seja $\sigma : s \to r$ um operador de uma assinatura Σ e $\mathcal{A}_\perp$ um Σ-modelo plano em que $\mathcal{A}_\perp(\sigma)$ é uma função definida recursivamente. Para não sobrecarregar a notação, abreviaremos $\mathcal{A}_\perp(\sigma)$ para f, $\mathcal{A}_\perp(s)$ para S e $\mathcal{A}_\perp(r)$ para R. Sendo recursiva, f é da forma

$$\begin{array}{rcl} f: & : & S \to R \\ f(x) & \stackrel{\text{def}}{=} & F(f)(x) \end{array} \tag{3.46}$$

onde F designa uma expressão em f funcionalmente correcta envolvendo, possivelmente, outros Σ-operadores. Podemos então escrever, em lugar de (3.46),

$$f \quad \stackrel{\text{def}}{=} \quad \lambda x.F(f)(x)$$

ou, ao nível puramente funcional,

$$f \quad \stackrel{\text{def}}{=} \quad F(f) \tag{3.47}$$

Qual o significado da expressão recursiva (3.47)? Para fazermos uma interpretação coerente com o Teorema 2.2 (Teorema de Kleene) atrás, precisamos de caracterizar o espaço de funções $S \to R$ como uma *c.p.o.* — $\langle S \to R; \leq \rangle$ — e garantir que

$$F : (S \to R) \longrightarrow (S \to R)$$

é uma *funcional* [6] contínua em tal *c.p.o.*. A ordem $\leq$ sobre funções será definida à custa das ordens planas que o modelo de trabalho, $\mathcal{A}$, garante sobre S e R. Assim, dados $f, g : S \to R$, definiremos

$$f \leq g \quad \stackrel{\text{def}}{=} \quad \forall a \in S : f(a) \leq_R g(a) \tag{3.48}$$

[6] A função F designa-se por *funcional* para tornar explícito que se trata de uma função cujos argumentos e resultados são, eles próprios, funções.

onde $\leq_R$ abrevia $\leq_{\mathcal{A}_\perp(r)}$. É fácil de ver que existe, em particular, uma função constante

$$\begin{array}{rcl} \overline{\perp} & : & S \to R \\ \overline{\perp}(x) & \stackrel{\text{def}}{=} & \perp_R \end{array}$$

que limita $\langle S \to R; \leq \rangle$ inferiormente, *i.é* tal que, para todo o

$$g : S \to R$$

se tem

$$\overline{\perp} \leq g$$

Isto vai-nos permitir interpretar o significado de (3.47) segundo o Teorema de Kleene, *i.é* construindo

$$\mu F = \bigvee_{i=0}^{\infty} F^i(\overline{\perp})$$

desde que F seja contínua. Mas vejamos, antes de mais, um exemplo, para $S = R = I\!N \cup \{\perp\}$:

$$f(n) \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow f(n+1) \end{array} \right.$$

quer dizer,

$$\begin{array}{rcl} f(n) & \stackrel{\text{def}}{=} & F(f)(n) \\ & onde & F(f) = \lambda n. \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow f(n+1) \end{array} \right. \end{array}$$

Intuitivamente, f é a identidade para $n = 1$ e irá divergir para qualquer argumento $n \neq 1$. Vejamos como a correspondente cadeia de Kleene confirma essa intuição:

$$\begin{array}{rcl} F^0(\overline{\perp}) & = & \overline{\perp} \\ & = & \lambda n.\perp \\ F^1(\overline{\perp}) & = & F(\lambda n.\perp) \\ & = & \lambda n. \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow \overline{\perp}(n+1) \end{array} \right. \\ & = & \lambda n. \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow \perp \end{array} \right. \\ F^2(\overline{\perp}) & = & F(F^1(\lambda n.\perp)) \\ & = & F(\lambda n. \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow \perp \end{array} \right.) \end{array}$$

$$
\begin{array}{ll}
= & \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow (\lambda n. \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow \perp \end{array} \right.)(n+1) \end{array} \right. \\
= & \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow \left\{ \begin{array}{ll} n+1 = 1 & \Rightarrow 1 \\ n+1 \neq 1 & \Rightarrow \perp \end{array} \right. \end{array} \right. \\
= & \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow \left\{ \begin{array}{ll} F & \Rightarrow 1 \\ V & \Rightarrow \perp \end{array} \right. \end{array} \right. \\
 & \textit{(pois } n+1 = 1 \textit{ é impossível em } I\!N\textit{)} \\
= & \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow \perp \end{array} \right.
\end{array}
$$

tendo-se assim obtido já o menor dos pontos fixos,

$$
\mu F \quad = \quad \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow \perp \end{array} \right.
$$

que é a função que se esperava.

Note-se que (3.49) não é o único ponto fixo de F; de facto, para todo o $m \in I\!N \cup \{\perp\}$, a função

$$
f_m \quad = \quad \lambda n. \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow m \end{array} \right.
$$

é ponto fixo de F, pois

$$
\begin{array}{lll}
F(f_m) & = & \lambda n. \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow f_m(n+1) \end{array} \right. \\
 & = & \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow \left\{ \begin{array}{ll} n+1 = 1 & \Rightarrow 1 \\ n+1 \neq 1 & \Rightarrow m \end{array} \right. \end{array} \right. \\
 & = & \lambda n. \left\{ \begin{array}{ll} n = 1 & \Rightarrow 1 \\ n \neq 1 & \Rightarrow m \end{array} \right. \\
 & = & f_m
\end{array}
$$

Repare-se que $\mu F = f_\perp \leq f_m$, para $m \geq 1$. Logo, o conjunto de todos os pontos fixos de F forma, ele próprio, uma *c.p.o.* plana (*cf.* Figura 3.3).

Faltou averiguar sobre a *continuidade* do funcional F. Em geral, o Teorema 2.3 poderá ser invocado para todas as funcionais envolvendo funções crescentes, como é o caso da funcional *composição* e a funcional *condição* — *cf.* os exercícios seguintes.

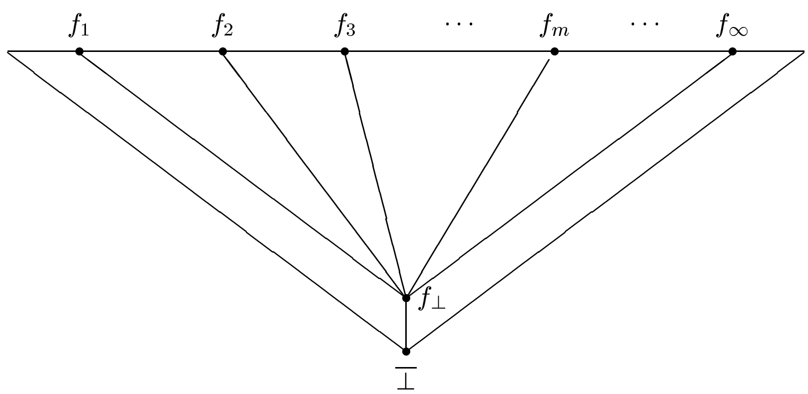


Figura 3.3: Sub-*c.p.o.* plana de pontos-fixos de F.

Exercício 3.12 Mostre que, para todo o f,

$$f \circ \overline{\bot} = \overline{\bot} \circ f = \overline{\bot}$$

□

Exercício 3.13 Verifique se a composição de funções (estritas ou, no mínimo, crescentes) é uma funcional crescente nos seus dois argumentos em relação à ordem de definição de funções expressa por (3.48), isto é se

$$\begin{array}{lcl} f \leq g & \Rightarrow & h \circ f \leq h \circ g \\ f \leq g & \Rightarrow & f \circ h \leq g \circ h \end{array}$$

□

Exercício 3.14 Seja $p \rightarrow g; h$ uma expressão funcional que designa a mesma função que a λ-expressão

$$\lambda x. \left\{ \begin{array}{lcl} p(x) & \Rightarrow & g(x) \\ \neg p(x) & \Rightarrow & h(x) \end{array} \right.$$

onde se assume sempre $p(x) \neq \bot$. Verifique quais dos factos

$$\begin{array}{lcl} f \leq g & \Rightarrow & p \rightarrow f; h \leq p \rightarrow g; h \\ f \leq g & \Rightarrow & p \rightarrow h; f \leq p \rightarrow h; g \end{array}$$

são válidos.

□

Exercício 3.15 Seja $[f, g]$ uma expressão funcional que designa a mesma função que a λ-expressão

$$\lambda x.\langle f(x), g(x)\rangle$$

— recordar (1.20). Verifique se

$$\begin{array}{lcl} f \leq h & \Rightarrow & [f, g] \leq [h, g] \\ g \leq h & \Rightarrow & [f, g] \leq [f, h] \end{array}$$

□

Falta apenas discutir o caso das funções poliádicas, *i.é*

$$\begin{array}{rcl} f : & : & S_1 \times \ldots \times S_n \to R \\ f(x_1, \ldots, x_n) & \stackrel{\text{def}}{=} & F(f)(x_1, \ldots, x_n) \end{array}$$

em lugar de (3.46), o que não apresenta problemas adicionais já que, com modelos planos, estamos a trabalhar apenas com funções naturalmente estendidas, que são sempre crescentes — veja-se o teorema que se segue.

Teorema 3.4 (Monotonia de Funções Naturalmente Estendidas)
Toda a função naturalmente estendida é monótona.
Demonstração*: Ver [Man74].* □

Exercício 3.16 Seja Σ a assinatura de um grupóide, isto é, constando apenas de uma espécie e um operador binário. Seja $2_\perp$ o conjunto parcialmente ordenado

$$\begin{tikzcd} F \arrow[dr, dash] & & V \arrow[dl, dash] \\ & \perp & \end{tikzcd}$$

e sejam $\mathcal{A}$ e $\mathcal{B}$ duas Σ_{po}-álgebras cujo portador é $2_\perp$ e cujos operadores satisfazem as seguintes tabelas de verdade, respectivamente em $\mathcal{A}$ ($\dot\wedge$) e em $\mathcal{B}$ ($\dot\vee$):

$\dot\wedge$	V	F	$\perp$
V	V	F	$\perp$
F	F	F	F
$\perp$	$\perp$	F	$\perp$

$\dot\vee$	V	F	$\perp$
V	V	V	V
F	V	F	$\perp$
$\perp$	V	$\perp$	$\perp$

1. Mostre que $\dot\wedge$ e $\dot\vee$ são operadores que, embora crescentes, não são estritos.
2. Mostre que o operador sobre $2_\perp$

$$neg(b) \stackrel{\text{def}}{=} \begin{cases} b = \perp & \Rightarrow \quad \perp \\ b > \perp & \Rightarrow \quad \neg b \end{cases}$$

estabelece um Σ_{po}-isomorfismo entre $\mathcal{A}$ e $\mathcal{B}$ (e vice versa) que pode ser encarado como uma extensão ordenada às clássicas *Leis de Morgan*.

$\square$

Exercício 3.17 Verifique se a função $\lambda(n,m).n-m$ (subtracção de inteiros) é ponto fixo da funcional seguinte:

$$F(f) \stackrel{\text{def}}{=} \lambda(x,y).\begin{cases} x = y & \Rightarrow & 0 \\ x > y & \Rightarrow & 1 + f(x-1,y) \end{cases}$$

$\square$

3.4.3 Pré-ordens

Temos visto como os conceitos fundamentais da teoria algébrica das Σ-álgebras estudada no Capítulo 1 se estendem naturalmente a Σ_{po}-álgebras. A questão que se põe agora é: qual é o Σ_{po}-equivalente da noção de Σ-congruência?

Na base do conceito de Σ_{po}-álgebra está a noção de ordem parcial. Esta noção "colide" com a propridade de *simetria* das relações de congruência. Veremos de seguida que a própria *antissimetria* das ordens parciais é uma propriedade demasiado forte para a noção que pretendemos que venha a estender o conceito de Σ-congruência. Comecemos por reflectir sobre a definição seguinte:

Definição 3.14 (Σ-pré-ordem) *Seja $\mathcal{A} : \Sigma \to Pos$ uma Σ_{po}-álgebra, e seja*

$$\sqsubseteq = (\sqsubseteq_s)_{s \in S}$$

uma família S-indexada de relações sobre $\mathcal{A}$ que é Σ-compatível.

Seja cada $\sqsubseteq_s$ uma relação reflexiva e transitiva. Vamos ainda supor que, para cada $s \in S$, e $a, a' \in \mathcal{A}(s)$,

$$a \leq_{\mathcal{A}(s)} a' \Rightarrow a \sqsubseteq_s a'$$

quer dizer, $\sqsubseteq$ estende $\leq_{\mathcal{A}}$ na medida em que

$$\leq_{\mathcal{A}(s)} \quad \subseteq \quad \sqsubseteq_s$$

para cada $s \in S'$.

Então, $\sqsubseteq$ dir-se-á uma Σ-pré-ordem *sobre $\mathcal{A}$.* $\square$

Quando comparada com uma Σ-equivalência ou uma Σ-ordem-parcial, uma Σ-pré-ordem é mais geral na medida em que relaxa qualquer requisito no que diz respeito à *simetria* (ou *antissimetria*).

Vejamos um exemplo típico de construção de uma pré-ordem sobre os termos de uma dada Σ-linguagem. Seja $\mathcal{A} : \Sigma \to Pos$ uma Σ_{po}-álgebra. Seja $s \in S$ uma Σ-espécie, $t, t' \in W_{\Sigma,s}$, e $\sqsubseteq_s$ a relação em $W_{\Sigma,s}$ definida por

$$t \sqsubseteq_s t' \quad sse \quad \mathcal{A}(t) \leq_s \mathcal{A}(t') \tag{3.49}$$

onde se abreviou $\leq_{\mathcal{A}(s)}$ em $\leq_s$, para não sobrecarregar a notação. Seja

$$\sqsubseteq = (\sqsubseteq_s)_{s \in S}$$

a família S-indexada de relações $\sqsubseteq_s$ que satisfazem (3.49). Que tipo de Σ-relação é $\sqsubseteq$? Não é difícil verificar que se trata de uma família de relações reflexivas e transitivas. Qual o seu comportamento no "eixo" da simetria? Quer dizer, que podemos dizer sobre dois termos t e t' tal que $t \sqsubseteq_s t'$ e $t' \sqsubseteq_s t$? Teremos

$$\begin{array}{rcl} t \sqsubseteq_s t' & \Rightarrow & \mathcal{A}(t) \leq_s \mathcal{A}(t') \\ t' \sqsubseteq_s t & \Rightarrow & \mathcal{A}(t') \leq_s \mathcal{A}(t) \\ \hline t \sqsubseteq_s t' \wedge t' \sqsubseteq_s t & \Rightarrow & \mathcal{A}(t') = \mathcal{A}(t) \end{array}$$

pois $\leq_s$ é, por definição, antissimétrica. Quer dizer,

$$(t \sqsubseteq_s t' \wedge t' \sqsubseteq_s t) \Rightarrow t \cong_{\mathcal{A}} t' \tag{3.50}$$

Com mais generalidade, podemos afirmar o seguinte teorema:

Teorema 3.5 (Fecho Antissimétrico de Σ-pré-ordens) *Seja $\sqsubseteq$ uma Σ-pré-ordem, e seja $\simeq$ o seu núcleo,* i.é

$$a \simeq_s a' \; sse \; a \sqsubseteq_s a' \wedge a' \sqsubseteq_s a \tag{3.51}$$

para $s \in S$. Então $\simeq$ é uma Σ-congruência.

Demonstração*: É imediato que $\simeq$ seja reflexiva e transitiva. Provemos que é simétrica,* i.é *que cada $\simeq_s$ é tal que*

$$a \simeq_s a' \Rightarrow a' \simeq_s a \tag{3.52}$$

De acordo com (3.51), a implicação (3.52) re-escreve para

$$(a \sqsubseteq_s a' \wedge a' \sqsubseteq_s a) \Rightarrow (a' \sqsubseteq_s a \wedge a \sqsubseteq_s a') \tag{3.53}$$

Como a conectiva lógica $\wedge$ é comutativa, (3.53) verifica-se trivialmente pois a implicação lógica é uma conectiva reflexiva, i.é, *para todo o predicado p,*

$$p \Rightarrow p$$

Sendo $\sqsubseteq$ uma Σ-pré-ordem, é Σ-compatível. Logo é uma Σ-congruência. $\Box$

O Teorema 3.5 mostra uma maneira de se "partirem" Σ_{po}-álgebras por Σ-pré-ordens obtendo novas Σ_{po}-álgebras, *i.é* temos para Σ_{po}-álgebras uma construção análoga à de Σ-álgebras quociente, de acordo com a definição que se segue.

Definição 3.15 (Σ_{po}-quocientes) *Seja $\mathcal{A} : \Sigma \to Pos$ uma Σ_{po}-álgebra, e seja $\sqsubseteq$ uma pré-ordem sobre $\mathcal{A}$. A Σ_{po}-álgebra $\mathcal{A}/\sqsubseteq$, dita "quociente de $\mathcal{A}$ por $\sqsubseteq$", define-se como se segue:*

- *seja $\simeq$ o núcleo de $\sqsubseteq$,* cf. *(3.51)*
- *para cada $s \in S$, defina-se*

$$\mathcal{A}/\sqsubseteq(s) = \mathcal{A}(s)/\simeq_s \tag{3.54}$$

- *para cada portador de $\mathcal{A}/\sqsubseteq$ definido por (3.54), defina-se a seguinte ordem parcial sobre $\simeq_s$-classes de equivalência:*

$$\forall [a], [a'] \in \mathcal{A}/\sqsubseteq(s) : [a] \leq_s [a'] \Leftrightarrow a \sqsubseteq_s a' \tag{3.55}$$

  Não é difícil provar que cada relação $\leq_s$ definida por (3.55) sobre $\mathcal{A}/\sqsubseteq(s)$ é uma ordem parcial, onde a igualdade da antissimetria é a igualdade de conjuntos (classes de congruência).
- *para cada Σ-operador $\sigma : s_1 \times \ldots \times s_n \to s$, e $a_i \in \mathcal{A}(s_i)$ para $1 \leq i \leq n$, defina-se*

$$\mathcal{A}/\sqsubseteq(\sigma)([a_1], \ldots, [a_n]) = [\mathcal{A}(\sigma)(a_1, \ldots, a_n)] \tag{3.56}$$

  cf. *Definição 1.14.*

Será cada $\mathcal{A}/\sqsubseteq(\sigma)$ definida por (3.56) uma função crescente? Seja $[a_i] \leq_{s_i} [a'_i]$ em (3.56) de acordo com (3.55). Então $a_i \sqsubseteq_{s_i} a'_i$. Como $\sqsubseteq$ é Σ-compatível,

$$\mathcal{A}(\sigma)(\ldots, a_i, \ldots) \sqsubseteq_s \mathcal{A}(\sigma)(\ldots, a'_i, \ldots)$$

isto é,

$$[\mathcal{A}(\sigma)(\ldots, a_i, \ldots)] \sqsubseteq_s [\mathcal{A}(\sigma)(\ldots, a'_i, \ldots)]$$

ou seja

$$\mathcal{A}/\sqsubseteq(\sigma)(\ldots, a_i, \ldots) \sqsubseteq_s \mathcal{A}/\sqsubseteq(\sigma)(\ldots, a'_i, \ldots)$$

como queríamos. $\Box$

Tal como aconteceu com Σ-álgebras, interessa-nos construir Σ_{po}-quocientes sobre a álgebra dos termos $\mathcal{W}$, no sentido de conseguirmos explicar o significado de um conjunto de Σ-inequações da forma $t \leq t'$.

Reparemos que, dado um Σ_{po}-modelo $\mathcal{A} : \Sigma \rightarrow Pos$, continua a ser a álgebra $\mathcal{W}/\cong_{\mathcal{A}}$ o Σ_{po}-quociente desejado, *cf.* (3.49) e (3.50). Isto porque o fecho antissimétrico de uma ordem parcial é a própria igualdade. A questão que se põe agora é a seguinte: que Σ_{po}-quociente de $\mathcal{W}$ estaremos a definir sempre que escrevemos uma família

$$E = (E_s)_{s \in S} \tag{3.57}$$

de Σ-inequações?

3.4.4 O Sistema Lógico-Dedutivo DIN(E)

Tal como fizemos para o caso equacional, comecemos por apresentar o sistema lógico-dedutivo

$$DIN(E)$$

associado a E, e que é uma reafirmação do sistema $DEQ(E)$ uma vez retirada a regra de simetria:

R_1 *Reflexividade* — para todo o termo t,

$$\frac{}{t \leq t}$$

R_2 *Transitividade* —

$$\frac{t \leq t', t' \leq t''}{t \leq t''}$$

R_3 *Substituição* — para cada operador $\sigma : s_1 \times \ldots \times s_n \rightarrow s$ e $t_i, t_i' \in W_{\Sigma, s_i}$ $(1 \leq i \leq n)$,

$$\frac{t_1 \leq t_1', \ldots, t_n \leq t_n'}{\sigma(t_1, \ldots, t_n) \leq \sigma(t_1', \ldots, t_n')}$$

R_4 *Instanciação* — para $\rho \in V_{\mathcal{W}}$,

$$\frac{t \leq t'}{\rho(t) \leq \rho(t')}$$

R_5 *Equações* — para cada equação $(t, t') \in E$

$$\frac{}{t \leq t'}$$

Este sistema lógico-dedutivo dita — tal como no caso equacional — a construção da congruência $\simeq_E$ (contudo diferente de $\cong_E$) que fica definida pela apresentação de axiomas interpretados como inequações, *cf.* (3.57), tal como se segue.

Teorema 3.6 (Fecho por Pré-ordem) *Seja $R = (R_s)_{s \in S}$ uma família S-indexada de relações sobre um Σ_{po}-modelo $\mathcal{A}$, isto é, $R_s \subseteq \mathcal{A}(s) \times \mathcal{A}(s)$ para cada $s \in S$. Vamos designar por $R^{\oplus}$ o fecho reflexivo, transitivo e substitutivo de R,* i.é *a menor família de relações que satisfaz as cláusulas seguintes:*

1. Base*: para cada $s \in S$, $R_s^{\oplus} \supseteq R_s$*
2. Fecho reflexivo*: para cada $s \in S$, $R_s^{\oplus} \supseteq \{(a,a) \mid\ a \in \mathcal{A}(s)\}$*
3. Fecho transitivo*: para cada $s \in S$, se $aR_s^{\oplus}a'$ e $a'R_s^{\oplus}a''$ então $aR_s^{\oplus}a''$*
4. Fecho substitutivo*: seja $\sigma : s_1 \times \ldots \times s_n \to s$ com Σ-operador, e sejam $a_i, a_i' \in \mathcal{A}(s_i)$ tal que $a_i R_{s_i}^{\oplus} a_i'$, para $1 \leq i \leq n$; então*

$$\sigma(a_1, \ldots, a_i, \ldots, a_n) R_s^{\oplus} \sigma(a_1', \ldots, a_i', \ldots, a_n')$$

Tem-se que $R^{\oplus}$ é a menor pré-ordem gerada por R.

Demonstração*: imediata a partir da construção de $R^{\oplus}$.* □

Definição 3.16 (Pré-ordem Inequacional Mínima) *Seja $E = (E_s)_{s \in S}$ uma família de Σ-axiomas (t, t') que se pretendem interpretar como inequações $t \leq t'$. Seja $R = (R_s)_{s \in S}$ a família de relações $R_s \subseteq W_{\Sigma,s} \times W_{\Sigma,s}$ definidas por*

$$R_s = \{(\rho(t), \rho(t')) \mid\ (t, t') \in E_s \wedge \rho \in V_{\mathcal{W}}\}$$

Então à relação $\sqsubseteq_E = R^{\oplus}$ dá-se o nome de pré-ordem inequacional mínima *associada ao conjunto E de inequações.* □

Definição 3.17 (Σ_{po}-quociente Inequacional) *Seja $\simeq_E$ o fecho antissimétrico (Teorema 3.5) da pré-ordem equacional mínima $\sqsubseteq_E$ associada a um conjunto E de Σ-inequações. Então o Σ_{po}-quociente $\mathcal{W}/\sqsubseteq_E$, definido à custa de $\simeq_E$ —* cf. *Definição 3.15 — é a Σ_{po}-álgebra canónica para representar a semântica de um conjunto E de Σ-inequações.* □

Exercício 3.18 Relembre a assinatura do Exercício 1.8:

$$\begin{array}{rcl} \Sigma & : & \{0,1,2,\sigma\} \to S^* \times S \\ \Sigma(0) & = & \langle <>, s \rangle \\ \Sigma(1) & = & \langle <>, s \rangle \\ \Sigma(2) & = & \langle <>, r \rangle \\ \Sigma(\sigma) & = & \langle <s,s>, r \rangle \end{array}$$

Suponha que esta assinatura é acompanhada das seguintes Σ-inequações:

$$\begin{array}{l} 0 \leq 1 \\ \forall x \in r : 2 \leq x \end{array}$$

Calcule a pré-ordem inequacional mínima associada a este conjunto de Σ-inequações. (Use uma representação gráfica adequada.)

□

Reparemos que, se E é um conjunto vazio de axiomas, então a sua interpretação equacional e inequacional coincidem, *i.é* $\mathcal{W}/\cong_\emptyset$ coincide com $\mathcal{W}/\sqsubseteq_\emptyset$. Por outro lado, faz sentido definir axiomatizações "híbridas" combinando equações com inequações desde que cada uma das primeiras, da forma

$$t \equiv t' \tag{3.58}$$

seja considerada uma mera abreviatura de duas inequações,

$$t \leq t' \tag{3.59}$$

e

$$t' \leq t \tag{3.60}$$

Neste contexto, podemos finalmente voltar ao axioma (3.26), que na altura ficou incompleto e que agora conseguimos completar escrevendo a inequação

$$\texttt{while suc}(n)\ \texttt{do}\ c \leq c'$$

para quaisquer c, c' da espécie $<Cmd>$. Quer dizer, $W_{\Sigma,<Cmd>}$ forma uma ordem parcial plana onde o termo $\texttt{while suc}(n)\ \texttt{do}\ c$ pertence à classe de equivalência de todos os comandos que não terminam, *i.é* que têm semântica indefinida. A abreviatura (3.58) para a conjunção de (3.59) e (3.60) leva-nos a introduzir uma última regra em $DIN(E)$ para a *igualdade*:

R_6 *Igualdade* — para todos os termos t, t',

$$\frac{t \leq t', t' \leq t}{t \equiv t'}, \frac{t \equiv t'}{t \leq t', t' \leq t}, \frac{t \equiv t'}{t' \equiv t}$$

3.4.5 Especificação Inequacional versus Modelos

As definições e teoremas da Secção 3.3.2 para o caso equacional podem transportar-se para o caso inequacional sem dificuldade, feitas as óbvias adaptações. Por exemplo, uma Σ-inequação $t \leq t'$, sobre uma espécie $s \in S$, será satisfeita num Σ_{po}-modelo $\mathcal{A}$ desde que

$$\forall \rho \in V_\mathcal{A} : \rho(t) \leq_{\mathcal{A}(s)} \rho(t') \tag{3.61}$$

Dada uma colecção de Σ-inequações E, $\mathcal{A}$ satisfaz E sse o facto (3.61) se verificar para todas as inequações de E e a especificação inequacional onde E se insere, (Σ, X, E), dir-se-á uma *interface para* $\mathcal{A}$, tal como anteriormente. O quociente $\mathcal{W}/\sqsubseteq_E$ continua a ser o limite universal mínimo de todos os quocientes induzidos por Σ_{po}-modelos $\mathcal{A}$ que satisfazem E, e o Σ_{po}-modelo trivial $\mathcal{T}$ a ser o limite universal máximo. No reticulado de tais quocientes, os homomorfismos que ordenam os Σ_{po}-modelos são agora Σ_{po}-homomorfismos.

3.5 Alguma Notação Útil

Para finalizar, vamos introduzir uma notação concreta para a declaração, na prática, de especificações axiomáticas. Seja $I = (\Sigma, X, E)$ uma especificação axiomática (interface) de acordo com as Definições 3.7 e 3.10. Seja $S = \{s_1, s_2, \ldots, s_n\}$ o conjunto das suas espécies, $\Omega = \{\sigma_1, \ldots, \sigma_m\}$ o conjunto dos seus operadores, $V = \{v_1, \ldots, v_k\}$ o conjunto das variáveis que X envolve e $E = (t_i \leq t'_i)_{1 \leq i \leq l}$ o conjunto dos seus axiomas. A notação concreta que se utilizará será a seguinte:

$$\begin{array}{rl} interface & I \\ sorts & s_1, s_2, \ldots, s_n \\ ops & \sigma_1 : \ldots \times \ldots \times \ldots \to \ldots \\ & \sigma_2 : \ldots \times \ldots \times \ldots \to \ldots \\ & \vdots \\ & \sigma_m : \ldots \times \ldots \times \ldots \to \ldots \\ axioms & \forall v_1, \ldots, v_k : \;\; t_1 \leq t'_1 \\ & \qquad\qquad\qquad \vdots \\ & \qquad\qquad\quad\; t_l \leq t'_l \end{array} \tag{3.62}$$

Não existindo algumas das entidades presentes em (3.62), omitir-se-ão as respectivas entradas. Por exemplo, na seguinte declaração de interface,

$$\begin{array}{rl} interface & ITEM \\ sorts & Item \end{array} \tag{3.63}$$

não são referidas as entradas ops e $axioms$ pois não há nem operadores nem axiomas. Reparemos que, sendo $ITEM$ uma interface minimal [7], faz sentido tomá-la como ponto de partida para coisas mais elaboradas. Por exemplo, se acrescentarmos a $ITEM$ um operador binário,

$$\theta : ITEM \times Item \to Item$$

obtemos a interface $GRUPOID$ para grupóides,

$$\begin{array}{rl} interface & GRUPOID \\ sorts & Item \\ ops & \theta : Item \times Item \to Item \end{array} \tag{3.64}$$

etc.

[7] Relembra-se que, de acordo com a Definição 1.1, o conjunto S de espécies de uma assinatura é sempre não-vazio.

3.6 Exercícios

Exercício 3.19 Considere a assinatura

$$\Sigma = \left\{ \begin{array}{rcl} init & : & \rightarrow Db \\ add & : & Record \times Db \rightarrow Db \\ del & : & Key \times Db \rightarrow Db \\ pack & : & Db \rightarrow Db \\ mkInf & : & Str \rightarrow Inf \\ mkKey & : & Str \rightarrow Key \\ mkRec & : & Key \times Inf \rightarrow Record \\ mkStr & : & Char \rightarrow Str \\ strCons & : & Char \times Str \rightarrow Str \\ undel & : & Key \times Db \rightarrow Db \end{array} \right.$$

referente à gestão de um dicionário de registos cuja chave e informação são 'strings' (sequências de caracteres), com a seguinte funcionalidade:

> *Inicialização ($init$);*
> *Entrada de novo registo (add);*
> *Marcação de chave de registo a apagar (del);*
> *Empacotamento, ou destruição* de facto *de todos os registos marcados para apagar ($pack$);*
> *Recuperação de um dado registo, conhecida a sua chave ($undel$)).*

Tome como ponto de partida para a definição de um modelo $\mathsf{A} : \Sigma \longrightarrow \mathbf{Set}$, que capte formalmente a semântica acima, as seguintes cláusulas referentes a Σ-operadores:

$$\mathcal{A}(init) \stackrel{\text{def}}{=} \langle \left(\quad \right), \{\} \rangle$$

$$\mathcal{A}(add) \stackrel{\text{def}}{=} \lambda(r,\sigma).\ \begin{array}{ll} let & k = \pi_1(r) \\ & i = \pi_2(r) \\ & \sigma_1 = \pi_1(\sigma) \\ & \sigma_2 = \pi_2(\sigma) \\ in & \langle \sigma_1 \dagger \begin{pmatrix} k \\ i \end{pmatrix}, \sigma_2 \rangle \end{array}$$

$$\mathcal{A}(del) \stackrel{\text{def}}{=} \lambda(k,\sigma).\ \begin{array}{ll} let & \sigma_1 = \pi_1(\sigma) \\ & \sigma_2 = \pi_2(\sigma) \\ in & \langle \sigma_1, \sigma_2 \cup \{k\} \rangle \end{array}$$

$$\mathcal{A}(pack) \stackrel{\text{def}}{=} \lambda\sigma.\ \begin{array}{ll} let & \sigma_1 = \pi_1(\sigma) \\ & \sigma_2 = \pi_2(\sigma) \\ in & \langle \sigma_1 \setminus \sigma_2, \{\} \rangle \end{array}$$

$$\mathcal{A}(undel) \stackrel{\text{def}}{=} \lambda(k,\sigma).\ \begin{array}{ll} let & \sigma_1 = \pi_1(\sigma) \\ & \sigma_2 = \pi_2(\sigma) \\ in & \langle \sigma_1, \sigma_2 - \{k\} \rangle \end{array}$$

$$\mathcal{A}(mkInf) \stackrel{\text{def}}{=} \lambda s.s$$

$$\mathcal{A}(mkKey) \stackrel{\text{def}}{=} \lambda s.s$$

$$\mathcal{A}(mkRec) \stackrel{\text{def}}{=} \lambda(k,i).\langle k,i \rangle$$

$$\mathcal{A}(mkStr) \stackrel{\text{def}}{=} \lambda c.{<}c{>}$$

$$\mathcal{A}(strCons) \stackrel{\text{def}}{=} cons$$

1. Infira o resto do modelo $\mathcal{A}$ (espécies).
2. Formule um $\Sigma(X)$-axioma que registe a propriedade: *marcar um registo para apagar e a seguir recuperá-lo não altera o dicionário*. Verifique se o modelo proposto, $\mathcal{A}$, satisfaz esse axioma.

□

Exercício 3.20 Verifique quais das funções em $I\!N$ $\lambda(n,m).n \times m$ (multiplicação de naturais) e $\lambda(n,m).n^m$ (exponenciação em $I\!N$) são pontos fixos da funcional seguinte:

$$F(f) \stackrel{\text{def}}{=} \lambda(x,y).\begin{cases} x = 1 & \Rightarrow \quad y \\ x > 1 & \Rightarrow \quad y + f(x-1,y) \end{cases}$$

□

Exercício 3.21 Considere a seguinte gramática para um fragmento de uma linguagem de Lógica Temporal:

$$\begin{array}{rcl} G & = & \langle NT, T, \langle \textit{Fórmula} \rangle, P \rangle \\ NT & = & \{\langle \textit{Fórmula} \rangle, \langle \textit{Variável} \rangle\} \\ T & = & \{\ \texttt{(}\ ,\ \texttt{)}\ ,\ \texttt{True}\ ,\ \texttt{False}\ ,\ \texttt{not}\ ,\ \texttt{and}\ ,\ \texttt{or}\ ,\ \bigcirc\ ,\ \mathcal{U}\ ,\ \odot\ ,\ \mathcal{S}\ ,\ \texttt{p}\ ,\ \texttt{q}\ ,\ \texttt{r}\ ,\ \texttt{x}\ ,\ \texttt{y}\ ,\ldots\} \end{array}$$

$$P = \left\{ \begin{array}{lll} \langle \textit{Fórmula} \rangle & ::= & \langle \textit{Variável} \rangle \mid \\ & & \texttt{(}\ \langle \textit{Fórmula} \rangle\ \texttt{)} \mid \\ & & \texttt{True} \mid \texttt{False} \mid \\ & & \texttt{not}\ \langle \textit{Fórmula} \rangle \mid \qquad \textit{/*negação da lógica clássica */} \\ & & \langle \textit{Fórmula} \rangle\ \texttt{and}\ \langle \textit{Fórmula} \rangle \mid \qquad \textit{/*conjunção da lógica clássica */} \\ & & \langle \textit{Fórmula} \rangle\ \texttt{or}\ \langle \textit{Fórmula} \rangle \mid \qquad \textit{/*disjunção da lógica clássica */} \\ & & \bigcirc \langle \textit{Fórmula} \rangle \mid \qquad \textit{/*operador temporal 'next' */} \\ & & \langle \textit{Fórmula} \rangle\ \mathcal{U}\ \langle \textit{Fórmula} \rangle \mid \qquad \textit{/*operador temporal 'until' */} \\ & & \odot \langle \textit{Fórmula} \rangle \mid \qquad \textit{/*operador temporal 'last' */} \\ & & \langle \textit{Fórmula} \rangle\ \mathcal{S}\ \langle \textit{Fórmula} \rangle \qquad \textit{/*operador temporal 'since' */} \\ \langle \textit{Variável} \rangle & ::= & \texttt{p} \mid \texttt{q} \mid \texttt{r} \mid \texttt{x} \mid \texttt{y} \mid \ldots \end{array} \right.$$

Pretendendo-se dotar a linguagem de uma semântica com passado finito e futuro infinito numerável, definiram-se os seguintes domínios semânticos:

$$[\![\langle \textit{Fórmula} \rangle]\!] = [\![\langle \textit{Variável} \rangle]\!] \stackrel{\text{def}}{=} I\!N_0 \to 2$$

que permitem indexar no tempo linear (modelado por $I\!N_0$) os valores booleanos de cada fórmula ou variável.

Definiu-se então a semântica:

$$\left.\begin{array}{rcl} [\![\ \texttt{True}\ ]\!] & \stackrel{\text{def}}{=} & \lambda i.V \\ [\![\ \texttt{False}\ ]\!] & \stackrel{\text{def}}{=} & \lambda i.F \\ [\![f\ \texttt{and}\ f']\!] & \stackrel{\text{def}}{=} & \lambda i.[\![f]\!](i) \wedge [\![f']\!](i) \\ [\![f\ \texttt{or}\ f']\!] & \stackrel{\text{def}}{=} & \lambda i.[\![f]\!](i) \vee [\![f']\!](i) \\ [\![\ \texttt{not}\ f]\!] & \stackrel{\text{def}}{=} & \lambda i.\neg([\![f]\!](i)) \end{array}\right\} \text{lógica clássica (atemporal)}$$

$$
\left.\begin{array}{rcl}
[\![\bigcirc f]\!] & \stackrel{\mathrm{def}}{=} & \lambda i.[\![f]\!](i+1) \\
[\![f\, \mathcal{U}\, f']\!] & \stackrel{\mathrm{def}}{=} & \lambda i.(\exists j \geq i : [\![f']\!](j) \wedge \forall i \leq k \leq j : [\![f]\!](k))
\end{array}\right\}\ \text{lógica temporal “futura”}
$$

$$
\left.\begin{array}{rcl}
[\![\odot f]\!] & \stackrel{\mathrm{def}}{=} & \lambda i.\left\{\begin{array}{l} i = 0 \Rightarrow F \\ i > 0 \Rightarrow [\![f]\!](i-1) \end{array}\right. \\
[\![f\, \mathcal{S}\, f']\!] & \stackrel{\mathrm{def}}{=} & \lambda i.\left\{\begin{array}{l} i = 0 \Rightarrow F \\ i > 0 \Rightarrow \exists 0 \leq j < i : [\![f']\!](j) \wedge \forall j < k < i : [\![f]\!](k) \end{array}\right.
\end{array}\right\}\ \text{lógica temporal “passada”}
$$

$$
[\![\ (\ f\ )\ ]\!] \stackrel{\mathrm{def}}{=} [\![f]\!]
$$

$$
[\![v]\!]_{\langle \mathit{Fórmula} \rangle} \stackrel{\mathrm{def}}{=} [\![v]\!]_{\langle \mathit{Variável} \rangle} \tag{3.65}
$$

1. Desenhe a assinatura Σ_G que descreve G algebricamente.
2. Explique o significado da cláusula semântica (3.65).
3. Mostre que a semântica apresentada satisfaz o seguinte axioma

$$a\, \mathcal{S}\, b \equiv \odot(b \ \mathtt{or}\ (a \ \mathtt{and}\ (a\, \mathcal{S}\, b)))$$

4. Mostre que a mesma semântica satisfaz a inequação

$$\odot \bigcirc a \leq a$$

para a ordem $\leq$ (=*implicação lógica temporal*) sobre $I\!N_0 \to 2$ que se define por:

$$v \leq v' \stackrel{\mathrm{def}}{=} \forall n \in I\!N_0 : v(n) \Rightarrow v'(n)$$

5. Que frase da linguagem tem como semântica o limite inferior da ordem $\leq$? Justifique.

□

Exercício 3.22 Verifique se o modelo semântico definido no exercício 3.21 como semântica de uma pequena linguagem para Lógica Temporal satisfaz os seguintes axiomas:

$$
\begin{array}{rcl}
\odot(f\ \mathtt{and}\ f') & \equiv & (\odot f)\ \mathtt{and}\ (\odot f') \qquad (3.66) \\
\mathtt{not}\ (\bigcirc f) & \equiv & \bigcirc(\ \mathtt{not}\ f) \qquad (3.67)
\end{array}
$$

□

3.7 Notas Bibliográficas

No presente capítulo fez-se um apanhado da notação e principais resultados da teoria algébrica da especificação axiomática de tipos abstractos de dados, que o leitor poderá encontrar muito mais desenvolvida em livros como [Bp82], [EM85], [Hen88] e outros. Em particular, os teoremas da *adequação* e *completude* dos sistemas dedutivos $DEQ(E)$ e $DIN(E)$ — que atrás se omitiram por razões de economia de espaço — podem encontrar-se em [Hen88], respectivamente nas pp.37-38 e p.52.

Ficaram por tratar dois aspectos relevantes desta teoria. Primeiro, a especificação por *equações condicionais*, *i.é* axiomas da forma

$$(\bigwedge_{i=1}^{q} t_i \equiv t'_i) \Rightarrow t \equiv t' \tag{3.68}$$

(para $q = 0$ teremos $t \equiv t'$, *i.é* o caso estudado) que são muito relevantes na prática. A teoria subjacente é análoga à estudada. Contudo, o subreticulado de congruências que satisfazem um conjunto de axiomas condicionais pode não ser completo, isto na medida em que o $l.u.b.$ de duas congruências satisfazendo os axiomas pode não fazer o mesmo. Por exemplo (tirado de [BWP84]), sejam *solteiro*, *casado*, *viúvo*, *verdadeiro* e *falso* Σ-constantes de uma especificação com o axioma condicional

$$casado \equiv solteiro \Rightarrow verdadeiro \equiv falso.$$

Duas congruências $\sim$ e $\sim'$ tal que

$$\begin{array}{rcl} \textit{solteiro} & \sim & \textit{viúvo} \\ \textit{solteiro} & \not\sim & \textit{casado} \\ \textit{verdadeiro} & \not\sim & \textit{falso} \end{array}$$

e

$$\begin{array}{rcl} \textit{casado} & \sim' & \textit{viúvo} \\ \textit{solteiro} & \not\sim' & \textit{casado} \\ \textit{verdadeiro} & \not\sim' & \textit{falso} \end{array}$$

satisfazem os axiomas; contudo, o seu $l.u.b.$ $\sim \cup \sim' = \sim''$ é tal que

$$\begin{array}{rcl} \textit{solteiro} & \sim'' & \textit{viúvo} \sim'' \textit{casado} \\ \textit{verdadeiro} & \not\sim'' & \textit{falso} \end{array}$$

Isto prende-se com o segundo aspecto por tratar: o estudo dos modelos e congruências hierárquicas associadas a assinaturas construídas hierarquicamente, (e equipadas com axiomas condicionais), assunto do capítulo que se segue.

Finalmente, o tratamento das funções recursivas parciais é extensivamente tratado no conhecido livro de Zohar Manna [Man74].

Capítulo 4

Especificação Modular e Parametrização

4.1 Introdução

É conhecido que quaisquer ambições em programação estão limitadas pelo volume dos programas cujo 'debug' e manutenção se conseguem fazer. Quando um programa ultrapassa um dado tamanho... ninguém mais no mundo o consegue entender! Mas espera-se que cada uma das suas partes seja entendida por, pelo menos, uma pessoa, e que outras entendam como as partes encaixam umas nas outras.

Saber encaixar blocos de 'software' uns nos outros é o objectivo da chamada *programação em larga escala*, enquanto que a *programação em pequena escala* se dedica à construção de tais blocos de código, usando por exemplo variáveis, comandos de atribuição, ciclos 'while', procedimentos *etc.*

PASCAL, LISP, PROLOG *etc.* são exemplos de linguagens conhecidas para programação em pequena escala; já CLU, MESA, MODULA 2, ADA, OBJ, ML *etc.* são linguagens que se preocupam mais com primitivas — *e.g.* 'clusters', módulos, 'packages', 'abstract data types' *etc.* — para programação em larga escala.

Quando se combinam *módulos* uns com os outros temos que nos preocupar com a *interface* entre si. Uma interface informa-nos quais os requisitos e resultados de um módulo. É claro, as interfaces têm que ser verificadas umas perante as outras. Mas... o que é uma interface? E um módulo?

Infelizmente, conceitos básicos como estes aparecem nos manuais de programação de forma confusa, informal, e acabam por tornar-se em mais um nível de complexidade das (ditas) novas linguagens de programação.

Como temos visto, a *especificação matemática* de sistemas ou programas tem sido proposta como alternativa para aumentar o rigor e a inteligibilidade da documentação do 'software'. Infelizmente, o "síndrome" acima descrito que afecta programas grandes também afecta especificações acima de determinado tamanho. É claro, a quantidade de informação contida num determinado número de páginas de um documento matemático de especificação (*e.g.* um modelo em $Sets$) é incomparavelmente superior à informação contida no mesmo número de páginas de uma listagem de COBOL, ou FORTRAN. Portanto, já ganhamos alguma coisa. Mas, na verdade, apenas fizemos subir (ainda que significativamente) o volume de informação a partir do qual um problema deixa de ser "inteligível". Torna-se, portanto, necessária a divisão de uma especificação em partes orgânicas (módulos) que sejam 'mind-sized' e que, articuladas umas com as outras, produzam um resultado desejado. Diferentes articulações de partes elementares poderão produzir resultados diferentes; quer dizer, torna-se possível a *re-utilização* de especificações por outras especificações.

Algumas questões se levantam de imediato:

- terá significado matemático a "partição" de uma especificação em partes?
- qual o significado matemático da "junção" de sub-especificações?
- como garantir que essa junção produz um efeito desejado?

Fornecer respostas para estas questões no âmbito da metodologia de especificação *por modelos* que vem sendo exposta neste texto é o principal objectivo do presente capítulo. Começaremos por uma inspecção do "estado da arte" no tocante à programação modular. Dessa inspecção resultará a intuição que motiva a abordagem formal que se lhe seguirá.

4.2 Programas e Módulos: Revisão do Estado da "Arte"

Comecemos por alguns exemplos de programação cássica em, por exemplo, PASCAL. Pretendemos que seja feita uma ideia da prática e intuição comuns, a ponto de podermos evoluir para a caracterização matemática de conceitos como *implementação*, *interface* e *módulo*.

Consideremos o seguinte fragmento de programa em PASCAL:

```
type Point = ^record xcoord: Real;
                     ycoord: Real
             end;

function mkpoint(x:Real; y: Real): Point;
```

```
 var p: Point;
 begin new(p);
       p^.xcoord := x; p^.ycoord := y;
       mkpoint := p
 end;

procedure rotate(var p:Point; theta: Real);
 var ....;
 begin ....end;
```

... que pode ser modificado no sentido de se separarem, em secções de código distintas, as declarações de tipos, procedimentos e funções, dos corpos que constituem estes últimos.

É o que se pode fazer (*e.g.* em UCSD PASCAL) conduzindo a:

```
/* POINT Interface */

type Point = ^record xcoord: Real;
                     ycoord: Real
             end;
function mkpoint(x:Real; y: Real): Point;
procedure rotate(var p:Point; theta: Real);
```

quanto a declarações, e a:

```
/* POINT Implementation */

function mkpoint;
 var p: Point;
 begin new(p);
       p^.xcoord := x; p^.ycoord := y;
       mkpoint := p
 end;

procedure rotate;
 var ....;
 begin ....end;
```

quanto à implementação das entidades acima declaradas.

Aqui, uma "interface" introduz nomes (*e.g.* `Point`, `mkpoint` *etc.*) e associa-lhes um tipo, mas nenhum código executável; este é dado na correspondente implementação.

Não é difícil imaginar a construção de programas usando este tipo de estruturação; por exemplo, `POINT` poderia ser a base de uma nova entidade, `LINE`,

```
/* LINE Interface */

USES POINT;
type Line = record ... end;
function mkline(p:Point; q:Point): Line;
function intersects(l1:Line; l2:Line): Point;

/* Implementation of LINE */

.....
```

por sua vez conducente a `PICTURE`,

```
/* PICTURE Interface */

USES POINT, LINE;
type Pic = record ... end;
function mkpic(p:Point; al:array[...] of Line): Pic;
procedure display(p:Pic; var a:array[...] of boolean);

/* Implementation of PICTURE */
.....
```

e assim sucessivamente. Notar a palavra-chave `USES` que é empregue, nesta sintaxe [1], para indicar "colagem" de código (implementações) formulada a nível das correspondentes interfaces, encadeadas hierarquicamente.

No entanto, interfaces como `POINT`, `LINE` *etc.* acima são "demasiado informativas". Há muito tempo que as técnicas de abstracção de dados recomendam que, em casos como estes, o utilizador não precise de (nem deva!) saber "de que é que os pontos, as linhas *etc.* são feitos" transformando-se em *tipos abstractos de dados* (TADs) cujos pormenores de implementação não são "públicos". Somos conduzidos a novas versões das interfaces acima, que escondem agora as próprias descrições dos tipos de dados presentes, *e.g.*

```
/* POINT Interface */

type Point;
function mkpoint(x:Real; y: Real): Point;
procedure rotate(var p:Point; theta: Real);
```

[1] A sintaxe concreta da linguagem-exemplo que utilizaremos a partir de agora não é a de nenhuma linguagem comercial para programação modular. Trata-se de uma extensão 'ad hoc' ao PASCAL, intuitivamente simples e suficiente para ilustrar a prática que vem sendo adoptada neste domínio.

passando essas descrições para as correspondentes implementações, *e.g.*

```
/* POINT Implementation */

type Point = ^record xcoord: Real;
                     ycoord: Real
             end;
function mkpoint;
 var p: Point;
 begin .... end;

procedure rotate;
 var ....;
 begin ....end;
```

Uma boa ideia, a seguir, é separar interfaces das suas implementações, criando dois tipos de código fonte:

- documentos de *interface*, contendo todas as interfaces envolvidas num programa;

- documentos de *implementação*, contendo todo o código executável.

Esta estratégia é tanto mais correcta quanto, de facto, até pode haver mais do que uma maneira de implementar a mesma interface. Por exemplo, nada nos impede de escrever

```
/* POINT Second Implementation */
type Point = array [1..2] of Real;
...
```

aceitando ambas as implementações. Portanto, é em geral de *muitos para um* a relação entre implementações e interfaces, o que nos indica que dar o mesmo nome a interfaces e implementações não será praticável, afinal.

4.2.1 Parametrização

Até agora vimos implementações simples, construídas hierarquicamente. Acontece, por vezes, que a implementação não é fixa e depende de alguns parâmetros. Por exemplo, suponhamos agora que pretendemos trabalhar com pontos ordenados pelas suas coordenadas. Podemos aumentar a interface `POINT` nesse sentido, construindo:

```
/* OPOINT Interface */

USES POINT;
type Points;
function pointleast(p,q:Point): Point;
procedure pointsort(var a:Points);
```

acrescentando o tipo `Points` — "plural" de `Point` (*e.g.* `array of Point`) —, uma função `pointleast` capaz de escolher o menor de dois pontos segundo as suas coordenadas, e um procedimento `pointsort` capaz de ordenar pontos segundo `pointleast`. Quer dizer, uma implementação de `OPOINT` poderia ser a que a seguir se esboça:

```
/* OPOINT Implementation */

USES POINT;
type Points = array[1..n] of Point;
     index  = 1..n;
function pointleast;
var r:Point;
begin
  r := p;
  if p^.xcoord > q^.xcoord
     then r := q
     else if p^.xcoord = q^.xcoord
          then if p^.ycoord > q^.ycoord then r := q
pointleast := r
end;

procedure pointsort;

  function pointless(p,q:Point):Boolean;
    begin pointless := (p = pointleast(p,q)) end;
  procedure qsort(l,r:index);
    var i,j:index; x,w:Point;
    begin
    i := l; j := r; x := a[(l+r) div 2];
    repeat
     while pointless(a[i],x) do i := i+1;
     while pointless(x,a[j]) do j := j-1;
     if i<=j then
        begin w := a[i]; a[i] := a[j]; a[j] := w;
```

```
                i := i+1; j := j-1
          end
      until i>j;
      if l<j then qsort(l,j); if i<r then qsort(i,r)
      end;

   begin qsort(1,n) end;
```

onde `pointsort` usa o conhecido algorítmo de ‘quicksort’.

Suponhamos ainda que, no mesmo contexto, precisamos também de ordenar números segundo a sua ordem ($\leq$) habitual, com base na interface:

```
/* ONUM Interface */

type Num;
     Nums;
function numleast(p,q:Num): Num;
procedure numsort(var x:Nums);
```

com a implementação bem simples que se segue:

```
/* ONUM Implementation */

type Num = Real;
     Nums = array[1..n] of Real;

 function numleast;
   var r : Num;
   begin
     if (p <= q) then r := p else r := q; numleast := r
   end;

procedure numsort;
  numless(p,q:Num):Boolean;
    begin numless := (p = numleast(p,q)) end;
  procedure qsort(l,r:index);
    var i,j:index; x,w:Num;
    begin
    i := l; j := r; x := a[(l+r) div 2];
    repeat
     while numless(a[i],x) do i := i+1;
     while numless(x,a[j]) do j := j-1;
     if i<=j then
```

```
            begin w := a[i]; a[i] := a[j]; a[j] := w;
                  i := i+1; j := j-1
            end
        until i>j;
        if l<j then qsort(l,j); if i<r then qsort(i,r)
        end;

     begin qsort(1,n) end;
```

Reparemos agora que, coexistindo esta implementação com a de `OPOINT`, no mesmo programa, temos um certo grau de duplicação de código na media em que,

- quanto a interfaces, elas são "idênticas" à luz da correspondência que se segue:

OPOINT	ONUM
Point	Num
Points	Nums
pointleast	numleast
pointsort	numsort

- quanto às implementações, `Point` e `Num` são certamente diferentes, o que se reflecte nas funções `pointleast` e `numleast`. Já a construção de `Points` (sobre `Point`) é a mesma de `Nums` (sobre `Num`), o que torna `pointsort` a *mesma* função que `numsort`, a menos da correspondência acima.

Vem logo à ideia construir um procedimento *genérico* `sort` que deve ser informado não só quanto ao tipo dos elementos a ordenar, mas também quanto ao critério a seguir (ordem total). Uma implementação contendo funções ou procedimentos genéricos deste tipo designa-se um *módulo*, que poderá ser escrito, neste caso, como a seguir se ilustra:

```
/* SORT Module */
requires (type Elem; function least(e1,e2:Elem):Elem);
type Elems = array[1..n] of Elem;

procedure sort(var a:Elems);
........
```

Assim, quando escrevemos `SORT(num;numleast)` queremos designar `ONUM`. Para obtermos `OPOINT` bastará escrevermos `SORT(Point;pointleast)`.

Num módulo parametrizado, a cláusula adicional `requires` indica os seus requisitos, *i.é* parâmetros formais. Estes assumem a forma de, afinal, interfaces.

Faz portanto sentido que os correspondentes parâmetros reais venham a ser implementações suas.

Assim, o verdadeiro parâmetro de `SORT` é uma *interface* que, baptizada com o nome de `ORDERING`, se pode declarar escrevendo

```
interface ORDERING;
type Elem;
function least(e1,e2: Elem): Elem;
```

Reparemos que acabamos de introduzir na linguagem a 'keyword'

```
interface
```

para declarar interfaces (anteriormente, a informação sobre interfaces era dada em comentário). Quer dizer, assim como usamos `type` para declarar tipos, `procedure` para declarar procedimentos, *etc.*, também `interface` será usada para declarar interfaces. A mesma estratégia será usada para declarar implementações ('keyword' `implementation`) e módulos ('keyword' `module`). Assim,

```
implementation NUM@ORD: ORDERING;
type Elem = Real;
function least(i1,i2: Real): Real;
  begin if i1<=i2 then i1 else i2
  end;
```

designa a implementação de `ORDERING` que implementa a habitual ordem sobre reais. Outra implementação de `ORDERING` que nos interessa é

```
implementation POINT@ORD: ORDERING;
type Elem = Point;
function least(i1,i2: Point): Point;
  var j: Point;
  begin ...... (* cf. pointleast *)
  end;
```

que captura o parâmetro real de `OPOINT` acima.

Passemos agora à declaração de `SORT`, usando a 'keyword' `module` em lugar de `implementation` para indicar que `SORT` é parametrizado. `SORT` é implementação de quê? Temos que declarar primeiro a sua interface, qualquer coisa como

```
interface SORTING;
type Elems;
procedure sort(var a:Elems);
```

e então começar por escrever

```
module SORT ... : SORTING;
type Elems = ...
procedure sort(var a:Elems);
.....
```

Deixamos, propositadamente, reticências em lugar do parâmetro formal de `SORT` que é, como já vimos, a interface `ORDERING`. Para reforçarmos a ideia de que esta parametrização está em pé de igualdade com a de um procedimento ou função, usaremos a habitual notação (“Algólica)” para parâmetros [2], completando o cabeçalho da declaração de `SORT` como se segue:

```
module SORT(P:ORDERING): SORTING;
```

(prescindindo da cláusula de `requires`), onde `P` é o nome do único parâmetro de `SORT`. Por fim, é só completar o corpo deste módulo com o corpo já desenvolvido [3]:

```
module SORT(P:ORDERING): SORTING;
type Elems = array[1..n] of P.Elem;
     index  = 1..n;
procedure sort(var a:Elems);
  procedure qsort(l,r:index);
 var i,j:index; x,w:P.Elem;
 begin
   i := l; j := r;
  x := a[(l+r) div 2];
  repeat
          ... P.least(a[i],x) do ...
  until i>j;
  if l<j then qsort(l,j);
  if i<r then qsort(i,r)
 end;

begin qsort(1,n)
end;
```

Reparemos que o corpo de `SORT` acede à informação do seu parâmetro usando expressões da forma

$$\texttt{P.x} \tag{4.1}$$

---

[2] É o que acontece em linguagens como MESA ou CLEAR.

[3] Os parâmetros `P` e `Q` são do tipo `ORDERING`; assim, não é necessário listar explìcitamente os seus tipos individuais e procedimentos, como atrás.

onde `P` é o nome do parâmetro a ser acedido, e `x` é uma entidade sintática declarada na interface de `P` (`ORDERING`, neste caso).

Vejamos agora a construção, sobre `ORDERING`, de um novo módulo, `LEX@ORD`, para ordenação lexicográfica. Por exemplo, `LEX@ORD` poderá ser usado para ordenar sequências de pares

$$(nome, data)$$

primeiro por *nome* e, para iguais nomes, segundo uma ordem sobre *data*. Por exemplo, a sequência

$$\left(\begin{array}{r} (''gaspar'', 1956) \\ (''maria'', 1947) \\ (''gaspar'', 1966) \\ (''manuel'', 1980) \end{array}\right)$$

será ordenada:

$$\left(\begin{array}{r} (''gaspar'', 1956) \\ (''gaspar'', 1966) \\ (''manuel'', 1980) \\ (''maria'', 1947) \end{array}\right)$$

Portanto, essa ordem só poderá ser construída se se tiver, à partida, a ordem que se pretende para *nome* e aquela que se pretende para *data*. Logo, `LEX@ORD` é um módulo duplamente parametrizado:

```
module LEX@ORD (P: ORDERING, Q: ORDERING): ORDERING;
type Elem = record x: P.Elem;
                   y: Q.Elem
            end;
function least(e1,e2:Elem): Elem;
  var r: Elem;
      p1,p2: P.Elem;
      q1,q2: Q.Elem;
  begin p1 := e1.x; p2 := e2.x;
  if p1 = p2 then ... else ...
  least := r;
  end;
```

Para completarmos esta nossa exemplificação, notemos que `SORT` não é o único módulo “possível” para construir implementações de `SORTING`. Outros módulos, por exemplo

```
module SHELL@SORT(P: ORDERING): SORTING;
type Elems = ... ;
procedure sort(var a:Elems);
begin ... end;
```

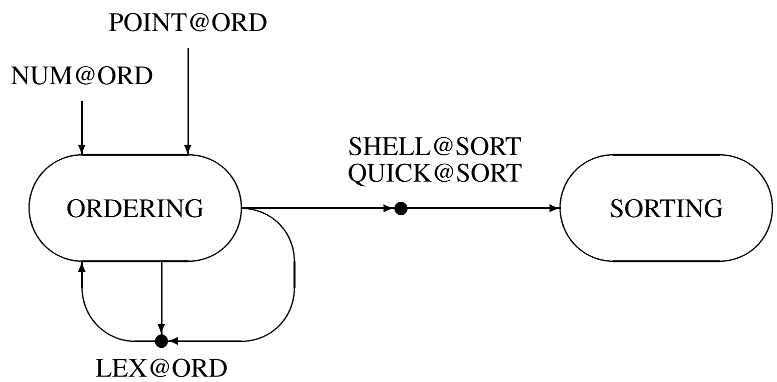


Figura 4.1: Diagrama ‘ADJ’ do plano de um programa.

são possíveis, diferindo quer nas estruturas de dados escolhidas, quer nos algoritmos presentes; aliás, vamos re-baptizar `SORT` para `QUICK@SORT`, para indicar mais claramente a sua natureza.

Finalmente, se dos exemplos acima construídos registarmos apenas a linha de cabeçalho de cada bloco de texto, obtemos uma ideia muito importante — o *plano* de todo o programa:

```
interface ORDERING, SORTING;
implementation NUM@ORD: ORDERING;
               POINT@ORD: ORDERING;
 module LEX@ORD (P: ORDERING, Q: ORDERING): ORDERING;
        SHELL@SORT(P: ORDERING): SORTING;
        QUICK@SORT(P: ORDERING): SORTING;
```

É neste ponto que acontece o “pensar-se em ponto grande”. Temos uma abstracção de todo o programa, omitindo todos os seus promenores. Trata-se de uma abstracção precisa e susceptível de mais refinamento. No exemplo em curso, tal abstracção é representável sob a forma do diagrama da Figura 4.1.

Podemos usar este diagrama para nos “divertirmos” a conceber vários programas, tal como os diagramas sintáticos de um manual de PASCAL nos ajudam a conceber *instruções*, *expressões etc.* Por exemplo, a expressão

```
LEX@ORD(NUM@ORD,NUM@ORD)
```

constrói uma implementação de uma ordem sobre pares ordenados de reais que é, afinal, a ordem (lexicográfica) sobre `Point` de que acima se falou. Ou ainda, podemos definir

```
QuSortNP := QUICK@SORT(LEX@ORD(NUM@ORD,POINT@ORD));
```

construindo o programa `QuSortNP` ('quicksort' sobre uma ordem lexicográfica cuja chave primária são reais e cuja chave secundária são pontos).

Em resumo, em programação em larga-escala (modular) um programa surge-nos como uma *expressão* envolvendo implementações e módulos, que pode ser 'type-checked' em termos da compatibilidade entre as interfaces envolvidas, exactamente da mesma maneira que as expressões a nível da programação em pequena-escala são 'type-checked' em termos da compatibilidade entre os tipos de dados envolvidos ('strong typing'). Qualquer segmento de código passa a "estar tipificado" e deixa de ser possível a mera colagem, ou ligação ('linking') de código, que não é estrutural e é propícia a erros.

Trata-se de uma maneira sistemática e económica de construir programas "grandes" re-utilizando código já desenvolvido ou a desenvolver.

Este tipo de estratégia para programação em larga-escala é a que está disponível em linguagens de programação como MESA e outras. Em que medida é rigorosa e fiável? Qual a sua semântica formal? Em que medida pode ser tudo isto transposto para o nível da *especificação formal* em "larga-escala"?

É o que se verá de seguida.

4.3 Modularidade em Especificação por Modelos

Passemos agora a uma tentativa de caracterização algébrica dos conceitos apresentados na secção anterior.

Do que atrás foi dito, depreende-se que a programação em larga escala tem procurado disciplinar as operações de "ligação" de código ('link') através de duas componentes:

- associação de um *tipo* a um bloco de código, designado por *interface* desse bloco.

- "tipificação" dos operadores de *ligação*, forçando-os a respeitar as interfaces dos módulos-argumento e a produzir resultados com tipos explícitos conhecidos.

Também não é dificil constatarmos que, se restringirmos *interfaces* à declaração de funções (*i.é* impedindo, para já, procedimentos),

- uma *interface* "é" uma assinatura Σ;

- uma *implementação* de uma interface Σ "é" uma Σ-álgebra, *i.é* um *modelo* de Σ;

- um *módulo* "é" uma operação de ligação que pega numa ou mais álgebras (que podem ter assinaturas diferentes), e dá como resultado é também uma álgebra.

O principal resultado desta disciplina é a constatação de que tais operadores de *ligação* formam, com as suas *interfaces*, assinaturas "modulares", Σ_M, que geram termos $t \in W_{\Sigma_M}$ que se designam vulgarmente por *expressões modulares* e que denotam a ligação estruturada, bem "tipificada", de blocos de código.

Mais concretamente, uma colecção de interfaces, implementações e módulos forma, ela própria, uma *(meta)assinatura*

$$\Sigma_M$$

cujas

- espécies são as *interfaces*;

- *constantes* são implementações de base;

- *módulos* são Σ_M-operadores (sobre implementações) [4].

Retomando o exemplo em curso, teremos a meta-assinatura

$$\Sigma_M : \Omega_M \to S_M^\star \times S_M$$

onde

$$S_M = \{ORDERING, SORTING\}$$

que contém os operadores

$$\left\{ \begin{array}{rcl} NUM@ORD & : & \to ORDERING \\ POINT@ORD & : & \to ORDERING \\ LEX@ORD & : & ORDERING \times ORDERING \to ORDERING \\ SHELL@SORT & : & ORDERING \to SORTING \\ QUICK@SORT & : & ORDERING \to SORTING \end{array} \right.$$

As implicações destes resultados só podem ser correctamente discutidas transpondo-os para a *especificação* modular. A este nível, permanece o conceito de interface axiomática, $I = (\Sigma, X, E)$ — que se pode escrever na notação que se propôs atrás,

[4] Notar que, logicamente, uma implementação pode e deve ser escrita sob a forma de um módulo sem parâmetros.

$$
\begin{array}{rl}
interface & ORDERING \\
sorts & Elem \\
ops & least : Elem \times Elem \to Elem \\
axioms & \ldots
\end{array}
\tag{4.2}
$$

cf. (3.62) — mas o de *programa*, ou segmento de *código executável* é substituído pelo seu arquétipo, *i.é* o seu modelo matemático. Todo o modelo $\mathcal{A}$ tem, pois, um tipo I — a sua interface,

$$\mathcal{A} : \ldots \to I$$

— e poderá ter parâmetros formais designados também por interfaces, *e.g.*

$$\mathcal{A} : X \to I$$

A apresentação do modelo assume assim a forma da definição de um operador,

$$
\begin{array}{rcl}
\mathcal{A} : X & \to & I \\
\mathcal{A}(x) & \stackrel{\mathrm{def}}{=} & \ldots
\end{array}
$$

cujo resultado deverá satisfazer a interface resultado I, quer dizer: se x_1 é um modelo de X, então $\mathcal{A}(x_1)$ é um modelo de I, *i.é*

$$\mathcal{A}(x_1) : I \to Sets$$

Vejamos um exemplo: suponhamos que queremos especificar o modelo

$$LEX@ORD : ORDERING \times ORDERING \to ORDERING \tag{4.3}$$

onde $ORDERING$ é a interface (4.2). Começaremos por escrever

$$LEX@ORD(P, Q) \stackrel{\mathrm{def}}{=} \ldots$$

onde as reticências devem ser preenchidas com a declaração dos modelos das espécies de $ORDERING$, neste caso uma só:

$$Elem \cong P.Elem \times Q.Elem \tag{4.4}$$

e dos modelos dos operadores, também um só, neste caso:

$$
\begin{array}{l}
least(e_1, e_2) \stackrel{\mathrm{def}}{=} \begin{array}[t]{ll}
let & p_1 = \pi_1(e_1) \\
 & q_1 = \pi_2(e_1) \\
 & p_2 = \pi_1(e_2) \\
 & q_2 = \pi_2(e_2) \\
in & \left\{ \begin{array}{lcl}
p_1 = p_2 & \Rightarrow & (p_1, Q.least(q_1, q_2)) \\
p_1 \neq p_2 & \Rightarrow & \begin{array}[t]{ll}
let & p = P.least(p_1, p_2) \\
in & \left\{ \begin{array}{lcl} p = p_1 & \Rightarrow & e_1 \\ p = p_2 & \Rightarrow & e_2 \end{array} \right.
\end{array}
\end{array} \right.
\end{array}
\end{array}
\tag{4.5}
$$

Juntando (4.4) com (4.5), teremos finalmente [5]:

$$LEX@ORD(P,Q) \stackrel{\text{def}}{=} \left\{ \begin{array}{rl} sorts & Elem \cong P.Elem \times Q.Elem \\ ops & least : Elem \times Elem \to Elem \\ & least(e_1, e_2) \stackrel{\text{def}}{=} \\ & \;\; \begin{array}{ll} let & p_1 = \pi_1(e_1) \\ & q_1 = \pi_2(e_1) \\ & p_2 = \pi_1(e_2) \\ & q_2 = \pi_2(e_2) \\ in & \left\{ \begin{array}{lll} p_1 = p_2 & \Rightarrow & (p_1, Q.least(q_1, q_2)) \\ p_1 \neq p_2 & \Rightarrow & \begin{array}{ll} let & p = P.least(p_1, p_2) \\ in & \left\{ \begin{array}{lll} p = p_1 & \Rightarrow & e_1 \\ p = p_2 & \Rightarrow & e_2 \end{array} \right. \end{array} \end{array} \right. \end{array} \end{array} \right. \tag{4.6}$$

Em termos formais, se designarmos por $\Sigma_{ORDERING}$ a assinatura associada à interface $ORDERING$, então não é difícil constatarmos que, se

$$P : \Sigma_{ORDERING} \to Sets$$

e

$$Q : \Sigma_{ORDERING} \to Sets$$

são dois modelos de $\Sigma_{ORDERING}$, então também

$$LEX@ORD(P,Q) : \Sigma_{ORDERING} \to Sets$$

é modelo de $\Sigma_{ORDERING}$, como queríamos. Portanto, $LEX@ORD$ é uma função binária sobre $\Sigma_{ORDERING}$-álgebras e dá como resultado uma $\Sigma_{ORDERING}$-álgebra.

Em resumo, se agruparmos numa assinatura Σ_M todos os símbolos de interface e todos os símbolos que designam modelos parametrizados, teremos uma interpretação 'standard' para a especificação modular correspondente, como se segue:

- as espécies de Σ_M são os símbolos que designam interfaces;
- os operadores de Σ_M são os nomes dos modelos parametrizados;
- as constantes de Σ_M são os modelos tal como foram entendidos nos capítulos anteriores, *i.é* sem parametrização;

[5] Note-se que a notação $P.Elem$, $Q.Elem$ *etc.* é adoptada aqui por ser "tradicional" em linguagens que admitem parametrização, *cf.* a equação (4.1). A natureza functorial de qualquer parâmetro — que é um modelo — levaria naturalmente a $P(Elem)$, $Q(Elem)$ *etc. cf.* Definição 1.7.

- o portador associado a uma dada interface I que é espécie de Σ_M é a classe $\mathcal{C}(I)$ de todas as álgebras que satisfazem I;

- a função associada a cada símbolo $\mathcal{M} : I_1 \times \ldots \times I_n \rightarrow J$ que é operador em Σ_M constrói uma J-álgebra desde que seja provido con n álgebras coerentes com a sua funcionalidade, *i.é* álgebras

  $$\mathcal{A}_i : I_i \rightarrow Sets$$

  para $1 \leq i \leq n$.

- as chamadas *expressões modulares* correspondem a termos gerados por Σ_M, *i.é* de W_{Σ_M}; se equipararmos Σ_M com variáveis $(\Sigma_M(X))$ teremos expressões modulares parametrizadas, *i.é* com "buracos" prontos a serem substituídos por sub-expressões modulares compatíveis.

Temos assim um mecanismo simples para construir álgebras (modelos) arbitrariamente elaborados à custa de sub-álgebras mais simples. Este mecanismo divide um modelo nos seus sub-modelos estruturalmente organizados, enquanto que convida à re-utilização, numa especificação, de sub-especificações já desenvolvidas.

Exercício 4.1 Considere a seguinte interface para gestão de informação organizada sob a forma de um dicionário:

$$\begin{array}{rl} interface & DIC \\ sorts & Key, Data, Dic \\ ops & \begin{array}{rcl} init & : & \rightarrow Dic \\ insert & : & Key \times Data \times Dic \rightarrow Dic \\ remove & : & Key \times Dic \rightarrow Dic \\ find & : & Key \times Dic \rightarrow Data \end{array} \end{array}$$

Relembrando a interface $ITEM$ (3.63), verifique se o seguinte módulo é um modelo de DIC:

$$DIC@SPEC \quad : \quad ITEM \times ITEM \rightarrow DIC$$

$$DIC@SPEC(C,I) \stackrel{\rm def}{=} \left\{ \begin{array}{llcl} sorts & Key & \cong & C.Item \\ & Data & \cong & I.Item \\ & Dic & \cong & Key \rightharpoonup Data \\ ops & init & \stackrel{\rm def}{=} & \left(\quad \right) \\ & insert(i,j,d) & \stackrel{\rm def}{=} & d \dagger \left(\begin{array}{c} i \\ j \end{array} \right) \\ & remove(i,d) & \stackrel{\rm def}{=} & d \setminus \{i\} \\ & find(i,d) & \stackrel{\rm def}{=} & \left\{ \; i \in dom(d) \;\; \Rightarrow \;\; d(i) \right. \end{array} \right.$$

□

Portanto, sempre que nos propomos especificar "qualquer coisa", devemos pensar primeiro — tal como já o fazíamos, aliás — na assinatura do modelo que nos propomos criar. Essa é a componente sintática da *interface* que "tipificará" o nosso modelo. Mas a interface poderá conter também informação semântica, fixada em termos de *axiomas*. Muitas vezes esses axiomas são propriedades desejáveis que o "cliente" da especificação teve o cuidado de registar nos seus requisitos. Outras vezes, os axiomas resultam do próprio processo de re-utilização de um modelo.

O modelo $LEX@ORD$ definido por (4.6) dá-nos um bom exemplo deste segundo caso, quanto à interface $ORDERING$ (4.2). Reparemos na definição de $least$ (4.5). Terminando esta definição com o teste, para $p = P.least(p_1, p_2)$

$$\begin{cases} p = p_1 & \Rightarrow \quad \ldots \\ p = p_2 & \Rightarrow \quad \ldots \end{cases}$$

está o especificador a assumir que esses dois testes são dicotómicos, *i.é* esgotam todas as possibilidades de valores de p; quer dizer, se $p_1 \neq p_2$, então

$$P.least(p_1, p_2) \equiv p_1 \vee P.least(p_1, p_2) \equiv p_2 \tag{4.7}$$

Onde está tal garantia, na interface $ORDERING$?

A definição (4.2) é omissa quanto a este requisito, que é semântico, aliás, e impõe que $ORDERING$ "seja" uma ordem total. Por exemplo, um modelo de $ORDERING$ em que $least$ seja a intersecção de dois conjuntos, ou o máximo divisor comum de dois inteiros ('g.l.b.'s de ordem parciais) não está nas condições impostas por (4.7) e, no entanto, satisfazem (4.2). De facto, é um "abuso" de nomenclatura chamar $ORDERING$ à interface definida por (4.2) — a designação $GRUPOID$ (3.64) é que seria apropriada!

Mais ainda, se o especificador tivesse escrito, em (4.6)

$$\ldots p = P.least(p_2, p_1)$$

em lugar de

$$\ldots p = P.least(p_1, p_2)$$

concerteza que gostaria que a definição de $least$ se não alterasse. Ou seja, está a ser assumida a propriedade *comutativa* do operador $least$:

$$least(p_1, p_2) \equiv least(p_2, p_1) \tag{4.8}$$

Em resumo, $LEX@ORD$ impõe a seguinte semântica aos seus parâmetros:

$$\begin{array}{rl} interface & ORDERING \\ sorts & Elem \\ ops & least : Elem \times Elem \rightarrow Elem \\ axioms & \forall e, e' : Elem \\ & least(e, e') \equiv least(e', e) \\ & least(e, e') \equiv e \vee least(e, e') \equiv e' \end{array} \tag{4.9}$$

Mas a própria especificação de $LEX@ORD$ sofre com isso, pois a sua interface de saída é $ORDERING$ também, *cf.* (4.3). Só se poderá dar o trabalho por terminado se se resolver a primeira alínea do primeiro dos exercícios que a seguir se propoem.

Exercício 4.2

1. Mostre que o modelo $LEX@ORD(A, B)$ satisfaz as duas equações da interface $ORDERING$ fixadas em (4.9), quaisquer que sejam os modelos

$$A, B :\rightarrow ORDERING$$

que as satisfaçam também.

2. Seja $\mathcal{T} :\rightarrow ORDERING$ o modelo trivial de $ORDERING$, *i.é*

$$\begin{array}{lll} \mathcal{T} : & \rightarrow & ORDERING \\ \mathcal{T} & \stackrel{\mathrm{def}}{=} & \begin{array}{ll} sorts & Elem \cong 1 \\ ops & least(e_1, e_2) \stackrel{\mathrm{def}}{=} e_1 \end{array} \end{array}$$

Mostre que, para todo o modelo m de $ORDERING$,

$$LEX@ORD(m, \mathcal{T}) \cong m \cong LEX@ORD(\mathcal{T}, m)$$

se verifica.

□

Exercício 4.3 Construa uma *interface axiomática* I (espécies + operadores + axiomas) adequada ao seguinte modelo para manipulação de grafos cujos nós são números naturais:

$$MOD@GRA :\rightarrow I$$
$$\left\{ \begin{array}{l} MOD@GRA(Node) \cong I\!N \\ MOD@GRA(Nodes) \cong 2^{Node} \\ MOD@GRA(Graph) \cong 2^{Node \times Node} \\ MOD@GRA(Bool) \cong 2 \\ MOD@GRA(init) \stackrel{\mathrm{def}}{=} \emptyset \\ MOD@GRA(allNodes) \stackrel{\mathrm{def}}{=} \lambda g.\pi_1[g] \cup \pi_2[g] \\ MOD@GRA(select) \stackrel{\mathrm{def}}{=} \lambda(a, g).\{t \in g \mid \pi_1(t) = a\} \\ MOD@GRA(sucs) \stackrel{\mathrm{def}}{=} \lambda(a, g).\pi_2[select(a, g)] \\ MOD@GRA(addArc) \stackrel{\mathrm{def}}{=} \lambda(a, a', g).g \cup \{\langle a, a' \rangle\} \\ MOD@GRA(isEmpty) \stackrel{\mathrm{def}}{=} \lambda g.g = \emptyset \end{array} \right.$$

(**NB**: Relembre o Exercício 2.28.)

□

Exercício 4.4 Repita o Exercício 4.1 supondo que à interface DIC se acrescentam os seguintes

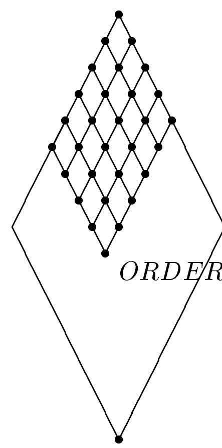


Figura 4.2: Especialização de uma interface.

axiomas:

$$
\begin{array}{lll}
axioms & \forall i : Key, j : Data, d : Dic & \\
 & find(i, insert(i, j, d)) \equiv & j \\
 & remove(i, insert(i, j, d)) \equiv & d \\
 & remove(i, init) \equiv & init \\
 & insert(i, j, insert(i, k, d)) \equiv & insert(i, j, d) \\
 & remove(i, remove(i, d)) \equiv & remove(i, d) \\
 & insert(i, j, remove(i, d)) \equiv & insert(i, j, d)
\end{array}
$$

□

4.3.1 Especialização e Classificação

Acabamos de ver como, uma vez definida uma interface ($ORDERING$) o exercício de desenvolvimento de um seu modelo ($LEX@ORD$) acarretou uma revisão dessa interface, no sentido de lhe impor mais propriedades. Podemos dizer que essa revisão foi no sentido de uma *especialização*, já que (*cf.* Figura 4.2) a classe de todos os modelos de $ORDERING$ após a revisão (4.9) é uma subclasse estrita da mesma classe para a versão inicial de $ORDERING$ (4.2) que é, afinal a classe de todos os *grupóides* livres.

Suponhamos que $ORDERING$ era uma interface, presente num grande sistema de 'software', da qual já existiam vários modelos antes do desenvolvimento de $LEX@ORD$. Qual o impacto da especialização de $ORDERING$? É óbvio que deveriam ser re-visitados todos os seus modelos no sentido de testar se as novas propriedades são verificadas ou não. Na negativa, tais modelos — ou o novo modelo $LEX@ORD$ — terão que ser "re-classificados", mudando-se o nome de uma das duas versões da interface em conflito.

É, pois, muito importante na prática reter o carácter "dinâmico" da classificação de módulos (modelos), que pode ser alterada como consequência da sua reutilização em contextos mais especializados.

4.4 Parametrização Parcial de Modelos

Recorde-se que o objectivo da modularidade é criar um método flexível para a construção 'mind-sized' de programas grandes.

É muitas vezes conveniente ter mais do que uma implementação de uma dada interface (por exemplo, uma implementação pode ser económica em espaço, mas lenta; uma outra, mais rápida mas mais extensa em código; uma outra, ainda, mais eficiente em ambientes paralelos, *etc.*).

Isso é difícil em MODULA 2 e ADA, por exemplo, que ligam interfaces a implementações 'by name'. Já MESA, PEBBLE, ML ou OBJ suportam múltiplas implementações, à partida.

No entanto, a múltipla implementação de uma interface traz problemas adicionais: como se indica que um módulo com dois parâmetros deve ser satisfeito por implementações que partilham uma sub-implementação comum?

Por exemplo, um módulo de álgebra linear poderá requerer dois parâmetros cujas interfaces sejam vectores e matrizes:

$$LIN@ALG : VECTORS \times MATRICES \rightarrow L@ALG@PACK$$

Mas isso só faz sentido se os vectores e as matrizes em questão forem constituídos sobre o mesmo tipo de elemento, algebricamente, o mesmo *corpo* (*e.g.* o corpo dos números reais). Comecemos, então, por redefinir:

$$LIN@ALG : FIELD \times VECTORS \times MATRICES \rightarrow L@ALG@PACK \tag{4.10}$$

Tendo, por exemplo, disponíveis os módulos:

$$\begin{array}{c} VECT : FIELD \rightarrow VECTORS \\ MAT : FIELD \rightarrow MATRICES \end{array}$$

só expressões modulares como,

$$\lambda f.LIN@ALG(f, VECT(f), MAT(f)) \tag{4.11}$$

"farão sentido", e não, por exemplo,

$$\lambda(f,v,f').LIN@ALG(f,v,MAT(f'))$$

nem

$$\lambda(f,f',f'').LIN@ALG(f,VECT(f'),MAT(f''))$$

para f, f' e f'' quaisquer. Definindo

$$LALG(f) \stackrel{\mathrm{def}}{=} LIN@ALG(f,VECT(f),MAT(f))$$

teremos que, por exemplo,

$$LALG(REALS)$$

é um modelo da álgebra linear sobre números reais, desde que

$$REALS :\rightarrow FIELD$$

seja um modelo conveniente do corpo real.

Exercício 4.5 Descreva as interfaces $FIELD$ (corpos algébricos) e $VECTORS$ (espaços vectoriais).
□

O facto de nem todas as expressões modulares, que são termos de W_{Σ_M}, terem "sentido", significa que existem operadores parciais em Σ_M, da mesma maneira que o termo $3+(2-3)$ não tem interpretação em $I\!N_0$ pelo facto de a operação de subtracção $(-)$ ser parcial em $I\!N_0$.

De facto, a definição de um modelo para $LIN@ALG$,

$$LIN@ALG(F,V,M) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{c} sorts \\ ops \\ \vdots \end{array} \right.$$

deverá impôr que a espécie definida pelo *corpo* F, *e.g.* $F.Field$, seja partilhada por V (sobre a qual são construídos vectores) e M (*ibid* para matrizes). Ou seja, $LIN@ALG$ terá uma pré-condição:

$$LIN@ALG(F,V,M) \stackrel{\mathrm{def}}{=} \{F.Field = V.Field = M.Field \Rightarrow \left\{ \begin{array}{c} sorts \ldots \\ ops \ldots \end{array} \right.$$

O exercício que se segue ilustra de forma simples a ocorrência, na prática, de modelos parciais.

Exercício 4.6 São dadas as interfaces seguintes:

$$\begin{array}{rl} interface & NOT \\ sorts & Bool \\ ops & 0,1 :\to Bool \\ & neg : Bool \to Bool \\ axioms & neg(0) \equiv 1 \\ & neg(1) \equiv 0 \end{array}$$

$$\begin{array}{rl} interface & OR \\ sorts & Bool \\ ops & 0,1 :\to Bool \\ & or : Bool \times Bool \to Bool \\ axioms & \forall p,q : Bool \\ & or(0,q) \equiv q \\ & or(1,q) \equiv 1 \\ & or(p,q) \equiv or(q,p) \end{array}$$

$$\begin{array}{rl} interface & NOR \\ sorts & Bool \\ ops & 0,1 :\to Bool \\ & nor : Bool \times Bool \to Bool \\ axioms & \forall p,q : Bool \\ & nor(p,q) \equiv nor(q,p) \\ & nor(1,q) \equiv 0 \\ & nor(0,0) \equiv 1 \end{array}$$

1. Verifique se $MKNOT : NOR \to NOT$

$$MKNOT(N) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{rl} sorts & Bool \cong N.Bool \\ ops & 0 :\to Bool \\ & 0 \stackrel{\mathrm{def}}{=} N.0 \\ & \\ & 1 :\to Bool \\ & 1 \stackrel{\mathrm{def}}{=} N.1 \\ & \\ & neg : Bool \to Bool \\ & neg(b) \stackrel{\mathrm{def}}{=} N.nor(b,0) \end{array} \right.$$

   é de facto modelo de NOT.

2. Complete a definição do modelo seguinte de OR:

$$\begin{array}{lcl} MKOR & : & NOR \times NOT \to OR \\ MKOR(X,Y) & \stackrel{\mathrm{def}}{=} & \{\ldots \Rightarrow \left\{ \begin{array}{rl} sorts & Bool \cong \ldots \\ ops & 0 :\to Bool \\ & 0 \stackrel{\mathrm{def}}{=} \ldots \\ & \\ & 1 :\to Bool \\ & 1 \stackrel{\mathrm{def}}{=} \ldots \\ & \\ & or : Bool \times Bool \to Bool \\ & or(p,q) \stackrel{\mathrm{def}}{=} \ldots \end{array} \right. \end{array}$$

□

Exercício 4.7 O conceito matemático de *multiconjunto*, apresentado no Exercício 1.38, é muito útil em especificação formal.

1. Considere a seguinte especificação equacional de uma interface para muticonjuntos:

$$\begin{array}{rl} interface & MULTISET < NAT0 \\ sorts & Multiset \\ ops & \phi :\to Multiset \\ & \oplus : Multiset \times Multiset \to Multiset \\ & count : Multiset \times Item \to Nat0 \\ & join : Multiset \times Item \to Multiset \\ axioms & \forall i \in Item, m \in Multiset \\ & count(\phi, i) \equiv 0 \\ & count(join(m,i),i) \equiv suc(count(m,i)) \end{array}$$

   construída hierarquicamente sobre:

$$\begin{array}{rl} interface & NAT0 < ITEM \\ sorts & Nat0 \\ ops & 0 :\to Nat0 \\ & suc : Nat0 \to Nat0 \\ & sum : Nat0 \times Nat0 \to Nat0 \end{array}$$

   Verifique que um modelo $\mathcal{A} : ITEM \to MULTISET$ em que

$$\begin{array}{rcl} \mathcal{A}(Nat0) & \cong & I\!N_0 \\ \mathcal{A}(Multiset) & \cong & Item \rightharpoonup I\!N \\ \mathcal{A}(sum) & \cong & \lambda(x,y).x+y \\ \mathcal{A}(0) & \cong & 0 \\ \mathcal{A}(\phi) & \cong & () \\ \mathcal{A}(suc) & \cong & \lambda(x).x+1 \\ \mathcal{A}(join) & \cong & \lambda(m,i).m \oplus \left(\begin{array}{c} i \\ 1 \end{array}\right) \\ \mathcal{A}(count) & \cong & \lambda(m,i).\left\{\begin{array}{lcl} i \in dom(m) & \Rightarrow & m(i) \\ i \notin dom(m) & \Rightarrow & 0 \end{array}\right. \end{array}$$

   e em que $\oplus$ é o operador definido por (1.102), satisfaz de facto as equações da interface $MULTISET$.

2. Acrescente à alínea anterior as definições dos operadores intersecção e diferença sobre multiconjuntos. Quais das equações (A.24) a (A.30) da álgebra de conjuntos se estendem a multiconjuntos?

□

4.5 Perspectiva Axiomática da Modularidade

Vimos nas secções anteriores que a interpretação "padrão" que fizemos do meta-nível de especificação Σ_M inclui os ingredientes típicos de qualquer modelo, inclusivamente operadores parciais.

Na presente secção vamos esboçar uma sua caracterização axiomática. Seja, dado a título de exemplo, o modelo trivial de $ORDERING$, *i.é* o modelo $\mathcal{T}$ definido por

$$\begin{array}{lcl} \mathcal{T} & : & \to ORDERING \\ \mathcal{T} & \stackrel{\text{def}}{=} & \left\{ \begin{array}{rl} sorts & Elem \cong 1 \\ ops & least(e_1, e_2) \stackrel{\text{def}}{=} e_1 \end{array} \right. \end{array}$$

Seja m um qualquer modelo de $ORDERING$. Tal como se viu no Exercício 4.1 não é difícil provar que

$$LEX@ORD(m, \mathcal{T}) \cong m \tag{4.12}$$

e que

$$LEX@ORD(\mathcal{T}, m) \cong m \tag{4.13}$$

onde $\cong$ designa *isomorfismo* entre $\Sigma_{ORDERING}$-álgebras. No mesmo exercício se viu que da expressão $LEX@ORD(m, \mathcal{T})$ se obtém, por substituição,

$$\left\{ \begin{array}{rl} sorts & Elem \cong m.Elem \\ ops & least(e_1, e_2) \stackrel{\text{def}}{=} (m.least(\pi_1(e_1), \pi_2(e_2)), NIL) \end{array} \right.$$

de um modelo de facto isomorfo de m — basta-nos lembrar da lei em *Sets*

$$A \times 1 \cong A$$

à qual está associada, como isomorfismo, a função de projecção π_1. Logo, temos (4.12) verificada, e o mesmo acontece para (4.13), usando a projecção π_2 como isomorfismo.

Que conclusão tirar de factos como (4.12) e (4.13)? É que, postulados abstractamente ao nível de Σ_M, *e.g.*

$$LEX@ORD(m, \mathcal{T}) \equiv m \equiv LEX@ORD(\mathcal{T}, m) \tag{4.14}$$

eles são, no fundo Σ_M-axiomas. $ORDERING$ pode ser ainda mais especializada, *e.g.* forçando a associatividade de $least$, *i.é* reescrevendo (4.9) para

$$\begin{array}{rl} interface & ORDERING \\ sorts & Elem \\ ops & least : Elem \times Elem \to Elem \\ axioms & \forall e, e', e'' : Elem \\ & least(e, e') \equiv least(e', e) \\ & least(e, e') \equiv e \vee least(e, e') \equiv e' \\ & least(least(e, e'), e'') \equiv least(e, least(e', e'')) \end{array}$$

Ora é possível provar a validade de mais um desses "meta"-axiomas:

$$\begin{aligned} LEX@ORD(LEX@ORD(m,n),p) &\equiv \\ LEX@ORD(m, LEX@ORD(n,p)) & \end{aligned} \tag{4.15}$$

ou seja, $LEX@ORD$ é um Σ_M-operador associativo. Juntando (4.14) com (4.15), estaremos em presença de uma *caracterização equacional* de $LEX@ORD$ bastante significativa, com a estrutura semântica de um monóide.

Faz também sentido pensarmos em caracterizações inequacionais. Por exemplo, entre dois $\Sigma_{ORDERING}$-modelos $\mathcal{A}$ e $\mathcal{B}$ tais que existe um $\Sigma_{ORDERING}$-homomorfismo de $\mathcal{B}$ para $\mathcal{A}$, podemos estabelecer o facto

$$\mathcal{A} \leq \mathcal{B} \tag{4.16}$$

Reparemos que qualquer interface I (*e.g.* $ORDERING$, $SORTING$ *etc.*) tem o seu modelo trivial $\mathcal{T}$, que é limite universal inferior de $\leq$ (4.16), *i.é*

$$\mathcal{T} \leq_I \mathcal{A}$$

para todo o modelo $\mathcal{A}$ de I. Quer dizer, estamos perante uma interpretação de Σ_M que é *parcialmente ordenada*, no sentido da Definição 3.11. Também será verdade que

$$\begin{array}{rcl} \mathcal{A} \leq \mathcal{B} \wedge \mathcal{B} \leq \mathcal{A} & \Rightarrow & \mathcal{A} \cong \mathcal{B} \\ \mathcal{A} & \leq & \mathcal{A} \\ \mathcal{A} \leq \mathcal{B} \wedge \mathcal{B} \leq \mathcal{C} & \Rightarrow & \mathcal{A} \leq \mathcal{C} \end{array}$$

ou seja, $\leq$ é uma Σ_M-pré-ordem, e faz sentido escrever Σ_M-inequações que registem relações úteis sobre Σ_M-expressões modulares.

Em síntese, acaba-se de mostrar que o "meta"-nível da especificação em larga-escala pode, ele próprio, ser especificado sob a forma axiomática (inequacional). No seu conjunto, a especificação em pequena e larga escalas gozam da mesma unidade conceptual, apenas se distinguindo no nível de abstracção em que é registada a informação sobre (grandes) sistemas de 'software'.

4.6 Exercícios

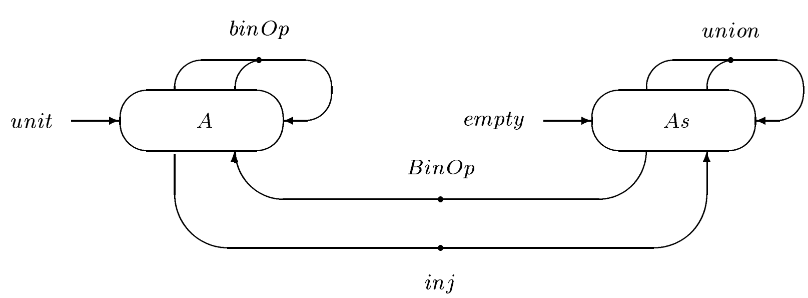
Exercício 4.8 A seguinte versão da interface $ORDERING$ (4.9)

$$\begin{array}{rl} interface & ORDERING \\ sorts & Elem \\ ops & least : Elem \times Elem \to Elem \\ & min : \to Elem \\ axioms & \forall e, e' : Elem \\ & least(e,e') \equiv least(e',e) \\ & least(e,min) \equiv min \\ & least(e,e') \equiv e \vee least(e,e') \equiv e' \end{array}$$

e o modelo $LEX@ORD$ (4.6) estão em desacordo. Reveja este último por forma a resolver tal diferendo entre interface e modelo.

□

Exercício 4.9 Considere uma interface axiomática J cuja assinatura subjacente é descrita pelo seguinte diagrama-ADJ,



que vem equipada com o seguinte conjunto de axiomas:

$$BinOp(empty) \equiv unit \tag{4.17}$$

$$BinOp(inj(i)) \equiv i \tag{4.18}$$

$$BinOp(union(x, y)) \equiv binOp(BinOp(x), BinOp(y)) \tag{4.19}$$

Suponha que alguém definiu o seguinte modelo,

$$\left\{\begin{array}{l}
MOD : I \longrightarrow J \\
MOD(A) \cong I.Item \\
MOD(As) \cong 2^A \\
MOD(empty) \stackrel{\mathrm{def}}{=} \emptyset \\
MOD(union) \stackrel{\mathrm{def}}{=} \lambda(x, y).x \cup y \\
MOD(unit) \stackrel{\mathrm{def}}{=} I.\epsilon \\
MOD(inj) \stackrel{\mathrm{def}}{=} \lambda a.\{a\} \\
MOD(binOp) \stackrel{\mathrm{def}}{=} I.\theta \\
MOD(BinOp) \stackrel{\mathrm{def}}{=} \lambda x. \left\{\begin{array}{rcll}
x = \emptyset & \Rightarrow & I.\epsilon & \\
\neg(x = \emptyset) & \Rightarrow & let & a \in x \\
 & & in & binOp(a, BinOp(x - \{a\}))
\end{array}\right.
\end{array}\right.$$

parametrizado por uma interface argumento I, e pretende agora demonstrar que se trata de facto de um J-modelo.

1. Mostre que, qualquer que seja I, MOD satisfaz pelo menos um dos três J-axiomas.
2. Descubra qual é a “menor” axiomatização de I que garante que MOD passa a satisfazer todos os J-axiomas.

□

Exercício 4.10 Na sequência da questão 4.9, mostre que o seguinte isomorfismo de J-álgerbras não se verifica,

$$MOD(\mathcal{T}_I) \quad \cong \quad \mathcal{T}_J \tag{4.20}$$

onde $\mathcal{T}_I$ e $\mathcal{T}_J$ designam respectivamente os modelos triviais das interfaces I e J.

□

Exercício 4.11 Seja $\leq$ uma ordem (dita de *implementabilidade*) sobre $ORDERING$-modelos, *i.é* para $\mathcal{A}, \mathcal{B} : \Sigma_{ORDERING} \rightarrow Sets$,

$$\mathcal{A} \leq \mathcal{B} \; sse \; \exists h : \mathcal{B} \rightarrow \mathcal{A} : h \text{ é um epimorfismo}$$

Verifique se $LEX@ORD$ (4.6) é um operador crescente quanto à ordem acima definida.

□

Exercício 4.12 Suponha um Σ_M-operador genérico para "ligação" de especificações,

$$\begin{array}{rcl} link & : & \Sigma_1 \times \Sigma_2 \rightarrow \Sigma \\ link(m_1, m_2) & \stackrel{\text{def}}{=} & m_1 \dagger m_2 \end{array}$$

que combina modelos de assinaturas $\Sigma_1 : \Omega_1 \rightarrow S_1^\star \times S_1$ e $\Sigma_2 : \Omega_2 \rightarrow S_2^\star \times S_2$ num modelo com assinatura

$$\Sigma = \Sigma_1 \dagger \Sigma_2$$

ie. $link(m_1, m_2)$ é a "sobreposição" do modelo m_2 no modelo m_1 (= "se m_2 exporta entidades que m_1 também exporta, então m_2 redefine-as").

Será *link* um "bom" operador de ligação? Estará bem definido? Será sempre possível sobrepôr dois modelos? Justifique a sua resposta.

□

Exercício 4.13 Considere o seguinte fragmento de um modelo $CFG : CFGI \rightarrow Sets$ para definição e manipulação de *gramáticas independentes de contexto* ("CFG" = 'context-free grammar'), onde

SYM designa em $Sets$ um domínio primitivo de "símbolos" que inclui o símbolo NIL:

$$\left\{\begin{array}{l}
CFG(Sym) \cong SYM \\
CFG(SetOfSyms) \cong 2^{SYM} \\
CFG(PairOfSetOfSyms) \cong SetOfSyms^2 \\
CFG(SeqOfSyms) \cong SYM^\star \\
CFG(Prod) \cong Sym \times SeqOfSyms \\
CFG(Grammar) \cong PairOfSetOfSyms \times Sym \times 2^{Prod} \\
CFG(Bool) \cong 2 \\
CFG(init) \stackrel{\text{def}}{=} \langle \emptyset, \emptyset, NIL, \emptyset \rangle \\
CFG(gramSym) \stackrel{\text{def}}{=} \lambda g.\pi_1(g) \\
CFG(ntermSym) \stackrel{\text{def}}{=} \lambda g.gramSym(g)(1) \\
CFG(termSym) \stackrel{\text{def}}{=} \lambda g.gramSym(g)(2) \\
CFG(axiom) \stackrel{\text{def}}{=} \lambda g.\pi_2(g) \\
CFG(prods) \stackrel{\text{def}}{=} \lambda g.\pi_3(g) \\
CFG(addNTSym) \stackrel{\text{def}}{=} \lambda(g,s).\langle gramSym(g) \dagger \begin{pmatrix} 1 \\ ntermSym(g) \cup \{s\} \end{pmatrix}, axiom(g), prods(g) \rangle \\
CFG(addTSym) \stackrel{\text{def}}{=} \lambda(g,s).\langle gramSym(g) \dagger \begin{pmatrix} 2 \\ termSym(g) \cup \{s\} \end{pmatrix}, axiom(g), prods(g) \rangle \\
CFG(addProd) \stackrel{\text{def}}{=} \lambda(g,p).\langle gramSym(g), axiom(g), prods(g) \cup \{p\} \rangle \\
CFG(isCFG) \stackrel{\text{def}}{=} \lambda g. \begin{array}[t]{ll} let & N = ntermSym(g) \\ & T = termSym(g) \\ & s = axiom(g) \\ & P = prods(g) \\ in & N \cap T = \emptyset \wedge s \in N \wedge \pi_1[P] \subseteq N \wedge \forall p \in P : elems(\pi_2(p)) \subseteq N \cup T \end{array}
\end{array}\right.$$

Após uma análise deste fragmento, construa a interface $CFGI$ por forma a CFG poder ser considerado de facto uma $CFGI$-álgebra, e descreva por palavras suas (breves!) o significado em CFG de cada operador de $CFGI$. Poderá $isCFG$ ser encarado como um invariante de $Grammar$? Justifique.

□

Exercício 4.14 Relembre do exercício 4.6 as interfaces NOT e NOR, bem como o modelo parametrizado $MKNOT : NOR \to NOT$. Mostre que

$$MKNOT(\mathcal{T}_{NOR}) \cong \mathcal{T}_{NOT}$$

onde $\mathcal{T}_I$ designa o modelo trivial de uma interface I qualquer. Será este resultado generalizável a qualquer modelo parametrizado $F : I \to J$, *i.é* será sempre verdade que

$$F(\mathcal{T}_I) \cong \mathcal{T}_J \ ?$$

Justifique as suas respostas.

□

Exercício 4.15 Suponha que ao modelo do exercício 4.13 se pretende acrescentar um operador

$$firstSyms : SeqOfSyms \times Grammar \to SetOfSyms$$

que calcule os 'first symbols' de uma dada sequência de símbolos gramaticais. Complete o seguinte esboço de especificação de $firstSyms$:

$$
firstSyms(\alpha, g) \stackrel{\text{def}}{=} \{\ldots \Rightarrow \left\{ \begin{array}{lcll}
\alpha = <> & \Rightarrow & \emptyset & \\
\neg(\alpha = <>) & \Rightarrow & let & \beta = tail(\alpha) \\
 & & & x = head(\alpha) \\
 & & & N = ntermSym(g) \\
 & & & T = termSym(g) \\
 & & & P = prods(g) \\
 & & in & \left\{ \begin{array}{lcl} x \in T & \Rightarrow & \{x\} \\ \neg(x \in T) & \Rightarrow & \cdots\cdots \end{array} \right.
\end{array} \right.
$$

□

Exercício 4.16 Recorde as funções $NatSort$ e $StrSort$ que ficaram por especificar no Exercício 2.75.

1. Interpretando-as cuidadosamente, no respectivo contexto, proponha um modelo parametrizado cuja interface de saída exporte uma única função $sort$ (especifique-a) e espécies associadas, e cuja interface de entrada importe o tipo de dados elementar que $sort$ usa para fazer a ordenação.

2. Indique expressões modulares que permitam obter, a partir de $sort$, as instâncias $NatSort$ e $StrSort$.

□

4.7 Notas Bibliográficas

A abordagem feita neste capítulo à especificação modular inspirou-se no trabalho de Burstall e Lampson sobre PEBBLE [BL84] — por sua vez derivada de uma análise sobre as sofisticadas primitivas modulares da linguagem MESA desenvolvida na Xerox PARC — *cf.* a seguinte citação extraída de [BL84]:

> "We believe that linking should not be described in a primitive and ad hoc special-purpose language; it deserves more systematic treatment. In our view the linking should be expressed in a functional applicative language, in which modules are regarded as *functions* from implementations to implementations. Furthermore this language should be typed, and the interfaces should play the rôle of types for the implementations. Thus we have the correspondence:
>
> implementation $\rightleftarrows$ value
> interface $\rightleftarrows$ type
> module $\rightleftarrows$ function"

Para um tratamento formal deste problema em termos de *tipos dependentes polimórficos* ver também [Bur84]. A teoria de Milner [Mil78] sobre tipos polimórficos de dados está subjacente a este tratamento.

A caracterização inequacional do nível modular (Σ_M) é muito relevante na prática, já que a ordem (4.16) regista a relação de *implementabilidade* entre modelos que satisfazem uma mesma interface. A referência [Oli90] mostra que é mesmo possível usar o cálculo de $Sets$ para raciocinar sobre essa ordem e obter assim um método transformacional para a síntese de implementações a partir das suas especificações.

A necessidade de se construirem bibliotecas de "componentes de 'software"' re-utilizáveis — essenciais a qualquer modelo de produção para "fábricas" de 'software' — é hoje de extrema relevância tecnológica, esperando-se um impacto significativo na "crise do 'software"' que persiste desde os anos 60 [E.S89]. A decomposição de especificações em componentes formalmente classificadas e combináveis segundo as suas interfaces é um passo decisivo nesse sentido.

Parte II

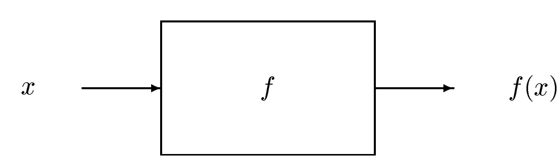
Especificação Não-Funcional

Capítulo 5

Semântica Relacional

5.1 Introdução

Nos capítulos anteriores estudamos uma técnica de modelação de problemas da vida real que assentava no conceito matemático de *função* enquanto descrição de uma actividade computacional, representada pelo processo de síntese de um resultado a partir de um ou mais argumentos:



Embora o paradigma funcional seja muito conveniente e útil em especificação, nem sempre se ajusta inteiramente aos problemas que se têm em mãos, e por vários motivos.

Em primeiro lugar, "especificar por funções" pode conduzir à sobre-especificação, *i.é* à fixação de uma decisão de projecto que seria desejável adiar até mais tarde no processo de desenvolvimento. Isto acontece sempre que há *indeterminação* nos requisitos. Por exemplo, se o problema que se nos põe é calcular uma das raízes de uma equação do 2.º grau,

$$ax^2 + bx + c = 0$$

se ela existir, poderemos optar entre dois modelos funcionais dessa actividade :

$$raiz(a,b,c) \stackrel{\mathrm{def}}{=} \frac{-b+\sqrt{b^2-4ac}}{2a} \tag{5.1}$$

ou

$$raiz(a,b,c) \stackrel{\mathrm{def}}{=} \frac{-b-\sqrt{b^2-4ac}}{2a} \tag{5.2}$$

No caso de existirem de facto duas raizes, nem (5.1) nem (5.2) são totalmente satisfatórias, pois optam sempre pela mesma dessas raízes quando o que se pedia é que fosse seleccionada uma dessas raízes, "aleatoriamente".

Ou seja, (5.1) ou (5.2) tomam uma decisão que se gostaria de adiar, ou melhor, encontram-se demasiado *determinadas*. O que verdadeiramente devemos fixar quanto à semântica de $raiz$ é a implicação formal

$$r = raiz(a,b,c) \Rightarrow ar^2 + br + c = 0$$

ou, se quisermos, a relação

$$\textit{é-raiz}(r,(a,b,c)) \Leftrightarrow ar^2 + br + c = 0$$

Reparemos que, ao contrário do que pode parecer à primeira vista, a indeterminação de especificações é desejável na prática, já que aumenta o seu espectro de implementabilidade, dando mais chance ao implementador de construir uma implementação correcta. É claro que isto só é verdadeiro se tal *indeterminação* corresponder aos requisitos em questão. Na negativa, incorre-se no erro de "sinal oposto", a *sub-especificação*, que conduzirá a implementações "incorrectas", ainda que aproximadas das desejáveis.

Mas há mais razões para trazer a indeterminação para o nível da especificação por modelos. É que muitos problemas da vida real são inerentemente não-deterministas. Por exemplo, suponhamos que queremos especificar a operação

$$getchar : Keyboard \to Char$$

pela qual um utilizador selecciona, de um dado teclado de caracteres disponíveis, aquele que no momento lhe interessa. O utilizador goza de "liberdade" suficiente para fazer escolhas imprevistas; portanto, $getchar$ nunca poderá ser modelado por uma função. A única coisa que podemos garantir sobre $getchar$ é que, para

$$Keyboard \cong 2^{Char}$$

então

$$c = getchar(k) \Rightarrow c \in k$$

ou, simplesmente,

$$getchar(k) \in k$$

Quer dizer, estamos a observar os resultados de $getchar$ como se de uma *relação* se tratasse.

Há, pois, componentes da interacção homem-máquina que só podem, assim, ser especificadas por modelos não-deterministas. De um modo geral, todos os problemas de comunicações contêm um certo grau de não-determinismo, dada a imprevisibilidade de se saber exactamente qual a unidade de informação que vai ser transmitida a seguir.

Noutros casos, os modelos não-deterministas são úteis para lidar com o nosso "desconhecimento" sobre a realidade. Por exemplo, um de três acontecimentos a_1, a_2, a_3 pode acontecer, aleatoriamente, no modelo, num dado instante, quando, afinal, se soubéssemos mais sobre a realidade em questão, talvez pudéssemos fixar que a_3 só acontece como consequência de a_1, por exemplo.

Vejamos agora, finalmente, uma terceira motivação para alargarmos a nossa noção de modelo-especificação. É que muitas transacções não são "inteiramente funcionais" no sentido em que os seus *resultados* não dependem apenas dos seus *argumentos*, mas também do valor de um "estado interno", *e.g.* a consulta a uma base de dados. Embora haja muitas situações em que esse estado interno se pode considerar um argumento adicional da função que modela a transacção, já não é tão natural a situação em que o processo de cálculo da função dá o seu resultado ao mesmo tempo que altera o valor desse estado interno, por efeito lateral ('side effect'). O exemplo tradicional desta situação é o operador POP sobre pilhas ('stacks'), cuja semântica vulgarmente se entende como dando no resultado o valor que se encontra acessível no topo da pilha (se existir), ao mesmo tempo que o retira da pilha.

5.2 Noção de Estado

A persistência temporal do "estado" nos objectos da vida real é uma constatação do dia-a-dia. Numa primeira aproximação, esse estado não é mais do que o produto cartesiano dos atributos característicos e persistentes desse objecto. Por exemplo, um *indivíduo* tem atributos como *idade*, *peso*, *altura*, *estado civil etc.* cujos valores mudam ao longo do tempo como consequência de *eventos* que afectam esse indivíduo (*e.g.* o divórcio altera o seu *estado civil*).

Uma noção "clássica" de *estado* é-nos transmitida pela teoria dos *autómatos de estados finitos* (AEFs). Aí se assume pura e simplesmente a existência de um conjunto finito Q de *etapas* ou *estádios* por que pode passar a "vida" do autómato enquanto *processo*, vida essa que é prescrita por uma tabela de transição de estados,

$$\delta \subseteq (Q \times I) \rightarrow (Q \times O)$$

que prevê a evolução do autómato de acordo com a sua reação a *estímulos* de um conjunto I; essa evolução traduz-se pelo 'output' de um valor de O e por uma

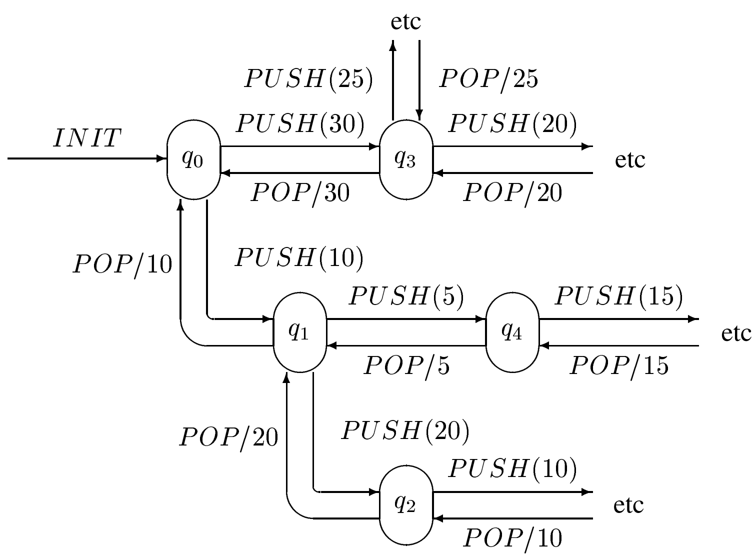


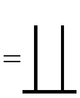
Figura 5.1: Fragmento de grafo descrevendo tabela de AEF para um 'stack' de inteiros.

transição de estado. Para exprimir indeterminismo, basta "relaxar" as tabelas de transição de estados, de funções para relações,

$$\delta \subseteq Q \times I \times Q \times O \tag{5.3}$$

Por exemplo, a Figura 5.1 representa um fragmento da tabela de um autómato de estados q_0, q_1, ...*etc.* que descreve o comportamento (determinístico) de um 'stack'. Trata-se de um grafo pesado por pares i/o onde $i \in I$ e $o \in O$. Neste caso I e O contêm 'strings' de caracteres representando entradas e saídas, respectivamente [1].

Qual é, então, o significado de cada estado q_i no digrama da Figura 5.1? Aparentemente, é um elemento de um conjunto de *posições* atómicas. Compare-se agora o diagrama da Figura 5.1 com o "desenho" da Figura 5.2 feito por alguém que, intuitivamente, nos tenta explicar o "funcionamento" de um 'stack'. As semelhanças são imediatas e o contraste é óbvio: o desenho obtém-se do diagrama substituindo cada *símbolo* q_0, q_1, ...*etc.* por um desenho de um 'stack', *e.g.*

$q_0 =$ 

[1] Aqui, o 'string' vazio ϵ é omitido, *e.g.* i/ϵ simplifica-se em i.

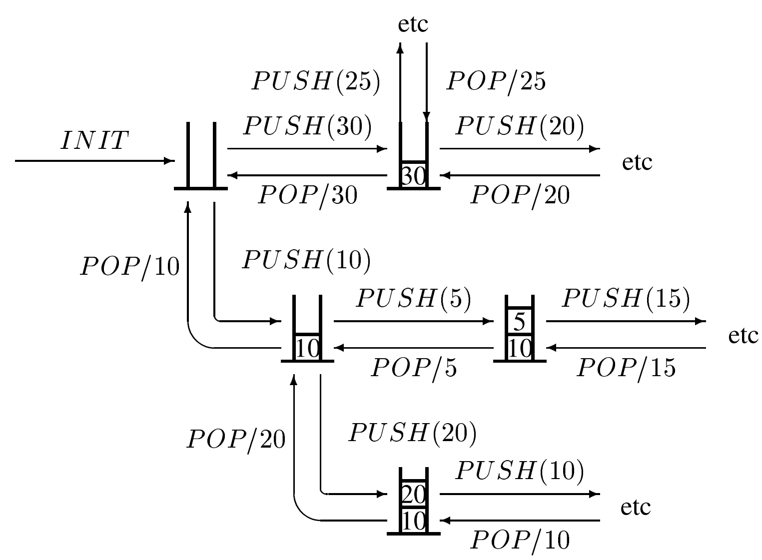
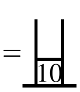
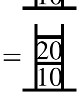


Figura 5.2: Desenho descrevendo um 'stack' de inteiros.

$q_1 =$ 

$q_2 =$ 

etc.

Como cada desenho pode, por sua vez, ser representado abstractamente por uma sequência de inteiros que regista a ordem de visita do correspondente 'stack', *e.g.*

$$\begin{aligned} q_0 &= <> \\ q_1 &= <10> \\ q_2 &= <20, 10> \end{aligned}$$

etc., chegamos à conclusão que o conjunto de estados Q "tem estrutura", pois

$$Q = I\!N^*$$

para este caso. Quer dizer, o *estado* não é mais que uma *estrutura de dados* que — na linha da hoje muito substancial tradição em linguagens de programação — "armazena" a informação "útil" sobre o passado do autómato até ao momento [2].

[2] Vale a pena reflectir sobre este ponto. Primeiro, reparemos que a capacidade expressiva de $Q =$

O estado de um objecto pode, assim, ser modelado por uma expressão em $Sets$ que designe um conjunto finito. Vejamos agora uma maneira simples de caracterizar matematicamente a semântica de uma transacção que altera o valor desse mesmo estado. Basear-se-á em relações que, em lugar de descreverem (explicitamente) tabelas de transição de estados, registam observações sobre o efeito da transacção, relacionando argumentos e estado *antes* da transacção, com o resultado e novo valor do estado *após* a transacção se realizar.

Reparemos que o conjunto I de 'inputs' em (5.3) pode ser discriminado em

$$I = E \times A$$

onde E é um conjunto de nomes de eventos e A um conjunto argumentos possíveis. Podemos então escrever

$$\begin{array}{rcl}
\delta & \subseteq & Q \times (E \times A \times O) \times Q \\
 & \Updownarrow & \\
\delta & \in & 2^{Q \times (E \times A \times O) \times Q} \\
 & \cong & \left(2^{Q \times A \times O \times Q}\right)^E \\
 & \cong & \left(2^{(Q \times A) \times (Q \times O)}\right)^E
\end{array}$$

Quer dizer, δ desdobra-se em tantas relações quantos os símbolos de enventos em E, pois, para cada $e \in E$,

$$\delta(e) \quad \subseteq \quad (Q \times A) \times (Q \times O)$$

Temos que prever casos particulares de eventos $e \in E$ sem argumentos e/ou resultados. No primeiro caso, é como se $A \cong 1$; ausência de 'outputs' corresponde a $O \cong 1$. Por exemplo, para $e = POP$, $Q = Stack$ definido por

$$Stack \cong X^* \tag{5.4}$$

$I\!N^*$ é ainda a de um AEF. Só que é mais "sugestiva" enquanto expressão da "memória" do autómato. Há, pois, uma nítida relação entre *estado* e *comportamento* no passado de um autómato.

Na prática, a programação por transição entre estados numéricos — que é uma herança do 'hardware', onde o estado é uma combinação de 'bits' (*cf.* a lógica sequencial, 'flip-flops' *etc.*) que tem, naturalmente, expressão numérica — foi sendo abandonada à medida que os autómatos se foram complicando, o número de estados foi aumentando e, concorrentemente, as linguagens de programação passaram a permitir a modelação do estado por estruturas de dados. Permaneceram apenas como 'target language' de soluções eficientes geradas automaticamente. Um exemplo eloquente é, sem dúvida, o da geração de autómatos de reconhecimento-LR feita, por exemplo, por geradores como `LEX/YACC`: cada estado do autómato é um item numérico que, afinal, codifica uma estrutura de informação complexa, representável por um conjunto de LR-items sujeito a um invariante não-trivial (*cf.* ϵ-*closure*).

e $A = O = X$, teremos que a semântica de POP,

$$\delta(POP) \subseteq \underbrace{Stack}_{entrada} \times \underbrace{(Stack \times X)}_{saída}$$

é uma relação de *entrada/saída* e não uma função, como foi habitual nos capítulos anteriores. Ora esta relação pode ser definida por compreensão. Seja $q \in Stack$ o estado corrente de uma dada pilha, $q' \in Stack$ o estado da mesma pilha após a ocorrência do evento POP e $r' \in X$ o presumível resultado. Então podemos observar o efeito seguinte:

$$q \neq <> \stackrel{POP}{\longrightarrow} \begin{array}{lcl} q' & = & tail(q) \quad \wedge \\ r' & = & head(q) \end{array}$$

quer dizer, a relação que descreve a semântica de POP é constituída por pares da forma

$$\langle q, \langle tail(q), head(q) \rangle \rangle$$

para todo o $q \neq <>$, *i.é*

$$\delta(POP) = \{\langle q, \langle tail(q), head(q) \rangle \rangle \mid q \in Stack \wedge q \neq <>\}$$

Em suma, para os eventos habituais de uma autómato de 'stack', teremos qualquer coisa como:

$$q \quad \xrightarrow{PUSH(n)} \quad q' = <n> \frown q$$

$$q \neq <> \quad \xrightarrow{TOP/head(q)} \quad q' = q$$

$$q \neq <> \quad \xrightarrow{POP/head(q)} \quad q' = tail(q)$$

$$q \quad \xrightarrow{EMPTY/(q = <>)} \quad q' = q$$

$$\xrightarrow{INIT} \quad q' = <>$$

Veremos agora, com mais detalhe, algumas das implicações teóricas deste tipo de especificação, nomeadamente no que diz respeito às necessárias extensões dos conceitos de *interface* e *modelo*.

5.3 Modelos com Estado Interno

O estilo de especificação que se esboçou na secção anterior tem uma larga tradição em ciência da especificação formal de sistemas. Na base dessa tradição está a constatação de que um sistema, uma máquina, um fragmento de programa, um *objecto* qualquer em geral, são entidades com propriedades persistentes e variáveis no tempo, cujo valor num dado instante define o seu *estado interno*.

Quando esse objecto interage com outros, é sujeito a estímulos aos quais pode reagir alterando o seu estado interno e/ou respondendo com a emissão de outros estímulos, e assim por diante. Cada etapa deste processo de interacção pode ser encarado como a ocorrência de um *evento*, cujo efeito é observável comparando a situação em que o objecto se encontrava, antes de aquele acontecer, com a situação consequente. O objecto reserva-se no direito de não aceitar todos os estímulos possíveis, dependendo da sua configuração corrente, emitindo possíveis mensagens de erro.

Como vimos, a "semântica" de um evento pode ser estabelecida por uma relação da forma

$$\rho \subseteq (Estado \times Argumentos) \times (Estado \times Resultado) \tag{5.5}$$

Definiremos o domínio de ρ como a projecção habitual, *i.é*

$$dom(\rho) = \{a \mid \ (a, b) \in \rho\}$$

Sempre que $dom(\rho) \subset Estado \times Argumentos$, temos uma relação *parcial i.é*, o evento não está definido para determinadas situações "à entrada". É vulgar caracterizar essas situações escrevendo um predicado — chamado a *pré-condição* do evento,

$$pre : Estado \times Argumentos \longrightarrow 2$$

— que mais não é que a *função característica* de $dom(\rho)$ em $Estado \times Argumentos$ [3].

Quanto à relação (5.5) propriamente dita, é vulgar caracterizá-la também pela sua função característica,

$$post : (Estado \times Argumentos) \times (Estado \times Resultado) \longrightarrow 2$$

a que é vulgar dar o nome de *pós-condição*, ou condição de *entrada/saída*.

Vejamos o exemplo do evento POP, já considerado atrás. Aqui, $Estado = Stack = X^*$, não há argumentos e $Resultado = X$, para qualquer X (conjunto de elementos a "empilhar"). Teremos então, neste caso,

$$\begin{array}{ll} pre_{POP} & : Stack \longrightarrow 2 \\ post_{POP} & : Stack \times (Stack \times X) \longrightarrow 2 \end{array}$$

[3] Chamamos função característica de um dado $S \subseteq A$ à imagem de S dada pelo isomorfismo $\mathcal{P}(A) \cong 2^A$.

Em face do que foi atrás exposto, é imediato definirmos

$$\begin{array}{ll} pre_{POP}(s) & \stackrel{\text{def}}{=} s \neq <> \\ post_{POP}(s,(s',r)) & \stackrel{\text{def}}{=} r = head(s) \wedge s' = tail(s) \end{array} \tag{5.6}$$

Fica assim completa a definição relacional da semântica do evento POP.

Quais as implicações formais de aceitarmos que um modelo tenha eventos? Quando tal acontece é porque o modelo em causa não é puramente funcional e tem *estado interno*. As definições de *interface* e de *modelo* que temos vindo a adoptar (*cf.* respectivamente as definições 3.10 e 1.7) terão que ser revistas. No primeiro caso, não é difícil imaginar como uma interface "funcional" possa ser aumentada com eventos — bastará dizer *quais* e qual a sua *operacionalidade*, *i.é* as espécies que determinam os seus argumentos e resultados. Podemos assim passar à definição que se segue.

Definição 5.1 (Interface com Eventos) *Seja* $I = (\Sigma, X, E)$ *uma interface axiomática, para* $\Sigma : \Omega \rightarrow S^\star \times S$. *Seja* Π *um conjunto de símbolos que são nomes de* eventos*, disjunto de* Ω*, sobre os quais se constrói uma aplicação*

$$\Gamma : \Pi \longrightarrow S^* \times S^*$$

caracterizando a "funcionalidade" de cada símbolo de evento.

Ao par $I' = \langle I, \Gamma \rangle$ *daremos o nome de* interface com eventos*, ou interface* procedimental. $\Box$

Esta definição exige algumas explicações. Em primeiro lugar, eventos e funções podem coexistir num mesmo modelo. Na nossa "meta-linguagem", uma interface com eventos escrever-se-á como a seguir se ilustra para o caso de $STACKS$:

$$\begin{array}{rl} interface & STACKS < ITEM \\ sorts & Stack, Bool \\ ops & isEmpty : Stack \rightarrow Bool \\ events & INIT :\rightarrow \\ & PUSH : Item \rightarrow \\ & POP :\rightarrow Item \end{array} \tag{5.7}$$

que enriquece

$$\begin{array}{rl} interface & ITEM \\ sorts & Item \end{array} \tag{5.8}$$

A parte funcional Σ da interface (5.7) é, assim,

$$\Sigma \quad = \quad \left(\begin{array}{c} isEmpty \\ \langle <Stack>, Bool \rangle \end{array} \right)$$

enquanto que Γ envolve três eventos,

$$\Pi = \{INIT, PUSH, POP\}$$

Reparemos que, enquanto uma função como $isEmpty$ está condicionada à produção de um e um só resultado, um evento poderá produzir vários ou mesmo nenhum. Por exemplo, $INIT$ é um evento de inicialização, sem argumentos nem resultados, *i.é* $\Gamma(INIT) = \langle <>, <> \rangle$. Genericamente, todo o evento τ tal que $\Gamma(\tau) = \langle <>, <> \rangle$, sem argumentos nem resultados, limita-se a ler e a afectar o estado.

Em segundo lugar, uma interface como $STACKS$ pressupõe a existência de um estado interno que, contudo, não é aí explicitado e só o virá a ser no modelo semântico respectivo. Por exemplo, vejamos o seguinte modelo para $STACKS$, parametrizado em $ITEM$ e baseado em sequências, como é habitual. Começaremos por escrevê-lo na nossa linguagem, deixando para depois as explicações devidas e a correspondente formalização:

$$\begin{array}{rcl}
STACKLIST & : & ITEM \rightarrow STACKS \\
STACKLIST(E) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{rl}
sorts & Item \cong E.Item \\
 & Stack \cong Item^{*} \\
ops & isEmpty(s) \stackrel{\mathrm{def}}{=} s = <> \\
state & Stack \\
events & INIT \stackrel{\mathrm{def}}{=} \sigma' = <> \\
 & PUSH(i) \stackrel{\mathrm{def}}{=} \sigma' = cons(i, \sigma) \\
 & POP(j') \stackrel{\mathrm{def}}{=} \{\sigma \neq <> \Rightarrow \begin{array}{ll} j' = head(\sigma) & \wedge \\ \sigma' = tail(\sigma) & \end{array}
\end{array} \right.
\end{array} \tag{5.9}$$

Comentários a (5.9):

- Os modelos para $Item$, $Stack$ e $isEmpty$ definem-se como habitualmente, *cf.* Capítulo 4.

- A cláusula $state\ \ldots$ especifica o estado interno de $STACKLIST$, neste caso uma sequência de $Items$. É uma convenção habitual reservar as variáveis σ e σ' para denotar os valores do estado, respectivamente antes e depois da ocorrência de um evento.

- $INIT$ não tem argumentos nem resultados; a sua semântica será pois uma relação
$$post_{INIT} \subseteq Stack \times Stack$$
Neste caso, optou-se por $post_{INIT}$ tal que
$$\sigma\ post_{INIT}\ \sigma' \Leftrightarrow \sigma' = <>$$

- $PUSH$ tem um argumento do tipo $Item$; a sua semântica será então uma relação
$$post_{PUSH} \subseteq Stack \times Item \times Stack$$
tendo-se definido
$$post_{PUSH}(\sigma, i, \sigma') \Leftrightarrow \sigma' = cons(i, \sigma)$$

- POP não tem argumentos e dá um resultado em $Item$, desde que a sua pré-condição se verifique, *cf.* (5.6). Para eventos com pré-condição (eventos parciais) adopta-se uma notação semelhante à notação para funções parciais, da forma
$$EVENTO(\ldots) \stackrel{\mathrm{def}}{=} \left\{ \; pre_{EVENTO}(\sigma, \ldots) \Rightarrow post_{EVENTO}(\sigma, \ldots, \sigma', \ldots) \right.$$
cf. POP em (5.9).

- Para simplificar a leitura, usa-se a convenção de associar variáveis simples (*e.g.* i, σ) para argumentos ou estado à entrada, e variáveis decoradas (*e.g.* σ', j') para estado ou resultado à saída.

Quer dizer, se τ é um evento tal que $\Pi(\tau) = \langle <s_1, \ldots, s_n>, <s> \rangle$ então, num modelo de τ, esperamos qualquer coisa como:

$$\begin{array}{rcl} pre_\tau & : & Q \times s_1 \times \ldots \times s_n \rightarrow 2 \\ post_\tau & : & Q \times s_1 \times \ldots \times s_n \times s_0 \times s \rightarrow 2 \end{array}$$

onde Q designa a classe de estados do modelo.

Note-se que a designação *interface procedimental* acima pretende reflectir o facto de um *procedimento* (*e.g.* em PASCAL) poder ser sempre considerado como uma codificação da semântica de um evento. Por exemplo, quando escrevemos

```
procedure P (var s:  Estado; a:  Argumento);
```

ou ainda

```
var s:  Estado;
⋮
procedure P (a:  Argumento);
⋮
```

estamos a pensar no evento P tal que

$$\begin{array}{ll} pre_P & : Estado \times Argumento \rightarrow 2 \\ post_P & : Estado \times Argumento \times Estado \rightarrow 2 \end{array}$$

Estamos agora em condições de formalizar o conceito de modelo de uma interface com eventos.

Definição 5.2 (Modelo com Eventos) *Seja $J = (I, \Gamma)$ uma interface com eventos tal como foi definida atrás (Definição 5.1). Um J-modelo $\mathcal{A}$ é um triplo $\langle \mathcal{A}_I, Q, \mathcal{A}_\Gamma \rangle$ tal que:*

1. *$\mathcal{A}_I$ é um I-modelo,* i.é *uma Σ-álgebra que satisfaz todos os axiomas em E,* cf. *Definição 5.1.*
2. *Q é uma designação do conjunto finito (ou infinito numerável) de estados internos de $\mathcal{A}$.*
3. *$\mathcal{A}_\Gamma$ define a tabela de transição de estados do modelo da forma seguinte: para cada símbolo de evento $\tau \in \Pi$ tal que $\Gamma(\tau) = (w, w')$ e $w = <s_1, \ldots, s_n>$, $w' = <s'_1, \ldots, s'_m>$, $\mathcal{A}_\Gamma(\tau)$ é uma relação binária em*

$$2^{(Q \times S_1 \times \ldots \times S_n) \times (Q \times S'_1 \times \ldots \times S'_m)}$$

onde S_i abrevia $\mathcal{A}_I(s_i)$ e S'_j abrevia $\mathcal{A}_I(s'_j)$.

Abreviaremos ainda $\mathcal{A}_\Gamma(\tau)$ em $\mathcal{A}(\tau)$ e $\mathcal{A}_I(x)$ em $\mathcal{A}(x)$. A função característica de $\mathcal{A}(\tau)$ designa-se pós-condição *de τ (em $\mathcal{A}$). Dá-se ainda a designação de* pré-condição *de τ em $\mathcal{A}$ à função característica do domínio da relação $\mathcal{A}(\tau)$.*
$\square$

O facto de o modelo de cada evento ser uma relação e não uma função justifica, só por si, o título "semântica relacional" que se deu a este capítulo. Como relações são mais gerais que funções, temos que qualquer especificação funcional pode ser convertida numa especificação relacional. Em muitos casos, é mesmo desejável essa conversão, não sendo difícil encontrar um critério prático para se decidir se uma especificação é "naturalmente" relacional ou não.

Veja-se, a título de exemplo, o problema $SGIB$ que foi usado extensivamente para ilustrar o Capítulo 1. Repara-se imediatamente que todos os operadores de Σ_{SGIB} contêm a espécie $Sistema$ como espécie-argumento. Mais do que isso, apenas um (*balanço*) não tem essa espécie como espécie-resultado. Esta "omnipresença" de $Sistema$ faz-nos pensar em omitir esta espécie de Σ_{SGIB}, convertendo os operadores que a manipulam em *eventos* que a assumam como estado interno. Por exemplo, ao operador

$$levantamento : IdConta \times Quantia \times Sistema \to Sistema$$

far-se-ia corresponder o evento

$$LEVANTAMENTO : IdConta \times Quantia \to$$

Note-se que $levantamento$ pode co-existir com $LEVANTAMENTO$. E até é desejável que isso aconteça, pois fica assim documentada a sua essência funcional e a sua origem no processo de desenvolvimento. Assim, se

$$\mathcal{A}(levantamento) \stackrel{\mathrm{def}}{=} \lambda(id, q, s) \ldots$$

já tiver sido definida num modelo $\mathcal{A}$, então é imediato definirmos

$$\mathcal{A}(LEVANTAMENTO)(id, q) \stackrel{\mathrm{def}}{=} \sigma' = \mathcal{A}(levantamento)(id, q, \sigma)$$

e assim para todos os outros operadores.

Exercício 5.1 Complete a especificação dos seguintes eventos genéricos, válidos sobre qualquer modelo com estado interno Q:

$$\begin{array}{rcll}
NOP & : & \rightarrow & \\
NOP & \stackrel{\mathrm{def}}{=} & \{ \ \ldots \ \Rightarrow \ \ldots & \text{/*evento que nada faz */} \\
NOISE & : & \rightarrow & \\
NOISE & \stackrel{\mathrm{def}}{=} & \{ \ \ldots \ \Rightarrow \ \ldots & \text{/*evento que altera indeterministicamente o estado */} \\
ABORT & : & \rightarrow & \\
ABORT & \stackrel{\mathrm{def}}{=} & \{ \ \ldots \ \Rightarrow \ \ldots & \text{/*evento que nada mais deixa acontecer */} \\
SET & : & Q \rightarrow & \\
SET(q) & \stackrel{\mathrm{def}}{=} & \{ \ \ldots \ \Rightarrow \ \ldots & \text{/*evento que altera deterministicamente o estado */}
\end{array}$$

□

Exercício 5.2 Pretende-se especificar a unidade de recepção de mensagens de um sistema de *correio electrónico*, a basear em duas partes estruturais: o *descodificador de endereços* e a *mailbox*. Entende-se por *endereço* uma sequência de *domínios*,

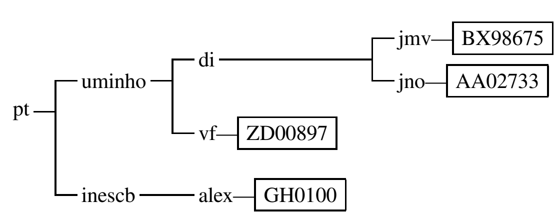
$$Endereco = Dominio^{\star}$$

normalmente listada por ordem inversa. Por exemplo,

`jno@di.uminho.pt`

é a representação habitual do endereço $<pt, uminho, di, jno>$.

A *mailbox* regista, para cada caixa de correio, a fila de mensagens que ainda não foram lidas. O *descodificador de endereços* é uma hierarquia de domínios que representa o processo de 'routing' de mensagens, *e.g.*



Assim, as folhas desta hierarquia serão identificadores de caixas de correio da *mailbox*:

$$\begin{array}{rcl}
Descodificador & \cong & Dominio \rightharpoonup SubDominio \\
SubDominio & \cong & Descodificador + Caixa
\end{array}$$

Especifique a unidade pretendida, não omitindo os eventos seguintes:

- $INI : \to$. Operação de inicialização da unidade.
- $REC : Endereco \times Mensagem \to$. Operação correspondente à chegada de uma mensagem à unidade. O endereço começa por ser descodificado e, sendo existente e suficiente, a mensagem é transmitida à caixa correspondente na *mailbox*.
- $LER : Endereco \to Mensagem$. O utente da caixa de correio endereçada lê a sua próxima mensagem, que é então retirada da *mailbox*.

□

Exercício 5.3 Relembre o Exercício 2.19 sobre a especificação funcional de *árvores de decisão*.

Suponha que se propôs o seguinte modelo como especificação não funcional de um gestor de árvores de decisão:

$$DEC@TREE \quad : \quad ITEM \times ITEM \to DTREE$$

$$DEC@TREE(A,W) \stackrel{\text{def}}{=} \left\{ \begin{array}{llll} sorts & What & \cong & W.Item \\ & Answer & \cong & A.Item \\ & Menu & \cong & 2^{Answer} \\ & DecTree & \cong & What \times (Answer \rightharpoonup DecTree) \\ ops & \multicolumn{3}{l}{menu(dt) \stackrel{\text{def}}{=} dom(\pi_2(dt))} \\ state & \multicolumn{3}{l}{DecTree} \\ events & LOAD(dt) & \stackrel{\text{def}}{=} & \sigma' = dt \\ & MENU(r') & \stackrel{\text{def}}{=} & \sigma' = \sigma \wedge \\ & & & r' = menu(\sigma) \\ & ASK(q') & \stackrel{\text{def}}{=} & \sigma' = \sigma \wedge \\ & & & q' = \pi_1(\sigma) \\ & ANSWER(a) & \stackrel{\text{def}}{=} & \{a \in menu(\sigma) \Rightarrow \\ & & & \sigma' = \pi_2(\sigma)(a) \end{array} \right.$$

1. Após uma análise cuidadosa deste modelo com eventos, faça a inferência da interface $DTREE$ que ele satisfaz.
2. Suponha que se pretende enriquecer essa interface com mais um evento,

$$HISTORY : \cdots \to \cdots$$

   que deverá responder à sua activação dando o "rol" de todas as respostas dadas até ao momento. Aumente $DTREE$ nesse sentido e estude o impacto da sua modelação em $DEC@TREE$.
3. Repita a alínea anterior para mais um evento,

$$BACKTRACK : \cdots \to \cdots$$

   que deverá recuperar de uma resposta mal dada.

□

5.3.1 Invariantes de Estado

Na Secção 2.4 foi introduzido o conceito de predicado *invariante* sobre uma dada espécie $s \in S$ de uma assinatura $\Sigma : \Omega \to S^\star \times S$, num dado Σ-modelo $\mathcal{A}$. Tal predicado, que designamos por *inv-s*, diz-se um invariante por se entender que ele se deve verificar para todos os valores de $\mathcal{A}(s)$, conjunto portador da espécie s em $\mathcal{A}$. Nessa secção foram apresentadas técnicas de prova para a verificação de invariantes.

Num modelo $\mathcal{A}$ com eventos (Definição 5.2) é perfeitamente natural que a definição da classe Q de estados internos do modelo exija também um invariante inv-Q. Tem então interesse considerar o problema da preservação "temporal" desse invariante pelos eventos que alteram o estado, ou seja, cuja pós-condição não implica a condição $\sigma = \sigma'$. Seja

$$\tau : s_1 \times \ldots \times s_n \to s$$

um evento dessa classe de eventos que alteram o estado. Antecipemos que o argumento da preservação do invariante por τ terá o aspecto de um predicado em *Lógica Dinâmica*, relacionando condições lógicas válidas sobre um estado σ anterior à ocorrência de τ com outras condições que se pretendem válidas sobre qualquer estado σ' para que o modelo possa vir a transitar após a ocorrência de τ. Assim, a desejável extensão da fórmula (2.131) a eventos será

$$\left.\begin{array}{l} \text{inv-}Q(\sigma)\wedge \\ pre_\tau(\sigma, t_1, \ldots, t_n)\wedge \\ post_\tau(\sigma, t_1, \ldots, t_n, \sigma', t')\wedge \\ \forall 1 \leq i \leq n : \textit{inv-}s_i(t_i) \end{array}\right\} \Rightarrow \left\{\begin{array}{l} \text{inv-}Q(\sigma')\wedge \\ \textit{inv-}s(t') \end{array}\right. \tag{5.10}$$

Note-se que, da mesma maneira que o estado de um modelo normalmente "põe em evidência" uma espécie que estaria omnipresente nos argumentos dos operadores funcionais do correspondente modelo sem estado, também o invariante de estado "põe em evidência" condições sobre essa espécie que podem ser retiradas das pré-condições dos eventos do modelo. Quer dizer, para maior economia e melhor inteligibilidade do texto de uma especificação, retiram-se das pré-condições as cláusulas lógicas já garantidas pelo invariante. Tal procedimento prático é inofensivo desde que as conjunções lógicas do antecedente da implicação (5.10) sejam encaradas como não estritas, *cf.* equação (3.44). Aliás, esta não é única situação em que é preciso entender as conectivas lógicas não-estritamente: no fundo, todo o encadeamento lógico entre pré e pós-condições pressupõe lógica não estrita, já que a avaliação de uma pós-condição só pode ser feita em contextos onde a pré-condição seja verdadeira.

5.4 Exercícios

Exercício 5.4 Na aplicação prática da especificação formal a problemas concretos usam-se muitas vezes "regras de algibeira" para simplificar argumentos de preservação de invariantes. É esse o caso em modelos cujo estado interno está sujeito a um invariante da forma

$$\text{inv-}Q \stackrel{\text{def}}{=} A \rightharpoonup \phi$$

onde $\phi : B \to 2$ é um dado predicado e os eventos a tratar são determinísticos e tais que as suas pós-condições são da forma:

$$\begin{array}{rcl} E_1(S) & \stackrel{\text{def}}{=} & \sigma' = \sigma \setminus S \\ E_2(a,b) & \stackrel{\text{def}}{=} & \sigma' = \sigma \dagger \begin{pmatrix} a \\ b \end{pmatrix} \end{array}$$

1. Com os dados disponíveis, complete da forma mais geral possível (justifique) as seguintes definições relativas a esse modelo:

$$\begin{array}{rcl} E_1 & : & \ldots\ldots\ldots\ldots\ldots \to \ldots\ldots\ldots\ldots\ldots \\ E_2 & : & \ldots\ldots\ldots\ldots\ldots \to \ldots\ldots\ldots\ldots\ldots \\ Q & \cong & \ldots\ldots\ldots\ldots\ldots \end{array}$$

2. Recorra a propriedades conhecidas da construção $A \rightharpoonup \phi$ para justificar as seguintes duas "regras de algibeira":

– $E_1(S)$ preserva sempre inv-Q, qualquer que seja S. (5.11)

– Para demonstrar que $E_2(a,b)$ preserva inv-Q, basta provar $\phi(b)$. (5.12)

□

Exercício 5.5 No contexto do Exercício 5.4, suponha que o evento em causa é determinístico e tal que a sua pós-condição é da forma:

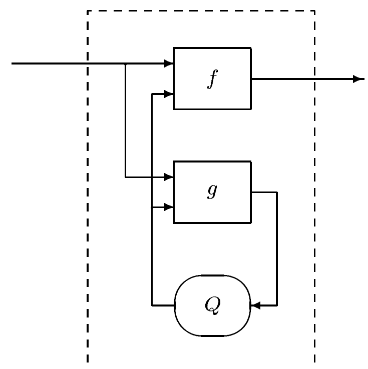
$$E(b') \stackrel{\text{def}}{=} \sigma' = (A \rightharpoonup f)(\sigma) \wedge b' = g(\sigma)$$

1. Tipifique, de forma mais geral possível, as funções f e g.
2. Recorra à definição formal do predicado $A \rightharpoonup \phi$ para justificar ou refutar a seguinte "regra de algibeira":

– Para demonstrar que E preserva inv-Q, basta provar que f preserva ϕ. (5.13)

□

Exercício 5.6 O diagrama seguinte representa a estrutura bi-funcional de uma dada acção (evento) determinística E, com apenas um argumento e uma saída, num modelo com estado interno Q:



Dadas as seguintes definições de f e g,

$$
\begin{array}{rcl}
f & = & \lambda(\sigma, a).\ let \quad n = H(a) \\
 & & \quad\ in \quad \left\{ \begin{array}{lcl} n \in dom(\sigma) & \Rightarrow & a \in \sigma(n) \\ n \notin dom(\sigma) & \Rightarrow & F \end{array} \right. \\
g & = & \lambda(\sigma, a).\ let \quad n = H(a) \\
 & & \quad\ in \quad \sigma \dagger \left(\begin{array}{c} n \\ \{a\} \cup \left\{ \begin{array}{lcl} n \in dom(\sigma) & \Rightarrow & \sigma(n) \\ n \notin dom(\sigma) & \Rightarrow & \emptyset \end{array} \right. \end{array} \right)
\end{array}
$$

onde $H : A \to I\!N$ é uma função pré-definida,

1. complete a seguinte definição da assinatura do evento,

$$E : \ldots\ldots\ldots\ldots\ldots \to \ldots\ldots\ldots\ldots\ldots$$

   e do modelo formal de Q:

$$Q \cong \ldots\ldots\ldots\ldots\ldots$$

2. descreva por palavras suas o significado deste evento, relacionando-o com um modelo de gestão da informação bem conhecido na literatura algorítmica;
3. mostre que E satisfaz o seguinte invariante:

$$\text{inv-}Q(\sigma) \stackrel{\text{def}}{=} \forall n \in dom(\sigma) : (\emptyset \neq \sigma(n) \wedge \forall i \in \sigma(n) : H(i) = n)$$

   i.é, prove o facto seguinte, para a e σ universalmente quantificadas:

$$\text{inv-}Q(\sigma) \wedge \sigma' = g(\sigma, a) \quad \Rightarrow \quad \text{inv-}Q(\sigma')$$

   Sugestão: desdobre o facto a provar em duas quantificações separadas e prove-as independentemente.

□

Exercício 5.7 No contexto do Exercício 1.41, Após uma breve análise dos requisitos e com base no fragmento de modelo sugerido (1.105), construa a especificação das seguintes operações sobre um modelo cujo estado interno é $System$:

$$
\begin{array}{lll}
VI & : & \rightarrow VI@INT \\
VI() & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{llll}
sorts & Text & \cong & Line^* \\
 & Line & \cong & Char^* \\
 & Nr & \cong & I\!N \\
 & Cursor & \cong & L : Nr \times C : I\!N \\
 & Bool & \cong & 2 \\
 & Char & \cong & \{0, \ldots, 9, a, \ldots, z, A, \ldots, Z, \ldots\} \\
 & State & \cong & C : Cursor \times T : Text \\
state & State & & \\
events & G(n) & \stackrel{\mathrm{def}}{=} & \begin{array}{ll} let & c = C(\sigma) \\ & t = T(\sigma) \\ in & \left\{ \; n \leq length(t) \quad \Rightarrow \quad T(\sigma') = t \wedge C(\sigma') = (n, 1) \right. \end{array} \\
 & r(c) & \stackrel{\mathrm{def}}{=} & \begin{array}{ll} let & c' = C(\sigma) \\ & t = T(\sigma) \\ in & \{ t \neq <> \Rightarrow \begin{array}{ll} let & x = C(c') \\ & y = L(c') \\ & linha = t(y) \\ & linha' = linha \dagger \left(\begin{array}{c} x \\ c \end{array} \right) \\ in & C(\sigma') = c \wedge \\ & T(\sigma') = t \dagger \left(\begin{array}{c} y \\ linha' \end{array} \right) \end{array} \end{array} \\
 & \vdots & &
\end{array} \right.
\end{array}
$$

Figura 5.3: Esboço de modelo VI para $VI@INT$.

1. Operação
$$NULL\_INV : IdL \longrightarrow 2^{InvNr}$$
que deverá indicar os números de todas as facturas anuladas numa dada loja.
2. Operação
$$SALES\_BY\_DATE : Date \longrightarrow (IdA \rightharpoonup I\!N)$$
que deverá totalizar as vendas de um dado dia (em todas as lojas).

□

Exercício 5.8 Pretende-se neste exercício esboçar uma especificação do editor *VI*, o editor "clássico" do ambiente UNIX. Suponha que se começa por a construir a interface

$$
\begin{array}{rl}
interface & VI@INT \\
sorts & Text, Line, Nr, Char, Cursor, Bool, State \\
events & G : Nr \rightarrow \\
 & r : Char \rightarrow \\
 & \vdots
\end{array}
$$

seguindo-se a construção de um modelo VI para $VI@INT$, cujo esboço se representa na Figura 5.3 [4].

1. A que comandos de edição do *VI* correspodem os eventos presentes na interface e no modelo acima?
2. Da análise do que está já especificado decerto conclui que é necessário acrescentar à interface um predicado
$$inv : State \rightarrow Bool$$
que seja invariante sobre $State$. Apresente a definição desse predicado no modelo VI.
3. Escreva a definição no modelo VI da semântica dos eventos
$$h, j, k, l : Nr \rightarrow$$
que deverão corresponder aos 4 comandos de navegação que conhece (o argumento numérico corresponde ao índice de repetição, *e.g.* ao comando `3h` corresponde no modelo o evento $h(3)$).
4. Escreva a definição em VI da semântica de mais um evento
$$dd : Nr \rightarrow$$
cuja consequência é apagar tantas linhas a partir do cursor quantas as especificadas no seu argumento.
5. Seja agora $VI@INT$ enriquecida com uma nova espécie, $String$, e mais dois eventos de edição:

$D :$		$\rightarrow$	*/* apaga todos os caracteres desde a posição corrente do cursor até ao fim da linha */*
$/ :$	$String$	$\rightarrow$	*/* procura a primeira ocorrência do "string" argumento e leva para aí o cursor */*

Seja $VI(String) \cong Char^\star$. Apresente definições para $VI(D)$ e $VI(/)$.

□

Exercício 5.9 Considere a seguinte interface e modelo para especificar um sistema de gestão de base de dados relacional (deliberadamente simplificado e incompleto):

$$\begin{array}{rl}
interface & RDBMS \\
sorts & Dbs, Bool, Id, Table, Tuple, Atrib, Schema, Value, Query \\
ops & join : Table \times Table \rightarrow Table \\
& project : Table \times Schema \rightarrow Table \\
& select : Table \times Query \rightarrow Table \\
& \text{inv-}Table : Table \rightarrow Bool \\
& \text{inv-}Dbs : Dbs \rightarrow Bool \\
events & INIT : \rightarrow \\
& SCHEMA : Id \rightarrow Schema \\
& JOIN : Id \times Id \times Id \rightarrow \\
& SELECT : Id \times Query \times Id \rightarrow \\
& PROJECT : Id \times Schema \times Id \rightarrow
\end{array} \tag{5.14}$$

[4] Neste modelo ocorrem produtos cartesianos prefixados por selectores (relembrar Secção 1.3.1), o que se usa muito em especificação por modelos para melhorar a sua legibilidade.

$$
\begin{array}{l}
DBMS : ITEM \times ITEM \times ITEM \longrightarrow RDBMS \\
DBMS(I, A, V) \stackrel{\text{def}}{=} \\
\left\{ \begin{array}{rl}
sorts & Bool \cong 2 \\
& Atrib \cong A.Item \\
& Schema \cong 2^{Atrib} \\
& Value \cong V.Item \\
& Tuple \cong Atrib \rightharpoonup Value \\
& Query \cong Atrib \rightharpoonup Value \\
& Table \cong 2^{Tuple} \\
& Id \cong I.Item \\
& Dbs \cong Id \rightharpoonup Table \\
ops & join(r, s) \stackrel{\text{def}}{=} \{t \cup t' \mid \ t \in r \wedge t' \in s \wedge t \Delta t'\} \\
& select(r, q) \stackrel{\text{def}}{=} \{t \in r \mid \ (q \mid dom(t)) \subseteq t\} \\
& project(r, S) \stackrel{\text{def}}{=} \{(t \mid S) \mid \ t \in r\} \\
& \text{inv-}Table(r) \stackrel{\text{def}}{=} \forall t, t' \in r : dom(t) = dom(t') \\
& \text{inv-}Dbs(\sigma) \stackrel{\text{def}}{=} \forall i \in dom(\sigma) : \text{inv-}Table(\sigma(i)) \\
state & Dbs \\
events & INIT \stackrel{\text{def}}{=} \sigma' = \left(\quad \right) \\
& SCHEMA(i) \stackrel{\text{def}}{=} \{i \in dom(\sigma) \Rightarrow let\ t \in \sigma(i)\ in\ dom(t) \\
& JOIN(i, j, k) \stackrel{\text{def}}{=} \{i, j \in dom(\sigma) \wedge k \notin dom(\sigma) \Rightarrow \sigma' = \sigma \cup \begin{pmatrix} k \\ join(\sigma(i), \sigma(j)) \end{pmatrix} \\
& SELECT(i, q, k) \stackrel{\text{def}}{=} \{i \in dom(\sigma) \wedge k \notin dom(\sigma) \Rightarrow \sigma' = \sigma \cup \begin{pmatrix} k \\ select(\sigma(i), q) \end{pmatrix} \\
& PROJECT(i, S, k) \stackrel{\text{def}}{=} \{i \in dom(\sigma) \wedge k \notin dom(\sigma) \Rightarrow \sigma' = \sigma \cup \begin{pmatrix} k \\ project(\sigma(i), q) \end{pmatrix}
\end{array} \right.
\end{array}
\tag{5.15}
$$

1. Interprete cuidadosamente este modelo e tente descrever por palavras suas a semântica informal de cada operação.
2. Prove que todos os eventos que alteram o estado preservam o invariante inv-Dbs.
3. Seja Σ_{RDBMS} a assinatura da componente funcional da interface $RDBMS$. Verifique se a seguinte igualdade

   $$select(project(r, s), q) \quad \equiv \quad project(select(r, q), s)$$

   é um $\Sigma_{RDBMS}(X)$-axioma válido no modelo $DBMS$, para a $X = \begin{pmatrix} r & s & q \\ Table & Schema & Query \end{pmatrix}$.
4. Pretende-se agora enriquecer o modelo por forma a ser possível registar e verificar dependências funcionais associadas a tabelas da base de dados subjacente. Para isso, à estrutura de tabelas

   $$Dbs \quad \cong \quad Id \rightharpoonup Table$$

   é acrescentada a informação de quais tabelas estão (ou poderão vir a estar) sujeitas a dependências funcionais ($FDeps$),

   $$Dbs \quad \cong \quad (Id \rightharpoonup Table) \times (Id \rightharpoonup FDeps)$$

   introduzindo as seguintes novas espécies no modelo e sua interface:

   $$\begin{array}{rcl} FDeps & \cong & 2^{FDep} \\ FDep & \cong & Attribs \times Attribs \end{array}$$

   Reforce o invariante inv-Dbs por forma a garantir a salvaguarda das seguintes condições adicionais:

- Todas as dependências funcionais estão bem definidas *i.é*, não envolvem atributos inexistentes;
- Se uma tabela está sujeita a uma dependência funcional, então todos os tuplos dessa tabela satisfazem essa dependência.

□

Exercício 5.10 Na sequência do exercício anterior, suponha que se pretende remover desse modelo o seu invariante de estado, *i.é*, qualquer tabela pode ter atributos omissos, por exemplo:

A	B	C
a_1		c_1
	b_2	c_2
a_3		

que corresponde à seguinte tabela $t \in Table$:

$$t = \{ \begin{pmatrix} A & C \\ a_1 & c_1 \end{pmatrix}, \begin{pmatrix} B & C \\ b_2 & c_2 \end{pmatrix}, \begin{pmatrix} A \\ a_3 \end{pmatrix} \}$$

Meça o impacto desta generalização do modelo re-especificando os eventos $SCHEMA$ e $JOIN$.

□

Exercício 5.11 Na sequência do exercício anterior, pretende-se enriquecer $RDBMS$ com um evento

$$SORT : Id \times AtribList \times Id \rightarrow$$

tal que $SORT(i, l, k)$ signifique que se guarda em k o resultado de se ordenar a tabela i lexicograficamente segundo a lista de atributos l.

Não especifique $SORT$, mas estude o impacto em $DBMS@MOD$ do enriquecimento de $RDBMS$ com esse evento (*e.g.* o que terá de ser adicionado? o que terá de ser alterado?).

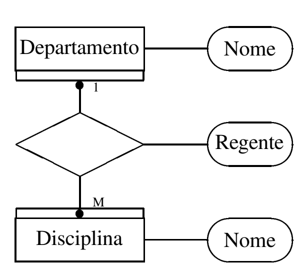
□

Exercício 5.12 Neste capítulo foi estudada um técnica para especificação da semântica formal de *eventos* (="funções + estado").

Relacione essa técnica com a notação lógica temporal que foi assunto do Exercício 3.21.

□

Exercício 5.13 Considere um modelo cujo estado interno Q corresponde à semântica em SETS do diagrama ERA seguinte:



1. Especifique formalmente o evento $NDISC$ sobre Q que regista uma nova disciplina, conhecidos o seu nome, regente e departamento.
2. Prove que a sua especificação de $NDISC$ satisfaz o invariante implícito em Q.

$\square$

5.5 Notas Bibliográficas

No capítulo que aqui termina pretendeu-se fazer a síntese entre três vertentes básicas da ciência da especificação formal de 'software', a saber:

1. Especificação construtiva com variáveis globais, com operações (eventos) cuja semântica *estática* é definida por pares de pré e pós-condições, seguindo o estilo de [Hoa69] e tal como vem sendo proposta em metodologias de larga aceitação como o 'Vienna Development Method' (VDM) [Jon80, Jon86].
2. Modularização de especificações formais, ampliando o que foi proposto no Capítulo 4 (ver as referências bibliográficas deste capítulo).

O conceito de modelo com *estado* e *eventos*, satisfazendo uma dada *interface* aproxima-se da noção de *objecto* (pertencente a uma dada *classe*) proposta pela *programação por objectos* [GR83] e *especificação orientada a objectos* [SFSE87]. Pode aind ser visto como um 'labelled transition system', *cf.* [Hen88].

Capítulo 6

Semântica Dinâmica

No capítulo anterior encontraram-se boas razões para deixar de especificar "por funções" e passar a especificar "por relações", *i.é* por eventos. A Definição 5.2 apresentou uma formalização do conceito de modelo com eventos, que foi previamente motivado com exemplos como $STACKLIST$, *cf.* (5.9). Assim ficou registada alguma da nossa *intuição* do que "é" um evento, formalizável por *relações matemáticas*.

Este tipo de semântica para eventos diz-se *estática* por ser "atemporal" e ignorar os aspectos dinâmicos de um sistema baseado em eventos (nomeadamente, o "acontecimento" temporal dos eventos, a sua sincronização, *etc.*). Abordar agora esta componente dinâmica de um sistema de software é o principal objectivo deste capítulo.

6.1 ..."o que é um Evento"?

Esta questão pode ser posta em paralelo com a questão (já respondida) "o que é um operador?". Relembremos que a semântica de um operador adquiriu "significado" a partir da altura em que se mostrou que ele, em conjunto com todos os outros da sua assinatura Σ, actuava como "gerador", ou "construtor" de uma linguagem W_Σ cujas frases ganhavam semântica a partir da altura em que se definia um modelo denotacional $\mathcal{A} : \Sigma \to Sets$ que permitia "calcular" o significado de cada frase.

Fomos ainda mais longe ao permitir que se criasse uma "versão simplificada" de $\mathcal{A}$ especificável à custa de um conjunto E de axiomas presente na $interface$ de $\mathcal{A}$. Formalmente, como vimos também, o (meta) significado destas construções constrói-se no reticulado de todas as Σ-congruências. O diagrama seguinte resume todo esse raciocínio:

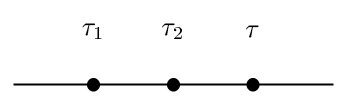
$$\begin{tikzcd} & & \cong_E \arrow[dr] & \\ \Sigma \arrow[r] & W_\Sigma \arrow[ur] \arrow[rr] & & \cong_{\mathcal{A}} \end{tikzcd} \tag{6.1}$$

Com a introdução de *eventos*, "emparelhamos" Σ com uma "assinatura" de eventos Γ. Como raciocinar sobre Γ? Somos tentados a construir algo semelhante a (6.1). Em primeiro lugar, fará sentido construir W_Γ, o conjunto de todas as frases com Γ-eventos?

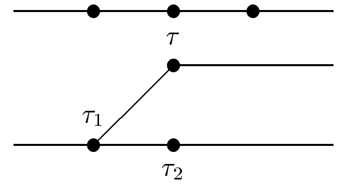
Um qualquer termo $t \in W_\Sigma$ é uma "expressão bem tipificada" sobre Σ-operadores. O que é uma "expressão bem tipificada" de Γ-eventos? Reparemos que, se τ_1, τ_2 são dois eventos e τ é um terceiro evento, qualquer "termo" da forma

$$\tau(\tau_1, \tau_2)$$

não faz muito sentido, a não ser que o seu significado fosse qualquer coisa como "τ_1 ocorre primeiro, τ_2 ocorre a seguir e τ ocorre de seguida com argumentos debitados por τ_1 e τ_2". É mais natural representarmos esse significado por uma ordem linear de ocorrência de eventos,



ou mesmo não-linear, *e.g.*



(6.2)

Neste caso, diríamos que τ_1 ocorre primeiro, e depois ou ocorre τ ou ocorre τ_2.

Concluimos que qualquer forma de Γ-linguagem é mais "rica" em situações que a simples construção de termos. Por isso mesmo, qualquer frase gerada por Γ não será designada um $termo$, mas sim uma expressão de *comportamento* ('behaviour'). Vejamos como construir expressões de comportamento.

6.1.1 Expressões de Comportamento

Comecemos por designar por B_Γ (*cf.* W_Σ, por analogia) o conjunto de todas as expressões de comportamento "geráveis" a partir de

$$\Gamma : \Pi \to S^* \times S^*$$

Arrisquemos agora uma definição indutiva para B_Γ, baseada nas cláusulas seguintes:

- se b é um Γ-comportamento, e τ é um Γ-evento (*i.é* $\tau \in \Pi$), então designaremos por $\tau.b$ o comportamento seguinte: τ ocorre primeiro e depois o modelo comporta-se segundo b;

- se b e b' são dois Γ-comportamentos, então $b + b'$ é o Γ-comportamento que representa a possibilidade de o modelo se comportar segundo b ou segundo b';

- 0 é um Γ-comportamento especial que significa *inacção* ou "paragem" [1].

- não há mais Γ-comportamentos do que os resultantes das cláusulas anteriores.

Matematicamente, escreveremos a definição recursiva:

$$\begin{array}{rcl} B_\Gamma & \stackrel{\text{def}}{=} & \{0\} \\ & \cup & \{\tau.b \mid \tau \in \Gamma \wedge b \in B_\Gamma\} \\ & \cup & \{b + b' \mid b, b' \in B_\Gamma\} \end{array} \tag{6.3}$$

Falta-nos agora, apenas, sermos mais precisos quanto à condição $\tau \in \Gamma$ que ocorre em (6.3). Seja $\tau \in \Pi$ um evento tal que

$$\Gamma(\tau) = (<s_1, \ldots, s_n>, <s'_1, \ldots, s'_m>)$$

o que se escreve abreviadamente da forma

$$\tau : s_1 \times \ldots \times s_n \to s'_1 \times \ldots \times s'_m \tag{6.4}$$

Seja $t_i \in W_{\Sigma,s_i}$ e $t'_j \in W_{\Sigma,s'_j}$, para $1 \leq i \leq n$ e $1 \leq j \leq m$. Então τ não "é" apenas um evento, mas uma colecção deles,

$$\tau(\ldots t_i \ldots, \ldots t'_j \ldots)$$

indexada pelos possíveis valores t_i, t'_j dos seus argumentos ou resultados. Quer dizer, $PUSH(10)$ não é, por exemplo, o mesmo evento que $PUSH(11)$. Se definirmos

$$[\tau] = \{\tau(t_1, \ldots, t_i, \ldots, t_n, t'_1, \ldots, t'_j. \ldots, t'_m) \mid t_i \in W_{\Sigma,s_i} \wedge t'_j \in W_{\Sigma,s'_j}\}$$

podemos caracterizar o conjunto de todos os eventos "atómicos" gerados por Γ como sendo

[1] Este Γ-comportamento toma por vezes a designação NIL ou $STOP$, *cf.* notações como EPL, CCS ou CSP referidas na Secção 5.5.

$$Eve(\Gamma) = \bigcup_{\tau \in \Pi} [\tau] \tag{6.5}$$

e corrigir, em (6.3), a condição $\tau \in \Gamma$ para $\tau \in Eve(\Gamma)$. Em resumo, temos a definição seguinte.

Definição 6.1 (Γ-comportamento) *Seja dada uma assinatura de eventos* $\Gamma : \Pi \to S^* \times S^*$*,* cf. *Definição 5.1. O conjunto de todos os "*Γ*-comportamentos" é o menor conjunto que satisfaz as seguintes cláusulas:*

1. *Prefixação — se* $\alpha \in Eve(\Gamma)$ *é um evento e* b *é um* Γ*-comportamento, então* $\alpha.b$ *é também um* Γ*-comportamento;*
2. *Alternativa — se* b *e* b' *são* Γ*-comportamentos, então* $b + b'$ *também o é;*
3. *Inacção — existe um* Γ*-comportamento particular* 0 *que tem a propriedade de, em nenhuma circunstância, fazer o que quer que seja.*

O conjunto de todos os Γ*-comportamentos designa-se por* B_Γ*.* $\square$

A definição anterior introduziu o conceito "sintático" de Γ-comportamento. Esse conceito carece de interpretação semântica, e esta fundamentalmente depende da nossa intuição sobre o que "é" um comportamento. Na verdade, há mais do que uma interpretação possível. Comecemos por aquela que designaremos por interpretação PS.

6.1.2 A Interpretação PS

Nesta interpretação, vamos estar apenas interessados nas sequências de eventos — acções — que um agente pode fazer de acordo com uma expressão de comportamento $b \in B_\Gamma$. Assim, a interpretação associada a cada uma dessas expressões é o conjunto daquelas sequências, *i.é* subconjunto de $Eve(\Gamma)^*$, *cf.* (6.5). Cada uma dessas sequências, *e.g.*

$$<\alpha, \beta, \ldots, \omega> \tag{6.6}$$

será designada um Γ-*traço* (*cf.* 'trace') ou Γ-*história*, para evidenciar o seu carácter temporal, e poder-se-á também escrever:

$$\alpha; \beta; \ldots; \omega \tag{6.7}$$

lendo-se "α aconteceu primeiro, seguido de β, seguido de ..., seguido de ω". Portanto, um traço ou história representa um instante concreto da vida de um agente, de acordo com o seu comportamento. Mas se o agente já "viveu" até ω em (6.6), isso significa que existiram instantes anteriores em que ele viveu "menos", *e.g.* o instante em que ainda só aconteceu o traço $<\alpha, \beta>$. Na verdade, o traço

(6.6) pressupõe todos os traços que são *prefixos* seus, incluindo $<>$, o traço que corresponde ao instante em que "ainda não aconteceu nada":

$$\begin{array}{lcl} t_n & = & <\alpha, \beta, \ldots, \tau, \omega> \\ t_{n-1} & = & <\alpha, \beta, \ldots, \tau> \\ & \vdots & \\ t_i & = & <\alpha, \beta, \ldots> \\ & \vdots & \\ t_2 & = & <\alpha, \beta> \\ t_1 & = & <\alpha> \\ t_0 & = & <> \end{array} \tag{6.8}$$

Temos, portanto, cada instante i caracterizado pelo traço t_i, verificando-se

$$i \leq j \Rightarrow t_i \leq t_j$$

onde a ordem $s \leq s'$ ("prefixo de") sobre sequências arbitrárias $s, s' \in X^*$ se pode definir por

$$s \leq s' \Leftrightarrow \exists r \in X^* : s' = s \frown r$$

e tem $<>$ como mínimo, *i.é*

$$\forall s \in X^* : <> \leq s \tag{6.9}$$

Compreende-se agora que a interpretação de um comportamento não pode ser qualquer subconjunto de traços — só faz sentido que seja um subconjunto de traços *fechado por prefixos*. Genericamente, um subconjunto $S \subseteq X^*$ está nessas condições se

$$s \in S \wedge r \leq s \Rightarrow r \in S \tag{6.10}$$

Definição 6.2 (Domínio PS) *Designaremos por* domínio PS *(de 'Prefix-closed Subsets') o conjunto de todos os subconjuntos não vazios de* $Eve(\Gamma)^*$ *fechados por prefixos,* i.é

$$PS = \{\emptyset \subset S \subseteq Eve(\Gamma)^* \mid S \text{ satisfaz (6.10)}\}$$

Constata-se imediatamente que $<> \in S$ *para todo o* $S \in PS$*,* cf. *(6.9).* □

A interpretação de B_Γ em PS define-se facilmente através de um homomorfismo

$$h_{PS} : B_\Gamma \to PS \tag{6.11}$$

com as cláusulas seguintes (abreviando h_{PS} para PS, como habitualmente):

$$PS(b+b') = PS(b) \cup PS(b') \tag{6.12}$$
$$PS(0) = \{<>\} \tag{6.13}$$
$$PS(\alpha.b) = \{<>\} \cup \{cons(\alpha,s) \mid s \in PS(b)\} \tag{6.14}$$

A título de exemplo, calculemos a interpretação do comportamento $\alpha.(0+\beta.0)$. Teremos:

$$PS(\alpha.(0+\beta.0)) = \{<>\} \cup \{cons(\alpha,s) \mid s \in PS(0+\beta.0)\} \tag{6.15}$$

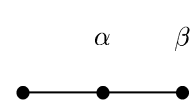
Ora

$$\begin{aligned} PS(0+\beta.0) &= PS(0) \cup PS(\beta.0) \\ &= \{<>\} \cup \{cons(\beta,s) \mid s \in PS(0)\} \\ &= \{<>\} \cup \{cons(\beta,s) \mid s = <>\} \\ &= \{<>,<\beta>\} \end{aligned} \tag{6.16}$$

Substituindo (6.16) em (6.15) obtemos

$$\begin{aligned} PS(\alpha.(0+\beta.0)) &= \{<>\} \cup \{cons(\alpha,s) \mid s \in \{<>,<\beta>\}\} \\ &= \{<>\} \cup \{<\alpha>,<\alpha,\beta>\} \\ &= \{<>,<\alpha>,<\alpha,\beta>\} \end{aligned}$$

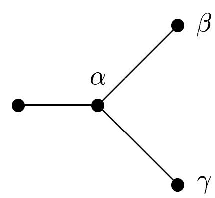
cf. o diagrama



Por comodidade, muitas vezes abreviam-se expressões de comportamento da forma "$\alpha.0$" para "α". É o que acontece nas igualdades seguintes:

$$PS(\alpha.(\beta+\gamma)) = PS(\alpha.\beta+\alpha.\gamma) \tag{6.17}$$
$$= PS(\alpha.\beta+\alpha.(\beta+\alpha)) \tag{6.18}$$
$$= \{<>,<\alpha>,<\alpha,\beta>,<\alpha,\gamma>\}$$

que podem ser verificadas usando as cláusulas (6.12) e (6.14), *cf.* o diagrama



Exercício 6.1 Sejam $x, y, z \in B_\Gamma$ expressões de comportamento, e $\alpha \in Eve(\Gamma)$ um evento. Para cada uma das igualdades

$$i \equiv j$$

que se seguem, verifique que $PS(i) = PS(j)$:

$$x + x \equiv x \quad (6.19)$$
$$x + y \equiv y + x \quad (6.20)$$
$$x + (y + z) \equiv (x + y) + z \quad (6.21)$$
$$x + 0 \equiv x \quad (6.22)$$
$$\alpha.(x + y) \equiv \alpha.x + \alpha.y \quad (6.23)$$

□

Resolvido o exercício anterior, ficamos com a possibilidade de, na interpretação PS transformar expressões de comportamento de acordo com as equações (6.19) a (6.23), uma vez que se mostrou que estas são válidas nessa interpretação. Por exemplo, (6.17) decorre imediatamente de (6.23), (6.18) decorre de (6.19) e de (6.23) *etc.*

Ora é também possível mostrar que se $PS(i) = PS(j)$, então $i \equiv j$ de acordo com as equações(6.19) a (6.23) [2] o que significa que esse conjunto de equações é *completo* em relação a PS e pode ser usado seguramente para raciocinar sobre comportamentos sem haver necessidade de recorrer à interpretação fixada pelas cláusulas (6.12) a (6.14) do homomorfismo (6.11). Tipicamente, o operador de alternativa $+$ e a constante 0 formam um monóide abeliano idempotente, sendo a prefixação uma operação distributiva em relação à alternativa.

Exercício 6.2 Suponha que se introduz, no modelo PS, uma conectiva de sequenciação $b; b'$ com a seguinte definição:

$$PS(b;b) \stackrel{\mathrm{def}}{=} \{x \frown y) \mid \ x \in PS(b) \wedge y \in PS(b')\}$$

onde $conc$ designa concatenação de sequências (*cf.* Exercício 2.9).

[2] *cf.* Secção 5.5.

Verifique quais dos axiomas seguintes são válidos para esta conectiva:

$$\alpha.(b;b') = (\alpha.b);b' \tag{6.24}$$
$$b;(b'+b'') = (b;b')+(b;b'') \tag{6.25}$$
$$b;(b';b'') = (b;b');b'' \tag{6.26}$$
$$b;0 = b \tag{6.27}$$
$$0;b = b \tag{6.28}$$
$$(b'+b'');b = (b';b)+(b'';b) \tag{6.29}$$

□

Exercício 6.3 Considere a extensão ao modelo **PS** para semântica de comportamentos, na qual é acrescentado um novo operador de prefixação, dito de *prefixação activa* e designado por ":", com o significado informal que se segue: *A prefixação $\alpha : b$ difere de $\alpha.b$ por ser mais activa que esta, na medida em que α acontece imediatamente e não chega a ser observável a história vazia $<>$.*

Nesta extensão, designada PS', teremos então:

$$PS'(\alpha : b) = \{cons(\alpha, t) \mid \ t \in PS'(b)\}$$

sendo igual a PS quanto aos restantes operadores comportamentais.

1. Mostre que o domínio da interpretação PS' não é fechado por prefixos.
2. Mostre que o axioma
$$\forall b \in B_\Gamma : b + 0 \equiv b$$
não se verifica em PS'.
3. Mostre que, para todo o comportamento b e evento α, se verifica em PS' o axioma
$$(\alpha : b) + 0 \equiv \alpha.b$$

□

Exercício 6.4 Considere os seguintes operadores sobre B_Γ onde, para simplificar a notação, se escreve Π em lugar de $Eve(\Gamma)$:

$b[\phi] : B_\Gamma \times (\Pi \rightharpoonup \Pi) \longrightarrow B_\Gamma$	*/* cujo efeito é renomear os eventos em b de acordo com a substituição ϕ */*
$b \setminus E : B_\Gamma \times 2^\Pi \longrightarrow B_\Gamma$	*/* cujo efeito é truncar o comportamento b a partir da ocorrência de qualquer dos eventos em E */*

Defina a interpretação PS dos dois operadores acima introduzidos e verifique se são válidas, em PS, as igualdades seguintes:

$$(b+b')[\phi] \equiv b[\phi] + b'[\phi] \tag{6.30}$$
$$(b+b') \setminus E \equiv b \setminus E + b' \setminus E \tag{6.31}$$

□

6.1.3 A Interpretação RT

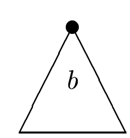
Há, é claro, muitas outras interpretações semânticas para expressões de comportamento, para além de PS. É provável que se tenha sentido, aliás, que a interpretação PS identifica comportamentos que intuitivamente parecem diferentes, por exemplo os dois membros da equação (6.23). A "diferença" é que no membro esquerdo a possibilidade de escolha (alternativa) aparece num estádio da computação posterior ao do membro direito, onde essa escolha é feita logo no início.

A interpretação RT — acrónimo de 'Rooted Trees' — que se segue, faz essa distinção e baseia-se nos diagramas de comportamento que se desenharam na secção anterior. Seja RT o conjunto de todas as árvores de comportamento finitas cujos ramos estão etiquetados por eventos de $Eve(\Gamma)$. Seja

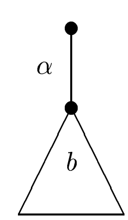
$$h_{RT} : B_\Gamma \to RT$$

o novo homomorfismo, alternativo a (6.11), tal que

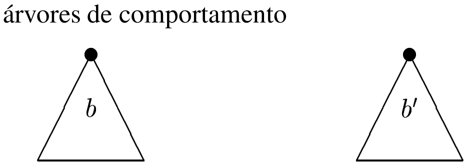
1. $RT(0)$ (*i.é* $h_{RT}(0)$) é a árvore trivial sem ramos, que se representa pela sua raíz $\bullet$;
2. Se $RT(b)$ é a árvore de comportamento



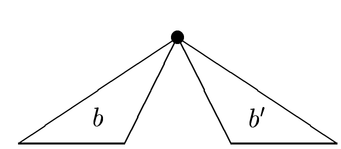
então $RT(\alpha.b)$ é a árvore



3. Sendo $RT(b)$ e $RT(b')$ as árvores de comportamento

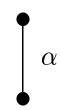


então $RT(b + b')$ será a árvore



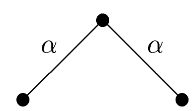
que se obtém identificando as raízes das árvores de comportamento de b e b'.

Por exemplo, $RT(\alpha) = RT(\alpha.0)$ será a árvore



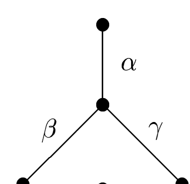
(6.32)

Já $RT(\alpha + \alpha) = RT(\alpha.0 + \alpha.0)$ é a árvore



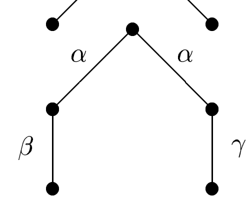
(6.33)

Logo, temos já um contra-exemplo, nesta interpretação, da equação (6.19), considerando que as árvores (6.32) e (6.33) são "diferentes". A mesma coisa acontece com a equação (6.23), já que, por exemplo, a interpretação RT de $\alpha.(\beta + \gamma)$ é



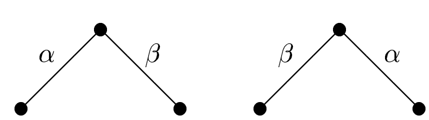
(6.34)

enquanto que a de $\alpha.\beta + \alpha.\gamma$ é



(6.35)

Assim, a interpretação RT parece satisfazer apenas as equações (6.20), (6.21) e (6.22), isto desde que se ache que, com respeito a (6.20), as árvores



(6.36)

por exemplo, "sejam" a mesma árvore. Carecemos, pois, de um modelo matemático para o domínio RT, já que o uso de diagramas de árvores de comportamento é demasiado informal.

Uma primeira aproximação pode ser considerar uma árvore de comportamento semelhante a uma árvore de decisão (relembrar Exercício 6.1) sem nós etiquetados e com ramos etiquetados com eventos de Eve(Γ):

$$RT \cong Eve(\Gamma) \rightharpoonup RT \tag{6.37}$$

Contudo, nesta aproximação, árvores como (6.33) não são representáveis, pois α não pode ocorrer duas vezes no domínio de uma função $\phi \in RT$ de acordo com (6.37).

Relaxando então a funcionalidade implícita em (6.37), teremos uma nova aproximação,

$$RT \cong 2^{Eve(\Gamma) \times RT} \tag{6.38}$$

de natureza relacional que já nos permite repetir o mesmo evento ao mesmo nível, mas ainda sem o efeito desejado de multi-ocorrência, pois as relações

$$\{(\alpha, \phi), (\alpha, \phi)\}$$

— formalização da árvore (6.33) — e

$$\{(\alpha.\phi)\}$$

— formalização da árvore (6.32) — são a mesma relação!

A evolução mais natural de (6.38) no sentido de registar a repetição de sub-árvores é converter conjuntos em sequências, *e.g.*

$$RT \cong (Eve(\Gamma) \times RT)^* \tag{6.39}$$

interpretando a alternativa de comportamentos pela concatenação de sequências (*cf.* Exercício 2.9),

$$RT(b + b') = RT(b) \frown RT(b')$$

mas agora temos ordem entre as sub-árvores, o que não queríamos pois vai-nos distinguir as duas árvores de (6.36).

A noção "intermédia" entre conjuntos (2^A) e sequências (A^*) que nos interessa é a de *multi-conjunto*, *i.é* um conjunto em que qualquer elemento pode ocorrer mais do que uma vez (relembrar o Exercício 1.38). Precisamos pois de uma extensão a 2^A. Reparemos que

$$2^A \cong (1 + 1)^A \tag{6.40}$$

pois $1+1 \cong 2$, relembrando (2.48) no Exercício 2.6. Na sequência desse exercício, vimos ainda que, para A e B em *Sets*,

$$A \rightharpoonup B \quad \cong \quad (B + 1)^A \tag{6.41}$$

cf. (2.108). Combinando agora (6.40) com (6.41), teremos

$$2^A \cong A \rightharpoonup 1 \tag{6.42}$$

quer dizer, um conjunto $s \in 2^A$ pode ser encarado como uma função finita que indica, para cada um dos elementos do seu domínio, que este ocorre uma vez no conjunto. Ora, se queremos conjuntos onde a ocorrência possa ser múltipla, basta-nos estender (6.42) a

$$A \rightharpoonup I\!N \tag{6.43}$$

e teremos aqui o desejado modelo de multiconjunto, tal como o especificamos no exercício 1.38. Assim, o multiconjunto vazio é representado pela função finita vazia $\left(\quad\right)$, e a união de multiconjuntos, designada $x \oplus y$, corresponde à adição de multiciplidades tal como vem definida em (1.102).

Voltando a (6.38), teremos para o domínio de RT o modelo seguinte,

$$RT \cong (Eve(\Gamma) \times RT) \rightharpoonup I\!N \tag{6.44}$$

sobre o qual se define a interpretação seguinte:

$$\begin{array}{rcl} RT(0) & = & () \\ RT(b+b') & = & RT(b) \oplus RT(b') \\ RT(\alpha.b) & = & \left(\begin{array}{c} (\alpha, RT(b)) \\ 1 \end{array} \right) \end{array} \tag{6.45}$$

Reparemos agora que, para qualquer $b \in B_\Gamma$

$$\begin{array}{rcl} RT(b+b) & = & RT(b) \oplus RT(b) \\ & = & \left(\begin{array}{c} a \\ 2RT(b)(a) \end{array} \right)_{a \in dom(RT(b))} \\ & \neq & RT(b) \end{array}$$

Logo $+$ não é idempotente (6.19) no modelo RT, como queríamos. Não sendo difícil provar que $\oplus$ é associativa e comutativa — *cf.* (A.24) e (A.25) no Exercício 4.7 — teremos verificadas as equações (6.20) e (6.21). Como $RT(0) = ()$, *cf.* (6.45) e $()$ é elemento neutro de $\oplus$, temos também verificada a equação (6.22). Falta apenas verificar a (6.23). Pegando no contra-exemplo $\alpha.(\beta + \gamma)$ (6.34), teremos

$$RT(\alpha.(\beta+\gamma)) = \left(\begin{array}{c} (\alpha, RT(\beta+\gamma)) \\ 1 \end{array} \right) = \left(\begin{array}{c} (\alpha, RT(\beta) \oplus RT(\gamma)) \\ 1 \end{array} \right)$$

enquanto que

$$\begin{array}{rcl} RT(\alpha.\beta+\alpha.\gamma) & = & RT(\alpha.\beta) \oplus RT(\alpha.\gamma) \\ & = & \left(\begin{array}{c} (\alpha, RT(\beta)) \\ 1 \end{array}\right) \oplus \left(\begin{array}{c} (\alpha, RT(\gamma)) \\ 1 \end{array}\right) \end{array}$$

ou seja,

$$RT(\alpha.(\beta+\gamma)) \neq RT(\alpha.\beta+\alpha.\gamma)$$

o que significa que (6.23) também não se verifica.

Em resumo, na interpretação RT, $+$ e 0 formam apenas um monóide abeliano e a prefixação não distribui em relação a $+$. Assim, a congruência equacional $\cong_{RT}$ tem grão semântico mais fino que $\cong_{PS}$.

Note-se que as interpretações PS e RT não são as únicas possíveis. Na verdade, elas representam dois "limites" matemáticamente simples entre os quais outras interpretações, mais próximas da realidade mas formalmente muito mais elaboradas, se podem definir. A sua discussão que sai fora do âmbito do presente estudo, é referida na Secção 5.5.

Exercício 6.5 No estudo da interpretação RT deu-se, como primeira aproximação, o modelo $RT \cong Eve(\Gamma) \rightharpoonup RT$ — *cf.* (6.37) — que, na verdade, é ainda semelhante ao modelo PS. Mostre que, de facto, é possível definir uma função

$$\begin{array}{rcl} \textit{traços} : RT & \to & PS \\ \textit{traços}(a) & \stackrel{\text{def}}{=} & \ldots \end{array}$$

(onde RT é dado por (6.37)) que dê, para cada árvore $a \in RT$, o conjunto-PS de todas as suas trajectórias — traços — desde a raíz até cada folha.

Como definiria, nesta versão de RT, os operadores comportamentais $0, +$ e prefixação?

□

6.2 Comportamentos Recursivos

Todas as expressões de comportamento que foram referidas nas secções anteriores são demasiadamente simples para captar a realidade comportamental que muitas vezes se pretende formalizar. Por exemplo, pensemos num cronómetro que faz contagem de tempo entre o accionamento de um evento $START$ até ao accionamento de um evento $STOP$. A contagem é feita em termos da ocorrência de sucessiva do evento $TICK$. Teremos qualquer coisa como

$$b = START.\underbrace{(TICK.(TICK.(\cdots.(TICK}_{n}.(STOP.0)))))$$

ou, abreviadamente,

$$b = START.\underbrace{TICK.TICK.\cdots.TICK}_{n}.STOP.0 \tag{6.46}$$

Reparemos que a expressão (6.46) se encontra "parametrizada" pelo valor n de cada contagem de tempo. De facto, para cada contagem concreta, o valor de n será diferente. Será que não podemos encontrar uma expressão genérica para o comportamento do nosso cronómetro qualquer que seja a sua contagem? Vejamos como.

O cronómetro estará completamente inactivo a não ser que seja accionado o "botão" $START$, *i.é*

$$b = 0 + START.x \tag{6.47}$$

onde x é o comportamento, para já ainda não especificado, correspondente ao que acontece depois de $START$ ser accionado. É razoável, à luz das interpretações semânticas que atrás se viram, reduzir (6.47) a

$$b = START.x \tag{6.48}$$

Passemos agora a especificar x, o comportamento associado ao processo de contagem. Se $STOP$ fôr accionado, a contagem pára imediatamente:

$$x = STOP + \ldots$$

Caso contrário, o evento $TICK$ marcará uma unidade de tempo e a contagem prosseguirá:

$$x = STOP + TICK.x \tag{6.49}$$

Bom, (6.48) e (6.49) especificam um comportamento para cronómetros do tipo "use e deite fora", pois após $STOP$ o cronómetro ficará inactivo para sempre. Um cronómetro "reutilizável" terá, então, o comportamento

$$\left\{ \begin{array}{lcl} b & = & START.x \\ x & = & STOP.b + TICK.x \end{array} \right. \tag{6.50}$$

Reparemos agora que tanto a equação (6.49) como as duas equações de (6.50) são definições (mutuamente) recursivas. O que nos faz perguntar, de imediato, se estaremos num contexto algébrico em que equações recursivas tenham significado.

Não trataremos este problema com a profundidade que ele merece, mas antes mostraremos que o tratamento semântico da linguagem de eventos B_Γ é crucial para a sua resolução. Peguemos na equação recursiva (6.49), e generalizemos os eventos concretos $STOP$ e $TICK$ por símbolos de eventos abstractos α e β:

$$x = \alpha + \beta.x \tag{6.51}$$

Consideremos a interpretação-PS, por exemplo, segundo a qual teremos a seguinte tradução semântica de (6.51):

$$\begin{array}{rcl} x & = & PS(\alpha + \beta.x) \\ & = & \{<>, <\alpha>\} \cup \{cons(\beta, b) \mid \ b \in x\} \end{array} \tag{6.52}$$

Não é difícil conjecturar uma $c.p.o.$ sobre o domínio-PS na qual (6.52) pode vir a ser resolvida pelo Teorema de Kleene: $\perp$ será o menor conjunto de traços fechados por prefixos, $\{<>\}$, e $\leq$ a inclusão de conjuntos fechados por prefixos.

Exercício 6.6 Mostre que a simples união de conjuntos fechados por prefixos é um conjunto fechado por prefixos.
□

Contudo, relembremos que a axiomatização que consta do Exercício 5.3 é completa para a interpretação-PS. Logo, essa axiomatização deverá ser suficiente para resolvermos (6.51) sem nos preocuparmos com os cálculos de (6.52). O comportamento 0 corresponde à base de iteração de Kleene (relembrar que $PS(0) = \{<>\}$) e a ordem será caracterizável pelas cláusulas

$$\begin{array}{rcl} b & \leq & b + b' \\ b' & \leq & b + b' \end{array}$$

para todo o $b, b' \in B_\Gamma$.

Na seguinte construção da cadeia de Kleene correspondente à equação (6.51), para $f(x) = \alpha + \beta.x$, usaremos a seguinte convenção de notação, para qualquer evento $\alpha \in Eve(\Gamma)$, $b \in B_\Gamma$ e $n \in I\!\!N_0$:

$$\begin{array}{rcl} \alpha^{n+1}.b & = & \alpha.\alpha^n.b \\ \alpha^0.b & = & b \end{array}$$

abreviando-se ainda $\alpha^n.0$ para α^n. Teremos então:

$$\begin{array}{rcl} f^0(0) & = & 0 \\ f^1(0) & = & f(f^0(0)) = f(0) = \alpha + \beta.0 = \alpha + \beta \\ f^2(0) & = & \alpha + \beta.(\alpha + \beta) = \alpha + \beta.\alpha + \beta^2 \\ f^3(0) & = & \alpha + \beta.(\alpha + \beta.\alpha + \beta^2) = \alpha + \beta.\alpha + \beta^2.\alpha + \beta^3 \\ \vdots & & \\ f^i(0) & = & \beta^i + \sum_{j=0}^{i-1} \beta^j \alpha \\ \vdots & & \end{array}$$

ou seja, no limite $i = \infty$,

$$\mu f = \beta^* + \beta^*.\alpha \tag{6.53}$$

designando por β^* o comportamento cuja interpretação-PS é $\{\beta\}^*$. Como $\beta^n \leq \beta^n.\alpha$, teremos que $\beta^n + \beta^n.\alpha = \beta^n.\alpha$, ou seja, obtemos o menor dos pontos fixos

$$\mu f = \beta^*.\alpha \tag{6.54}$$

O resultado (6.54) confirma a nossa intuição: no comportamento especificado por (6.51) pode indefinidamente acontecer o evento β, atingindo-se a inactividade após acontecer α.

Exercício 6.7 Mostre que $PS(\beta^n) \subseteq PS(\beta^n.\alpha)$.
□

Voltando ao cronómetro especificado recursivamente por (6.49), teremos que o seu comportamento vem a ser, de acordo com (6.54),

$$TICK^*.STOP$$

ou melhor,

$$START.TICK^*.STOP$$

combinando este resultado com (6.48).

6.3 Interfaces e Modelos com Comportamento

A noção de *interface com eventos* (Definição 5.1) pode agora ser enriquecida com a inclusão de uma expressão de comportamento (recursiva, em geral) que prescreva o comportamento de qualquer modelo seu, ou que, se quizermos, indique qual a actividade possível (e esperável) de um tal modelo.

Definição 6.3 (Interface Comportamental) *Seja $J = (I, s_0, \Gamma)$ uma interface com eventos tal como foi formalizada pela Definição 5.1. Seja $b \in B_\Gamma$ um Γ-comportamento.*

Então ao quádruplo (I, s_0, Γ, b) daremos o nome de interface comportamental. □

Note-se desde já que, uma expressão comportamental $b \in B_\Gamma$ apresentada numa interface com esse comportamento pode ser apresentada recursivamente por um sistema de equações de comportamento, entendendo-se por b o menor dos pontos-fixos da equação que é apresentada em primeiro lugar. É o que acontece com a seguinte interface $CRONO$ de um cronómetro, cujo comportamento se vai buscar a (6.50):

$$\begin{array}{rl} interface & CRONO \\ sorts & Tempo \\ ops & 0 :\to Tempo \\ & suc : Tempo \to Tempo \\ state & \Sigma : Tempo \\ events & START :\to \\ & STOP :\to Tempo \\ & TICK :\to \\ behaviour & b = START.x \\ & x = STOP(t').b + TICK.x \end{array} \tag{6.55}$$

Em (6.55) aparece, assim, uma cláusula adicional de comportamento referenciada pela palavra-chave *behaviour*.

Vejamos agora um dos possíveis modelos de $CRONO$ — um cronómetro capaz de contar tempo módulo 2^{32}:

$$\begin{array}{lcl} CRONO32BITS & : & \to CRONO \\ CRONO32BITS & \stackrel{\text{def}}{=} & \left\{ \begin{array}{rl} sorts & Tempo \cong \{t-1 \mid \ t \in 2^{32}\} \\ ops & 0 \stackrel{\text{def}}{=} 0 \\ & suc(t) \stackrel{\text{def}}{=} rem(\frac{t+1}{2^{32}}) \\ events & START \stackrel{\text{def}}{=} \sigma' = 0 \\ & STOP(t') \stackrel{\text{def}}{=} t' = \sigma \wedge \sigma' = \sigma \\ & TICK \stackrel{\text{def}}{=} \sigma' = suc(\sigma) \end{array} \right. \end{array} \tag{6.56}$$

Falta apenas discutir a noção de satisfação, por parte de um modelo, com relação a uma interface com comportamento. Se o modelo der semântica total a todos os eventos (é o que acontece em (6.56)) ele satisfaz trivialmente a sua interface, já que, estando semânticamente definida qualquer combinação de eventos, estará também o comportamento oferecido pela interface. Se tal não acontecer, porém, não pode a interface oferecer traços que vêm a ser impossíveis de "correr" no modelo, por haver desajuste entre as pré/pós-condições dos eventos envolvidos.

Seja $t \in PS(b)$ um traço de eventos admissível pelo comportamento b presente numa interface J. Seja $\mathcal{A}$ um (pretenso) modelo de J. Definiremos $\mathcal{A}(t)$ — a avaliação de t em $\mathcal{A}$ — como se segue:

$$\mathcal{A}(<>) = 1_{\mathcal{A}(s_0)} \text{ (identidade sobre o estado } s_0\text{)} \tag{6.57}$$

$$\mathcal{A}(<\alpha_1, \ldots, \alpha_n>) = \mathcal{A}(\alpha_1) \circ \ldots \circ \mathcal{A}(\alpha_n) \tag{6.58}$$

onde "$\circ$" designa a composição de relações, *i.é* para A um conjunto e $R, S \subseteq A \times A$,

$$R \circ S = \{(a, a') \in A \times A \mid \exists a'' \in A : aRa'' \wedge a''Sa'\}$$

cf. (1.52). Note-se ainda que cada $\mathcal{A}(\alpha_i)$ em (6.58) designa a projecção a $\mathcal{A}(s_0)$ da relação especificada em $\mathcal{A}$ como semântica de α_i, *cf.* Definição 5.2. Por exemplo, considere-se o traço

$$t = <START, TICK, STOP(t')>$$

admissível em $CRONO$ (6.55). A avaliação de t em $CRONO32BITS$ será a composição de relações

$$\begin{array}{rc} CRONO32BITS(START) & \circ \\ CRONO32BITS(TICK) & \circ \\ CRONO32BITS(STOP(t')) & \end{array}$$

i.é, após conveniente renomeação de variáveis,

$$(\sigma'' = 0) \circ (\sigma''' = \sigma'' + 1) \circ (\sigma' = \sigma''')$$

ou seja, simplificando,

$$\begin{array}{rc} (\sigma'' = 0) \circ (\sigma' = \sigma'' + 1) & \equiv \\ \sigma' = 0 + 1 & \equiv \\ \sigma' = 1 & \end{array}$$

qualquer que seja o estado inicial σ.

Podemos finalmente estabelecer que $\mathcal{A}$ satisfaz uma interface

$$J = (I, s_0, \Gamma, b)$$

desde que, para todo o $t \in PS(b)$, $\mathcal{A}(t) \supset \emptyset$, ou seja nenhum traço oferecido na interface é impraticável no modelo. Veja-se o seguinte contra-exemplo: suponhamos que à interface $STACKS$, satisfeita pelo modelo $STACKLIST$ (5.9), é adicionada uma expressão de comportamento que oferece o traço

$$t = <INIT, POP(j')>$$

Ora é fácil de ver que a avaliação de t em $STACKLIST$ conduz a uma relação binária sobre a espécie $Stack$ que é descrita pela condição

$$\sigma'' = <> \wedge \sigma'' \neq <> \wedge \sigma' = tail(\sigma)$$

(em que o segundo $\wedge$ é não estrito) que, ao ser universalmente falsa, descreve a relação vazia. Logo, $SACKLIST$ passa a não satisfazer esse enriquecimento comportamental da interface $STACKS$.

6.4 Exercícios

Exercício 6.8 Suponha que se enriquece a interface $STACKS$ com o comportamento:

$$\begin{aligned} b &= INIT.PUSH(i).x \\ x &= PUSH(j).x + POP(j').x \end{aligned}$$

Verifique se $SACKLIST$ satisfaz essa interface comportamental.
□

Exercício 6.9 É muito popular um modelo de relógio digital com um visor de quatro dígitos e dois botões, $S1$ e $S2$ — por vezes designados *Mode* e *Set*, respectivamente — que obedece às seguintes instruções:

Operation of Your Watch

- RUN I MODE

  Hours & Minutes : Normal display shows hours and minutes.
  Month & Date : Press $S1$ once and release.
  Seconds : Assuming normal display, press $S1$ and release.
  Normal Display : Press $S1$ once and release.

- RUN II MODE

  Hours & Minutes/Month & Date : Assuming normal display, press $S2$ once, the date and time will begin flashing back and forth for 2 seconds each.
  Seconds : Press $S1$ once and release.

- SETTING YOUR WATCH

  Month : Press $S2$ once if at RUN II MODE, or press $S2$ twice if at RUN I MODE. Advance by pressing $S1$.
  Date : Press $S2$ once and release. Advance by pressing $S1$.
  Hours : Press $S2$ once and release. Advance by pressing $S1$.
  Minutes : Press $S2$ once and release. Advance by pressing $S1$. The seconds are reset to zero and placed on hold.
  Normal Display : Press $S2$ once to change to hold time display. Then, press $S1$ once to change to normal display.

Suponha que alguém pretende especificar formalmente a interface deste tipo de relógio, tendo chegado ao seguinte esboço:

$$
\begin{array}{rl}
interface & WATCH \\
sorts & \ldots \\
ops & \ldots \\
state & \ldots \\
events & S1, S2 :\rightarrow \\
behaviour & \begin{array}{rcl}
b & = & S1.h\&d + S2.flash \\
h\&d & = & S1.seconds \\
seconds & = & S1.\cdots \\
flash & = & S1.\cdots + S2.month \\
month & = & S1.\cdots + S2.\cdots \\
date & = & S1.\cdots + S2.\cdots \\
hours & = & \cdots \\
minutes & = & \cdots
\end{array}
\end{array}
$$

Complete a cláusula de comportamento ($behaviour$) da interface $WATCH$ de acordo com as instruções acima.

□

Exercício 6.10 Um assunto que ficou por abordar neste capítulo foi a composição modular de modelos cuja interface exibe comportamento. Uma forma de exprimir o comportamento do modelo resultante passa pela introdução de mais um operador sobre B_Γ,

$$b \mid b' : B_\Gamma \times B_\Gamma \longrightarrow B_\Gamma$$

cujo efeito é colocar *em paralelo* os comportamentos b e b', emulando a sua "execução em partilha de tempo" típica de qualquer sistema operativo. Assim, $b \mid b'$ pode realizar todos os eventos em b ou b', intercalando-os de forma arbitrária, mas respeitando sempre a ordem da sua ocorrência em cada uma das parcelas.

Defina a interpretação PS do operador $\mid$ e verifique se são válidas, em PS, as igualdades seguintes:

$$b \mid b \equiv b \tag{6.59}$$

$$b \mid 0 \equiv 0 \tag{6.60}$$

□

Exercício 6.11 No exercício anterior definiu-se a interpretação-PS da construção $b = b_1 \mid b_2$, de forma simplificada.

De facto, sempre que, em $b_1 \mid b_2$, um evento α ocorre tanto em b_1 como em b_2, dizemos que α é uma *comunicação* entre b_1 e b_2. Assim, a correspondente interpretação PS deverá possibilitar quer a ocorrência separada de α em b_1 e em b_2 (comunicação não realizada) quer a ocorrência simultânea de α em b_1 e em b_2 (comunicação realizada), que se representa por um evento especial τ. Por exemplo, no comportamento

$$b = \alpha.\beta.0 \mid \gamma.\alpha.0$$

a ocorrência simultânea de α traduzir-se-á na adição dos traços $<\gamma, \tau>$ e $<\gamma, \tau, \beta>$ e seus prefixos.

Redefina $PS(b_1 \mid b_2)$ para o caso geral, tendo em conta esta sofisticação.

□

Exercício 6.12 Nos exercícios anteriores foi abordada a interpretação PS de um operador sobre B_Γ,

$$b_1 \mid b_2 : B_\Gamma \times B_\Gamma \longrightarrow B_\Gamma$$

cujo efeito é colocar *em paralelo* os comportamentos b_1 e b_2, emulando a sua "execução em partilha de tempo" típica de qualquer sistema operativo. Em $b_1 \mid b_2$, se um evento α ocorre tanto em b_1 como em b_2, dizemos que α é uma *comunicação* entre b_1 e b_2. Assim, a correspondente interpretação PS possibilita quer a ocorrência separada de α em b_1 e em b_2 (comunicação não realizada) quer a ocorrência simultânea de α em b_1 e em b_2 (comunicação realizada), que se representa por um evento especial τ. Por exemplo, no comportamento

$$b = \alpha.\beta.0 \mid \gamma.\alpha.0$$

a ocorrência simultânea de α traduzir-se-á na adição dos traços $<\gamma, \tau>$ e $<\gamma, \tau, \beta>$ e seus prefixos.

Suponha agora um outro conectivo de composição paralela,

$$b_1 \parallel b_2 : B_\Gamma \times B_\Gamma \longrightarrow B_\Gamma$$

idêntico ao anterior mas que impede a realização de qualquer comunicação não realizada. No exemplo anterior, os traços $<\alpha, \gamma>$ ou $<\gamma, \alpha, \beta, \alpha>$, entre outros, não são mais possíveis.

Complete a seguinte definição de $b_1 \parallel b_2$ em termos de outros conectivos já estudados:

$$b_1 \parallel b_2 \stackrel{\text{def}}{=} b_1 \mid b_2 \ldots$$

Justifique a sua resposta.
□

6.5 Notas Bibliográficas

O objectivo este capítulo foi apresentar ao leitor a técnica de especificação da semântica *dinâmica* de modelos com eventos (operações) através de expressões de comportamento (possivelmente recursivas) 'à la' Hennessy [Hen88], que descrevem a articulação e encadeamento previsível dos eventos oferecidos na sua interface (visíveis), num contexto modular.

É de referir que, neste estudo da *vertente comportamental* da especificação formal, as abordagens aqui estudadas apenas "tocam a ponta do 'iceberg"' que é a teoria algébrica de processos, como o leitor poderá ver continuando a ler [Hen88], de onde se foram buscar apenas as interpretações PS e RT [3]. A completude da axiomatização proposta para o modelo PS (*cf.* Exercício 6.1) é demonstrada a pp.41-43 de [Hen88].

O estudo da linguagem EPL de [Hen88] para descrição de processos pode ser encarado como motivação para a análise de outras notações mais elaboradas para processos, como CCS [Mil89] e CSP [Hoa85]. A linguagem comercial OCCAM [JG86] anima especificações CSP e a sua concepção tem muito a ver com o desenvolvimento, na última década, de um novo componente electrónico, o 'transputer'.

[3] O Exercício 6.3 inspirou-se em [Cos89].

Finalmente, refira-se o trabalho pioneiro de Park [Par80] na definição de uma semântica relacional dinâmica para o não-determinismo computacional, numa abordagem por pontos-fixos. Note-se que o estudo de processos conduz à extensão dos domínios denotacionais convencionais, por exemplo, de traços finitos para traços infinitos, e de pontos-fixos mínimos para pontos-fixos máximos. Por exemplo, [Par80] [4] mostra que o maior dos pontos-fixos da equação (6.51) é $\beta^\omega + \beta^*.\alpha$, onde β^ω designa o traço infinito de eventos β. Este ponto-fixo condiz, afinal, com a nossa intuição quanto à possibilidade de um processo, que se comporta segundo essa equação, optar para sempre pelo ramo $\beta.x$ da alternativa $\alpha + \beta.x$ e, portanto, não terminar nunca.

[4] Ver também pp.248–252 de [MA86]

Parte III

Reificação

Capítulo 7

Introdução

7.1 Sobre o Binómio Especificação-Reificação

A investigação em tecnologia da programação de computadores tem evidenciado, ao longo dos últimos anos, a necessidade de se separar qualquer projecto de 'software' em duas fases complementares: primeiro a sua *especificação formal* — em que se escreve um texto matemático que prescreve sem ambiguidade "o que" o 'software' em causa deve vir a fazer — seguida da sua *implementação* — em que se acabará por produzir código máquina instruindo o 'hardware' sobre "como" fazer o que se preescreveu na especificação de que se partiu.

Nos capítulos anteriores preocupámo-nos em estudar técnicas de *especificação formal*. É agora a altura de abordarmos técnicas para a *implementação* de especificações.

Começaremos por notar que, uma vez construída uma especificação, há em geral mais do que uma maneira de instruir uma dada máquina para que esta realize "o que" o especificador desenhou (pense-se, por exemplo, na liberdade de escolha de algorítmos, de estruturas de dados, da linguagens de programação, do sistema operativo, do compilador para essa linguagem *etc. etc.*). Portanto, a relação entre especificação e implementação é *de uma para muitas*, quer dizer, especificações são mais *abstractas* que implementações.

De facto, por detrás da escrita de uma especificação está a ideia que ela vai ser lida, entendida e sujeita a raciocínios por seres humanos, que só querem pensar no problema em causa *abstraindo* de todos os pormenores do foro computacional. Já uma implementação se destina apenas a ser executada por uma máquina, tão eficientemente quanto possível. Assim, todos os "truques" de programação introduzidos numa implementação para aumentar a sua eficiência são, por inerência, detalhes irrelevantes e sem significado ao nível da especificação de que se partiu [1]. Em suma, o salto "epistemológico" entre especificações e implementações está longe de ser "suave" e é a principal preocupação da chamada tecnologia da *reificação* (ou *refinamento*), um ramo relativamente recente da engenharia da programação de computadores baseada em métodos formais.

É claro que pretendemos sempre que o comportamento de uma implementação seja *exactamente* aquele que ficou prescrito na sua especificação formal, requisito sem o qual a tecnologia do refinamento perderá toda a sua relevância. Assim, o princípio da *verificação matemática da correcção* de uma implementação face à sua especificação é central a qualquer metodologia de reificação e, na verdade, aquilo que há de mais essencial no método formal adoptado. Trata-se, antes de mais, de caracterizar matematicamente a palavra "exactamente" que acima se escreveu. Depois, há que viabilizar tecnologicamente o processo de verificação formal, esta sendo historicamente a dificuldade mais difícil de superar, como se discute em seguida.

7.2 Análise do "Estado da Arte"

O problema da *verificação de correcção* de um programa insere-se num contexto mais lato que é o da chamada *validação* de programas, ou, se quisermos, no do *controlo de qualidade* do 'software' enquanto produto industrial.

A primeira forma de controlo da qualidade de um programa que se conhece designa-se por *teste* — 'debug' em jargão informático anglo-saxónico — e é, ainda hoje e em larga medida, a mais utilizada. Consiste em corrigir um programa por execuções sucessivas, num processo de "erro e nova tentativa" próprio da actuação dos seres irracionais. Os seus inconvenientes são bem conhecidos:

- é assumidamente um método anti-científico;

- é um método ineficaz pois, como ficou registado na frase histórica de E.W. Dijkstra,

  > *... o teste de um programa apenas demonstra a* existência *de erros, nunca a sua ausência.*

  Assim [2],

$$\begin{array}{rcll} \text{teste revela erros} & \Rightarrow & \text{programa tem erros} & (p \Rightarrow q) \\ \text{teste não revela erros} & \not\Rightarrow & \text{programa não tem erros} & (\neg p \not\Rightarrow \neg q) \end{array}$$

[1] É precisamente por criar uma confusão entre esses detalhes e a essência do problema em causa que, um programa que o resolva — mesmo profusamente equipado com comentários adequados — não pode ser considerado a documentação de si próprio.

[2] Faça-se a analogia com a verificação de uma soma aritmética via "prova dos 9"...

- é um método de aplicação restrita, pois muitas aplicações em "tempo real" são virtualmente intestáveis, *e.g.* em áreas como a aquisição de dados 'on line', o controlo anti-míssil *etc.* [3];

- é um método que implica custos elevados, pois "errando e tentando de novo" o programador necessitará de mais tempo para o desenvolvimento e o seu empregador terá maiores encargos financeiros (salários *etc.*);

- é um método que conduz à excessiva utilização de recursos computacionais, agravando os encargos financeiros do desenvolvimento pela compra de mais máquinas do que necessário;

- é um método irresponsável, pois a impossibilidade humana de testar até à exaustão um programa "iliba" qualquer programador das suas responsabilidades profissionais, o que é contrário ao espírito da moderna produção tecnológica;

- é um método de "cura" deficiente, já que a experiência mostra que muitos erros detectados na fase de teste de um programa foram originados muito cedo no seu processo de desenvolvimento; remover tais erros no fim de tudo (em vez da sua detecção no início) é obviamente mais difícil e trabalhoso — *e.g.* recuperar de uma má escolha de uma estrutura de dados pode forçar a substituição de todos os algorítmos a ela associados;

- finalmente, é um método que agrava o processo de manutenção pois, com os inconvenientes acima referidos, a manutenção do código ocupará inevitavelmente um lugar predominante na contabilização total dos custos de produção de 'software'.

Uma forma alternativa para controlo da qualidade dos programas — a chamada *verificação* — demarca-se em relação ao *teste* ao pretender verificar a correcção de um programa sem que se precise de o executar. Esta ideia para controlo de qualidade em 'software' surgiu nos anos 60 a partir de trabalhos de investigação que mostraram ser possível "calcular" o significado matemático de um programa de computador [4]. Passava assim a poder ser equacionada uma especificação E de um problema a resolver, com um programa P candidato a solução. Surgiu então o princípio básico da *verificação*: dada a especificação E, constrói-se um programa P que suposta e assumidamente satisfaz essa especificação; o passo de verificação consiste em provar que

$$[\![P]\!] \;\underline{\mathrm{sat}}\; E \tag{7.1}$$

[3] Faça-se aqui a analogia com a impossibilidade de simular em laboratório a carga de um gerador de energia da ordem de *GWatts* de potência.

[4] Cf. trabalhos pioneiros como os de McCarthy [McC63] e Floyd [Flo67] que estão na origem das técnicas de *semântica denotacional* das linguagens de programação.

onde $[\![P]\!]$ designa a semântica formal, ou denotacional de P (*cf.* secção 7.7) e $\underline{\mathrm{sat}}$ é uma relação lógica de *satisfação* adequada ao formalismo de especificação.

O processo de controlo de qualidade transformava-se assim num exercício de matemática. Mas é de suspeitar que, para especificações E de programas "reais" P, com alguns milhares de linhas de código, tanto o cálculo de $[\![P]\!]$ como a demonstração de (7.1) sejam incomportáveis na prática. Daí o "falhanço histórico" do método, hoje designado por *verificação estática* ou *a posteriori*. O termo *a posteriori* retém o facto de, neste método de verificação, só ser iniciado o argumento de correcção *após* todo o programa P se encontrar realizado. Ora o "bom senso" aponta para a necessidade de se fazerem argumentos de correcção comportáveis, o que, para programas "reais", só é possível se o argumento se iniciar, de alguma forma, *antes* da síntese final do programa a desenvolver. A ideia foi então a de reduzir a complexidade do argumento de correcção estruturando-o em sub-argumentos, da mesma forma que um matemático, perante a necessidade de provar um teorema elaborado, estrategicamente o decompõe em teoremas auxiliares (*lemas*), 'mind sized', que prova isoladamente.

Suponhamos que a especificação E, de onde se parte para a construção de P, é — por muito complicada que seja — uma expressão matemática da forma abstracta

$$E = \varphi(E_1, \ldots, E_n)$$

Será estratégico ver em cada E_i uma "sub-especificação" do problema em causa, sub-especificação no sentido de especificar "uma parte" do problema original. Salta à ideia pensar-se que talvez exista um programa P_i, com toda a probabilidade mais "pequeno" que P, que implementa cada E_i. Que cada P_i satisfaz E_i pode verificar-se demonstrando n-lemas da forma

$$[\![P_i]\!] \ \underline{\mathrm{sat}}\ E_i$$

Faltará agora, apenas, descobrir como combinar os n-subprogramas P_i entre si, por exemplo, construindo

$$\mathcal{E}(P_1, \ldots, P_n)$$

onde $\mathcal{E}$ designa uma construção sintática adequada disponível na linguagem de programação que está a ser usada (*e.g.* `if` P_1 `then` P_2 `else` P_3), por forma a que o teorema "final",

$$\frac{[\![P_1]\!] \ \underline{\mathrm{sat}}\ E_1 \wedge \ldots \wedge [\![P_n]\!] \ \underline{\mathrm{sat}}\ E_n}{[\![\underbrace{\mathcal{E}(P_1, \ldots, P_n)}_{P}]\!] \ \underline{\mathrm{sat}}\ \underbrace{\varphi(E_1, \ldots, E_n)}_{E}} \tag{7.2}$$

possa ser de complexidade aceitável. É claro que a estratégia seguida pode ser repetida estruturalmente: se demonstrar um dos lemas

$$[\![P_i]\!] \ \underline{\mathrm{sat}}\ E_i$$

é ainda um processo complicado, nada nos impede de pensar em decompor

$$P_i = \mathcal{E}_i(P_{i_1}, \ldots, P_{i_m})$$

tendo inspeccionado entretanto que

$$E_i = \varphi_i(E_{i_1}, \ldots, E_{i_m})$$

e assim sucessivamente. Constrói-se assim um processo de verificação estruturado em *árvore*, por lemas aninhados, típica dos raciocínios matemáticos extensos, que acaba por gerar, ao fim e ao cabo, um programa P cuja estrutura sintática reflecte essa árvore de prova.

Designa-se por *construtiva* esta variante da verificação de programas que suplantou a verificação estática, *cf.* o esquema

$$\text{Validação} \left\{ \begin{array}{l} \text{Teste} \\ \text{Verificação} \left\{ \begin{array}{l} \text{Estática ('a posteriori')} \\ \text{Construtiva} \end{array} \right. \end{array} \right. \tag{7.3}$$

É claro que nem todas as linguagens de programação são "igualmente boas" para se fazer verificação construtiva. É preciso que aos seus combinadores sintáticos (*cf.* $\mathcal{E}$ acima) se possam associar combinadores matemáticos (*cf. e.g.* φ_i, φ acima). Data de então a hoje bem conhecida técnica da *Programação Estruturada*, iniciada com linguagens como o ALGOL 60 mas sobretudo divulgada pelo PASCAL, embora à maior parte da comunidade informática que a adoptou tenham sido alheias as razões (do foro teórico) que motivaram a mudança de tecnologia [5]. Na nova ordem das coisas salvaram-se as primitivas de estruturação sintática "bem comportadas" como, por exemplo, `if ...then ...else ...`, `while ...do ...`, `repeat ...until ...` *etc.*, e baniram-se aquelas que, como `goto ...`, tornavam muito complexos — senão impossíveis — raciocínios estruturais como (7.2) [6].

7.3 Refinamento (Reificação) por Fases

A "fórmula" de Niklaus Wirth,

Algorithms + Data Structures = Programs

[5] Ainda hoje muita gente ignora esta origem de facto da Programação Estruturada (que fez furor nos anos 70), que não foi apenas uma simples mudança de "sensibilidade" conducente a um "novo" estilo de programação baseado numa sintaxe (aparentemente!) mais "rica" para a programação de computadores.

[6] De facto, não se trata de banir *todos* os `gotos` da programação, mas apenas o seu uso indiscriminado, *cf.* o título "provocatório" do conhecido artigo de D. Knuth: *Structured Programming with Goto Statements* [Knu74].

que ficou conhecida pelo título do seu livro de texto [Wir76], hoje um "clássico" da literatura em Programação Estruturada dos anos 70, pode talvez ser encarada como a primeira mensagem ao "grande público" de que *programas "são" estruturas algébricas* — *cf.*

Operadores + *Conjuntos* = *Álgebras*

— de facto o 'moto' de uma vasta quantidade de investigação levada a cabo nas duas últimas décadas sobre a aplicação da Álgebra Universal às ciências da programação.

Uma das escolas do pensamento em especificação algébrica designa-se *orientada a modelos*, ou *construtiva* porque, segundo ela, as especificações escrevem-se explicitamente como *modelos*, *i.é* álgebras, em vez de serem definidas (implicitamente) por conjuntos de axiomas ou propriedades. As linguagens de especificação construtiva (*e.g.* VDM [Jon86] ou Z [Spi89]) permitem operações não-determinísticas sobre um estado local, que dizer, trabalham com *álgebras relacionais* [7]. Neste contexto, a "fórmula" acima re-escrever-se-á em

Relações + *Conjuntos* = *Álgebras Relacionais*

A visão algébrica da programação tem o benefício de nos permitir decompôr os nossos raciocínios sobre os programas segundo dois eixos ortogonais (*cf.* a Figura 7.1): as *entidades* manipuladas pelos programas (*i.é* estruturas de dados) e as *operações* que as manipulam (*e.g.* procedimentos, funções *etc.*).

O refinamento não é excepção a este respeito. A maneira como em VDM se procede de especificações abstractas em direcção a modelos cada vez mais concretos (cada vez mais próximos da máquina destino, disponível por 'hardware' ou sob a forma de uma linguagem de programação) designa-se por "refinamento de modelos". Em VDM recomenda-se que um dos eixos acima — o das *estruturas de dados* — seja considerado em primeiro lugar, possivelmente envolvendo várias iterações. Logo que os modelos para as estruturas de dados assim encontrados se consideram satisfatórios (no sentido de se encontrarem já disponíveis na máquina destino), as decisões de refinamento são tomadas na direcção ortogonal do *refinamento algorítmico*, por forma a, eventualmente, atingir código executável (*e.g.* codificado em C, PL/1, PASCAL *etc.*), *cf.* a Figura 7.2. É esta a estratégia que adoptaremos aqui e que passaremos a detalhar de seguida.

7.4 Eixos de Refinamento

Tradicionalmente, a estratégia acima esboçada para, gradualmente, atingir a complexidade de uma implementação, não é dedutiva. Como se mostrou na secção

[7] Relembrar a Parte II.

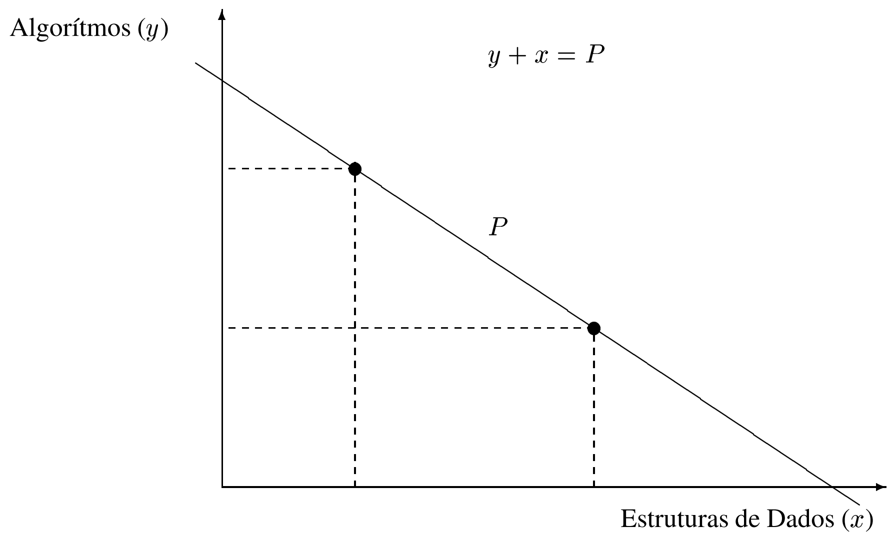


Figura 7.1: ‘Algorithms + Data Structures = Programs’ [Wi76]

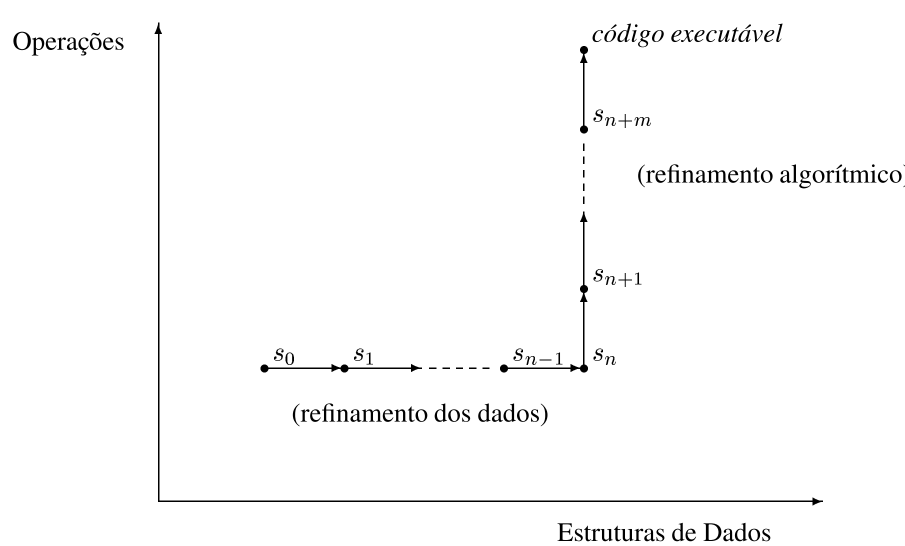


Figura 7.2: Refinamento dos Dados / das Operações

7.1, cada decisão toma-se indutivamente, conjecturando passos de refinamento baseados na intuição e na experiência (deveras arbitrária…) do programador. Assim, a qualidade das implementações ficará dependente da "boa" ou "má" experiência dos respectivos programadores. Contudo, o método acaba por funcionar globalmente ao exigir, em cada passo, a prova formal da sua correcção em face do anterior.

7.4.1 O Eixo das Estruturas de Dados

No que diz respeito às estruturas de dados, a justificação formal da correcção de uma estrutura em relação a uma outra baseia-se nas noções de *representação*, *redundância* e *abstracção*.

Representação

Comecemos por ver um exemplo clássico, que é o da implementação em computador de subconjuntos de um dado domínio de dados A, isto é, o da implementação de 2^A. Algumas linguagens (*e.g.* PASCAL) disponibilizam operadores para manipulação de subconjuntos de um tipo enumerado, desde que este não exceda determinado número de elementos. Este número é, normalmente, o comprimento (ou um seu múltiplo) da palavra-base da máquina sobre a qual a linguagem está implementada, já que a técnica de representação subjacente se baseia na ideia de atribuir uma posição da palavra ('bit') a cada elemento do tipo enumerado A em questão e assinalar a sua presença num dado subconjunto de A forçando esse 'bit' a 1.

Por exemplo, seja a máquina subjacente baseada em palavras de 16-bits [8] e seja $card(A) \leq 16$. Pretendemos que uma palavra represente um dado subconjunto $S \in 2^A$, isto é, $S \subseteq A$. Antes de mais, o compilador deverá ser responsável por atribuir a cada elemento de A uma e uma só posição i entre 0 e 15. Podemos assim designar por a_i o elemento de A cuja posição ('bit') é i. Será agora fácil de deduzir qual o subconjunto de A que uma dada palavra representa—bastará colectar os elementos associados às posições cujo 'bit' toma o valor 1. Por exemplo, a palavra

0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0

(7.4)

representa o conjunto

$$\{a_1, a_3, a_4\}$$

enquanto que a palavra

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[8] Como é óbvio, a estratégia para palavras de mais de 16-bits será em tudo análoga à que se apresenta.

representa o conjunto vazio $\emptyset$, *etc.*

Este exemplo é suficiente para motivar a noção de *representação* — a ideia de que todo o valor do tipo de dados da especificação (neste caso um dado subconjunto S de A) tenha pelo menos um "representante" ao nível da implementação (neste caso uma dada palavra de 16-bits).

Redundância

Reparemos que escrevemos, no parágrafo acima, "pelo menos um" e não "exactamente um". De facto, pode haver mais do que um representante ao nível da implementação para um dado valor da especificação. Por exemplo, suponhamos acima que $card(A) = 14$. Qual a diferença entre os conjuntos representados pela palavra (7.4) e pela que se segue?

0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1

Nenhuma, já que a representação de qualquer subconjunto de A, por este ter apenas 14 elementos, "consome" apenas os 'bits' 0 a 13 e ignora os 'bits' 14 e 15; portanto, estes podem considerar-se redundantes para a representação em causa, *i.é* são "desperdiçados".

A existência de *redundância* em representações fica ainda melhor caracterizada se pensarmos, agora, numa estratégia para representar 2^A quando A tem cardinal arbitrariamente grande (ainda que finito). Este é um problema que se põe inevitavelmente ao recém-iniciado em programação que, confrontado com a limitação da estratégia anterior, acaba por sentir a conveniência de representar subconjuntos de A por listas de A ($A^\star$), eventualmente *ligadas* por apontadores. A redundância desta representação é mais que evidente, pois qualquer subconjunto de A é representável por qualquer lista que o enumere por qualquer ordem, com ou sem repetição de elementos. Por exemplo, o conjunto

$$\{a_1, a_3, a_4\}$$

será representável tanto pela lista

$$<a_1, a_3, a_4>$$

como por

$$<a_3, a_4, a_1>$$

como por

$$<a_3, a_4, a_1, a_3>$$

etc.

Abstracção

Em síntese, como caracterizar formalmente as noções acima descritas, *i.é* como formalizar a relação que existe entre um domínio de dados e qualquer outro que o implemente?

Para garantirmos, a baixo nível, a representação de qualquer valor da especificação, precisamos de fixar a correspondência entre cada elemento a representar e os seus representantes. No exercício em curso (implementar 2^A por $A^\star$), essa correspondência poderá ser:

$$S \leadsto \{l \in A^\star \mid \; elems(l) = S\} \tag{7.5}$$

para todo o $S \in 2^A$. Ora reparemos que a relação pretendida surge mais simples se se inverter a correspondência acima, fixando, para cada sequência $l \in A^\star$ o conjunto $elems(l) \in 2^A$ que l "abstractamente" representa. Afinal, basta ver em $elems$ a essência do processo de representação: $elems$ é a função de *abstracção* que indica, para cada valor concreto qual o valor abstracto que aquele representa [9]. Genericamente, se A é um domínio de dados a implementar, B será uma implementação de A se fôr possível definir uma função de abstracção

$$f : B \longrightarrow A$$

que seja *sobrejectiva* (exigência imposta para garantir representantes para qualquer valor abstracto).

Exercício 7.1 Mostre, por indução sobre 2^A (para A finito) que

$$\begin{array}{lcl} elems & : & A^* \longrightarrow 2^A \\ elems(x) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} x = <> & \Rightarrow & \emptyset \\ x \neq <> & \Rightarrow & \{head(x)\} \cup elems(tail(x)) \end{array} \right. \end{array}$$

é uma função sobrejectiva, *i.é* que conjuntos podem ser adequadamente representadas por sequências.
□

7.4.2 O Eixo das Operações

No eixo ortogonal das operações, pretendemos agora caracterizar, também formalmente, a relação que deve existir entre um processo algorítmico e qualquer outro que o implemente. A justificação formal da correcção de um dado algorítmo em relação a um outro baseia-se nas noções de *simulação* e *concretização*.

[9] Formalmente, se se relembrar o Teorema Fundamental do Homomorfismo, poderemos ver em (7.5) a *função natural* que estabelece o isomorfismo entre 2^A e o quociente $A^\star/K_{elems}$. Logo 2^A é mais abstracto que $A^\star$.

Simulação

Em especificação algébrica, um processo algorítmico pode ser especificado por uma função matemática, possivelmente recursiva, da forma

$$\sigma : A \longrightarrow B$$

Suponhamos que, no eixo de refinamento das espécies dos dados se tomaram, entretanto, as seguintes decisões de refinamento:

$$\begin{tikzcd}
A \arrow[r, "\sigma"] & B \\
A_1 \arrow[u, "f"] & B_1 \arrow[u, "g"']
\end{tikzcd}$$

onde f e g são funções de abstracção (sobrejectivas). Antes de mais, vamos pretender encontrar uma função σ_1 que, ao nível de A_1 e de B_1, “simule” o comportamento de σ:

$$\begin{tikzcd}
A \arrow[r, "\sigma"] & B \\
A_1 \arrow[u, "f"] \arrow[r, "\sigma_1"'] & B_1 \arrow[u, "g"']
\end{tikzcd}$$

isto é, que verifique

$$\forall a_1 \in A_1 : g(\sigma_1(a_1)) = \sigma(f(a_1)) \tag{7.6}$$

numa espécie de “efeito ‘pullback’ ” [10]. Note-se desde já que pode existir mais do que um σ_1 que simule σ, *i.é* temos de novo a relação *de um para muitos* típica do refinamento. O efeito de “simulação” corresponde à ideia de que σ_1 pode ser invocada em lugar de σ para obter o mesmo efeito que esta última:

$$\begin{array}{lll} \sigma(a) \stackrel{\mathrm{def}}{=} & let & a_1 \in \{a' \in A_1 \mid f(a') = a\} \\ & in & g(\sigma_1(a_1)) \end{array}$$

[10] Analogia com a construção da teoria das categorias com o mesmo nome.

Começa-se por selecionar um qualquer representante válido do argumento de σ, que se submete, a baixo nível, à simulação σ_1; o resultado assim obtido é abstraído por g dando o mesmo resultado que σ daria, a alto nível.

Exercício 7.2 Considere o seguinte diagrama de refinamento do operador de pertença ($\in$) de conjuntos para sequências:

$$\begin{tikzcd} A \times 2^A \arrow[r, "\in"] & 2 \\ A \times A^\star \arrow[u, "1_A \times elems"] \arrow[r, "belongs"'] & 2 \arrow[u, "1_2"'] \end{tikzcd} \tag{7.7}$$

Mostre por indução sobre $A^\star$ que a função

$$belongs(x,y) \stackrel{\text{def}}{=} \left\{ \begin{array}{lll} y = <> & \Rightarrow & F \\ y \neq <> & \Rightarrow & let \quad h = head(y) \\ & & in \quad \left\{ \begin{array}{lll} h = x & \Rightarrow & T \\ h \neq x & \Rightarrow & belongs(x, tail(y)) \end{array} \right. \end{array} \right. \tag{7.8}$$

satisfaz esse diagrama.
□

Note-se que a garantia de simulação "fiel" afecta apenas a funcionalidade dos operadores. Não fica garantida uma boa eficiência, como pode ser avaliado pelo exercício que se segue.

Exercício 7.3 Mostre que a concatenação de sequências implementa correctamente a união de conjuntos, *i.é* que

$$elems(l \frown r) = elems(l) \cup elems(r)$$

Qual o problema desta implementação do ponto de vista de eficiência?
□

Como o nível da implementação é mais rico em pormenores concretos, não admira que haja processos algorítmicos a baixo nível que não se manifestam a alto nível. É o que se pode sentir no problema que se segue.

Exercício 7.4 Mostre que, quando uma sequência implementa um conjunto, a operação que lhe filtra elementos repetidos ('garbage collection') é um mero pormenor de implementação. (**Sugestão**: mostre que essa operação refina a função identidade entre conjuntos.)
□

Concretização Algorítmica

Uma vez fixado o nível de simulação a que uma implementação vai ser concretizada, importa agora investir no eixo algorítmico no sentido de obter funções computáveis e eficientes. O processo é sempre estrutural, factorizando a construção de funções complexas em termos de funções mais simples, até que estas sejam executáveis sob a forma de um algoritmo.

Vejamos, a título de exemplo, um processo de concretização da função de multiplicação de números inteiros:

$$\begin{array}{rcl} mult & : & \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z} \\ mult(x, y) & \stackrel{\text{def}}{=} & x \times y \end{array}$$

A principal conjectura do desenvovimento é explorar o sinal do primeiro argumento x de $mult$ e, com base na conhecida propriedade da multiplicação de inteiros,

$$x \times y \quad = \quad -((-x) \times y)$$

re-escrever a definição de $mult$ em

$$mult(x, y) \quad \stackrel{\text{def}}{=} \quad \left\{ \begin{array}{lcl} x \geq 0 & \Rightarrow & multp(x, y) \\ x < 0 & \Rightarrow & -multp(-x, y) \end{array} \right.$$

deferindo o esforço de desenvolvimento para $multp$, uma nova função que restringe $mult$ com a garantia de que o primeiro argumento é positivo, *i.é* um número natural:

$$\begin{array}{rcl} multp & : & \mathbb{N}_0 \times \mathbb{Z} \longrightarrow \mathbb{Z} \\ multp(x, y) & \stackrel{\text{def}}{=} & x \times y \end{array}$$

Sendo um natural, x poderá ser usado como contador no passo seguinte, em que se desenvolve $multp$ algoritmicamente:

$$\begin{array}{rcl} multp & : & \mathbb{N}_0 \times \mathbb{Z} \longrightarrow \mathbb{Z} \\ multp(x, y) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} x = 0 & \Rightarrow & 0 \\ x > 0 & \Rightarrow & y + multp(x - 1, y) \end{array} \right. \end{array} \tag{7.9}$$

sendo necessário mostrar, antes de mais, que $multp$ (7.9) satisfaz

$$multp(x, y) \quad = \quad x \times y$$

o que se fará sem dificuldade (*e.g.* por indução em $\mathbb{N}_0$).

Se assumirmos mecanicamente disponível a operação $+$, temos já em (7.9) uma função executável (recursiva). Mas prossigamos, já que a ideia é ilustrar a pormenorização algorítmica até à geração de código. O passo seguinte é procurar uma versão não-recursiva de $multp$,

$$\begin{array}{rcl} nrmultp & : & I\!N_0 \times \mathbb{Z} \longrightarrow \mathbb{Z} \\ nrmultp(x, y) & \stackrel{\mathrm{def}}{=} & ciclo(x, y, 0) \end{array}$$

introduzindo a função auxiliar

$$\begin{array}{rcl} ciclo & : & I\!N_0 \times \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z} \\ ciclo(i, y, r) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcl} i = 0 & \Rightarrow & r \\ i > 0 & \Rightarrow & ciclo(i-1, y, y+r) \end{array} \right. \end{array}$$

sendo aqui necessário justificar, que

$$\begin{array}{rcl} multp(x, y) & = & nrmultp(x, y) \\ & = & ciclo(x, y, 0) \end{array}$$

Será fácil a um razoável programador codificar as funções acima desenvolvidas numa linguagem estruturada, *e.g.* PASCAL,

```
function nrmultp(x:integer,y:integer):integer;
(* rely on x >= 0 *)
var r: integer;
    i: integer;
begin i := x; r := 0;
while i > 0 do
      begin r := r + y;
            i := i - 1
      end;
nrmultp := r
end;

function mult(x:integer,y:integer):integer;
var r: integer;
begin
if x >= 0 then r := nrmultp(x,y)
          else r := -nrmultp(-x,y);
mult := r
end;
```

embora um "purista" o pudesse obrigar, entretanto — e para a implementação ficar imaculadamente garantida — a ir substituindo, estruturalmente, as expressões funcionais por expressões denotacionais equivalentes da linguagem PASCAL, até os parêntesis de significado, $[\![\ldots]\!]$, desaparecerem do meio do código, relembrar (7.1) e (7.2). E, de facto, a codificação apresentada só funciona se a semântica matemática das estruturas `if ...then ...else ...`, `while ...do ...`, *etc.* condisser com as expressões matemáticas que se escreveram em $mult$ e $nrmultp\ldots$

7.5 Alternativa Transformacional

Apesar dos seus méritos, a verificação construtiva não é isenta de inconvenientes. Cada nível intermédio é primeiro proposto e depois prova-se que está correcto em face do anterior. Embora melhore muito a primitiva verificação estática — relembrar (7.3) — este estilo tipo "inventa-e-verifica" ainda é, muitas vezes, pouco eficaz, ao assumir que o engenheiro de 'software' tem intuição suficiente para "advinhar" implementações eficientes, o que é improvável, na generalidade dos casos. O ónus das provas de correcção é teoricamente reduzido mas a complexidade do raciocínio matemático exigido em provar passos intermédios de desenvolvimento de soluções para os problemas de 'software' da vida real continua a não ser desprezável. Mais ainda, é sempre frustante ver ser considerável o esforço de provar factos que são intuitivamente óbvios.

Por exemplo, considere-se uma especificação simples de um sistema-"brinquedo" para gestão de contas bancárias[11]:

$$\begin{array}{rcl} BAMS & \cong & AccNr \rightharpoonup Status \\ Status & \cong & H : 2^{AccHolder} \times \\ & & A : Amount \\ Amount & \cong & \mathbb{N}_0 \end{array}$$

onde deve verificar-se a seguinte propriedade invariante:

$$\text{inv-}BAMS(\sigma) \stackrel{\text{def}}{=} \forall n \in dom(\sigma) : H(\sigma(n)) \neq \emptyset$$

obrigando a que cada conta tenha, pelo menos, um titular.

A um praticante de especificação formal poderá custar algum tempo discutir a correcção formal da seguinte (e óbvia!) implementação de $BAMS$ no *modelo relational*, constando de duas relações binárias (vulg. "tabelas"):

$$\begin{array}{rcll} BAMSR & \cong & HT: \; 2^{Row1} \times & \text{/*}\textit{table of account-holders}\text{ */} \\ & & AT: \; 2^{Row2} & \text{/*}\textit{table of amounts}\text{ */} \\ Row1 & \cong & K : AccNr \times H : AccHolder & \\ Row2 & \cong & K : AccNr \times A : Amount & \end{array}$$

[11] Trata-se do mesmo sistema que serviu de exemplo base no início do capítulo 1.

sujeita ao seguinte invariante:

$$\begin{aligned}\text{inv-}BAMSR(\langle ht, at\rangle) \quad &\stackrel{\text{def}}{=} \quad \pi_1[at] = \pi_1[ht] \wedge \\ &\phantom{\stackrel{\text{def}}{=}} \quad depKA(at)\end{aligned} \tag{7.10}$$

onde

$$\pi_1[\rho] \stackrel{\text{def}}{=} \{\pi_1(t) \mid \ t \in \rho\} \tag{7.11}$$

é a projecção pelo primeiro atributo (o *domínio*) de uma relação binária ρ, e onde o predicado

$$\begin{aligned} depKA \quad &: \quad 2^{Row2} \longrightarrow 2 \\ depKA(\rho) \quad &\stackrel{\text{def}}{=} \quad \forall r, s \in \rho : (K(r) = K(s) \Rightarrow A(r) = A(s))\end{aligned} \tag{7.12}$$

exprime a *dependência funcional* $K \to A$.

Quando exemplos "de brinquedo" como estes são escalados a exemplos reais, as demonstrações formais ou são ignoradas (e o método deixa de ser aceitável como "formal"...), ou transformam-se num pesado fardo que estrangula o desenvolvimento. E há questões que se levantam: já que todo o processo se baseia, indutivamente, na conjectura e na invenção, que garantia temos que são inventadas as "melhores" implementações? Não será possível *derivar* equacionalmente (deduzir) implementações a partir das respectivas especificações?

A investigação mais recente tem sugerido estilos alternativos para conduzir a reificação. A ideia é desenvolver um *calculus* que permita que os *programas* possam ser de facto "calculados" dedutivamente a partir da sua própria especificação. Nesta aproximação, um passo intermédio de um projecto deduzir-se-á do anterior de acordo com alguma(s) lei(s) do cálculo, que por princípio será *estrutural* ao permitir que os componentes de uma expressão possam ser calculados isoladamente (*i.é* resultados de refinamento pré-existentes podem ser "re-utilizados"). A redução do ónus da prova matemática obtem-se fazendo cálculo estrutural em lugar de demonstrações a partir de princípios básicos. Aliás, esta é que é a ideia de um *calculus*, como o passado tem testemunhado noutros contextos (*cf.* o cálculo diferencial e integral, a álgebra linear, *etc.*).

É assim que, após uma década de intensiva investigação nos fundamentos dos *métodos formais* para projecto de 'software', é tentador antecipar que os anos 90 venham a assitir à fase de maturação da tecnologia do 'software', desenvolvendo e usando ferramentas a nível industrial baseadas em processos de cálculo. Teremos assim, finalmente, completo o esquema da evolução das técnicas de controlo de qualidade em 'software':

$$\text{Validação} \left\{ \begin{array}{l} \text{Teste} \\ \text{Verificação} \left\{ \begin{array}{l} \text{Estática ('a posteriori')} \\ \text{Construtiva} \end{array} \right. \\ \underline{\text{Cálculo}} \end{array} \right. \tag{7.13}$$

sendo o objectivo dos capítulos que se seguem ensinar a deduzir, ou calcular, programas a partir de especificações.

7.6 “Especificação Reversa”

Refira-se, para terminar, um tópico que naturalmente surge quando equacionamos especificações com implementações — o da *especificação reversa*. Que fazer da imensidade de programas desenvolvidos no passado sem recurso a métodos fiáveis? Será que tais programas estão condenados aos métodos deficientes com que nasceram? Ou é possível “recuperá-los” de alguma forma?

A chamada *especificação reversa* é uma disciplina nascente que pretende, por uma inversão do processo de cálculo de um programa, partir do seu código para a sua (provável!) especificação formal. Não é difícil ver na analogia da Figura 1.5 — entre as construções primitivas da notação que neste texto se utiliza ($Sets$) e estruturas de dados de linguagens de programação estruturada — um ponto de partida para exercícios de inversão de programas escritos nessas linguagens.

Veremos que o cálculo de programas que estudaremos pode ser usado nas duas “direcções”, directa e reversa. Mas, para já, ficaremos com o seguinte exercício de motivação.

Exercício 7.5 Considere o seguinte (pequeno) programa numa linguagem tipo PASCAL:

```
program Exemplo;
cons max = 2048;
type Elem = String;
var top:0..max; stack:array[1..max] of Elem;
function TOP: Elem;  begin TOP := stack[top]
                     end;
procedure INIT; begin top := 0 end;
function POP: Elem;  begin
                     POP := stack[top];
                     top:= top-1
                     end;
procedure PUSH (e:  Elem);  begin
                            top:= top+1;
                            stack [top]:= e
                            end;
function EMPTY: Bool;  begin EMPTY:= (top=0)
                       end;
begin (* main:  not yet implemented *)
end.
```

cujo “significado” lhe deve ser intuitivamente óbvio.

Conjecture uma especificação formal que possa ter sido ponto de partida para este programa.

□

7.7 Notas Bibliográficas

A tecnologia da verificação de correcção de programas teve origem, nos anos 60, nos trabalhos de McCarthy [McC63]. A sua estratégia consistia em descrever o significado de programas sob a forma de funções recursivas, dedutíveis dos 'flowcharts' respectivos, manipulando um vector de variáveis (estado da máquina). O seu trabalho pode ser considerado pioneiro do estudo da semântica denotacional de linguagens de programação, posteriormente desenvolvido e aprofundado em Oxford por Scott & Strachey.

O uso alternativo de *asserções*, *i.é* fórmulas do cálculo de predicados de 1.ª ordem sobre o espaço de estados de 'flowcharts' ou programas em linguagens tipo ALGOL, foi iniciado por Floyd [Flo67]. Sobre a aplicação desta técnica a 'flowcharts' escreveu Manna um capítulo do seu conhecido livro [Man74]. Para programas estruturados, desenvolveu-se uma importante técnica de raciocínio sobre asserções sob a forma de 'proof rules' associadas às estruturas algorítmicas mais habituais, *cf. e.g.* [Hoa69, Dij76].

As referências [Sto81, Gor79] são textos recomendados sobre semântica denotacional de linguagens de programação.

As recentes técnicas de cálculo de programas têm origem na importante escola de *transformação de programas* que se desenvolveu dentro da comunidade da programação funcional, em particular o chamado método 'fold/unfold' [BD77, Dar82, Dar84, Bp82]. Os items 'Automatic Implementation of Abstract Data Types' e 'Synthesis from Unrunnable Specifications', da referência [Dar82], são contribuições particularmente sugestivas.

Capítulo 8

Reificação dos Dados em SETS

Dentro do espírito e estratégia delineada no capítulo anterior, começa-se neste capítulo o estudo de técnicas de *cálculo* de programas a partir das suas especificações. Tal como vem sugerido na Figura 7.2, aborda-se em primeiro lugar o eixo das estruturas de dados. Sempre que é necessário manipular/simplificar expressões funcionais recorre-se ao cálculo funcional que é dado no apêndice B.

8.1 Ordens de Redundância

O *cálculo de estruturas de dados* que aqui se começa a estudar baseia-se em propriedades da notação dos $Sets$ [1]. Numa primeira aproximação, começa-se por ordenar as estruturas (conjuntos) segundo a ordem simples de cardinalidade entre conjuntos.

8.1.1 Redundância Simples

Definição 8.1 (Ordem de Redundância em Sets) $A \preceq B$ *(ler* "A é menos redundante que B" *ou* "B é mais redundante que A"*) é a ordem de cardinalidade sobre* $Sets$*,* i.é *a ordem definida por:*

$$A \preceq B \stackrel{\text{def}}{=} \exists B \stackrel{f}{\longrightarrow} A : f \text{ é sobrejectiva.} \tag{8.1}$$

A função f (que não é única, em geral) será referida como a função de abstracção *de B para A.* $\square$

[1] Cf. Capítulo 1.

Escreveremos $A \preceq_f B$ sempre que pretendemos explicitar funções de abstracção em $\preceq$-raciocínios. Por exemplo,

$$2^A \preceq_{elems} A^\star \tag{8.2}$$

para A finito. A ordem $\preceq$ é reflexiva e transitiva, e a $\preceq$-antissimetria induz o isomorfismo de conjuntos.

Exercício 8.1 Mostre que

$$A \quad \preceq_{1_A} \quad A \tag{8.3}$$
$$A \preceq_f B \wedge B \preceq_g C \quad \Rightarrow \quad A \preceq_{f \circ g} C \tag{8.4}$$
$$A \preceq_f B \wedge B \preceq_g A \quad \Rightarrow \quad A \cong B \tag{8.5}$$

□

A transitividade significa que podemos encadear $\preceq$-passos e sintetizar funções de abstracção para n $\preceq$-saltos. A correspondente versão finitária de (8.4) é, então,

$$\underbrace{A \preceq_{f_1} A_1 \preceq_{f_2} \cdots \preceq_{f_n} A_n}_{f = f_1 \circ \cdots \circ f_n}$$

8.1.2 Sobre-Redundância

Vejamos como é necessária uma elaboração do que acima se expôs, a partir do seguinte exemplo de motivação: pretende-se representar funções em $A \rightharpoonup B$ por relações (tabelas) em $2^{A \times B}$, *i.é* em direcção ao modelo relacional da informação.

É sabido que *toda a função finita "é" uma relação*. Mas nem todas as relações são funções! Só as que tiverem uma *dependência funcional* de A para B [2]. Logo,

$$A \rightharpoonup B \preceq 2^{A \times B}$$

não está correcto, mas sim o raciocínio:

$$\begin{aligned} A \rightharpoonup B \quad &\cong_{mkf} \quad \{\rho \subseteq A \times B \mid \forall \langle a, b \rangle, \langle a', b' \rangle \in \rho : (a = a' \Rightarrow b = b')\} \\ &= \quad \{\rho \in 2^{A \times B} \mid fdp(\rho)\} \\ &= \quad S \end{aligned} \tag{8.6}$$

onde $S = \{\rho \in 2^{A \times B} \mid fdp(\rho)\}$ é um conjunto cuja função característica em $2^{A \times B}$ é o predicado

$$fdp(\rho) \quad \stackrel{\text{def}}{=} \quad \forall \langle a, b \rangle, \langle a', b' \rangle \in \rho : (a = a' \Rightarrow b = b') \tag{8.7}$$

[2] Relembre-se $depKA$ (7.12), a título de exemplo.

e mkf é a função de abstracção que converte cada relação de S na correspondente função [3],

$$mkf(\rho) \stackrel{\text{def}}{=} \left(\begin{array}{c} a \\ the(\{b \in B \mid a\rho b\}) \end{array} \right)_{a \in \pi_1[\rho]} \qquad (8.8)$$

que só está definida para relações ρ satisfazendo (8.7). A condição fdp funciona como um *invariante induzido* pelo refinamento, sendo preciso escrever

$$A \rightharpoonup B \quad \trianglelefteq^{fdp}_{mkf} \quad 2^{A \times B} \qquad (8.9)$$

de acordo com a definição que se segue.

Definição 8.2 (Ordem de Sobre-redundância em Sets) *É a ordem definida como se segue:*

$$A \trianglelefteq B \quad \stackrel{\text{def}}{=} \quad \exists S \subseteq B : A \preceq_f S \qquad (8.10)$$

Seja ϕ a função característica de S em B. Designá-la-emos por invariante de refinamento*, e escreveremos*

$$A \trianglelefteq^{\phi}_{f} B \qquad (8.11)$$

para registar esse invariante e a função de abstracção f. □

Sempre que $A \trianglelefteq^{\phi}_{f} B$, o predicado em $A \times B$

$$\lambda(a, b).(a = f(b)) \wedge \phi(b) \qquad (8.12)$$

designa-se por *invariante de abstracção*.

Exercício 8.2 Os seguintes quatro exemplos de representação da sequência $<a, b, c>$,

$$\begin{array}{rcl} r_1 & = & \left(\begin{array}{ccc} 1 & 2 & 3 \\ a & b & c \end{array} \right) \\ r_2 & = & \{a, b, c\} \\ r_3 & = & \left(\begin{array}{ccc} 7 & 10 & 99 \\ a & b & c \end{array} \right) \\ r_4 & = & \left(\begin{array}{ccc} a & b & c \\ 1 & 2 & 3 \end{array} \right) \end{array}$$

sugerem outros tantos refinamentos possíveis para sequências. Defina o modelo de dados sugerido por cada exemplo e discuta a sua adequação (em cada caso, ou indica um contra-exemplo, *i.é* uma sequência não representável, ou apresenta o $\trianglelefteq$-raciocínio correspondente).
□

[3] O operador the é o que vem definido em (1.49).

Propriedades da Relação de Sobre-redundância

A relação de sobre-redundância é uma pré-ordem, *i.é* satisfaz as propriedades que se seguem:

- Reflexividade:
$$A \trianglelefteq_{1_A}^{\lambda a.TRUE} A$$

- Fecho antissimétrico:
$$A \trianglelefteq_f^{\phi} B \wedge B \trianglelefteq_g^{\gamma} A \Rightarrow A \cong B$$

- Transitividade (notar síntese de invariantes de refinamento):
$$A \trianglelefteq_f^{\phi} B \wedge B \trianglelefteq_g^{\gamma} C \Rightarrow A \trianglelefteq_{f \circ g}^{\rho} C$$

  para
$$\rho(c) = \gamma(c) \wedge \phi(g(c)) \tag{8.13}$$

  onde a conjunção lógica ($\wedge$) deve ser encarada como uma conectiva não-estrita no seu segundo argumento (*e.g.* $F \wedge \bot = F$).

  Para uma cadeia de n $\trianglelefteq$-saltos,
$$\trianglelefteq_i \left\{ \begin{array}{l} f_i \\ \phi_i \end{array} \right.$$

  o invariante e função de abstracção globais são dados por:
$$\trianglelefteq \left\{ \begin{array}{l} f \stackrel{\text{def}}{=} \bigcirc_{i=1}^{n} f_i \\ \phi \stackrel{\text{def}}{=} \lambda x. \bigwedge_{i=n}^{1} \phi_i((\bigcirc_{j=i+1}^{n} f_j)(x)) \end{array} \right. \tag{8.14}$$

Observação sobre a Relação de Sobre-redundância

A seguinte definição é mais forte do que (8.10):

$$A \trianglelefteq_f^{\phi} B \stackrel{\text{def}}{=} f : B_\phi \to A \textit{ existe e é sobrejectiva} \tag{8.15}$$

onde B_ϕ designa o subconjunto $\{b \in B \mid \ \phi(b)\}$. Esta definição é a de facto utilizada até à secção 8.8.1 e protege-nos da *sobre-especificação* de invariantes, pois
$$\phi(b) \stackrel{\text{def}}{=} b \in dom(f)$$
é o "menor" invariante induzível sobre B — menor no sentido em que qualquer enfraquecimento seu conduz à indefinição da função de abstracção f.

Exercício 8.3 O multiconjunto é uma noção intermédia entre o *conjunto* e a *lista*. É "mais do que um conjunto" na medida em que dá lugar à repetição de elementos, mas é "menos do que uma lista" na medida em que não regista qualquer ordem entre os seus elementos.

Verifique que a ordem $\trianglelefteq$ reflecte estas constatações, *i.é* que

$$2^A \quad \trianglelefteq_{dom} \quad A \rightharpoonup I\!N \quad \trianglelefteq_{mkms} \quad A^\star$$

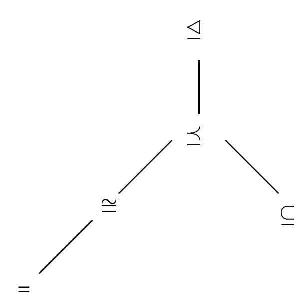
onde $mkms$ (="make multiset") é a função

$$mkms(l) \stackrel{\rm def}{=} \left\{ \begin{array}{lll} l = <> & \Rightarrow & \left(\ \right) \\ l \neq <> & \Rightarrow & let \quad h = head(l) \\ & & \qquad\ \ t = tail(l) \\ & & in \quad \left(\begin{array}{c} h \\ 1 \end{array} \right) \oplus mkms(t) \end{array} \right.$$

e onde $\oplus$ é a união de multiconjuntos que adiciona multiciplidades, *cf.* (1.102).

□

Exercício 8.4 Discuta a validade da seguinte hierarquia de relações em $Sets$:



□

8.2 Cálculo de Redundância

8.2.1 A Estrutura dos $Sets$

Relembrar os construtores primitivos de $Sets$:

- *produto cartesiano*: $A \times B$
- *união disjunta*: $A + B$
- *exponenciação*: A^B para B finito [4]

[4] Cf. $\aleph_0^n = \aleph_0$ para $n = |B|$.

à custa dos quais se definem os seguintes construtores *derivados*:

- *subconjuntos*: 2^A para A finito,
- *relações binárias*: $2^{A \times B}$ para A, B finitos,
- *funções parciais* finitas:

$$A \rightharpoonup B \quad = \quad \bigcup_{K \subseteq A} B^K \tag{8.16}$$

  para A finito,

- *sequências finitas* sobre A:

$$A^\star = \bigcup_{n \geq 0} A^{\overline{n}}$$

- *alternativa "nula"*:

$$A + 1 \tag{8.17}$$

  onde, tipicamente, $1 \cong \{NIL\}$, mas se tem

$$\{0\} \cong \{1\} \cong \ldots \cong \{x\} \tag{8.18}$$

  para qualquer x;

- *definições recursivas*:

$$X \quad \cong \quad \mathcal{F}(X) \tag{8.19}$$

  onde $\mathcal{F}$ é uma expressão em $Sets$ ("functor") envolvendo os construtores acima [5].

Um dos nossos 'running examples' será o seguinte modelo para *árvores de decisão*:

$$DecTree \cong What \times (Answer \rightharpoonup DecTree) \tag{8.20}$$

cujo functor é, esquematicamente:

$$\mathcal{F}(X) \quad \cong \quad A \times (B \rightharpoonup X) \tag{8.21}$$

[5] Notar que nem todos os $\mathcal{F}$ serão permitidos. Mais à frente serão estudadas restrições a $\mathcal{F}$ por forma a (8.19) ter solução em $Sets$.

8.3 O Subcálculo de Isomorfismo

Relembram-se nesta secção as propriedades da relação de isomorfismo em $Sets$ já referidas nas secções 2.3 e 2.3.7. Aí se viu formarem $\times$ e $+$ em $Sets$ um *semi-anel comutativo* [6], a menos de isomorfismo ($\cong$) [7]:

$$
\begin{aligned}
A \times B &\cong B \times A && (8.22)\\
A \times (B \times C) &\cong (A \times B) \times C && (8.23)\\
A \times 1 &\cong A && (8.24)\\
A + B &\cong B + A && (8.25)\\
A + (B + C) &\cong (A + B) + C && (8.26)\\
A + 0 &\cong A && (8.27)\\
A \times 0 &\cong 0 && (8.28)\\
A \times (B + C) &\cong (A \times B) + (A \times C) && (8.29)
\end{aligned}
$$

Assim, fará sentido escrevermos produtos finitos,

$$A_1 \times \ldots \times A_n \ \text{ i.é } \ \Pi_{i=1}^{n} A_i \tag{8.30}$$

assim como uniões disjuntas finitas,

$$A_1 + \cdots + A_n \ \text{ i.é } \ \sum_{i=1}^{n} A_i$$

e tem-se que

$$
\begin{aligned}
\underbrace{A \times \ldots \times A}_{n} &\cong A^n && (8.31)\\
\underbrace{A + \ldots + A}_{n} &\cong n \times A && (8.32)
\end{aligned}
$$

[6] Semi-anel e não anel porque não há inversos aditivos, isto é, $+, 0$ não formam um grupo (formam um monóide abeliano). Aliás, $\times, 1$ também formam um monóide abeliano.

[7] Recomenda-se ao leitor que acompanhe o estudo destas leis com uma análise informal do seu significado prático à luz da correspondência da figura 1.5. Por exemplo, a associatividade do produto (8.23) converte-se na seguinte equivalência óbvia entre estruturas de dados `record`:

```
record
    F: A;
    S: record
           F: B;
           S: C;
       end
end;
```

equivalente a

```
record
    F: record
           F: A;
           S: B
       end;
    S: C;
end;
```

em PASCAL.

como se esperava.

Quanto à exponenciação, será útil relembrarmos os factos seguintes,

$$\begin{aligned} A^0 &\cong 1 && (8.33)\\ A^{(B+C)} &\cong A^B \times A^C && (8.34)\\ A^1 &\cong A && (8.35)\\ 1^A &\cong 1 && (8.36)\\ (B\times C)^A &\cong B^A \times C^A && (8.37)\\ C^{A\times B} &\cong \left(C^B\right)^A && (8.38)\\ A \rightharpoonup B &\cong (B+1)^A && (8.39)\end{aligned}$$

bem como

$$\begin{aligned} A^\star &\cong \sum_{n\geq 0} A^n && (8.40)\\ A \rightharpoonup B &\cong \sum_{K\subseteq A} B^K && (8.41)\end{aligned}$$

pois

$$\begin{aligned} A \neq B &\Rightarrow X^A \cap X^B = \emptyset && (8.42)\\ A\cap B = \emptyset &\Rightarrow A \cup B \cong A + B && (8.43)\end{aligned}$$

e, finalmente,

$$\begin{aligned} A^B \cong A^X \times A^{B-X} &\Leftarrow X \subseteq B && (8.44)\\ A^n &\cong A \times A^{n-1} && (8.45)\\ n \neq m &\Rightarrow X^n \cap X^m = \emptyset && (8.46)\end{aligned}$$

onde as leis (8.45,8.46) podem ser encaradas como instâncias de (8.44,8.42), respectivamente.

Exercício 8.5 Na sequência do Exercício 2.6, defina os isomorfismos correspondentes às leis (8.24) a (8.29) e (8.33) a (8.37), da direita para a esquerda.
□

Exercício 8.6 Com respeito ao isomorfismo implícito na lei (8.38),

$$\begin{aligned} uncurry : (C^B)^A &\rightarrow C^{A\times B}\\ \sigma &\leadsto \lambda(a,b).(\sigma(a))(b)\end{aligned}$$

mostre que

$$\begin{array}{rcl} curry : C^{A\times B} & \to & (C^B)^A \\ \sigma & \leadsto & \lambda a.(\lambda b.\sigma(a,b)) \end{array}$$

é a respectiva aplicação inversa, *i.é* que $curry = uncurry^{-1}$.
□

8.3.1 Exemplos de Aplicação

À primeira vista, a relação de isomorfismo entre domínios de dados não parece ser de grande utilidade para o refinamento, uma vez que não há introdução de redundância. Na prática, acontece o contrário: os resultados que se derivam dentro deste subcálculo são altamente relevantes, como se pode ver por alguns exemplos que se desenvolvem a seguir.

Cálculo da Implementação de $A^\star$ em Lista Recursiva

Neste exemplo desenvolver-se-á o processo de cálculo de listas ligadas como implementação de sequências em $A^\star$. Comecemos por alguns passos óbvios,

$$\begin{array}{rcl} A^\star & = & \bigcup_{n=0}^{\infty} A^n \\ & \cong & \sum_{n=0}^{\infty} A^n \\ & \cong & 1 + A + A^2 + \ldots \end{array} \tag{8.47}$$

cf. respectivamente (8.40), (8.33) e (8.35).
Introduzindo agora duas variáveis auxiliares L e N,

$$\begin{array}{rcl} N & = & A + A^2 + \ldots \\ L & = & 1 + N \end{array} \tag{8.48}$$

teremos, por substituição,

$$A^\star \cong L \tag{8.49}$$

e

$$\begin{array}{rcl} N & = & A + A^2 + \ldots \\ & \cong & A \times 1 + A \times A + A \times A^2 + \ldots \\ & \cong & A \times (1 + A + A^2 + \ldots) \\ & \cong & A \times L \end{array} \tag{8.50}$$

em que se recorreu às leis (8.35), (8.45) e (8.33), à versão finitária de (8.29), e onde o passo (8.50) se obteve por 'folding' sobre o passo (8.47). Em suma,

$$\begin{array}{lllll} A^\star & \cong & L & & \\ & & onde & L & = & 1 + N \\ & & & N & \cong & A \times L \end{array} \tag{8.51}$$

À luz da correspondência da figura 1.5, L (8.51) vem a ser codificável em, por exemplo,

```
L  =  ^N;
N  =  record
         P: A;
         S: L
      end;
```

na linguagem PASCAL [8]. Se em (8.51) se removerem as variáveis L e N, teremos

$$A^\star \cong 1 + A \times A^\star$$

cf. passos (8.48) e (8.50). O processo acima é, pois, uma versão construtiva da demonstração de que $A^\star$ é ponto fixo da equação

$$L \cong 1 + A \times L \tag{8.52}$$

que se realizou na secção 2.3 — *cf.* equação (2.29) e seguintes.

Exercício 8.7 O seguinte encadeamento de refinamentos,

$$2^A \quad \preceq_{elems} \quad A^\star \quad \cong_g \quad L$$

onde

$$L \cong 1 + A \times L$$

e

$$g(x) \stackrel{\text{def}}{=} \begin{cases} <> & \Leftarrow \quad x = \langle 1, NIL \rangle \\ cons(a, g(l)) & \Leftarrow \quad x = \langle 2, \langle a, l \rangle \rangle \end{cases}$$

permite-nos refinar conjuntos directamente em listas ligadas. Componha as duas funções globais de abstracção,

$$f = elems \circ g$$

e simplifique, usando leis básicas do cálculo funcional (apêndice B), a função f que assim documenta esse refinamento.

□

[8] Mas veja-se oportunamente a secção 8.7.1 sobre este este tipo de codificação.

Factorização de Relações

Este exemplo mostra como transformar *relações binárias* em *funções parciais*, e vice-versa. É um de vários resultados úteis para refinamentos cujo objectivo seja a implementação no *modelo relacional* da informação, hoje tão difundido como suporte de base de dados em sistemas de informação.

O nosso ponto de partida é o domínio de relações $2^{A\times B}$. Teremos, de imediato:

$$2^{A\times B} \quad \cong \quad (2^B)^A \tag{8.53}$$

cf. (8.38). Seja $2^B_+ = 2^B - \{\lambda b.F\}$, onde $\lambda b.F$ designa o predicado *sempre falso* em B, *i.é* o predicado que induz o conjunto vazio $\emptyset$ em B. Portanto,

$$2^B_+ \cong \mathcal{P}(B) - \{\emptyset\}$$

sabendo que

$$2^A \quad \cong \quad \mathcal{P}(A) \tag{8.54}$$

Dos factos (8.43) e (8.18) deriva-se:

$$\begin{array}{rcl} 2^B & = & 2^B_+ \cup \{\lambda b.F\} \\ & \cong & 2^B_+ + 1 \end{array}$$

à custa do que a equação (8.53) — combinada com a lei (8.39) — re-escreve em:

$$\begin{array}{rcl} 2^{A\times B} & \cong & (2^B_+ + 1)^A \\ & \cong & A \rightharpoonup 2^B_+ \end{array} \tag{8.55}$$

Em suma, toda a família A-indexada de subconjuntos de B não-vazios "é" uma relação e como tal pode ser representada em suporte tabular.

Exercício 8.8 Mostre que a função de abstracção:

$$\begin{array}{rcl} collect & : & 2^{A\times B} \longrightarrow A \rightharpoonup 2^B_+ \\ collect(\rho) & \stackrel{\mathrm{def}}{=} & \left(\begin{array}{c} a \\ \{x \in B \mid \; a\rho x\} \end{array}\right)_{a\in\pi_1[\rho]} \end{array} \tag{8.56}$$

estabelece o isomorfismo (8.55) e defina

$$discollect \quad = \quad collect^{-1} \tag{8.57}$$

□

Exercício 8.9 Demonstre ou refute a seguinte igualdade, onde letras maiúsculas designam espécies de dados e minúsculas designam funções,

$$2^{(f \circ \pi_2)} \circ discollect \quad = \quad 2^{\pi_2} \circ discollect \circ (A \rightharpoonup 2^f)$$

□

Exercício 8.10 Demonstre o facto seguinte, em $Sets$:

$$(A \rightharpoonup B)^C \quad \cong \quad (C \times A) \rightharpoonup B \tag{8.58}$$

□

Exercício 8.11 Estará correcta a dedução seguinte, em $Sets$?

$$\begin{array}{rcl}
(A \rightharpoonup B) \rightharpoonup C & \cong & (C+1)^{(A \rightharpoonup B)} \\
& \cong & (C+1)^{((B+1)^A)} \\
& \cong & (C+1)^{(A \times (B+1))} \\
& \cong & (C+1)^{(A \times B + A)} \\
& \cong & (C+1)^{A \times B} \times (C+1)^A \\
& \cong & (A \times B \rightharpoonup C) \times (A \rightharpoonup C)
\end{array}$$

Justifique devidamente a sua resposta.

□

Exercício 8.12 Verifique se o seguinte isomorfismo é válido no cálculo SETS:

$$(A \rightharpoonup B) \times (A \rightharpoonup C) \quad \cong \quad A \rightharpoonup (B + C + B \times C)$$

□

8.4 O $\trianglelefteq$-Subcálculo

A equação (8.2) acima registou um $\trianglelefteq$-resultado básico àcerca da implementação de conjuntos finitos. Dois outros $\trianglelefteq$-resultados elementares têm a ver com a introdução explícita de redundância via construtores primitivos do cálculo, por exemplo

$$A \trianglelefteq_{\pi_1} A \times B \tag{8.59}$$

$$B \trianglelefteq_{\pi_2} A \times B \tag{8.60}$$

que permite emparelhar uma estrutura de dados com qualquer outra, e

$$A \trianglelefteq^{\lambda x.x=\langle 1,a\rangle}_{\lambda\langle i,a\rangle.a} A + B \tag{8.61}$$

$$B \trianglelefteq^{\lambda x.x=\langle 2,b\rangle}_{\lambda\langle i,b\rangle.b} A + B \tag{8.62}$$

que permite injectar uma estrutura num 'record' variante — recordar a analogia da figura 1.5.

A maior parte dos resultados elaborados do $\trianglelefteq$-sub-cálculo têm a ver com o refinamento de funções parciais finitas. Comecemos por relembrar as leis (8.33) a (8.38) sobre a exponenciação, quer dizer, sobre conjuntos de funções totais. Será que tais leis se continuam a verificar para funções parciais, quer dizer, para expressões da forma $A \rightharpoonup B$ em lugar de B^A ?

Pode facilmente verificar-se que, de facto, apenas (8.34) e (8.33) são preservadas, *cf.* respectivamente,

$$\begin{aligned}(B + C) \rightharpoonup A &\cong (A+1)^{B+C} \\ &\cong (A+1)^B \times (A+1)^C \\ &\cong (B \rightharpoonup A) \times (C \rightharpoonup A)\end{aligned} \tag{8.63}$$

e

$$\begin{aligned}0 \rightharpoonup A &\cong (A+1)^0 \\ &\cong 1\end{aligned}$$

No que diz respeito a (8.35) e (8.36), obteem-se resultados tangencialmente diferentes: quanto a (8.36), é fácil mostrar que

$$\begin{aligned}A \rightharpoonup 1 &\cong (1+1)^A \\ &\cong 2^A\end{aligned} \tag{8.64}$$

— aplicar (8.39) e *cf.* $2 \cong 1+1$ (8.32). Quer dizer, obtemos uma forma de modelar conjuntos finitos por funções finitas; quanto a (8.35) tem-se que

$$\begin{aligned}1 \rightharpoonup A &\cong (A+1)^1 \\ &\cong A+1\end{aligned}$$

resultado que permanece no $\cong$-subcálculo, *cf.* (8.39) e a própria (8.35).

Ver-se-á de seguida que as duas leis que faltam — (8.37) e (8.38) — conduzem a $\trianglelefteq$-inequações. Antes disso, porém, faça-se o seguinte exercício.

Exercício 8.13 Mostre que, para $B \ncong 0$ e $C \ncong 0$, se tem

$$A \rightharpoonup (B \rightharpoonup C)_+ \cong A \times B \rightharpoonup C$$

onde o invariante $+$ se define por

$$+(\sigma) \stackrel{\rm def}{=} \sigma \neq \left(\quad\right)$$

(**sugestão**: faça um estudo de cardinalidades).

Que lei do cálculo estudou que é desta um caso particular?

□

A lei distributiva (8.37) "verifica-se" a nível de $\rightharpoonup$ desde que $\trianglelefteq$ substitua $\cong$,

$$A \rightharpoonup B \times C \trianglelefteq (A \rightharpoonup B) \times (A \rightharpoonup C) \tag{8.65}$$

como se pode ver pelo raciocínio seguinte:

$$\begin{aligned}
A \rightharpoonup (B \times C) &\cong \bigcup_{K \subseteq A} (B \times C)^K \\
&\cong \bigcup_{K \subseteq A} (B^K) \times (C^K) \\
&\cong \{\langle\sigma,\tau\rangle \mid \sigma \in B^K \wedge \tau \in C^K \wedge K \subseteq A\} \\
&= \{\langle\sigma,\tau\rangle \mid \sigma \in A \rightharpoonup B \wedge \tau \in A \rightharpoonup C \wedge dom(\sigma) = dom(\tau)\}
\end{aligned}$$

cf. (8.16) e a própria lei (8.37). Assim, existe um $S \subseteq (A \rightharpoonup B) \times (A \rightharpoonup C)$ tal que:

$$A \rightharpoonup (B \times C) \cong S$$

induzindo o invariante $\phi = eqd$ para:

$$eqd(\langle\sigma,\tau\rangle) \stackrel{\rm def}{=} dom(\sigma) = dom(\tau) \tag{8.66}$$

A função de abstracção correspondente é

$$f(\langle\sigma,\tau\rangle) \quad = \quad \sigma \bowtie \tau \tag{8.67}$$

onde $\bowtie$ designa o seguinte operador de "emparelhamento" sobre funções parciais finitas que satisfazem (8.66):

$$\sigma \bowtie \tau \quad \stackrel{\rm def}{=} \quad \left(\begin{array}{c} a \\ \langle\sigma(a),\tau(a)\rangle \end{array}\right)_{a \in dom(\sigma)} \tag{8.68}$$

Em suma, temos que

$$A \rightharpoonup (B \times C) \trianglelefteq^{eqd}_{\bowtie} (A \rightharpoonup B) \times (A \rightharpoonup C) \tag{8.69}$$

que nos indica como $\rightharpoonup$ "distribui" pelo produto $\times$.

Exercício 8.14 Será $\bowtie$ uma função de abstracção injectiva (sobre eqd)? Justifique.
$\square$

Exercício 8.15 Será verdadeiro o seguinte facto,

$$(A \times B)^\star \quad \trianglelefteq_f^\phi \quad A^\star \times B^\star$$

em SETS, para quaisquer A e B?

Na negativa, apresente um contra-exemplo. Na afirmativa, proponha f e ϕ.
$\square$

A $\rightharpoonup$-versão da lei de "currying" da exponenciação (8.38) é outro $\trianglelefteq$-resultado,

$$(A \times B) \rightharpoonup C \trianglelefteq A \rightharpoonup (B \rightharpoonup C) \tag{8.70}$$

e não o pretendido,

$$(A \times B) \rightharpoonup C \cong A \rightharpoonup (B \rightharpoonup C)$$

porque as funções de abstracção $curry$ e $uncurry$ (*cf.* Exercício 8.6) são aqui, respectivamente, não sobrejectiva e parcial — pense-se nos valores de $A \rightharpoonup (B \rightharpoonup C)$ contendo $(\quad)$ no contra-domínio.

O invariante induzido por (8.70) é, assim,

$$\phi(\sigma) \stackrel{\rm def}{=} (\quad) \notin rng(\sigma)$$

para a função de abstracção

$$uncurry(\sigma) \quad \stackrel{\rm def}{=} \quad \left(\begin{array}{c} \langle a, b \rangle \\ (\sigma(a))(b) \end{array} \right)_{a \in dom(\sigma) \wedge b \in dom(\sigma(a))}$$

A "simétrica" da lei (8.70),

$$A \rightharpoonup (B \rightharpoonup C) \trianglelefteq (A \times B) \rightharpoonup C$$

vem a ser, portanto, inválida, embora se verifique um facto algo mais complicado,

$$A \rightharpoonup (B \rightharpoonup C) \trianglelefteq 2^A \times ((A \times B) \rightharpoonup C) \tag{8.71}$$

cujo factor extra (em 2^A) vem a ser o conjunto dos $a \in A$ que têm como imagem a função vazia. Mas é preferível encarar (8.71) como um caso particular de

$$A \rightharpoonup D \times (B \rightharpoonup C) \trianglelefteq (A \rightharpoonup D) \times ((A \times B) \rightharpoonup C) \tag{8.72}$$

— faça-se $D = 1$ e apliquem-se propriedades elementares de $\times$ e (8.64). O invariante induzido por (8.72) é $\phi = dpi$ [9]

$$dpi(\sigma, \tau) \stackrel{\mathrm{def}}{=} \pi_1[dom(\tau)] \subseteq dom(\sigma) \tag{8.73}$$

onde $\pi_1[dom(\tau)]$ designa, como habitualmente, $\{\pi_1(p) \mid p \in dom(\tau)\}$, para o que se relembram os selectores π_1, π_2 associados ao produto cartesiano:

$$\begin{aligned} \pi_1(a, b) &= a \\ \pi_2(a, b) &= b \end{aligned}$$

Recomenda este invariante que todos os $a \in A$, que aparecem no domínio de $\sigma \in A \rightharpoonup D$ mas não aparecem no domínio de $\tau \in A \times B \rightharpoonup C$, sejam exactamente as entradas, a nível abstracto, a aplicar na função vazia (em $B \rightharpoonup C$). A função de abstracção correspondente será, então, uma espécie de 'nested join':

$$\bowtie_n (\sigma, \tau) \stackrel{\mathrm{def}}{=} \left(\langle \sigma(a), \left(\begin{array}{c} \pi_2(p) \\ \tau(p) \end{array} \right)^{a}_{p \in dom(\tau) \wedge \pi_1(p) = a} \rangle \right)_{a \in dom(\sigma)} \tag{8.74}$$

Refira-se, para terminar, que o $\trianglelefteq$-subcálculo inclui $\rightharpoonup$-leis que não podem ser adaptadas de resultados semelhantes sobre a exponenciação. Por exemplo,

$$(B + C)^A \ncong B^A \times C^A$$

e, no entanto, é válida a seguinte lei para decomposição de funções finitas:

$$A \rightharpoonup (B + C) \trianglelefteq^{djd}_{\overset{+}{\bowtie}} (A \rightharpoonup B) \times (A \rightharpoonup C) \tag{8.75}$$

para

$$\begin{aligned} djd(\langle \sigma, \tau \rangle) &\stackrel{\mathrm{def}}{=} dom(\sigma) \cap dom(\tau) = \emptyset \\ \overset{+}{\bowtie}(\langle \sigma, \tau \rangle) &\stackrel{\mathrm{def}}{=} i_1 \circ \sigma \cup i_2 \circ \tau \\ &= (A \rightharpoonup i_1)(\sigma) \cup (A \rightharpoonup i_2)(\tau) \end{aligned}$$

onde djd (='disjoint domains') impõe domínios disjuntos e i_1, i_2 são as "injecções" implícitas no co-produto, recordar (1.28).

Exercício 8.16 Repare que o domínio $A \rightharpoonup 2^B_+$ da lei (8.55) é menos geral que $A \rightharpoonup 2^B$, ao não permitir conjuntos vazios nos contradomínios das funções finitas; essa situação é contemplada por

$$2^{A \times B} \preceq_{discollect} A \rightharpoonup 2^B \tag{8.76}$$

[9] dpi='domain projection inclusion'.

onde, contudo, $discollect$ deixa de ser injectiva. Identifique a classe de funções finitas em $A \rightharpoonup 2^B$ que $discollect$ abstrai na relação vazia. Mostre ainda que $collect$, estendida a $A \rightharpoonup 2^B$, deixa de ser sobrejectiva.
□

As leis (8.65,8.71,8.72) e (8.75) desempenham um papel importante na reificação de funções finitas. Podem ser usadas para "decompor" funções complexas ou aninhadas em tuplos de funções mais simples. Se as combinarmos com (8.76) e (8.55), estamos em condições de afirmar que o fragmento que até aqui se estudou do $\trianglelefteq$-subcálculo é adequado ao refinamento de modelos de dados elaborados, baseados em funções finitas, para esquemas de bases de dados relacionais.

Mas, antes disso, precisamos de estudar um resultado essencial a todo o processo de cálculo.

8.5 Cálculo Estrutural

O nosso exemplo motivador será, agora, o seguinte: temos $A \rightharpoonup 2^B$ (famílias de conjuntos) e sabemos que $2^B \trianglelefteq B^\star$. Em que medida podemos deduzir que

$$A \rightharpoonup 2^B \quad \trianglelefteq \quad A \rightharpoonup B^\star \quad ?$$

Só se $\lambda X.A \rightharpoonup X$ for "monótono" em relação a $\trianglelefteq$, *i.é*, se $X \trianglelefteq Y \Rightarrow (A \rightharpoonup X) \trianglelefteq (A \rightharpoonup Y)$ se verificar. De facto, para raciocínios estruturais como este precisamos do teorema que se segue.

Teorema 8.1 ($\trianglelefteq$-monotonicidade dos construtores de $Sets$) *Os construtores de $Sets$ $\times$ e $+$ são crescentes (monótonos) com respeito a $\trianglelefteq$ (8.10); a exponenciação requer expoentes isomorfos.*

Quer dizer, dados os objectos de $Sets$ A, B, X, Y, M, N tais que $A \trianglelefteq X$, $B \trianglelefteq Y$ e $M \cong N$, então verificam-se os factos

$$A \times B \quad \trianglelefteq \quad X \times Y \tag{8.77}$$
$$A + B \quad \trianglelefteq \quad X + Y \tag{8.78}$$
$$A^M \quad \trianglelefteq \quad X^N \tag{8.79}$$

Esquema de Prova: *Seja*

$$A \trianglelefteq_f^\phi X, \;\; B \trianglelefteq_g^\varphi Y, \;\; M \cong_h N \tag{8.80}$$

Então

1. *Equação (8.77):*
$$A \times B \trianglelefteq_{f\times g}^{\phi\times\varphi} X \times Y$$
onde o produto de duas funções f e g é a sua aplicação "em paralelo"[10]*:*
$$(f \times g)\langle a, b\rangle = \langle f(a), g(b)\rangle \quad (8.81)$$
e o produto de dois predicados ϕ e φ é a sua conjunção "em paralelo":
$$(\phi \times \varphi)\langle a, b\rangle = \phi(a) \wedge \varphi(b)$$

2. *Equação (8.78):*
$$A + B \trianglelefteq_{f+g}^{\phi+\varphi} X + Y$$
onde a soma de duas funções f e g é
$$\begin{cases} (f+g)\langle 1, a\rangle &=& \langle 1, f(a)\rangle \\ (f+g)\langle 2, b\rangle &=& \langle 2, g(b)\rangle \end{cases} \quad (8.82)$$
— recordar (1.29) — e a soma de dois predicados ϕ e φ é
$$\begin{cases} (\phi+\varphi)\langle 1, a\rangle &=& \phi(a) \\ (\phi+\varphi)\langle 2, b\rangle &=& \varphi(b) \end{cases}$$

3. *Equação (8.79): Como M e N têm a mesma cardinalidade, seja ela m, então*
$$\begin{array}{rcl} A^M & \cong & A^m \\ & \cong & \underbrace{A \times \ldots \times A}_{m} \end{array}$$
e
$$\begin{array}{rcl} X^N & \cong & X^m \\ & \cong & \underbrace{X \times \ldots \times X}_{m} \end{array}$$
cf. *(8.31). Quer dizer, (8.79) pode reduzir-se a uma versão finitária de (8.77) em que*
$$A^m \trianglelefteq_{f^m}^{\phi^m} X^m$$
onde
$$\begin{array}{l} f^m\langle a_1, \ldots, a_m\rangle = \langle f(a_1), \ldots, f(a_m)\rangle \\ \phi^m\langle a_1, \ldots, a_m\rangle = \forall 1 \leq i \leq m : \phi(a_i) \end{array} \quad (8.83)$$

[10] Recordar (1.19). Para melhor leitura, a aplicação funcional envolvendo argumentos que são tuplos, *e.g.* $f(\langle x_1, \ldots, x_n\rangle)$, abreviar-se-á para $f\langle x_1, \ldots, x_n\rangle$ ou $f(x_1, \ldots, x_n)$.

Alternativamente, podemos definir a função de abstracção global t *e o correspondente invariante* τ *em*

$$A^M \trianglelefteq_t^\tau X^N$$

como se segue, para $\sigma \in X^N$ *e sendo* h *o isomorfismo entre* M *e* N *(8.80):*

$$\begin{array}{rcl} t(\sigma) & = & f \circ \sigma \circ h^{-1} \\ \tau(\sigma) & = & \forall n \in dom(\sigma) : \phi(\sigma(n)) \end{array}$$

Fica por provar que as três funções de abstracção compostas são sobrejectivas e totais sobre os respectivos invariantes, o que não é muito difícil. □

Exercício 8.17 Faça as demonstrações que ficaram por fazer na prova do teorema anterior.
□

A relevância do Teorema 8.1 é permitir-nos refinar *isoladamente* os componentes estruturais de uma expressão que designe, em $Sets$, um domínio de dados arbitrariamente complexo. Torna assim possível o cálculo progressivo das estruturas de dados de um dado programa a partir da sua especificação, por introdução gradual de redundância, sintetizando automaticamente funções de abstracção e os invariantes induzidos pelas sucessivas decisões de refinamento. Veremos de imediato um exemplo de aplicação deste teorema.

8.5.1 Exemplo de Aplicação

Estamos já em condições de calcular a implementação $BAMSR$ referida na secção 7.5:

$$\begin{array}{rcl} BAMS & = & AccNr \rightharpoonup (2_+^{AccHolder} \times Amount) \\ & \trianglelefteq_1 & (AccNr \rightharpoonup 2_+^{AccHolder}) \times (AccNr \rightharpoonup Amount) \\ & \cong_2 & (2^{AccNr \times AccHolder}) \times (AccNr \rightharpoonup Amount) \\ & \trianglelefteq_3 & (2^{AccNr \times AccHolder}) \times (2^{AccNr \times Amount}) \\ & = & BAMSR \end{array}$$

onde

$$\begin{array}{lll} \trianglelefteq_1 & \left\{ \begin{array}{l} f_1 = \bowtie \\ \phi_1 = eqd \end{array} \right. & \textit{cf.}\ (8.69) \\ \trianglelefteq_2 & \left\{ \begin{array}{l} f_2 = collect \times id \\ \phi_2 = \lambda\rho.V \times \lambda\gamma.V \end{array} \right. & \textit{cf.}\ \text{Teorema 8.1 e (8.56)} \\ \trianglelefteq_3 & \left\{ \begin{array}{l} f_3 = id \times mkf \\ \phi_3 = \lambda\rho.V \times fdp \end{array} \right. & \textit{cf.}\ \text{Teorema 8.1 e (8.9)} \end{array}$$

Assim,

$$
\begin{aligned}
f &= f_1 \circ f_2 \circ f_3 \\
&= \bowtie \circ (collect \times id) \circ (id \times mkf) \\
&= \bowtie \circ (collect \times mkf) \\
&= \lambda\langle \rho, \gamma \rangle.\ \ let \quad x = collect(\rho) \\
&\qquad\qquad\quad in \quad \left(\begin{array}{c} a \\ \langle x(a), mkf(\gamma)(a) \rangle \end{array} \right)_{a \in \pi_1[\rho]}
\end{aligned} \tag{8.84}
$$

Quanto ao invariante, teremos:

$$
\begin{aligned}
\phi(\langle \rho, \gamma \rangle) &= \phi_1(f_2(f_3(\rho, \gamma))) \wedge \\
&\quad\ \phi_2(f_3(\rho, \gamma)) \wedge \\
&\quad\ \phi_3(\rho, \gamma) \\
&= \phi_1(f_2(\rho, mkf(\gamma))) \wedge \\
&\quad\ T \wedge \\
&\quad\ fdp(\gamma) \\
&= \phi_1(collect(\rho), mkf(\gamma)) \wedge \\
&\quad\ fdp(\gamma) \\
&= (dom(collect(\rho)) = dom(mkf(\gamma))) \wedge \\
&\quad\ fdp(\gamma) \\
&= \pi_1[\rho] = \pi_1[\gamma] \wedge fdp(\gamma)
\end{aligned} \tag{8.85}
$$

cuja análise condiz com a nossa intuição:

- qualquer número de conta de que se conhecem titulares tem sempre uma quantia associada ($\pi_1[\rho] \subseteq \pi_1[\gamma]$) e vice-versa ($\pi_1[\rho] \supseteq \pi_1[\gamma]$);
- há uma dependência funcional de números de conta para quantias ($fdp(\gamma)$).

Exercício 8.18 Suponha que, no exemplo de cálculo acima, $BAMS$ não exige contas com pelo menos um titular, *i.é* temos $2^{AccHolder}$ em lugar de $2_{+}^{AccHolder}$.

Que alteração se verifica no invariante global ϕ a impôr ao nível relacional? (**Sugestão**: faça $C = 1$ em (8.72) e aplique aqui o resultado.)

□

Exercício 8.19 Suponha que, a partir da especificação que se fez de um sistema para planeamento da produção (relembrar o Exercício 2.59):

$$
\begin{array}{rcl}
DPP & \cong & Stock \times Pricelist \times Equip \\
Stock & \cong & \#Unid \rightharpoonup I\!N \\
Pricelist & \cong & \#Comp \rightharpoonup I\!N \\
Equip & \cong & \#Equip \rightharpoonup (\#Unid \rightharpoonup I\!N)_{+} \\
\#Unid & \cong & \#Comp + \#Equip
\end{array}
$$

se calculou a seguinte implementação relacional:

$$
\begin{array}{rclll}
DPP1 & \cong & CC: & 2^{Row1} \quad \times & \textit{/*tabela de custo/`stock' dos componentes */} \\
 & & SE: & 2^{Row2} \quad \times & \textit{/*tabela de `stock' de equipamentos */} \\
 & & EC: & 2^{Row3} \quad \times & \textit{/*tabela de decomposição de */} \\
 & & & & \textit{/*equipamentos em seus componentes */} \\
 & & EE: & 2^{Row4} & \textit{/*tabela de decomposição de */} \\
 & & & & \textit{/*equipamentos em sub-equipamentos */} \\
Row1 & \cong & Co: & \#Comp \quad \times & \\
 & & QC: & I\!N \quad \times & \textit{/*quantidade ou custo */} \\
 & & T: & 2 \quad \times & \textit{/*etiqueta */} \\
Row2 & \cong & Eq: & \#Equip \quad \times & \\
 & & St: & I\!N & \\
Row3 & \cong & Eq: & \#Equip \quad \times & \\
 & & Co: & \#Comp \quad \times & \\
 & & Nr: & I\!N & \\
Row4 & \cong & Eq: & \#Equip \quad \times & \\
 & & Se: & \#Equip \quad \times & \\
 & & Nr: & I\!N &
\end{array}
$$

Sintetize a função de abstracção correspondente e o invariante induzido ao nível relacional.
□

Exercício 8.20 Uma maneira habitual de representar sequências de $A^\star$ em sistemas relacionais é sob a forma de tabelas

$$2^{I\!N \times A}$$

em que cada linha da tabela representa um elemento de uma sequência e a sua posição na sequência. Verifique então que

$$A^\star \quad \trianglelefteq \quad I\!N \rightharpoonup A \quad \trianglelefteq \quad 2^{I\!N \times A}$$

e sintetize o invariante induzido ao nível relacional.
□

Exercício 8.21 Suponha que se perdeu toda a informação de um $\trianglelefteq$-raciocínio, exceptuando a função de abstracção global f, que é a seguinte:

$$f \quad = \quad (A \rightharpoonup \langle dom \circ \pi_2, \pi_1 \rangle) \circ \bowtie \circ (mkf \times g)$$

onde

$$g(r) \quad = \quad \left(\begin{array}{c} p \\ NIL \end{array} \right)_{p \in r}$$

Será possível reconstituir o processo de cálculo que se perdeu? Na afirmativa, faça-o, indicando a estrutura original, a implementação que se obteve, e as leis do cálculo SETS que foram utilizadas. Na negativa, explique a sua resposta.
□

8.6 Aproximação Functorial

Recorreremos agora à noção categorial de *functor* (em $Sets$) com o objectivo de sistematizar e estender o processo de cálculo.

8.6.1 Noção de Functor

Definição 8.3 *Sejam C,D duas categorias. Diz-se que $\mathcal{F}$ é um* functor *de C para D, escrevendo-se*

$$\mathcal{F} : C \longrightarrow D$$

se $\mathcal{F}$ é um par de aplicações (ambas designadas por $\mathcal{F}$), uma que aplica objectos de C em objectos de D e outra uma que aplica morfismos de C em morfismos de D, tal que

$$\mathcal{F}(X) \stackrel{\mathcal{F}(f)}{\longrightarrow} \mathcal{F}(Y)$$

é um morfismo em D sempre que $X \stackrel{f}{\longrightarrow} Y$ é morfismo de C, e se tem:

$$\begin{array}{rcl} \mathcal{F}(1_C) & = & 1_{\mathcal{F}(C)} \\ \mathcal{F}(f \circ g) & = & \mathcal{F}(f) \circ \mathcal{F}(g) \end{array}$$

□

8.6.2 Functores Polinomiais

Sejam A, B, C objectos em $Sets$. Seja $f : A \to B$ um morfismo em $Sets$ e seja $1_C : C \to C$ a designação do morfismo identidade sobre C. Então C pode ser encarado como o *functor constante*

$$C : Sets \longrightarrow Sets \tag{8.86}$$

tal que $C(A) = C$ e $C(f) = 1_C$.

Sejam $\mathcal{F}, \mathcal{G} : Sets \longrightarrow Sets$ functores. Define-se o seu *produto*

$$\mathcal{F} \times \mathcal{G} : Sets \longrightarrow Sets$$

por $(\mathcal{F} \times \mathcal{G})(A) = \mathcal{F}(A) \times \mathcal{G}(A)$ e por $(\mathcal{F} \times \mathcal{G})(f) = \mathcal{F}(f) \times \mathcal{G}(f)$, *cf.* (8.81). Por exemplo, $\mathcal{F}(X) = X \times C$ define o functor produto do functor identidade $\lambda X.X$ com o functor constante C (8.86), e tem-se $\mathcal{F}(f)\langle a, c\rangle = (f \times 1_C)\langle a, c\rangle = \langle f(a), c\rangle$.

Podemos ter produtos n-ários de functores, dos quais $\mathcal{F}(X) = X^n$ é um caso particular. Sendo f o morfismo acima, teremos neste caso

$$\mathcal{F}(f) = f^n : A^n \to B^n$$

cf. (8.83).

A noção de functor produto tem uma dual — a de *functor co-produto* (ou $soma$ de dois functores) — que se baseia em (8.82):

$$\begin{array}{rcl} \mathcal{F}+\mathcal{G} & : & Sets \longrightarrow Sets \\ (\mathcal{F}+\mathcal{G})(A) & = & \mathcal{F}(A)+\mathcal{G}(A) \\ (\mathcal{F}+\mathcal{G})(f) & = & \mathcal{F}(f)+\mathcal{G}(f) \end{array}$$

Functores co-produto podem também ser iterados a n-argumentos. Finalmente, temos a definição:

Definição 8.4 *Diz-se* polinomial *todo o functor* $\mathcal{F} : Sets \longrightarrow Sets$ *que é*

- *um functor constante, ou*
- *o functor identidade, ou*
- *a composição ou produto finitário de functores polinomiais.*

(Relembre (2.116.)) □

Por exemplo,

$$\mathcal{F}(X) = 1 + A \times X \tag{8.87}$$

—*cf.* (8.52) — é um functor polinomial, assim como $\mathcal{F}(X) = X^\star$, *cf.* (8.40). Já o functor $\mathcal{F}(X) = C \rightharpoonup X$ será polinomial apenas se C for finito [11], tendo-se

$$\begin{array}{rcl} (C \rightharpoonup f)(\sigma) & = & f \circ \sigma \\ & = & \left(\begin{array}{c} c \\ f(\sigma(c)) \end{array} \right)_{c \in dom(\sigma)} \end{array} \tag{8.88}$$

O resultado que se segue é válido em $Sets$:

Teorema 8.2 *Todo o functor polinomial*

$$\mathcal{F} : Sets \longrightarrow Sets$$

pode ser reduzido à forma canónica:

$$\begin{array}{rcl} \mathcal{F}(X) & \cong & C_0 + (C_1 \times X) + (C_2 \times X^2) + \cdots + (C_n \times X^n) \\ & = & \sum_{i=0}^{n} C_i \times X^i \end{array} \tag{8.89}$$

Demonstração: Ver [MA86]. □

---

[11] Cf.

$$\begin{array}{rcll} C \rightharpoonup X & \cong & (X+1)^C & \\ & \cong & (X+1)^n & \textit{para } n = card(C) \end{array}$$

Por exemplo, para $\mathcal{F}(X) = X^\star$ tem-se $C_i = 1$ para todo o i, pois

$$\begin{aligned} X^\star &\cong \sum_{i\geq 0} X^i \\ &\cong \sum_{i\geq 0} 1 \times X^i \end{aligned}$$

— relembrar (8.40) e o facto de 1 ser o elemento neutro de $\times$ [12].

Um resultado sugestivo para converter functores polinomiais na forma canónica é a fórmula do próprio *binómio de Newton*,

$$(A+B)^n \cong \sum_{p=0}^{n} {}^nC_p \times A^{n-p} \times B^p \tag{8.90}$$

como se ilustra no seguinte tratamento do functor 'pedigree':

$$\mathcal{F}(X) \;=\; I \times (X+1)^2$$

já atrás referido (2.156,2.118). Ter-se-á,

$$\begin{aligned} \mathcal{F}(X) &= I \times (X+1)^2 \\ &\cong I \times (X^2 + 2 \times X + 1) \\ &\cong I \times X^2 + 2 \times I \times X + I \end{aligned}$$

que, literalmente, significa: *"sobre um indivíduo I ou ambos o pai e a mãe são conhecidos (X^2), ou um de ou o pai ou a mãe é conhecido ($2 \times X$), ou tanto o pai como a mãe são desconhecidos (1)"*.

Combinando (8.89,8.90) com a lei (8.75) para decomposição de funções, obtemos uma regra geral para decompor uma função da forma

$$A \rightharpoonup \mathcal{F}(X)$$

num produto cartesiano de funções, *i.é*

$$\Pi_{i=0}^{n}(A \rightharpoonup C_i \times X^i) \tag{8.91}$$

cada uma das quais pode, por sua vez, ser refinada para a forma tabular ($2^{A\times B}$), regra geral essa que será de muita utilidade em projecto de bases de dados relacionais.

[12] Estamos a encarar somas numeráveis de functores polinomiais como sendo functores polinomiais.

8.6.3 Cálculo Functorial

É agora a altura de, com a ajuda da noção de functor polinomial, estendermos o Teorema 8.1 como se segue:

Teorema 8.3 *Seja $\mathcal{F}$ um (endo)functor polinomial em $Sets$. Sempre que*

$$A \trianglelefteq_f^\phi B \tag{8.92}$$

então

$$\mathcal{F}(A) \trianglelefteq_{\mathcal{F}(f)}^{\mathcal{F}(\phi)} \mathcal{F}(B) \tag{8.93}$$

Demonstração: Pode fazer-se por indução sobre a estructura do functor polinomial $\mathcal{F}$. A soma e o produto (e a exponenciação) já se trataram no Teorema 8.1.

Um dos casos base consiste no functor identidade $\mathcal{F}(X) = X$, que é trivial, pois $\mathcal{F}(f) = f$ e $\mathcal{F}(\phi) = \phi$, *i.é* (8.93) reduz-se a (8.92). O outro caso de base é o do functor constante $\mathcal{F}(X) = C$. Tem-se $\mathcal{F}(f) = 1_C$ e $\mathcal{F}(\phi) = \lambda b.V$, verificando-se (8.93) trivialmente. □

Este teorema fornece-nos uma estratégia elegante, functorial, para cálculo de invariantes de abstracção complexos ao longo de processos de reificação. Embora $\mathcal{F}(X) = 2^X$ não seja polinomial, (8.93) verifica-se para este functor também, para A finito.

A Figura 8.2 apresenta um sumário deste "cálculo functorial" para as construções mais habituais (primitivas e derivadas) em $Sets$. Reconhecem-se, aqui devidamente tratadas, as "construções" (1.19), (1.29), (1.83) *etc.* apresentadas no Capítulo 1.

8.6.4 Exemplo

Estamos agora em condições de tratar o exemplo que motivou a secção 8.5. Partindo de

$$2^B \trianglelefteq_{elems}^{\lambda l.T} B^\star$$

inferimos directamente

$$A \rightharpoonup 2^B \quad \trianglelefteq_f^\phi \quad A \rightharpoonup B^\star$$

deduzindo

$$\begin{aligned} f &= A \rightharpoonup elems \\ &= \lambda\sigma.(elems \circ \sigma) \\ &= \lambda\sigma.\left(\begin{array}{c} a \\ elems(\sigma(a)) \end{array}\right)_{a \in dom(\sigma)} \end{aligned}$$

Figura 8.1: Summary of Functorial Calculus.

$\mathsf{F}(X)$	$\mathsf{F}(f)$	$\mathsf{F}(\phi)$
X	f	ϕ
C	1_C	$\lambda c.V$
2^X	$2^f = \lambda s.\{f(x) \mid x \in s\}$	$2^\phi = \lambda s.\forall b \in s : \phi(b)$
$X^\star$	$f^\star = \lambda l.{<}f(x) \mid x \leftarrow l{>}$	$\phi^\star = \lambda l.\forall 1 \leq i \leq length(l) : \phi(l(i))$
X^C	$\begin{array}{rcl} f^C & = & \lambda\sigma.f \circ \sigma \\ & = & \lambda\sigma.\left(\begin{array}{c} c \\ f(\sigma(c)) \end{array}\right)_{c \in C} \end{array}$	$\phi^C = \lambda\sigma.\forall c \in C : \phi(\sigma(c))$
$C \rightharpoonup X$	$\begin{array}{rcl} C \rightharpoonup f & = & \lambda\sigma.f \circ \sigma \\ & = & \lambda\sigma.\left(\begin{array}{c} c \\ f(\sigma(c)) \end{array}\right)_{c \in dom(\sigma)} \end{array}$	$C \rightharpoonup \phi = \lambda\sigma.\forall x \in rng(\sigma) : \phi(x)$
$\mathsf{G}(X) \times \mathsf{H}(X)$	$\mathsf{G}(f) \times \mathsf{H}(f)$	$\lambda(x,y).(\mathsf{G}(\phi))x \wedge (\mathsf{H}(\phi))y$
$\mathsf{G}(X) + \mathsf{H}(X)$	$\mathsf{G}(f) + \mathsf{H}(f)$	$\lambda x.\begin{cases} x = i_1(a) & \Rightarrow \quad (\mathsf{G}(\phi))a \\ x = i_2(b) & \Rightarrow \quad (\mathsf{H}(\phi))b \end{cases}$
$\mathsf{G}(X) + 1$	$\lambda x.\begin{cases} x = i_2(NIL) & \Rightarrow \quad x \\ x = i_1(a) & \Rightarrow \quad i_1\langle(\mathsf{G}(f))a\rangle \end{cases}$	$\lambda x.\begin{cases} x = i_2(NIL) & \Rightarrow \quad V \\ x = i_1(a) & \Rightarrow \quad (\mathsf{G}(\phi))a \end{cases}$

Figura 8.2: Sumário do Cálculo Functorial

e

$$\begin{array}{rcl} \phi & = & A \rightharpoonup \lambda l.T \\ & = & \lambda\sigma.\forall x \in rng(\sigma) : T \\ & = & \lambda\sigma.T \end{array}$$

Exercício 8.22 Considere o seguinte modelo para filas com prioridade,

$$\Sigma \cong A \rightharpoonup B^\star$$

sujeito ao invariante

$$\phi(\sigma) \stackrel{\text{def}}{=} {<>} \notin rng(\sigma)$$

onde B é o domínio de objectos a enfileirar e A é um domínio de prioridades sobre o qual se supõe definida uma ordem total.

Suponha que alguém chegou à seguinte implementação relacional de Σ:

$$\Sigma' \cong 2^{A \times I\!N \times B}$$

sujeito ao invariante

$$\phi'(t) \stackrel{\text{def}}{=} \forall r, r' \in t : \pi_1(r) = \pi_1(r') \Rightarrow \pi_2(r) \neq \pi_2(r')$$

Mostre (aplicando o cálculo functorial) que este invariante é, desnecessariamente, forte demais.

□

Exercício 8.23 Uma fábrica de cabos condutores eléctricos pretende uma aplicação para controlo dos seus 'stocks'. Há vários tipos de cabos condutores, cada um identificado por um *código de artigo*. Cada segmento de cabo constitui uma *ponta*, é de um determinado tipo, tem um dado comprimento (múltiplo de $1m$) e está armazenado algures numa *bobine*. Cada *bobine* tem uma *matrícula* que a identifica e está num determinado *estado*, a saber: armazenada num dado *armazém*, em manutenção, em produção, em controlo de qualidade, no cliente, abatida ou morta. Na mesma bobine só podem estar armazenadas pontas do mesmo tipo. Todas as bobines do mesmo tipo estão armazenadas no mesmo armazém.

1. Complete a especificação seguinte da estrutura de base para gestão deste 'stock',

$$\begin{array}{rcl} DBase & \cong & \#Art \rightharpoonup \#Arm \times Bobines \\ Bobines & \cong & \#Bob \rightharpoonup Ponta^{\star} \times Estado \\ Ponta & \cong & \ldots \\ Estado & \cong & \ldots \end{array}$$

   onde *#Art (código de artigo)*, *#Arm (referência de armazém)* e *#Bob (matrícula de bobine)* são tipos primitivos.

2. Calcule (usando $Sets$) uma implementação relacional de $DBase$.
3. Escreva um invariante sobre $DBase$ que garanta que a mesma matrícula de bobine não é usada para artigos diferentes.
4. Projecte esse invariante segundo o refinamento relacional que calculou.

□

Exercício 8.24 A informação constante dos *registos prediais* duma Conservatória de Registo Predial (CRP) reparte-se por várias séries de livros de acordo com os objectos registados e as relações jurídicas entre eles. Só nos vamos aqui preocupar com as séries "B" e "C". Na primeira, cada entrada descreve um prédio, *e.g.* um edifício, um terreno, *etc.* No chamado *regime de propriedade horizontal* a informação de cada prédio consta sobretudo da descrição das suas fracções, hipotecas e prédio origem. A situação típica é a seguinte: um empreiteiro hipoteca um terreno (prédio origem) para poder suportar financeiramente a construção de um prédio de n-andares (fracções). Os andares acabam por ser comprados por particulares que normalmente recorrem a empréstimos que lhes são facultados mediante novas hipotecas, agora sobre as fracções em causa. Todas as hipotecas são registadas em livros da série "C".

1. Complete a especificação seguinte da estrutura de base para gestão de uma CRP, onde *#Pre (código de prédio)*, *Text (texto sumário)*, *#Fra (identificador de fracção)* e *#Hip (referência a uma hipoteca)* são tipos primitivos.

$$\begin{array}{rcl} CRP & \cong & B \times C \\ B & \cong & \#Pre \rightharpoonup Desc \times 2^{\#Hip} \times \textit{Fracções} \times Origem \times Valor \\ C & \cong & \#Hip \rightharpoonup Credor \times Valor \\ \textit{Fracções} & \cong & \#Fra \rightharpoonup Desc \times 2^{\#Hip} \times \% \times Valor \times Rend \\ Desc & \cong & Text \\ Origem & \cong & \ldots \\ Credor & \cong & Text \quad \textit{/*de uma hipoteca */} \\ Valor & \cong & \ldots \\ \% & \cong & 99 \quad \textit{/*percentagem da fracção */} \\ Rend & \cong & \ldots \quad \textit{/*rendimento colectável */} \end{array}$$

2. Calcule (usando $Sets$) uma implementação relacional de CRP.

3. Escreva um invariante sobre CRP que garanta que uma fracção nunca é 0% do valor total do prédio e que todas as fracções do mesmo prédio prefazem 100%.
4. Escreva um invariante sobre CRP que garanta que *nunhum prédio vem a ser origem de si próprio*.
5. Projecte esses invariantes segundo o refinamento relacional que calculou.

□

8.7 Refinamento de Modelos Recursivos

8.7.1 Motivação

Equações como (8.20) e (2.156) atrás são exemplos de modelos de dados definidos recursivamente em $Sets$. Muitos outros podiam ter sido apresentados como exemplo, já que a recursividade é uma técnica "elegante" para especificar muitos construtores de dados não primitivos em $Sets$, por exemplo:

$$X \cong 1 + A \times X \quad \text{/*\textit{listas finitas sobre } } A, \text{ cf. \textit{(8.52)} */} \tag{8.94}$$

$$X \cong 1 + A \times X^2 \quad \text{/*\textit{árvores binárias sobre } } A \text{ */} \tag{8.95}$$

$$X \cong 1 + A \times X^\star \quad \text{/*\textit{árvores generalizadas sobre } } A \text{ */}$$

$$X \cong V + A \times X^\star \quad \text{/*\textit{"frases" sobre } } V \text{\textit{ e }} A \text{ */}$$

etc.

Que fazer quando, num processo de refinamento, encontramos um modelo recursivo?

Algumas linguagens suportam directamente a recursividade (*e.g.* LISP, ML) mas muitas outras não (*e.g.* 'assemblers' e o primitivo FORTRAN). Linguagens como o PASCAL ou o C ficam "a meio caminho" entre estas situações extremas — a recursividade é implementável de forma indirecta, com recurso a *apontadores* que endereçam segmentos de *memória dinâmica* ('heaps'). A programação com apontadores é uma actividade sujeita a erros (*cf.* problemas como a indefinição de apontadores, a não-terminação causada por referências cíclicas *etc.*) e tende a produzir código "manhoso", *i.é* difícil de entender por quem não o escreveu [13]. Uma maneira de ultrapassar estes problemas é pensar em regras genéricas que refinem estruturas de dados recursivas em equivalentes não-recursivas que sejam seguras, eliminando toda a manipulação não-sistemática de apontadores [14].

[13] Como bem observou Wirth [Wir76], os apontadores são equivalentes, no eixo das estruturas de dados, aos 'gotos' algorítmicos que a programação estruturada tanto se preocupou em abolir.

[14] Da mesma forma que se pode fazer "Structured Programming with Goto Statements" *cf.* [Knu74].

Olhando para a analogia entre as construções primitivas de $Sets$ e algumas estruturas de dados do PASCAL, apresentada na Figura 1.5, é tentador arriscarmos uma "codificação" de definições como, por exemplo, (8.94) acima. Obtemos, introduzindo uma variável auxiliar `Node` para facilitar a codificação,

```
X     =  ^Node;
Node  =  record
            P: A;
            S: X
         end;
```
(8.96)

De facto, seria concerteza esta a estrutura de dados que um programador de PASCAL escreveria, em primeira mão, para poder representar sequências do tipo `A`. Contudo, qual o significado real da construção `^Node` ? Qual é o "valor" de um apontador? Onde está a prevenção contra apontadores indefinidos e ciclos de apontadores? Estas são limitações da "analogia" invocada que, como todas as analogias, carece de rigor formal.

Um dos primeiros trabalhos que abordaram formalmente este problema foi o de Fielding [Fie80]. Entre várias estruturas estudadas, é considerado o refinamento de *árvores binárias de procura* [15],

$$X \quad \cong \quad 1 + Key \times Data \times X^2 \tag{8.97}$$

que corresponde à definição (8.95) em $Sets$ fazendo $A \cong Key \times Data$.

Um dos refinamentos propostos para esta estrutura em [Fie80] 'maps a binary tree onto linear storage':

$$\begin{array}{rcl} S2 & \cong & (K+1) \times ARRAY \\ ARRAY & \cong & K \rightharpoonup Key \times Data \times (K+1)^2 \end{array} \tag{8.98}$$

onde $K \cong I\!N$ é um conjunto infinito numerável de apontadores. Este refinamento é proposto em [Fie80] sob um pesado invariante induzido, que impede apontadores indefinidos e ciclos de apontadores. Ora um invariante semelhante é também requerido para justificarmos a representação em "lista-ligada" de sequências finitas, que se apresentou acima em (8.96) e que, por analogia com (8.98), escreveríamos formalmente como se segue:

$$X_1 \quad \cong \quad (K+1) \times (K \rightharpoonup A \times (K+1))$$

Poderão estes dois saltos de refinamento que "removem recursividade" ser generalizados?

[15] Estamos a transliterar para $Sets$ a notação VDM usada em [Fie80].

8.7.2 Leis de Des-recursivação

No caso geral, suponhamos que nos é dada uma definição recursiva em $Sets$ da forma

$$X \cong 1 + \mathcal{G}(X), \tag{8.99}$$

onde $\mathcal{G}$ é um (endo)functor em $Sets$ (relembrar a Definição 8.3).

Podemos conjecturar que, se X_1 é um ponto fixo em $Sets$ de (8.99), então

$$X_1 \trianglelefteq_f^\phi (K+1) \times (K \rightharpoonup \mathcal{G}(K+1)) \tag{8.100}$$

para $K \cong I\!N$. Para justificarmos esta conjectura será preciso encontrar o par f, ϕ aí referido, para qualquer $\mathcal{G}$.

O primeiro resultado que desenvolveremos será aplicável a definições mais gerais que (8.99), da forma

$$X \cong \mathcal{F}(X) \tag{8.101}$$

da qual (8.99) se obtém fazendo $\mathcal{F}(X) = 1 + \mathcal{G}(X)$.

Seja $\langle X_0, \delta : \mathcal{F}(X_0) \to X_0 \rangle$ um ponto fixo do functor $\mathcal{F}$ (8.101) em $Sets$. A existência do menor dos pontos fixos de (8.101) está garantida sempre que $\mathcal{F}$ é *co-contínuo*. Ignoraremos aqui alguns detalhes técnicos, sendo suficiente sabermos que todo o functor polinomial é *co-contínuo* [16].

Queremos agora discutir o seguinte salto de refinamento:

$$X_0 \trianglelefteq_f^\phi K \times (K \rightharpoonup \mathcal{F}(K)) \tag{8.102}$$

onde K é um domínio de "apontadores" tal que $K \cong I\!N$. Ora (8.102) será garantido pela existência de uma sobrejecção

$$f : K \times (K \rightharpoonup \mathcal{F}(K)) \longrightarrow X_0$$

total sobre $\{\langle k, \sigma \rangle \in K \times (K \rightharpoonup \mathcal{F}(K)) \mid \phi(k, \sigma)\}$. Para simplificarmos inicialmente o problema, assumiremos temporariamente que $\sigma \in K \rightharpoonup \mathcal{F}(K)$ é uma função total e desenharemos o diagrama seguinte:

$$\begin{array}{c} K \\ \downarrow \sigma \\ \mathcal{F}(K) \end{array} \tag{8.103}$$

Fixado um dado σ — que intuitivamente veremos como um segmento de 'heap' endereçado por K, ou como uma "base de dados" com chave K — f_σ designará a função que, para cada "apontador" $k \in K$, abstrai o valor de X_0 que corresponde

[16] Ver [MA86], p.262,270.

a uma "navegação" sobre σ tomando k como endereço de partida. Podemos acrescentar f_σ a (8.103), ou seja desenhar

$$\begin{array}{ccc} X_0 & \xleftarrow{f_\sigma} & K \\ & & \downarrow \sigma \\ & & \mathcal{F}(K) \end{array}$$

Como (X_0, δ) é ponto-fixo de $\mathcal{F}$, podemos adicionar δ ao diagrama e obter

$$\begin{array}{ccc} X_0 & \xleftarrow{f_\sigma} & K \\ \delta \uparrow & & \downarrow \sigma \\ \mathcal{F}(X_0) & & \mathcal{F}(K) \end{array}$$

Finalmente, o diagrama pode ser fechado por $\mathcal{F}(f_\sigma)$,

$$\begin{array}{ccc} X_0 & \xleftarrow{f_\sigma} & K \\ \delta \uparrow & & \downarrow \sigma \\ \mathcal{F}(X_0) & \xleftarrow{\mathcal{F}(f_\sigma)} & \mathcal{F}(K) \end{array} \tag{8.104}$$

já que $\mathcal{F}$ é um functor.

A equação implícita no diagrama comutativo (8.104) é

$$f_\sigma(k) = \delta(\mathcal{F}(f_\sigma)(\sigma(k)))$$

Escreveremos agora $f(k, \sigma)$ em lugar de $f_\sigma(k)$, obtendo a seguinte função de abstracção para (8.102)

$$\begin{array}{rcl} f & : & K \times (K \rightharpoonup \mathcal{F}(K)) \longrightarrow X_0 \\ f(k, \sigma) & \stackrel{\text{def}}{=} & \delta(\mathcal{F}(f_\sigma)(\sigma(k))) \end{array} \tag{8.105}$$

Podemos exemplificar este raciocínio com o próprio functor polinomial (8.87). $(A^\star, \delta)$ é um ponto-fixo bem-conhecido deste functor [17] onde

$$\begin{array}{rcl} \delta & : & 1 + A \times A^\star \longrightarrow A^\star \\ \delta(x) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} <> & \Leftarrow & x = \langle 1, NIL \rangle \\ cons(a, l) & \Leftarrow & x = \langle 2, \langle a, l \rangle \rangle \end{array} \right. \end{array} \tag{8.106}$$

Neste caso, para $\mathcal{F}(f_\sigma)$ obteremos

$$\begin{array}{rcl} \mathcal{F}(f_\sigma)(x) & = & (1 + A \times f_\sigma)(x) \\ & = & (1_1 + 1_A \times f_\sigma)(x) \\ & = & \left\{ \begin{array}{lcl} x & \Leftarrow & x = \langle 1, NIL \rangle \\ \langle 2, \langle a, f_\sigma(l) \rangle \rangle & \Leftarrow & x = \langle 2, \langle a, l \rangle \rangle \end{array} \right. \end{array} \tag{8.107}$$

[17] Consultar as secções 2.3 e 8.3.1.

Compondo (8.107) com (8.106) obteremos, segundo (8.105),

$$f(k,\sigma) \stackrel{\rm def}{=} \left\{ \begin{array}{lcl} <> & \Leftarrow & \sigma(k) = \langle 1, NIL\rangle \\ cons(a, f_\sigma(k')) & \Leftarrow & \sigma(k) = \langle 2, \langle a, k'\rangle\rangle \end{array} \right.$$

Deixando agora de assumir que σ é total, teremos de encarar o problema da indefinição de apontadores — $\sigma(k)$ em (8.105) será indefinido sempre que $k \notin dom(\sigma)$ e outros apontadores k' no contra-domínio de σ, acessíveis por k, poderão estar na mesma situação.

Esta noção de *acessibilidade* entre apontadores poderá caracterizar-se pelo fecho transitivo $<^+_{\mathcal{F}}$ da seguinte ordem sobre K, induzida por $\mathcal{F}$ e σ:

$$k_1 <_{\mathcal{F}} k \quad \stackrel{\rm def}{=} \quad k \in dom(\sigma) \wedge k_1 \in_{\mathcal{F}} \sigma(k) \tag{8.108}$$

onde a conjunção lógica é de novo assumida como uma conectiva não estrita — *cf.* (8.13) — e $\in_{\mathcal{F}}$ se define por indução sobre a estrutura polinomial de $\mathcal{F}$, como se segue:

$$\begin{array}{rcl} k \in_C x & \stackrel{\rm def}{=} & F \\ k \in_{\lambda X.X} x & \stackrel{\rm def}{=} & k = x \\ k \in_{\mathcal{F}\times\mathcal{G}} \langle x, y\rangle & \stackrel{\rm def}{=} & k \in_{\mathcal{F}} x \vee k \in_{\mathcal{G}} y \\ k \in_{\mathcal{F}+\mathcal{G}} x & \stackrel{\rm def}{=} & \left\{ \begin{array}{lcl} k \in_{\mathcal{F}} y & \Leftarrow & x = \langle 1, y\rangle \\ k \in_{\mathcal{G}} z & \Leftarrow & x = \langle 2, z\rangle \end{array} \right. \end{array} \tag{8.109}$$

O seguinte invariante para (8.102) impede qualquer apontador acessível de k de ser indefinido:

$$\phi(k,\sigma) \quad \stackrel{\rm def}{=} \quad \begin{array}{ll} let & P = \{k\} \cup \{k' \in K \mid k' <^+_{\mathcal{F}} k\} \\ in & P \subseteq dom(\sigma) \end{array}$$

Contudo, este invariante é insuficiente se pretendermos restringir a nossa interpretação ao menor dos pontos fixos, quer dizer, se pretendemos garantir que $f(k,\sigma)$ não dá resultados infinitos. Falta assim forçar que P seja um conjunto *bem-fundado* [18] com respeito a $<_{\mathcal{F}}$:

$$\phi(k,\sigma) \quad \stackrel{\rm def}{=} \quad \begin{array}{ll} let & P = \{k\} \cup \{k' \in K \mid k' <^+_{\mathcal{F}} k\} \\ in & P \subseteq dom(\sigma) \wedge \\ & \forall \emptyset \subset C \subseteq P : \exists m \in C : \forall k' <_{\mathcal{F}} m : k' \notin C \end{array} \tag{8.110}$$

Por exemplo, seja $\mathcal{F}(X) = C \rightharpoonup \mathcal{G}(X)$. Não será difícil obter $\in_{\mathcal{F}}$ para este caso,

$$k \in_{C \rightharpoonup \mathcal{G}} x \stackrel{\rm def}{=} \exists c \in dom(x) : k \in_{\mathcal{G}} x(c) \tag{8.111}$$

[18] Relembrar o exercício 1.22.

A condição (8.111) irá ser útil para entendermos a ilustração do cálculo de desrecursivação que se apresentará na secção 8.7.4.

Finalmente, então, apresenta-se o teorema que corporiza tudo o que acima se descreveu:

Teorema 8.4 (Primeira Lei de Desrecursivação)
Seja $\mathcal{F}$ um endofunctor polinomial em $Sets$ e seja $\langle X_0, \delta\rangle$ um ponto fixo de $\mathcal{F}$ tal que o conjunto X_0 é finito ou infinito numerável e tal que os elementos de X_0, considerados como estruturas arborescentes, são finitos.

Seja K um conjunto infinito numerável. Seja $\phi : K \times (K \rightharpoonup \mathcal{F}(K)) \to 2$ um predicado definido por

$$\begin{array}{rll} \phi(k_0, \sigma) \stackrel{\mathrm{def}}{=} k_0 \in dom(\sigma) & \wedge & (k' <_{\mathcal{F}} k \Rightarrow k' \in dom(\sigma)) \\ & \wedge & (\nexists k \in K : k <_{\mathcal{F}}^{+} k) \\ & \wedge & dom(\sigma) \text{ é finito} \end{array}$$

Seja enfim $f : (K \times (K \rightharpoonup \mathcal{F}(K)))_{\phi} \to X_0$ uma aplicação definida por $f(k_0, \sigma) = f_\sigma(k_0)$ onde $f_\sigma : dom(\sigma) \to X_0$ é uma aplicação auxiliar definida, recursivamente e à custa de σ, por

$$f_\sigma(k) = \delta(\mathcal{F}(f_\sigma)(\sigma(k)))$$

Tem-se então

$$X_0 \trianglelefteq_f^{\phi} (K \times (K \rightharpoonup \mathcal{F}(K)))$$

Demonstração: Ver [Jou92]. □

Até agora temos interpretado o domínio K como um conjunto qualquer, infinito numerável, cujos elementos são, de certo modo, referências para os objectos a implementar. Dissemos que um elemento de K podia ser considerado como um nome, uma chave ou um apontador. É de notar que existe uma diferença fundamental entre um apontador numa linguagem de programação (como *e.g.* PASCAL) e, por exemplo, um nome ou uma chave num sistema relacional. É que um apontador não é apenas o endereço de uma célula de memória. Um apontador pode também "não ser nada", isto é, pode ter o valor *NIL*. Por isso uma operação sobre uma estrutura arborescente, implementada à custa de apontadores p, tem geralmente a forma

$$\left\{ \begin{array}{rcl} p = NIL & \Rightarrow & \ldots \\ \neg(p = NIL) & \Rightarrow & \ldots \end{array} \right.$$

Assim, um domínio de apontadores é sempre da forma $1 + K$ (ou $K + 1$, obviamente), em que $1 = \{NIL\}$ e K é um conjunto (infinito numerável, pelo menos teoricamente) de endereços de células de memória.

Já atrás nos apareceram, mais do que uma vez, domínios da forma $1 + K$, ver *e.g.* (8.100). Vamos agora enunciar um segundo teorema, a que chamaremos *segunda lei de desrecursivação*, que permite obter esse tipo de representação e cuja demonstração (aqui omitida por razões de economia de espaço) mostra que se pode usar a representação fornecida pelo teorema 8.4 para obter a nova representação [19].

Teorema 8.5 (Segunda Lei de Desrecursivação)
Seja G um functor polinomial em $Sets$ da forma

$$G(L) = C_1 \times L + C_2 \times L \times L + \ldots + C_n \times \underbrace{L \times L \times \ldots \times L}_{n}$$

em que pelo menos um dos conjuntos C_i é não vazio.

Seja (X, δ) um ponto fixo da equação

$$L \cong 1 + G(L)$$

em que 1 representa o conjunto singular $\{NIL\}$, tal que todos os elementos de X, considerados como estruturas arborescentes, são finitos e tal que o próprio conjunto X é infinito numerável.

Seja K um conjunto infinito numerável, com $NIL \notin K$. Seja

$$I' : (1 + K) \times (K \rightharpoonup G(1 + K)) \longrightarrow Bool$$

um predicado definido por

$$\begin{array}{l} I'((e, p_0), \sigma') \stackrel{\mathrm{def}}{=} \\ \qquad ((p_0 \neq nil \Rightarrow p_0 \in dom(\sigma')) \quad \begin{array}{ll} \wedge & (k' <_{\sigma'} k \Rightarrow k' \in dom(\sigma')) \\ \wedge & (\not\exists k \in K : k <^{+}_{\sigma'} k) \\ \wedge & dom(\sigma') \text{ é finito}) \end{array} \end{array}$$

Seja enfim

$$r' : ((1 + K) \times (K \rightharpoonup G(1 + K)))_{I'} \rightarrow X$$

uma aplicação definida por

$$r'((e, p_0), \sigma') = r'^{\sigma'}(e, p_0)$$

sendo

$$r'^{\sigma'} : 1 + dom(\sigma') \rightarrow X$$

[19] Essencialmente, junta-se a K um elemento NIL e substitui-se por esse elemento todos os elementos de K que representam estruturas vazias.

uma aplicação auxiliar definida, recursivamente e à custa de σ', por

$$r'^{\sigma'}(e,p) = \delta((id_1 + G(r'^{\sigma'}))((id_1 + \sigma')(e,p)))$$

Tem-se então

$$X \trianglelefteq_{r'} ((1+K) \times (K \rightharpoonup G(1+K)))_{I'}$$

Demonstração: Ver [Jou92]. □

Exercício 8.25 Aplique as leis de desrecursivação estudadas neste curso a cada uma das seguintes definições recursivas,

$$\begin{array}{lcl} X & \cong & 1+X \\ X & \cong & 1+A\times X \\ X & \cong & 1+A\times X^2 \\ X & \cong & 1+A\times X^\star \\ X & \cong & V+A\times X^\star \\ X & \cong & I\times(X+1)^2 \\ X & \cong & I+I\times 2^X \\ X & \cong & (A+X)\rightharpoonup I\!N \end{array}$$

definindo $f(k,\sigma)$ e a ordem $<_{\mathcal{F}}$.

□

8.7.3 Casos Particulares com Interesse

É interessante analisar o significado de (8.102) para alguns functores $\mathcal{F}$ simples:

– Functor constante $\mathcal{F}(X) = C$. $\langle C, 1_C\rangle$ é, obviamente, um ponto-fixo de $\mathcal{F}$. De (8.105) e (8.86) obtém-se a função de abstracção

$$\begin{array}{rcl} f & : & K \times (K \rightharpoonup C) \longrightarrow C \\ f(k,\sigma) & \stackrel{\text{def}}{=} & \delta(\mathcal{F}(f_\sigma)(\sigma(k))) \\ & = & 1_C(C(f_\sigma)(\sigma(k))) \\ & = & 1_C(1_C(\sigma(k))) \\ & = & \sigma(k) \end{array}$$

De (8.108) e (8.109) obtém-se a ordem vazia $<^+_{\mathcal{F}}$ sobre apontadores (K), pelo que o invariante (8.110) se reduz a

$$\phi(k,\sigma) \stackrel{\text{def}}{=} k \in dom(\sigma) \qquad (8.112)$$

como se esperava. De notar que este caso particular corresponde à popular técnica de programação [20] pela qual, em vez de manipular directamente dados do tipo C (estaticamente armazenados), o programa manipula *identificadores* seus, ou referências *dinâmicas* para eles.

[20] Típica de C ou PASCAL, ou ainda da programação por objectos.

– $C = 0$ no caso acima. Este caso trivializa (8.101) à definição $X \cong 0$ de um domínio vazio; notar que $K \times (K \rightharpoonup \mathcal{F}(K))$ simplifica em $K \times (K \rightharpoonup 0)$, que se reduz a

$$\begin{aligned} K \times (K \rightharpoonup 0) &\cong K \times \bigcup_{X \subseteq K} 0^X \\ &\cong K \times 0^0 \end{aligned}$$

Quer dizer, $\sigma : 0 \to 0$ só pode ser completamente indefinida, e $dom(\sigma) = \emptyset$; então $k \in dom(\sigma) = F$ e (8.112) reduz-se a

$$\phi(k,\sigma) \stackrel{\mathrm{def}}{=} F$$

Como se esperava, chegou-se à reificação vazia (onde todo o dado é inválido).

– Functor identidade $\mathcal{F}(X) = X$. Neste caso (8.101) reduz-se a $X \cong X$, que aceita qualquer ponto fixo $(X_0, \delta : X_0 \to X_0)$ onde δ é uma bijecção. Vejamos como o menor dos pontos fixos implícitos em (8.110) reduz este caso ao anterior ($X_0 = 0$), mostrando que definição e a propriedade bem-fundada não podem ser verificadas ao mesmo tempo.

Assumir que P é bem-fundado com respeito a $k_1 <_{\lambda X.X} k$ (que se abreviará para $k_1 < k$ e é logicamente equivalente a $k \in dom(\sigma) \wedge k_1 = \sigma(k)$) implica

$$\exists m \in P : \sigma(m) \notin P \tag{8.113}$$

Mas se $m \in P$ então m é acessível de k ($m <^+ k$), $m \in dom(\sigma)$ e $\sigma(m) < m$. Logo

$$\sigma(m) < m <^+ k$$

i.é $\sigma(m) <^+ k$, que implica $\sigma(m) \in P$ e contradiz (8.113). Logo a conjunção da definição de apontadores e a ausência de ciclos em (8.112) é impossível, e obtemos

$$\phi(k,\sigma) \stackrel{\mathrm{def}}{=} F$$

Um sabor mais algorítmico pode ser emprestado a (8.110) introduzindo uma função auxiliar

$$reach(k,\sigma) \stackrel{\mathrm{def}}{=} \{k\} \cup \{k' \in K \mid k' <^+_{\mathcal{F}} k\}$$

(quer dizer, $P = reach(k,\sigma)$) sujeita às transformações que se seguem [21]:

$reach(k,\sigma) \stackrel{\mathrm{def}}{=}$

[21] Cf. as leis do apêndice B.

$$
\begin{array}{ll}
 & \{k\} \cup \{k' \in K \mid \ k' <^{+}_{\mathcal{F}} k\} \\
= & \{k\} \cup \left\{ \begin{array}{lll} \emptyset & \Leftarrow & k \notin dom(\sigma) \\ \bigcup_{k' <_{\mathcal{F}} k} \{k'\} \cup \{k'' \in K \mid \ k'' <^{+}_{\mathcal{F}} k'\} & \Leftarrow & k \in dom(\sigma) \end{array} \right. \\
= & \{k\} \cup \left\{ \begin{array}{lll} \emptyset & \Leftarrow & k \notin dom(\sigma) \\ \bigcup_{k' <_{\mathcal{F}} k} reach(k', \sigma) & \Leftarrow & k \in dom(\sigma) \end{array} \right.
\end{array}
$$

O teste de definição pode ser encapsulado num predicado

$$defined(k, \sigma) \stackrel{\rm def}{=} P \subseteq dom(\sigma)$$

Como $P = reach(k, \sigma)$ obtemos, depois das substituições e transformações necessárias:

$$defined(k, \sigma) \stackrel{\rm def}{=} \left\{ \begin{array}{lll} F & \Leftarrow & k \notin dom(\sigma) \\ \forall k' <_{\mathcal{F}} k : defined(k', \sigma) & \Leftarrow & k \in dom(\sigma) \end{array} \right.$$

Notar que $reach$ e $defined$ não calculam o pretendido ponto-fixo implícito no fecho transitivo de $<_{\mathcal{F}}$ no caso de esta relação ser cíclica. Um ciclo-$<_{\mathcal{F}}$ é detectado sempre que se revisita o mesmo $k \in K$. Como a detecção de ciclos se ajusta ao teste de definição de apontadores, podemos agregar os dois testes e escrever

$$\phi(k, \sigma) = \phi_{aux}(k, \sigma, \emptyset) \tag{8.114}$$

onde

$$
\begin{array}{l}
\phi_{aux}(k, \sigma, C) \stackrel{\rm def}{=} \\
\qquad \left\{ \begin{array}{lll} F & \Leftarrow & k \notin dom(\sigma) \vee k \in C \\ \forall k' <_{\mathcal{F}} k : \phi_{aux}(k', \sigma, C \cup \{k\}) & \Leftarrow & k \in dom(\sigma) \end{array} \right.
\end{array} \tag{8.115}
$$

8.7.4 Exemplo de Cálculo de Des-recursivação

O modelo $DecTree$ para árvores de decisão atrás apresentado (8.20) será explorado nesta secção e submetido a uma série de cálculos, no sentido de ilustrar os principais resultados deste capítulo. Os saltos de refinamento serão indexados por números naturais, de forma a facilitar a aplicação da regra (8.14) para inferência de funções de abstracção e invariantes.

Reificação dos Domínios dos Dados

A remoção de recursividade de acordo com (8.102) será a primeira transformação aplicável a (8.20). Obtém-se

$$DecTree \quad \trianglelefteq_1 \quad DecTree_1$$

onde

$$DecTree_1 \quad = \quad K \times (K \rightharpoonup What \times (Answer \rightharpoonup K)) \qquad (8.116)$$

Antes de prosseguirmos, pensemos um pouco sobre o que já conseguimos em $DecTree_1$ (8.116). São admissíveis duas interpretações (pelo menos):

1. Olhando para K como um domínio de *apontadores*,

   $$K \rightharpoonup What \times (Answer \rightharpoonup K) \qquad (8.117)$$

   modela um segmento de 'heap', *i.é* de memória dinâmica. Em PASCAL (onde o 'heap' está "escondido" no 'run-time system') escreveríamos qualquer coisa como [22]

   ```
   type  DecTree1 =  ^DecTree;
         DecTree =   record
                     Q: What;
                     R: array [Answer] of ^DecTree
                     end;
   ```
   (8.118)

2. Olhando para K como um domínio de *nomes de objectos*, (8.117) modela a base de objectos

   $$nome \rightarrow objecto$$

   implícita em qualquer ambiente de programação orientada aos objectos [Wol88]. Re-escrevendo (8.116) mais sugestivamente:

   $$\begin{array}{rcl} DecTree_1 & \cong & \begin{array}{rl} ObjName: & K \\ Archive: & ObjBase \end{array} \times \\ ObjBase & = & K \rightharpoonup Attributes \\ Attributes & \cong & \begin{array}{rl} Q: & What \\ SubObjs: & Answer \stackrel{m}{\to} K \end{array} \times \end{array}$$

Prosseguiremos agora via (8.72):

$$\begin{array}{rcl} DecTree_1 & = & K \times (K \rightharpoonup What \times (Answer \rightharpoonup K)) \\ & \trianglelefteq_2 & K \times ((K \rightharpoonup What) \times ((K \times Answer) \rightharpoonup K)) \\ & = & DecTree_2 \end{array} \qquad (8.119)$$

Agora, olhando para K como um domínio de *estados*, $DecTree_2$ aceita a interpretação seguinte:

[22] Estritmente falando, (8.118) é já um refinamento de (8.117) porque, como já se viu, os apontadores em PASCAL implementam $K + 1$ em lugar de K (a alternativa 1 corresponde ao valor `nil`). Portanto, um invariante sobre (8.118) será necessário, impedindo a alternativa `nil` em `DecTree1`.

- $(K \times Answer) \rightharpoonup K$ = *diagrama estados-transições* de um *autómato de estados finitos* (determinístico) onde $Answer$ = estímulos de entrada;
- o primeiro factor $k \in K$ em (8.119) = estado corrente do autómato;
- $K \rightharpoonup What$ = tabela semântica associando um significado a cada estado.

Assim, o salto 2 acaba de nos conduzir a um nível mais baixo onde, apesar de tudo, encontramos "velhos amigos": autómatos de estados finitos podem implementar-se com 'arrays' e 'jumps'! Continuando com o raciocínio teremos, sucessivamente,

$$\begin{array}{rcl} DecTree_2 & = & K \times ((K \rightharpoonup What) \times ((K \times Answer) \rightharpoonup K)) \\ & \trianglelefteq_3 & K \times (2^{K \times What} \times 2^{(K \times Answer) \times K}) \\ & = & DecTree_3 \\ & \cong_4 & K \times (2^{K \times What} \times 2^{K \times Answer \times K}) \\ & = & DecTree_4 \end{array}$$

cf. (8.9) and (8.30). O modelo final,

$$DecTree_4 = K \times (2^{K \times What} \times 2^{K \times Answer \times K}) \tag{8.120}$$

não é nem mais nem menos do que um esquema relacional para implementar $DecTree$ sob a forma de dois ficheiros de base de dados (tabelas) onde K assume agora o papel de um domínio de *chaves*.

Em resumo, o nosso raciocínio pode esquematizar-se como se segue:

$$DecTree \trianglelefteq_1 DecTree_1 \trianglelefteq_2 DecTree_2 \trianglelefteq_3 DecTree_3 \cong_4 DecTree_4 \tag{8.121}$$

Inferência do Invariante de Abstracção

A função de abstracção global para (8.121) pode inferir-se via (8.14), *i.é*

$$f(k, \langle t, t' \rangle) = f_1(f_2(f_3(f_4(k, \langle t, t' \rangle))))$$

onde f se obtém por composição functorial das funções de abstracção f_1 a f_4, como se segue: f_4 é dada por

$$f_4 = 1_K \times (1_{2^{K \times What}} \times 2^g)$$

onde g é o isomorfismo

$$(A \times B) \times C \cong_g A \times B \times C$$

definido por

$$g(\langle a, b, c\rangle) \quad = \quad \langle\langle a, b\rangle, c\rangle$$

isto é,

$$f_4(\langle k, \langle t, t'\rangle\rangle) \quad = \quad \langle k, \langle t, \{\langle\langle k, a\rangle, k'\rangle \mid \langle k, a, k'\rangle \in t'\}\rangle\rangle$$

De forma semelhante, f_3 baseia-se na abstracção mkf (8.9) de relações para funções,

$$f_3 \quad = \quad 1_K \times (mkf \times mkf)$$

enquanto que f_2 é, simplesmente,

$$f_2 = 1_K \times \bowtie_n$$

onde $\bowtie_n$ é dada por (8.74). Finalmente,

$$\begin{array}{rcl} f_1(k, \sigma) & = & f_\sigma(k) \\ f_\sigma(k) & \stackrel{\text{def}}{=} & let \quad \sigma(k) = \langle w, \sigma'\rangle \\ & & in \quad \langle w, \left(\begin{array}{c} a \\ f_\sigma(\sigma'(a)) \end{array}\right)_{a \in dom(\sigma')}\rangle \end{array} \tag{8.122}$$

cf. (8.105,8.81) e (8.88). Então

$$\begin{array}{rcl} f_3(f_4(\langle k, \langle t, t'\rangle\rangle)) & = & \langle k, \langle mkf(t), mkf(\{\langle\langle k, a\rangle, k'\rangle \mid \langle k, a, k'\rangle \in t'\})\rangle\rangle \\ & = & \langle k, \langle \left(\begin{array}{c} k \\ w \end{array}\right)_{\langle k, w\rangle \in t}, \left(\begin{array}{c} \langle k, a\rangle \\ k' \end{array}\right)_{\langle k, a, k'\rangle \in t'} \rangle\rangle \end{array}$$

e

$$\begin{array}{l} f_2(f_3(f_4(\langle k, \langle t, t'\rangle\rangle))) = \\ \qquad \langle k, \left(\begin{array}{c} x \\ \langle w, \left(\begin{array}{c} a \\ k' \end{array}\right)_{\langle k'', a, k'\rangle \in t' \wedge k'' = x} \rangle \end{array}\right)_{\langle x, w\rangle \in t} \rangle \end{array} \tag{8.123}$$

Combinando (8.123) com (8.122) obtemos, depois de algumas simplificações,

$$\begin{array}{rcl} f(k, \langle t, t'\rangle) & \stackrel{\text{def}}{=} & let \quad w = the(\{\pi_2(r) \mid r \in t \wedge \pi_1(r) = k\}) \\ & & in \quad \langle w, \left(\begin{array}{c} a \\ f(k', \langle t, t'\rangle) \end{array}\right)_{\langle k'', a, k'\rangle \in t' \wedge k'' = k} \rangle \end{array} \tag{8.124}$$

Passemos agora à inferência do invariante global, que é dado por:

$$\phi(k, \langle t, t'\rangle) = \phi_3(f_4(k, \langle t, t'\rangle)) \wedge \phi_2(f_3(f_4(k, \langle t, t'\rangle))) \wedge \phi_1(f_2(f_3(f_4(k, \langle t, t'\rangle))))$$

já que ϕ_4 é sempre verdadeiro. ϕ_3 recorre a fdp (8.7),

$$\phi_3 = (\lambda k.V) \times fdp \times fdp$$

e ϕ_2 é o predicado dpi (8.73). Pode verificar-se que $\phi_2(f_3(f_4(k, \langle t, t' \rangle)))$ se reduz a $\pi_1[t'] \subseteq \pi_1[t]$. Então

$$\begin{array}{lcl} \phi_3(f_4(k, \langle t, t' \rangle)) & = & V \wedge fdp(t) \wedge fdp(\{\langle \langle k, a \rangle, k' \rangle \mid \langle k, a, k' \rangle \in t'\}) \\ & = & fdp(t) \wedge fdp(\{\langle \langle k, a \rangle, k' \rangle \mid \langle k, a, k' \rangle \in t'\}) \end{array} \tag{8.125}$$

Continuando a calcular $fdp(\{\langle \langle k, a \rangle, k' \rangle \mid \langle k, a, k' \rangle \in t'\})$ em (8.125) teremos

$$\forall \langle k_1, a, k_1' \rangle, \langle k_2, b, k_2' \rangle \in t' : (k_1 = k_2 \wedge a = b) \Rightarrow k_1' = k_2'$$

Finalmente, ϕ_1 (8.110) baseia-se na ordem

$$\begin{array}{lcl} k_1 < k & \stackrel{\text{def}}{=} & k \in dom(\sigma) \wedge \\ & & (F \vee \exists a \in dom(\pi_2(\sigma(k))) : k_1 = \pi_2(\sigma(k))(a)) \\ & \Leftrightarrow & k \in dom(\sigma) \wedge \ \ let \ \ \sigma' = \pi_2(\sigma(k)) \\ & & \qquad\qquad\qquad in \ \ \exists a \in dom(\sigma') : k_1 = \sigma'(a) \end{array}$$

cf. (8.111). Relembrando (8.114,8.115) acima, podemos reduzir

$$\phi_1(f_2(f_3(f_4(k, \langle t, t' \rangle))))$$

a $aux(k, t, t', \emptyset)$ onde

$$\begin{array}{l} aux(k, t, t', C) \stackrel{\text{def}}{=} \\ \quad \left\{ \begin{array}{lcl} F & \Leftarrow & k \notin \pi_1[t] \vee \\ & & k \in C \\ \forall r \in \{\pi_1(s) = k \mid s \in t'\} : aux(\pi_3(r), t, t', C \cup \{k\}) & \Leftarrow & k \in \pi_1[t] \end{array} \right. \end{array} \tag{8.126}$$

Combinando os resultados anteriores, obtemos finalmente o seguinte invariante (não-trivial!) sobre (8.120):

$$\begin{array}{l} \phi(k, \langle t, t' \rangle) = \\ \qquad (\forall \langle k, w \rangle, \langle k', w' \rangle \in t : k = k' \Rightarrow w = w') \wedge \\ \qquad (\forall \langle k_1, a, k_1' \rangle, \langle k_2, b, k_2' \rangle \in t' : (k_1 = k_2 \wedge a = b) \Rightarrow k_1' = k_2') \wedge \\ \qquad \pi_1[t'] \subseteq \pi_1[t] \wedge \\ \qquad aux(k, t, t', \emptyset) \end{array} \tag{8.127}$$

onde aux é o predicado recursivo auxiliar dado por (8.126).

Por muito complicado que o leitor ache o invariante (8.127), deverá reflectir sobre o seguinte:

- é o "menor" invariante que garante, segundo o processo de cálculo seguido, a correcta implementação relacional de $DecTree$;
- é uma medida eloquente para se avaliar a "insegurança" e o risco que se corre ao utilizar essa implementação completamente desapoiada.

Exercício 8.26 Relembre do Exercício 2.38 o seguinte modelo recursivo, em $Sets$, para *árvores-B*

$$BT \cong 1 + BT \times (A \times BT)^\star$$

onde, sobre o tipo de elementos A se considera definida uma ordem total $<$. No contexto da des-recursivação deste modelo, caracterize a ordem $<_\mathcal{F}$ correspondente, *i.é* onde

$$\mathcal{F}(X) \cong 1 + X \times (A \times X)^\star$$

□

Exercício 8.27 Considere o seguinte modelo recursivo, em $Sets$, para *árvores de expressão*

$$E \quad \cong \quad V + O \times E^\star \tag{8.128}$$

onde V designa um conjunto de *variáveis* e O um conjunto de *operadores*.

1. Escreva um invariante sobre E que garanta que *em todas as ocorrências de um operador $o \in O$ numa expressão σ o seu número de argumentos é o mesmo*.
2. No contexto da des-recursivação deste modelo, caracterize a ordem $<_\mathcal{F}$ correspondente, onde
$$\mathcal{F}(X) \cong V + O \times X^\star$$
3. Complete as reticências no seguinte processo de refinamento de E:
Como $E^\star$ é ponto fixo de
$$\ldots$$
obteremos de (8.128), por substituição,
$$E \quad \cong \quad \ldots \tag{8.129}$$
$$\ldots \quad \cong \quad \ldots \tag{8.130}$$
Substituindo agora E (8.129) em (8.130), teremos
$$\ldots$$
Em suma, (8.128) implementa-se por
$$\left\{ \begin{array}{lcl} E & \cong & V + O \times L \\ L & \cong & 1 + V \times L + O \times L^2 \end{array} \right. \tag{8.131}$$
4. Aos olhos da analogia que foi dada nas aulas entre as construções primitivas de $Sets$ e construções semelhantes nas linguagens de programação estruturada, como codificaria (8.131) em C ou PASCAL? (escolha uma das linguagens).

□

8.8 Ainda Sobre a Manipulação de Invariantes

Vimos atrás como o processo de refinamento acarreta a indução de invariantes sobre os dados requeridos para garantir a representação fiel da informação da especificação. Esses invariantes são sintetizados em cada passo e são, de certa forma, "os menores" possíveis, *cf.* secção 8.1.2.

Esta estratégia tem a vantagem de impedir implementações "mais fortes" do que efectivamente necessário — como se pode apreciar ao resolver o Exercício 8.22, por exemplo — mas ignora outros aspectos da reificação dos dados que são relevantes na prática, nomeadamente,

- a própria especificação inicial de um projecto está, ela própria, sujeita a invariantes 'ad hoc', *i.é* abritrários; por exemplo, mesmo um tipo primitivo como

$$Date \cong 31 \times 12 \times I\!N$$

requer um invariante não-trivial:

$$\begin{array}{rcl} dateOk & : & Date \longrightarrow 2 \\ dateOk(d,m,a) & \stackrel{\rm def}{=} & \left\{ \begin{array}{lcl} m \in \{1,3,5,7,8,10,12\} & \Rightarrow & d \leq 31 \wedge \\ & & (\neg(a = 1582 \wedge m = 10) \\ & & \vee(d < 5) \vee (14 < d)) \\ m \in \{4,6,9,11\} & \Rightarrow & d \leq 30 \\ m = 2 \wedge leap(a) & \Rightarrow & d \leq 29 \\ m = 2 \wedge \neg leap(a) & \Rightarrow & d \leq 28 \end{array} \right. \end{array}$$

onde

$$\begin{array}{rcl} leap & : & I\!N \longrightarrow 2 \\ leap(a) & \stackrel{\rm def}{=} & rem(a, \left\{ \begin{array}{lcl} 1700 \leq a \wedge rem(a,100) = 0 & \Rightarrow & 400 \\ 1700 > a \vee rem(a,100) \neq 0 & \Rightarrow & 4 \end{array} \right.) = 0 \end{array}$$

- algumas regras do cálculo são "sensíveis ao contexto" no sentido em que só podem ser aplicadas a um tipo de dados se este estiver afectado por um dado invariante [23];

- pode ser desejável introduzir invariantes 'ad hoc' ao longo do refinamento que nada têm a ver com fidelidade de representação (*representatividade*), mas sim com outras facetas do refinamento como a procura de *eficiência*.

Por exemplo, sabemos que

$$2^A \trianglelefteq_{elems} A^\star \qquad (8.132)$$

Contudo, podemos estar interessados em implementar 2^A por listas de $A^\star$ *ordenadas* por uma ordem total sobre A, ordem essa que é irrelevante para a função de abstracção $elems$.

[23] Veja-se [Oli90] para mais detalhes.

Em geral, se A, B forem conjuntos finitos tais que $A \trianglelefteq_f B$, pode acontecer que $A \trianglelefteq_{f|\phi} B_\phi$ se verifique, onde $\phi : B \longrightarrow 2$ é um invariante sobre B e B_ϕ designa o conjunto $\{b \in B \mid \phi(b)\}$. ϕ pode ser tão forte quanto desejarmos, desde que a cardinalidade B_ϕ seja maior que a de A. No "limite", $A \cong B_\phi$. Tal será o caso em

$$2^A \cong_{elems|linear} (A^\star)_{linear}$$

onde $linear$ força que cada sequência $s \in A^\star$ esteja totalmente ordenada por uma dada ordem linear $\sqsubset \in A \times A$:

$$linear(s) \stackrel{\text{def}}{=} \forall 1 \leq i < j \leq length(s) : s(i) \sqsubset s(j)$$

O objectivo desta secção é o de analisar o impacto de tais invariantes 'ad hoc' nos resultados anteriormente estudados.

8.8.1 Invariantes 'Ad Hoc'

Antes de mais, será a partir de agora necessário decorar cada domínio de dados A com o "seu" invariante ϕ,

$$A_\phi$$

uma vez que ϕ pode, num dado estado do refinamento, ser mais forte do que o implicado pelas regras-$\trianglelefteq$ até aí aplicadas. De facto, um ou mais invariantes extra ('ad hoc') podem ter sido entretanto introduzidos ao arbítrio do implementador. Assim, passaremos a escrever

$$A \trianglelefteq_f^\phi B_\phi$$

em lugar de

$$A \trianglelefteq_f^\phi B$$

sendo ϕ omitido apenas se for o predicado sempre verdadeiro ($\lambda b.T$).

O teorema que segue permite-nos introduzir invariantes 'ad hoc' em qualquer passo de um refinamento.

Teorema 8.6 *Se $A \trianglelefteq_f^\phi B_\phi$ se verifica e se $\alpha : A \longrightarrow 2$ é um invariante extra sobre A, então α pode ser "empurrado" de A para B como se segue:*

$$A_\alpha \trianglelefteq_f^\phi B_{\phi \wedge \alpha \circ f}$$

em que $\phi \wedge \alpha \circ f$ abrevia o predicado $\lambda b.\phi(b) \wedge \alpha(f(b))$.
Demonstração: Ver [Jou92]. □

Precisaremos ainda de mais alguns resultados.

Teorema 8.7 *Sendo $\mathcal{F}$ um (endo)functor polinomial em $Sets$ ou o functor 2^X, A finito e $\phi : A \longrightarrow 2$ um invariante sobre A, tem-se que*

$$\mathcal{F}(A_\phi) \cong \mathcal{F}(A)_{\mathcal{F}(\phi)} \tag{8.133}$$

Demonstração: Ver [Oli94]. □

Teorema 8.8 *Seja $A \trianglelefteq_f B$ verdadeiro e $\phi : B \longrightarrow 2$ um predicado tal que a restrição de f por ϕ,*

$$f|\phi : B_\phi \longrightarrow A$$

é ainda sobrejectiva. Então

$$A \trianglelefteq_{f|\phi} B_\phi$$

Demonstração: Óbvia, a partir da definição 8.10. □

No que diz respeito à funcionalidade de um modelo, temos o seguinte teorema garantindo que, se uma operação preserva um invariante 'ad hoc', então qualquer uma sua implementação válida o faz, a baixo nível.

Teorema 8.9 *Seja $\sigma : A \to A$ a especificação de uma função transformando elementos de A e preservando um dado invariante 'ad hoc' α definido sobre A, isto é* [24],

$$\forall a \in A : \alpha(a) \Rightarrow \alpha(\sigma(a)) \tag{8.134}$$

Então, o refinamento σ' de σ implicado pelo salto (8.11),

$$\begin{tikzcd} A \arrow[r, "\sigma"] & A \\ B_\phi \arrow[u, "f"] \arrow[r, "\sigma'"'] & B_\phi \arrow[u, "f"'] \end{tikzcd}$$

preservará α ao nível de B.

Demonstração: σ' é qualquer função que satisfaça

$$\forall b \in B_\phi : f(\sigma'(b)) = \sigma(f(b)) \tag{8.135}$$

α será preservado por σ', ao nível de B, sse

$$\forall b \in B_\phi : \alpha(f(b)) \Rightarrow \alpha(f(\sigma'(b)))$$

[24] O facto de σ ser unário não corresponde a falta de generalidade: o teorema permanece válido para aridades arbitrárias de σ:

$$\sigma : \ldots \times A \times \ldots \to A$$

Por redução ao absurdo, α não será preservado se

$$\exists b \in B_\phi : \alpha(f(b)) \wedge \neg\alpha(f(\sigma'(b)))$$

isto é,

$$\exists b \in B_\phi : \alpha(f(b)) \wedge \neg\alpha(\sigma(f(b)))$$

via (8.135). Porque f é sobrejectiva, $f(b) = a$ para algum $a \in A$. Logo, teremos

$$\exists a \in A : \alpha(a) \wedge \neg\alpha(\sigma(a))$$

assim contradizendo a hipótese (8.134). $\Box$

Passaremos de imediato ao estudo de um exemplo de aplicação destes resultados.

8.8.2 Exemplo: Tabelas de 'Hashing'

As tabelas de 'hashing' são estruturas de dados bem conhecidas [Wir76, HS78] cujo objectivo é combinar eficientemente as vantagens tanto da memória estática como da dinâmica. Estruturas estáticas como os 'arrays' são de acesso aleatório mas têm a desvantagem de ocupar demasiada memória primária. As estruturas dinâmicas, baseadas em *apontadores* (*e.g.* listas lineares, árvores de procura *etc.*) são mais versáteis com respeito a requisitos de memória mas o acesso à informação não é tão imediato.

A ideia do 'hashing' é sugerida pelo significado da própria palavra: uma grande base de dados é "hashed" em tantos "pedaços" quanto possível. A procura dentro de cada "pedaço" não é imediata, mas cada pedaço é acedido aleatoriamente. Como cada uma dessas sub-bases de dados é mais pequena que a original, o tempo gasto na procura é encurtado por um dado factor dependente do número de sub-bases. O acesso aleatório obtem-se normalmente pela chamada função de 'hashing',

$$H : Data \longrightarrow Location$$

que calcula, para cada item, a sua localização na tabela de 'hashing'. A terminologia 'standard' encara como *sinónimos* todos os items que *colidem*, *i.é* competem para o mesmo local, ou endereço. Um conjunto de sinónimos chama-se *bucket*.

Há várias maneiras de tratar colisões, como *e.g.* 'linear probing' [Wir76] ou 'overflow handling' [HS78]. Apenas nos dedicaremos a esta última.

Processo de Cálculo

Seja $n \in \mathbb{N}$ o número de entradas previstas para a nossa tabela de 'hashing' e

$$H : A \longrightarrow n$$

uma dada função de 'hashing'. Normalmente, n é muito menor que o cardinal de A, isto é, H é claramente não-injectiva.

O objectivo é usar $\unlhd$-resultados para converter

$$2^A \unlhd \ldots$$

em algo que possamos identificar como um modelo aceitável de uma tabela de 'hashing'. O ponto de partida é o resuldado básico (8.60):

$$A \unlhd_{\pi_2} B \times A$$

Instanciando (8.60) para $B = n$,

$$A \unlhd_{\pi_2} n \times A \tag{8.136}$$

imporemos de imediato a $n \times A$ um invariante 'ad hoc':

$$\phi(i,a) \stackrel{\rm def}{=} i = H(a) \tag{8.137}$$

por forma a que cada item $a \in A$ só seja emparelhável com o seu índice de 'hashing' $H(a)$. Ora a restrição de π_2 a $(n \times A)_\phi$ é sobrejectiva sobre A, pois

$$\pi_2[\{(i,a) \in n \times A \mid \ i = H(a)\}] = A$$

e teremos

$$A \unlhd_{\pi_2|\phi} (n \times A)_\phi \tag{8.138}$$

pelo Teorema 8.8. Aplicando a (8.138) o functor "partes de X", $\lambda X.2^X$, obtemos

$$2^A \unlhd_{f_1} (2^{n\times A})_{\Phi_1} \tag{8.139}$$

onde

$$\begin{array}{rcl}\Phi_1 & = & 2^\phi \\ & = & \lambda s.(\forall x \in s : \phi(x)) \\ & = & \lambda s.(\forall (i,a) \in s : i = H(a))\end{array}$$

e

$$\begin{array}{rcl}f_1 & = & 2^{(\pi_2|\phi)} \\ & = & \lambda s.\{\pi_2(x) \mid \ x \in s\} \\ & = & \lambda s.\{a \in A \mid \ (i,a) \in s\}\end{array} \tag{8.140}$$

Relembremos agora a lei de 'currying' da exponenciação (8.38),

$$A^{B\times C} \quad \cong_{uncurry} \quad (A^C)^B$$

onde

$$uncurry(t) \stackrel{\text{def}}{=} \lambda(b,c).t(b)(c)$$

Para $A = 2$ obtem-se a seguinte versão de $uncurry$,

$$uncurry(t) \stackrel{\text{def}}{=} \{\langle b,c\rangle \mid\ b \in B \wedge c \in t(b)\}$$

A instância de (8.38) que nos interessa é — *cf.* o lado direito de (8.139) —

$$2^{n\times A} \cong (2^A)^n$$

Se manipularmos os invariantes de acordo com o Teorema 8.6 obteremos

$$(2^{n\times A})_\Phi \cong_{f_2} ((2^A)^n)_{\Phi_2} \tag{8.141}$$

de onde se deduz:

$$\begin{aligned} f_2 &= \lambda t.uncurry(t) \\ &= \lambda t.\{(i,a) \mid\ i \in n \wedge a \in t(i)\} \end{aligned} \tag{8.142}$$

e

$$\begin{aligned} \Phi_2 &= \Phi \circ uncurry \qquad (8.143) \\ &= \lambda t.\Phi(\{(i,a) \mid\ i \in n \wedge a \in t(i)\}) \\ &= \lambda t.\forall i \in n : (\forall a \in t(i) : i = H(a)) \end{aligned}$$

É fácil mostrar que $((2^A)^n)_{\Phi_2}$ acima — *i.é* o 'array' $(n \to 2^A)_{\Phi_2}$ — é de facto um modelo para tabelas de 'hashing'. De acordo com (8.144), todo o conjunto no codomínio de tais 'arrays' conterá *sinónimos* [HS78]. E todos esses conjuntos são mutuamente disjuntos, *cf.* o seguinte lema:

Lema 8.1 *Para cada tabela de 'hashing'* $t \in (n \to 2^A)_{\Phi_2}$ *verifica-se*

$$\forall i \neq j \in n : t(i) \cap t(j) = \emptyset$$

Demonstração: Por redução ao absurdo, supor que, para alguns $i \neq j$ e $a \in A$, se tem

$$a \in t(i) \wedge a \in t(j)$$

Então, por Φ_2 (8.144) tem-se

$$i = H(a) \wedge j = H(a)$$

i.é, $i = j$, logo contradizendo a hipótese inicial de que $i \neq j$. □

Em suma, cada $db \in 2^A$ é partida em n segmentos de colisão, disjuntos, cada um endereçado pelo correspondente índice de 'hashing' calculado por H. Contudo, a tarefa de refinar 2^A "recorre" no codomínio de $n \to 2^A$, um problema conhecido pela designação de *overflow handling* ou *collision handling* [HS78].

Manipulação de Colisões

Conhecida a propriedade composicional dos refinamentos em $Sets$, salta à ideia tentar refinar os próprios conjuntos de colisões em (sub)tabelas de 'hashing'. Esta solução tem o nome de 'rehashing' e conduz a qualquer coisa como

$$n \to (m \to 2^A) \tag{8.144}$$

sob um invariante elaborado envolvendo n (possivelmente diferentes) funções de 'sub-hashing' H_i $(i = 1, \ldots, n)$:

$$\Phi_3 \quad = \quad \lambda t.\forall i \in n : (\forall j \in m : (\forall a \in t(i)(j) : i = H(a) \wedge j = H_i(a))) \tag{8.145}$$

Mas, estritamente falando, *rehashing* não é mais do que multiplicar o espaço de endereçamento de uma dada tabela de 'hashing' por um dado factor (m), pois

$$((2^A)^m)^n \cong (2^A)^{n \times m}$$

— *cf.* a lei (8.38) — onde a função de 'hashing' é

$$H'(a) = \langle H(a), H_{H(a)}(a) \rangle$$

Tipicamente $m < n$, e os conjuntos de colisões tornam-se mais pequenos. Mas 2^A ainda recorre no co-domínio de (8.144)!

Uma melhor solução será analisar a teoria de refinamento de 2^A que já conhecemos, e aí muitas opções estão disponíveis, por exemplo $A^\star$ — *cf.* (8.2). Teremos então tabelas de 'hashing' convertidas em 'arrays' de sequências de colisões,

$$(n \to A^\star)_{\Phi_3} \tag{8.146}$$

onde Φ_3 se obtem via o Teorema 8.3,

$$2^A \trianglelefteq_{elems} A^\star \Rightarrow (2^A)^n \trianglelefteq_{elems^n} (A^\star)^n$$

onde

$$elems^n(t) = \lambda i.elems(t(i)) \tag{8.147}$$

e pelo Teorema 8.6,

$$(2^A)^n \trianglelefteq_{elems^n} (A^\star)^n \quad \Rightarrow \quad ((2^A)^n)_{\Phi_2} \trianglelefteq_{elems^n} ((A^\star)^n)_{\Phi_2 \circ elems^n}$$

ou seja,

$$\begin{aligned} \Phi_3 &= \Phi_2 \circ elems^n \\ &= \lambda t.\forall i \in n : (\forall a \in elems(t(i)) : i = H(a)) \end{aligned} \tag{8.148}$$

A função de abstracção global será, *cf.* (8.147), (8.142) e (8.140),

$$
\begin{aligned}
f(t) &= f_1(f_2(f_3(t))) \\
&= f_1(f_2(elems(t(i)))) \\
&= \{a \in A \mid \ (i,a) \in \{(i,a) \mid \ i \in n \wedge a \in elems(t(i))\}\} \\
&= \{a \in A \mid \ i \in n \wedge a \in elems(t(i))\} \\
&= \{a \in A \mid \ \exists i \in n : a \in elems(t(i))\} \\
&= \bigcup_{i \in n} \{a \in A \mid \ a \in elems(t(i))\} \\
&= \bigcup_{i \in n} elems(t(i)) \qquad (8.149)
\end{aligned}
$$

confirmando a intuição de que a base abstracta é a reunião de todos os seus "pedaços" guardados numa tabela de 'hashing'.

Exercício 8.28 Prossiga com o processo de cálculo que se apresentou ao longo desta secção, refinando A^* em lista ligada por apontadores e termine com uma codificação na linguagem C.
□

Exercício 8.29 Relembre o esquema relacional derivado no Exercício 8.19. Calcule o impacto de lhe associar, agora, o invariante ('ad hoc') que se segue e que deverá garantir, sobre DPP, as condições elaboradas que se seguem:

- que nenhum equipamento é feito de sub-equipamentos desconhecidos;
- que nenhum equipamento vem a incluir-se a si próprio como sub-equipamento;
- que se conhece o preço de todo o componente que é usado pelo menos num equipamento;
- que se não podem ter em 'stock' equipamentos que se não fabricam.

□

8.9 Invariantes 'Ad Hoc' Implícitos em Diagramas ERA

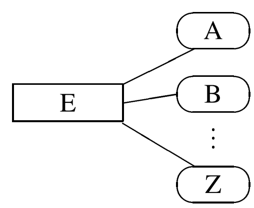
Antes de terminarmos, apresentaremos, a título ilustrativo, vários esquemas de inversão de Diagramas ERA ('Entity-Relationship-Attribute') para expressões $Sets$ em que ocorrem variados invariantes 'ad hoc'.

Nos diagramas ERA que se seguem, E, F são símbolos que designam entidades, $\#E$, $\#F$ designam os respectivos atributos chave e A, B, C, D, G, $\cdots$, Z designam atributos. Cada regra de inversão é da forma

$$d = e$$

onde d é um diagrama ERA e e é a correspondente “tradução semântica” em $Sets$.

Entidades



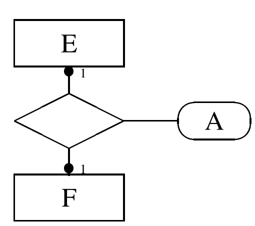
$$= \quad \#E \rightharpoonup A \times B \times \cdots \times Z \tag{8.150}$$

Relacionamentos

Genericamente, vamos supôr um atributo abstracto A associado a cada relacionamento, desdobrável em tantos quantos necessários em cada caso.

Relacionamentos 1:1 Várias situações a considerar conforme as entidades envolvidas (E ou F) são *opcionais* ou *obrigatórias*:

1. E e F são ambas opcionais:



$$= \quad \begin{array}{l} ((\#F \rightharpoonup \#E \times A)\times \\ (\#E \rightharpoonup \cdots)\times \\ (\#F \rightharpoonup \cdots))_\phi \end{array} \tag{8.151}$$

Invariante associado:

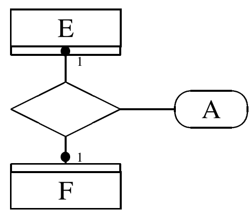
$$\begin{array}{l} \phi(\rho,\sigma,\sigma') \stackrel{\mathrm{def}}{=} \\ \qquad \forall x, x' \in dom(\rho) : x \neq x' \Rightarrow \pi_1(\rho(x)) \neq \pi_1(\rho(x'))\wedge \\ \qquad dom(\rho) \subseteq dom(\sigma')\wedge \\ \qquad \pi_1[rng(\rho)] \subseteq dom(\sigma) \end{array}$$

Se atributo A ausente ($A = 1$):

$$\begin{array}{rcl} \#F \rightharpoonup \#E \times A & \cong & F \rightharpoonup \#E \times 1 \\ & \cong & \#F \rightharpoonup \#E \end{array}$$

cf. lei (8.24).

2. E obrigatória e F opcional:

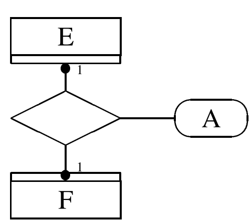


$$= \begin{array}{l}((\#F \rightharpoonup \#E \times A)\times \\ (\#E \rightharpoonup \cdots)\times \\ (\#F \rightharpoonup \cdots))_{\phi}\end{array} \quad (8.152)$$

Invariante associado:

$$\begin{array}{l}\phi(\rho,\sigma,\sigma') \stackrel{\text{def}}{=} \\ \qquad \forall x, x' \in dom(\rho) : x \neq x' \Rightarrow \pi_1(\rho(x)) \neq \pi_1(\rho(x'))\wedge \\ \qquad dom(\rho) \subseteq dom(\sigma')\wedge \\ \qquad \pi_1[rng(\rho)] = dom(\sigma)\end{array}$$

3. F obrigatória e E opcional:

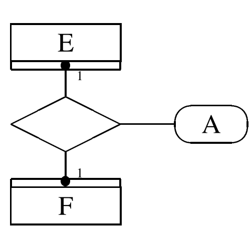


$$= \begin{array}{l}((\#F \rightharpoonup \#E \times A)\times \\ (\#E \rightharpoonup \cdots)\times \\ (\#F \rightharpoonup \cdots))_{\phi}\end{array} \quad (8.153)$$

Invariante associado:

$$\begin{array}{l}\phi(\rho,\sigma,\sigma') \stackrel{\text{def}}{=} \\ \qquad \forall x, x' \in dom(\rho) : x \neq x' \Rightarrow \pi_1(\rho(x)) \neq \pi_1(\rho(x'))\wedge \\ \qquad dom(\rho) = dom(\sigma')\wedge \\ \qquad \pi_1[rng(\rho)] \subseteq dom(\sigma)\end{array}$$

4. E e F ambas obrigatórias:



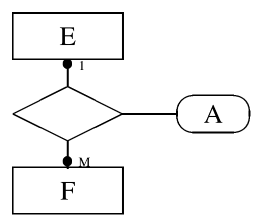
$$= \begin{array}{l}((\#F \rightharpoonup \#E \times A)\times \\ (\#E \rightharpoonup \cdots)\times \\ (\#F \rightharpoonup \cdots))_{\phi}\end{array} \quad (8.154)$$

Invariante associado:

$$
\begin{aligned}
&\phi(\rho, \sigma, \sigma') \stackrel{\text{def}}{=} \\
&\qquad \forall x, x' \in dom(\rho) : x \neq x' \Rightarrow \pi_1(\rho(x)) \neq \pi_1(\rho(x')) \wedge \\
&\qquad dom(\rho) = dom(\sigma') \wedge \\
&\qquad \pi_1[rng(\rho)] = dom(\sigma)
\end{aligned}
$$

Relacionamentos M:1 Novamente 4 regras, pela mesma ordem:

1.

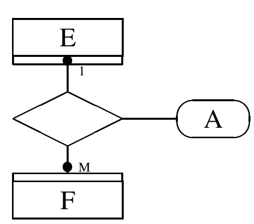


$$
\begin{aligned}
= \quad & ((\#F \rightharpoonup \#E \times A) \times \\
& (\#E \rightharpoonup \cdots) \times \\
& (\#F \rightharpoonup \cdots))_\gamma
\end{aligned}
\tag{8.155}
$$

Invariante associado:

$$
\begin{aligned}
&\gamma(\rho, \sigma, \sigma') \stackrel{\text{def}}{=} \\
&\qquad dom(\rho) \subseteq dom(\sigma') \wedge \\
&\qquad \pi_1[rng(\rho)] \subseteq dom(\sigma)
\end{aligned}
$$

2.

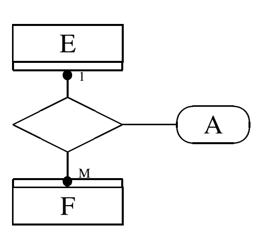


$$
\begin{aligned}
= \quad & ((\#F \rightharpoonup \#E \times A) \times \\
& (\#E \rightharpoonup \cdots) \times \\
& (\#F \rightharpoonup \cdots))_\gamma
\end{aligned}
\tag{8.156}
$$

Invariante associado:

$$
\begin{aligned}
&\gamma(\rho, \sigma, \sigma') \stackrel{\text{def}}{=} \\
&\qquad dom(\rho) \subseteq dom(\sigma') \wedge \\
&\qquad \pi_1[rng(\rho)] = dom(\sigma)
\end{aligned}
$$

3.

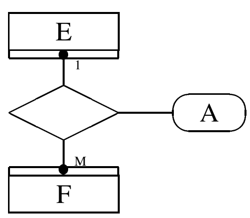


$$
\begin{aligned}
= \quad & ((\#F \rightharpoonup \#E \times A) \times \\
& (\#E \rightharpoonup \cdots) \times \\
& (\#F \rightharpoonup \cdots))_\gamma
\end{aligned}
\tag{8.157}
$$

Invariante associado:

$$\begin{array}{l} \gamma(\rho,\sigma,\sigma') \stackrel{\mathrm{def}}{=} \\ \qquad dom(\rho) = dom(\sigma')\wedge \\ \qquad \pi_1[rng(\rho)] \subseteq dom(\sigma) \end{array}$$

4.



$$= \quad \begin{array}{l} ((\#F \rightharpoonup \#E \times A)\times \\ (\#E \rightharpoonup \cdots)\times \\ (\#F \rightharpoonup \cdots))_\gamma \end{array} \tag{8.158}$$

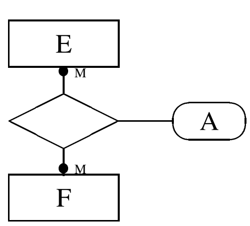
Invariante associado:

$$\begin{array}{l} \gamma(\rho,\sigma,\sigma') \stackrel{\mathrm{def}}{=} \\ \qquad dom(\rho) = dom(\sigma')\wedge \\ \qquad \pi_1[rng(\rho)] = dom(\sigma) \end{array}$$

— NB: Os relacionamentos $1:1$ são casos particulares dos relacionamentos $M:1$, apenas impondo um requisito de injectividade.

Relacionamentos M:M (pela mesma ordem):

1.



$$= \quad \begin{array}{l} ((\#E \times \#F \rightharpoonup A)\times \\ (\#E \rightharpoonup \cdots)\times \\ (\#F \rightharpoonup \cdots))_\varphi \end{array} \tag{8.159}$$

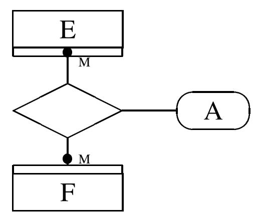
Invariante associado:

$$\varphi(\rho,\sigma,\sigma') \quad \stackrel{\mathrm{def}}{=} \quad \begin{array}{l} \pi_1[dom(\rho)] \subseteq dom(\sigma)\wedge \\ \pi_2[dom(\rho)] \subseteq dom(\sigma') \end{array}$$

De notar que, quando $A = 1$ (*i.é* relacionamento sem atributos) se tem

$$\begin{array}{rcl} \#E \times \#F \rightharpoonup 1 & \cong & (1+1)^{\#E\times\#F} \\ & \cong & 2^{\#E\times\#F} \end{array}$$

2.

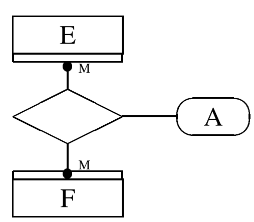


$$= \begin{array}{l} ((\#E \times \#F \rightharpoonup A)\times \\ (\#E \rightharpoonup \cdots)\times \\ (\#F \rightharpoonup \cdots))_{\varphi} \end{array} \tag{8.160}$$

Invariante associado:

$$\begin{array}{rcl} \varphi(\rho,\sigma,\sigma') & \stackrel{\mathrm{def}}{=} & \pi_1[dom(\rho)] = dom(\sigma)\wedge \\ & & \pi_2[dom(\rho)] \subseteq dom(\sigma') \end{array}$$

3.



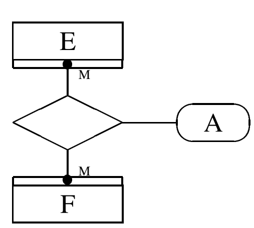
$$= \begin{array}{l} ((\#E \times \#F \rightharpoonup A)\times \\ (\#E \rightharpoonup \cdots)\times \\ (\#F \rightharpoonup \cdots))_{\varphi} \end{array} \tag{8.161}$$

Invariante associado:

$$\begin{array}{rcl} \varphi(\rho,\sigma,\sigma') & \stackrel{\mathrm{def}}{=} & \pi_1[dom(\rho)] \subseteq dom(\sigma)\wedge \\ & & \pi_2[dom(\rho)] = dom(\sigma') \end{array}$$

(observe-se que o diagrama desta regra é simétrico em relação ao diagrama da regra anterior)

4.



$$= \begin{array}{l} ((\#E \times \#F \rightharpoonup A)\times \\ (\#E \rightharpoonup \cdots)\times \\ (\#F \rightharpoonup \cdots))_{\varphi} \end{array} \tag{8.162}$$

Invariante associado:

$$\begin{array}{rcl} \varphi(\rho,\sigma,\sigma') & \stackrel{\mathrm{def}}{=} & \pi_1[dom(\rho)] = dom(\sigma)\wedge \\ & & \pi_2[dom(\rho)] = dom(\sigma') \end{array}$$

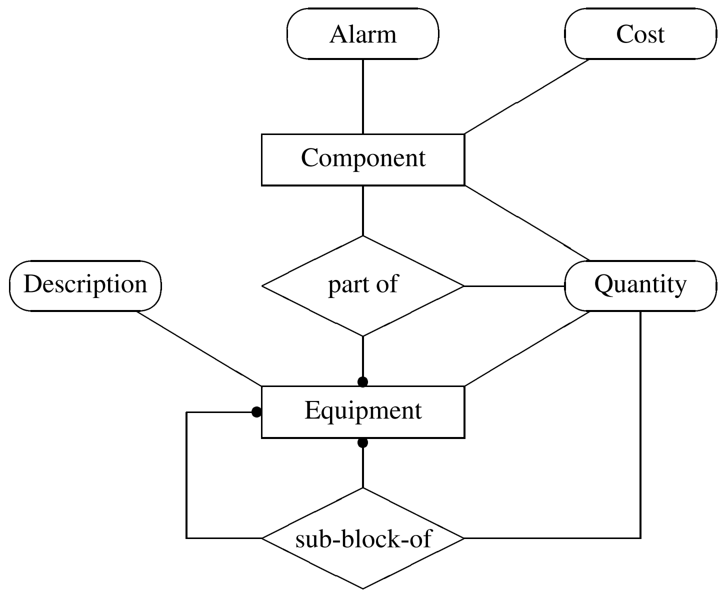


Figura 8.3: Exemplo de um diagrama ERA

8.9.1 Exemplo de Aplicação

Na figura 8.3 mostra-se um diagram-ERA que pretende registar a estrutura da base de dados de produção de uma fábrica de determinado tipo de equipamentos (relembrar os exercícios 1.3, 2.59 e 8.19). Este diagrama pretende registar as seguintes intuições sobre o problema:

- a fábrica constrói um determinado tipo de *equipmento* cuja produção envolve *componentes* individuais obtidos externamente (*e.g.* comprados a um fornecedor);

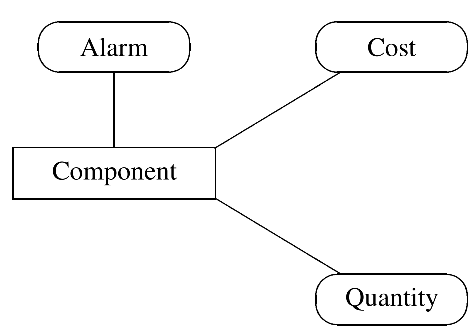
- cada componente individual tem um *custo* e está armazenado em determinada *quantidade*, verificada em relação a um dado valor mínimo de *alarme*;

- cada componente é, segundo uma quantidade determinada, *parte de* pelo menos um *equipmento* (*e.g.* o circuito ref. `X` tem n circuitos integrados de ref. `Y`);

- os equipmentos podem conter, segundo uma quantidade determinada, outros equipmentos como *sub-blocos* (*e.g.* o computador pessoal ref. `Z` tem m ‘PC boards’ de ref. `T`).

Em resumo, é possível reconstituir a árvore de produção de cada equipmento, árvore essa que pode envolver componentes individuais e/ou outras (sub-)árvores de produção.

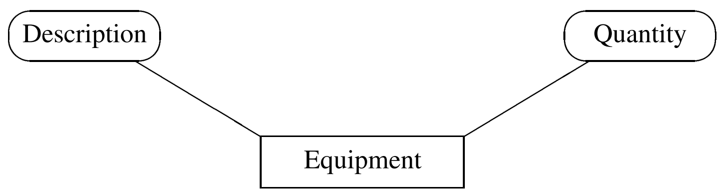
A tradução deste diagrama para SETS pode ser obtida de acordo com as regras acima descritas. O diagrama envolve apenas duas entidades, $Component$ e $Equipment$. A partir da regra (8.150) obtém-se:



— *i.é*, em SETS:

$$\#Component \rightharpoonup Alarm \times Cost \times Quantity$$

— e



— *i.é*, em SETS:

$$\#Equipment \rightharpoonup Description \times Quantity$$

O relacionamento *parte de* (‘part of’) pode ser tido em conta adicionando, via regra (8.160), uma função parcial finita extra:

$$\begin{array}{l}(\#Component \times \#Equipment \rightharpoonup Quantity) \times \\ (\#Component \rightharpoonup Alarm \times Cost \times Quantity) \times \\ (\#Equipment \rightharpoonup Description \times Quantity)\end{array}$$

Finalmente, o relacionamento *sub-bloco-de* (‘sub-block-of’) é incorporado segundo a regra (8.162):

$$\begin{array}{ll}(& (\#Equipment \times \#Equipment \rightharpoonup Quantity)\times \\ & (\#Component \times \#Equipment \rightharpoonup Quantity)\times \\ & (\#Component \rightharpoonup Alarm \times Cost \times Quantity)\times \\ & (\#Equipment \rightharpoonup Description \times Quantity) \\ )_\Phi & \end{array} \tag{8.163}$$

cujo invariante Φ é induzido pelas regras acima empregues:

$$\begin{array}{rcc}\Phi(r_1, r_2, e_1, e_2) & \stackrel{\text{def}}{=} & \pi_1[dom(r_1)] \subseteq dom(e_1) \\ & & \wedge \\ & & \pi_2[dom(r_1)] \subseteq dom(e_1) \\ & & \wedge \\ & & \pi_1[dom(r_2)] = dom(e_2) \\ & & \wedge \\ & & \pi_2[dom(r_2)] \subseteq dom(e_1)\end{array}$$

Independentemente deste invariante, pelas leis (8.63), (8.29) e (8.22) a expressão (8.163) pode ser transformada em

$$\begin{array}{l}((\#Equipment \times (\#Equipment + \#Component)) \rightharpoonup Quantity)\times \\ (\#Equipment \rightharpoonup Description \times Quantity)\times \\ (\#Component \rightharpoonup Alarm \times Cost \times Quantity)\end{array}$$

O mesmo invariante pode ser parcialmente aliviado via lei (8.72), finalmente conduzindo a

$$
\begin{array}{lll}
(& (\#Equipment \rightharpoonup & Description \times Quantity \times \\
 & & ((\#Equipment + \#Component) \rightharpoonup Quantity) \quad) \times \\
 & (\#Component \rightharpoonup Alarm \times Cost \times Quantity) & \\
)_\varphi & &
\end{array}
\tag{8.164}
$$

O invariante que resta é

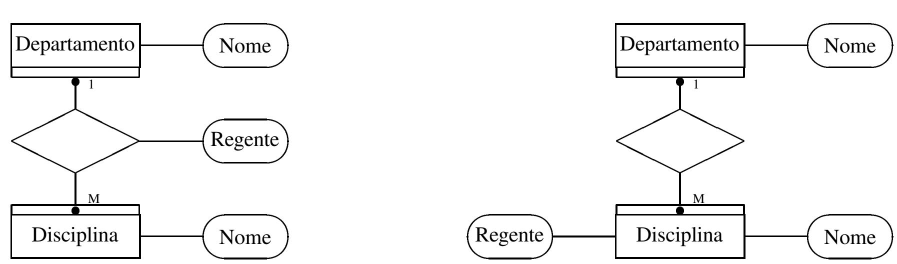
$$
\begin{array}{rcll}
\varphi(e_1, e_2) & \stackrel{\mathrm{def}}{=} & let & X = \bigcup_{k \in dom(e_1)} dom(\pi_3(e_1(k))) \\
 & & & E = \{x \in \#Equipment \mid \langle 1, x\rangle \in X\} \\
 & & & C = \{x \in \#Component \mid \langle 2, x\rangle \in X\} \\
 & & in & dom(e_2) = C \wedge E \subseteq dom(e_1)
\end{array}
\tag{8.165}
$$

Note-se que o “significado” assim obtido para o diagrama — vertido em SETS por (8.164) e (8.165) — é, em termos de especificação, relativamente pobre comparado com aquilo que o especificador provavelmente tinha na sua mente mas não teve meios para registar no diagrama. Por exemplo, onde é que em (8.165) vemos a garantia de que *nenhum equipmento pode recursivamente ser um sub-bloco de si próprio*? Será necessário fixar um invariante para o efeito, invariante esse que a notação ERA é incapaz só por si de explicitar. Em suma, um diagrama-ERA poderá ser sem dúvida um bom ponto de partida para arrancar com uma especificação formal de um domínio de dados. Mas na maioria dos casos será necessário acrescentar informação extra que o diagrama foi incapaz de registar.

Exercício 8.30 Perante o seguinte fragmento da formulação dos requisitos de uma aplicação de gestão pedagógica,

> *No sistema académico de uma dada universidade toda a disciplina é fornecida por um e um só departamento, que a entrega a um seu docente responsável (regente). Departamentos e disciplinas são entidades caracterizadas, entre outros atributos, pelo seu nome.*

dois programadores discutem sobre qual dos seguintes diagramas de Entidades-Relações (a) e (b), que se seguem, devem adoptar:



(a) (b)

1. Dê-lhes a sua ajuda, comparando (a) e (b) com base na semântica em SETS que estudou para diagramas deste tipo.
2. Escreva a função que, em notação SETS, converte a informação relacional de um dos formatos (a) ou (b) para o outro, se é que tal função pode ser definida.

□

8.10 Exercícios

Exercício 8.31 Demonstre ou refute os seguintes factos, em $Sets$:

$$A \times A^\star \trianglelefteq A^\star \tag{8.166}$$
$$A \rightharpoonup (B \times 2^B) \trianglelefteq 2^{A \times 2 \times B} \tag{8.167}$$
$$(A \times B \rightharpoonup C) \cong A \rightharpoonup (B \times C) \tag{8.168}$$
$$(A^\star)^B \cong B \rightharpoonup A^+ \tag{8.169}$$
$$(A \rightharpoonup B) \times (B \rightharpoonup C) \cong A \rightharpoonup C \tag{8.170}$$

para A, B e C quaisquer e onde

$$A^+ \stackrel{\mathrm{def}}{=} \bigcup_{n>0} A^n \tag{8.171}$$

designa o conjunto de todas as sequências não-vazias de elementos de A.

□

Exercício 8.32 A lei

$$A \rightharpoonup (B \times C) \quad \trianglelefteq_f^{fdpodom} \quad (A \times B) \rightharpoonup C \tag{8.172}$$

generaliza uma das leis básicas do cálculo $Sets$ que estudou nesta disciplina.

1. Identifique essa lei e mostre que ela é de facto uma particularização de (8.172).
2. Defina a função de abstracção $f_{(8.172)}$.

□

Exercício 8.33 Indique 4 leis $A \trianglelefteq_f^\phi B$ do cálculo SETS tais que a função de *abstracção* f, apesar de parcial, é injectiva. Para cada uma delas, escreva a definição da função de *representação* correspondente, *i.é*, da inversa f^{-1}.

□

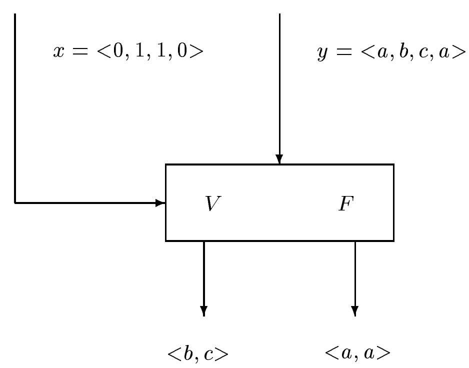
Exercício 8.34 No contexto do Exercício 2.50, repare que o facto

$$(A^\star)^2 \quad \trianglelefteq_f^\phi \quad 2^\star \times A^\star$$

se verifica, em SETS, para mais do que uma possibilidade de f e ϕ.

Proponha-os por forma a que $f(\langle x, y\rangle)$ funcione da seguinte forma, ilustrada pelo diagrama abaixo: cada booleano em x actua como filtro, separando a sequência y nas duas sub-sequências que são dadas como resultado.

Exemplo, para $x = <0, 1, 1, 0>$ e $y = <a, b, c, a>$:



Será a função de abstracção que acaba de propôr injectiva? Justifique.

□

Exercício 8.35 Será que o seguinte facto

$$A^\star \times (A \rightharpoonup B) \trianglelefteq_f^\phi (A \times B)^\star$$

se verifica em SETS? Na afirmativa proponha f e ϕ. Na negativa, apresente um contra-exemplo.

□

Exercício 8.36 Numa clínica trabalham vários médicos ($Doctor$). As consultas são marcadas antecipadamente, registando-se, por ordem de chegada, qual o doente ($Patient$) e qual o médico que o vai ver.

Pretendendo a clínica um sistema de informação para a gestão de e consultas e suas marcações, duas empresas de 'software' diferentes concorreram ao projecto, propondo dois modelos alternativos que diferem logo na estrutura da base de dados de suporte:

$$\begin{array}{rcl} DBase1 & \cong & (Patient \times Doctor)^\star \\ DBase2 & \cong & Doctor \rightharpoonup Patient^+ \end{array}$$

NB: interprete $Patient^+$ de acordo com (8.171) acima.

1. Explique por palavras suas qual lhe parece ser a principal diferença de 'design' entre $DBase1$ e $DBase2$.
2. Serão estes modelos igualmente representativos? Na negativa, qual deles implementa o outro? Justifique a sua resposta.

3. Especifique a operação seguinte:

$$consulta \quad : \quad Doctor \times DBase \to DBase$$

$$consulta(d, \sigma) \stackrel{\text{def}}{=} \ldots\ldots$$ */* o médico d consulta a base σ e, caso ainda tenha doentes seus para ver, vê o que está há mais tempo à espera, devolvendo a base já sem esse doente registado */*

   para as duas situações $DBase = DBase1$ e $DBase = DBase2$.

□

Exercício 8.37 Relembre do exercício 8.22 o seguinte modelo para filas com prioridade,

$$PQueue(A, B) \quad \cong \quad A \rightharpoonup B^{\star}$$

sujeito ao invariante

$$\phi(\sigma) \stackrel{\text{def}}{=} <> \notin rng(\sigma)$$

onde B é o domínio de objectos a enfileirar e A é um domínio de prioridades sobre o qual se supõe definida uma ordem total.

1. Calcule uma codificação na linguagem C da estrutura $PQueue$. Justifique o seu cálculo indicando as leis do cálculo SETS a que recorreu.
2. Calcule a respectiva função de abstracção.

Sugestão: relembre o exercício 8.7.

□

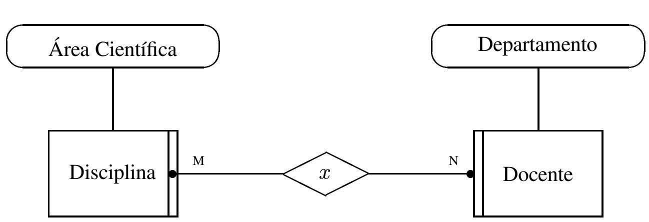
Exercício 8.38 Seja A um qualquer tipo de dados não vazio. A ideia comum em programação de que qualquer objecto $a \in A$ é representável por um apontador para esse objecto é transmitida pelo facto seguinte:

$$A \trianglelefteq_{f}^{\phi} 1 + A$$

Caracterize formalmente f e ϕ.

□

Exercício 8.39 Considere o seguinte diagrama E-R-A ("Entidades-Relacionamentos & Atributos") referente a um sistema de informação académico:



Calcule a semântica formal em $Sets$ deste diagrama, incluindo o respectivo invariante implícito. De acordo o modelo obtido, sugira um nome ou uma interpretação informal para o relacionamento x.
□

Exercício 8.40 Pretende-se especificar e depois implementar um sistema de informação tipo 'World Wide Web' que relacione páginas de informação ($Page$) entre si sob a forma de uma rede semântica, isto é, capaz de exprimir *ligações* ($LnkId$) arbitrárias entre os endereços dessas páginas (Id).

Uma das seguintes duas estruturas em SETS especifica esse sistema e a outra é uma sua implementação.

$$X = (Id \rightharpoonup Page) \times 2^{Id \times LnkId \times Id} \tag{8.173}$$

$$Y = Id \rightharpoonup (Page \times 2^{LnkId \times Id}) \tag{8.174}$$

Qual? Justifique e calcule o invariante que afecta a que é implementação.
□

8.11 Notas Bibliográficas

A maior parte deste capítulo é a tradução e fusão de partes dos artigos [Oli90], [Oli92] e relatórios técnicos [Oli94] e [OC93]. A "inversão" ERA-$Sets$ foi pela primeira vez apresentada em [Oli93] e posteriormente investigada em [Rod93] e [OC93].

Capítulo 9

Reificação das Operações

Uma vez obtidas, segundo o processo de cálculo que se estudou no capítulo anterior, as funções de abstracção envovidas no refinamento dos domínios de dados manipulados por uma dada operação de uma especificação, *e.g.* $\sigma : A \longrightarrow B$, é altura de nos procuparmos com o refinamento algorítmico da operação propriamente dita.

Relembremos da introdução que fizemos no capítulo 7 os ingredientes do refinamento no *eixo algorítmico*:

$$\left\{\begin{array}{l}\text{simulação (= obtenção de computabilidade)}\\ \text{concretização algorítmica (= obtenção de eficiência)}\end{array}\right.$$

Quanto a simulações, relembremos o diagrama comutativo

$$\begin{tikzcd} A \arrow[r, "\sigma"] & B \\ A_1 \arrow[u, "f"] \arrow[r, "\sigma_1"'] & B_1 \arrow[u, "g"'] \end{tikzcd}$$

de acordo com o qual se pretende calcular uma simulação σ_1 de σ:

$$g(\sigma_1(a_1)) = \sigma(f(a_1)) \tag{9.1}$$

Como então dissemos, a técnica clássica consiste em, indutivamente, conjecturar σ_1 e depois provar matematicamente que σ_1 satisfaz (9.1).

Alternativamente, a ideia que está subjacente ao *cálculo de simulações* que aqui desenvolveremos é encarar (9.1) como uma equação que tentaremos "resolver em ordem a σ_1". Na verdade, qualquer σ_1 que satisfaça (9.1) será aceitável como simulação. Se g fôr injectiva, existe g^{-1} e teremos que

$$g^{-1}(g(\sigma_1(a_1))) = g^{-1}(\sigma(f(a_1)))$$

i.é

$$\sigma_1(a_1) = g^{-1}(\sigma(f(a_1)))$$

Neste caso há uma única solução para a equação de simulação, bastando calcular e simplificar $g^{-1}(\sigma(f(a_1)))$:

$$\begin{array}{rcl}
\sigma_1(a_1)) & = & g^{-1}(\sigma(f(a_1))) \\
& = & \ldots \\
& \vdots & \\
& = & \ldots e \ldots \quad \textit{/*expressão livre de } f \textit{ e de } g \textit{ */}
\end{array}$$

Se g não fôr injectiva, pode haver mais do que uma solução, conforme processo de cálculo,

$$\begin{array}{rcl}
g(\sigma_1(a_1)) & = & \sigma(f(a_1)) \\
& = & \ldots \\
& \vdots & \\
& = & g(\ldots e \ldots)
\end{array}$$

obtida "eliminando g" de ambos os lados da equação final:

$$\sigma_1(a_1) \quad = \quad \ldots e \ldots$$

Isto significa que nada impede que, se existir um processo de cálculo alternativo tal que se chegue a

$$g(\sigma_1(a_1)) \quad = \quad g(\ldots e' \ldots)$$

então

$$\sigma_1(a_1) \quad = \quad \ldots e' \ldots$$

será outra solução possível para refinamento de σ.

Quanto à concretização algorítmica, trata-se de prolongar o processo de cálculo até se obter a semântica de um fragmento de código executável:

$$\begin{array}{rcl} & \vdots & \\ \sigma_1(a_1) & = & \ldots e \ldots \\ & = & \ldots \\ & \vdots & \\ & = & [\![P]\!] \end{array}$$

sendo P o executável final.

Que cálculo utilizaremos agora nestes raciocínios? Das alternativas possíveis [1] escolhemos o chamado *cálculo 'Fold/Unfold'* dos anos 70/80, por ser (talvez) o mais simples. Esse cálculo é sumariamente exposto no apêndice B.

Apresentar-se-ão neste capítulo alguns cálculos de simulações de operações sobre domínios de dados já refinados anteriormente. Apresenta-se depois, ainda que sem a suficiente introdução, um breve estudo de uma classe de processos de refinamento típicos da concretização no eixo algorítmico — a eliminação da "falsa" recursividade, *i.é* a geração de ciclos `while` ou `for` a partir de simulações (aparentemente) recursivas. O apêndice B deverá ser consultado à medida que se fôr progredindo na exemplificação.

9.1 Cálculo de Simulações Simples

Entendemos por simulação simples aquela em que a função de abstracção g da equação de refinamento (9.1) é uma bijecção, o que garante, como vimos, a unicidade da solução. A situação mais simples é a de g ser a própria indentidade.

Está nesta situação o cálculo da simulação do teste de pertença sobre sequências. Relembremos o Exercício 7.2 em que foi conjecturada uma definição recursiva para o operador $belongs$, que depois se provou que simulava o teste de pertença ($\in$) de conjuntos ao nível de sequências.

Pretendemos agora fazer o cálculo desse operador sem recurso a quaisquer demonstrações, através de transformações da equação definicional implícita no diagrama comutativo (7.7),

$$belongs(x, y) = x \in elems(y) \tag{9.2}$$

e da definição da correspondente função de abstracção:

$$elems(x) \stackrel{\text{def}}{=} \left\{ \begin{array}{lcl} x = < > & \Rightarrow & \emptyset \\ x \neq < > & \Rightarrow & \{head(x)\} \cup elems(tail(x)) \end{array} \right. \tag{9.3}$$

[1] Ver secção final deste capítulo.

cf. Exercício 7.1.

Teremos, por substituição [2]:

$$belongs(x,y) = x \in \left\{ \begin{array}{lll} y = <> & \Rightarrow & \emptyset \\ y \neq <> & \Rightarrow & \{head(y)\} \cup elems(tail(y)) \end{array} \right.$$

de onde, aplicando (B.1), obtemos

$$belongs(x,y) = \left\{ \begin{array}{lll} y = <> & \Rightarrow & x \in \emptyset \\ y \neq <> & \Rightarrow & x \in (\{head(y)\} \cup elems(tail(y))) \end{array} \right.$$

o que vem a dar, feitas algumas simplificações básicas,

$$belongs(x,y) = \left\{ \begin{array}{lll} y = <> & \Rightarrow & F \\ y \neq <> & \Rightarrow & x = head(y) \vee x \in elems(tail(y)) \end{array} \right. \tag{9.4}$$

Falta-nos apenas "eliminar" $elems$ na definição acima. Ora, se

$$belongs(x,y) = x \in elems(y)$$

então, substituindo y por $tail(y)$,

$$x \in elems(tail(y)) = belongs(x, tail(y))$$

que se pode por sua vez substituir em (9.4) obtendo-se

$$belongs(x,y) = \left\{ \begin{array}{lll} y = <> & \Rightarrow & F \\ y \neq <> & \Rightarrow & x = head(y) \vee belongs(x, tail(y)) \end{array} \right.$$

Finalmente, por (B.6), teremos

$$belongs(x,y) = \tag{9.5}$$

$$\left\{ \begin{array}{lll} y = <> & \Rightarrow & F \\ y \neq <> & \Rightarrow & \left\{ \begin{array}{lll} x = head(y) & \Rightarrow & T \\ x \neq head(y) & \Rightarrow & belongs(x, tail(y)) \end{array} \right. \end{array} \right. \tag{9.6}$$

que é exactamente (7.8).

9.2 Cálculo de Simulações com mais do que uma Solução

9.2.1 Simulação da Intersecção sobre Sequências

Após os exemplos anteriores fica a ideia de que qualquer operador sobre conjuntos pode, via $elems$, ser simulado por operações sobre sequências (listas). Em boa

[2] Quer dizer, 'unfold' de (9.3) em (9.2).

verdade, uma boa parte das funções sobre listas que qualquer programador acaba por escrever para seu uso pessoal acaba por ser afinal, uma biblioteca de funções de conjuntos implementados por listas.

Vejamos mais um exemplo que é aparentemente simples: pretendemos calcular a simulação int sobre listas da intersecção de conjuntos:

$$\begin{tikzcd}
2^X \times 2^X \arrow[r, "\cap"] & 2^X \\
X^\star \times X^\star \arrow[u, "elems"'] \arrow[r, "int"'] & X^\star \arrow[u, "elems"']
\end{tikzcd}$$

Teremos, sucessivamente,

$$
\begin{array}{l}
elems(int(l_1, l_2)) = elems(l_1) \cap elems(l_2) \\
\quad = \; (\left\{ \begin{array}{rcl} l_1 = <> & \Rightarrow & \{\} \\ \neg(l_1 = <>) & \Rightarrow & \{head(l_1)\} \cup elems(tail(l_1)) \end{array} \right.) \cap elems(l_2) \\
\quad = \; \left\{ \begin{array}{rcl} l_1 = <> & \Rightarrow & \{\} \\ \neg(l_1 = <>) & \Rightarrow & (\{head(l_1)\} \cup elems(tail(l_1))) \cap elems(l_2) \end{array} \right. \\
\quad = \; \left\{ \begin{array}{rcl} l_1 = <> & \Rightarrow & \{\} \\ \neg(l_1 = <>) & \Rightarrow & \underbrace{\underbrace{\{head(l_1)\} \cap elems(l_2)}_{A} \cup elems(int(tail(l_1), l_2))}_{B} \end{array} \right.
\end{array}
$$

Para se trabalharem agora as sub-expressões A e B convem anotar as seguintes propriedades básicas da teoria dos conjuntos:

$$\{a\} \cap X \; = \; \left\{ \begin{array}{rcl} a \in X & \Rightarrow & \{a\} \\ \neg(a \in X) & \Rightarrow & \emptyset \end{array} \right. \tag{9.7}$$

$$elems(x \frown y) \; = \; elems(x) \cup elems(y) \tag{9.8}$$

É imediato aproveitar (9.7) e (9.2) em A:

$$
\begin{array}{rcl}
A & = & \left\{ \begin{array}{rcl} head(l_1) \in elems(l_2) & \Rightarrow & \{head(l_1)\} \\ \neg(head(l_1) \in elems(l_2)) & \Rightarrow & \{\} \end{array} \right. \\
 & = & \left\{ \begin{array}{rcl} belongs(head(l_1), l_2) & \Rightarrow & \{head(l_1)\} \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & \{\} \end{array} \right.
\end{array}
$$

Substituição de A em B:

$$B \; = \; (\left\{ \begin{array}{rcl} belongs(head(l_1), l_2) & \Rightarrow & \{head(l_1)\} \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & \{\} \end{array} \right.) \cup elems(int(tail(l_1), l_2))$$

$$= \left\{ \begin{array}{lcl} belongs(head(l_1), l_2) & \Rightarrow & \underbrace{\{head(l_1)\} \cup elems(int(tail(l_1), l_2))}_{C} \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & elems(int(tail(l_1), l_2)) \end{array} \right.$$

onde, por (9.8):

$$C = elems(<head(l_1)> \frown int(tail(l_1), l_2))$$

Substituindo agora C em B:

$$\begin{array}{lcl} B & = & \left\{ \begin{array}{lcl} belongs(head(l_1), l_2) & \Rightarrow & elems(<head(l_1)> \frown int(tail(l_1), l_2)) \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & elems(int(tail(l_1), l_2)) \end{array} \right. \\ & = & elems(\left\{ \begin{array}{lcl} belongs(head(l_1), l_2) & \Rightarrow & <head(l_1)> \frown int(tail(l_1), l_2) \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & int(tail(l_1), l_2) \end{array} \right.) \end{array}$$

podemos finalmente substituir B na expressão de onde partimos, obtendo:

$$elems(int(l_1, l_2)) =$$

$$\left\{ \begin{array}{lcl} l_1 = <> & \Rightarrow & \{\} \\ \neg(l_1 = <>) & \Rightarrow & elems(\left\{ \begin{array}{lcl} belongs(head(l_1), l_2) & \Rightarrow & <head(l_1)> \frown int(tail(l_1), l_2) \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & int(tail(l_1), l_2) \end{array} \right.) \end{array} \right.$$

$$= elems(\underbrace{\left\{ \begin{array}{lcl} l_1 = <> & \Rightarrow & <> \\ \neg(l_1 = <>) & \Rightarrow & \left\{ \begin{array}{lcl} belongs(head(l_1), l_2) & \Rightarrow & <head(l_1)> \frown int(tail(l_1), l_2) \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & int(tail(l_1), l_2) \end{array} \right. \end{array} \right.)}_{D}$$

Chegamos assim a uma igualdade da forma

$$elems(int(l_1, l_2)) = elems(D)$$

sendo indistingíveis, quando abstraídas por $elems$, as expressões $int(l_1, l_2)$ e D.
Podemos pois "eliminar $elems$" de ambos os lados da equação e obter

$$int(l_1, l_2) \stackrel{\text{def}}{=}$$

$$\left\{ \begin{array}{lcl} l_1 = <> & \Rightarrow & <> \\ \neg(l_1 = <>) & \Rightarrow & \left\{ \begin{array}{lcl} belongs(head(l_1), l_2) & \Rightarrow & <head(l_1)> \frown int(tail(l_1), l_2) \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & int(tail(l_1), l_2))) \end{array} \right. \end{array} \right.$$

ou ainda,

$$int(l_1, l_2) \stackrel{\text{def}}{=}$$

$$\left\{ \begin{array}{lcl} l_1 = <> & \Rightarrow & <> \\ \neg(l_1 = <>) & \Rightarrow & let \quad x = \left\{ \begin{array}{lcl} belongs(head(l_1), l_2) & \Rightarrow & <head(l_1)> \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & <> \end{array} \right. \\ & & in \quad x \frown int(tail(l_1), l_2) \end{array} \right. \tag{9.9}$$

Como $elems$ não é um operador injectivo, a solução que obtivemos não é única. De facto, nada impede que outra variante do processo de cálculo conduza a outra solução, por exemplo uma mais sofisticada que filtre ocorrências repetidas do mesmo elemento de l_1. Finalmente, é óbvio que a decisão de fazer o 'unfold' inicial de $elems(l_1)$ em lugar do de $elems(l_2)$ foi arbitrária.

Exercício 9.1 Acaba de ser estudado um processo de cálculo da simulação, sobre listas, da operação de intersecção de dois conjuntos.

Adapte esse raciocínio ao cálculo de uma simulação da reunião de conjuntos representados por listas,

$$\begin{tikzcd}
2^X & \times & 2^X \arrow[rr, "\cup"] & & 2^X \\
X^\star \arrow[u, "elems"] & \times & X^\star \arrow[u, "elems"'] \arrow[rr, "x"'] & & X^\star \arrow[u, "elems"']
\end{tikzcd}$$

que garanta a propriedade de não repetição de elementos.

□

Exercício 9.2 No contexto do refinamento "clássico" de conjuntos para sequências (listas),

$$2^A \preceq A^*$$

que é estabelecido pela função *elems*, construa o processo de cálculo do seguinte operador sobre listas que implementa o cálculo do cardinal de um conjunto:

$$comp(l) \quad = \quad \left\{ \begin{array}{lll} l = <> & \Rightarrow & 0 \\ l \neq <> & \Rightarrow & \begin{array}{ll} let & h = head(l) \\ & t = tail(l) \\ in & comp(t) + \left\{ \begin{array}{lll} belongs(h,t) & \Rightarrow & 0 \\ \neg belongs(h,t) & \Rightarrow & 1 \end{array} \right. \end{array} \end{array} \right. \tag{9.10}$$

onde $belongs$ é o operador que implementa $\in$ ao mesmo nível.

□

Exercício 9.3 No mesmo contexto que o exercício anterior, suponha agora que se pretende calcular o operador sobre sequências que implementa a diferença de conjuntos, a designar por *dif*, partindo da equação

$$elems(dif(l,r))) = elems(l) - elems(r)$$

que resulta do respectivo diagrama de refinamento. Suponha ainda que alguém deu já os seguintes quatro passos desse cálculo:

$$\begin{array}{ll}
\multicolumn{2}{l}{elems(dif(l,r)))} \\
= & elems(l) - elems(r) \\
= & (\left\{ \begin{array}{lll} l = <> & \Rightarrow & \emptyset \\ l \neq <> & \Rightarrow & \{head(l)\} \cup elems(tail(l)) \end{array} \right.) - elems(r)
\end{array}$$

$$
\begin{array}{ll}
= & \left\{ \begin{array}{lll} l = <> & \Rightarrow & \emptyset \\ l \neq <> & \Rightarrow & (\{head(l)\} \cup elems(tail(l))) - elems(r) \end{array} \right. \\
= & \left\{ \begin{array}{lll} l = <> & \Rightarrow & \emptyset \\ l \neq <> & \Rightarrow & (\{head(l)\} - elems(r)) \cup (elems(tail(l)) - elems(r)) \end{array} \right. \\
= & \left\{ \begin{array}{lll} l = <> & \Rightarrow & \emptyset \\ l \neq <> & \Rightarrow & (\{head(l)\} - elems(r)) \cup elems(dif(tail(l), r)) \end{array} \right. \\
= & \cdots
\end{array}
$$

1. Justifique (ou refute) os passos que já foram dados.
2. Complete o processo de cálculo.

□

Exercício 9.4 Calcule as simulações dos operadores sobre listas — $elems$, $length$, $tail$, $head$ — induzidas pelos refinamentos estudados no Exercício 8.20.

□

9.2.2 Simulações sobre o Modelo Relacional da Informação

Vejamos agora dois exemplos de simulação de operações sobre o modelo relacional da informação, referentes a dois problemas cuja reificação dos dados já foi feita anteriormente.

Simulação de uma Operação de Inserção Relacional

Relembremos o modelo $BAMS$ da secção 7.5 cujo refinamento em $BAMSR$ (relacional) foi calculado na secção 8.5.1. Considere-se a seguinte operação que acrescenta um titular a uma conta:

$$
\begin{array}{l}
addAccHolder : AccNr \times AccHolder \times BAMS \longrightarrow BAMS \\
addAccHolder(k, t, \sigma) \stackrel{\text{def}}{=} \\
\left\{ \begin{array}{lll}
k \in dom(\sigma) & \Rightarrow & \begin{array}{ll} let & x = \sigma(k) \\ & t_k = \pi_1(x) \\ & q_k = \pi_2(x) \\ in & \sigma \dagger \left(\begin{array}{c} k \\ \langle t_k \cup \{t\}, q_k \rangle \end{array} \right) \end{array} \\
\neg(k \in dom(\sigma)) & \Rightarrow & \sigma
\end{array} \right.
\end{array}
\tag{9.11}
$$

Pretendemos calcular a sua simulação sobre $BAMSR$, o que corresponde a resolver em ordem a x a seguinte equação de reificação:

$$
f(x(k, t, \langle \rho, \gamma \rangle)) \quad = \quad addAccHolder(k, t, f(\langle \rho, \gamma \rangle)) \tag{9.12}
$$

onde f é a função de abstracção dada por (8.84).

Começaremos por desenvolver o lado direito desta equação, substituindo σ na definição de $addAccHolder$ por $f(\langle\rho,\gamma\rangle)$ e tentando simplificar depois. Note-se que essa substituição corresponde a re-escrever, no corpo de $addAccHolder$, as seguintes sub-expressões:

$$\begin{array}{rcl} \sigma & \longrightarrow & f(\langle\rho,\gamma\rangle) \\ k \in dom(\sigma) & \longrightarrow & k \in \pi_1[\rho] \\ \sigma(k) & \longrightarrow & \langle collect(\rho)(k), mkf(\gamma)(k)\rangle \\ t_k \cup \{t\} & \longrightarrow & collect(\rho)(k) \cup \{t\} \\ q_k & \longrightarrow & mkf(\gamma)(k) \end{array}$$

Ora reparemos que

$$collect(\rho)(k) \cup \{t\} \quad = \quad collect(\rho \cup \{\langle k,t\rangle\})(k)$$

e, consequentemente, que

$$\begin{array}{ll} \multicolumn{2}{l}{\sigma \dagger \left(\begin{array}{c} k \\ \langle t_k \cup \{t\}, q_k\rangle \end{array} \right)} \\ = & f(\langle\rho,\gamma\rangle) \dagger \left(\begin{array}{c} k \\ \langle collect(\rho \cup \{\langle k,t\rangle\})(k), mkf(\gamma)(k)\rangle \end{array} \right) \qquad (9.13) \\ = & f(\langle\rho \cup \{\langle k,t\rangle\},\gamma\rangle) \qquad (9.14) \end{array}$$

Assim, a equação (9.12) converte-se em,

$$f(x(k,t,\langle\rho,\gamma\rangle)) \quad = \quad \left\{ \begin{array}{rcl} k \in \pi_1[\rho] & \Rightarrow & f(\langle\rho \cup \{\langle k,t\rangle,\gamma\}\rangle) \\ \neg(k \in \pi_1[\rho]) & \Rightarrow & f(\langle\rho,\gamma\rangle) \end{array} \right.$$

ou, removendo os fs de ambos os seus lados e simplificando, em:

$$x(k,t,\langle\rho,\gamma\rangle) \quad = \quad \langle \left\{ \begin{array}{rcl} k \in \pi_1[\rho] & \Rightarrow & \rho \cup \{\langle k,t\rangle\} \\ \neg(k \in \pi_1[\rho]) & \Rightarrow & \rho \end{array} \right. ,\gamma\rangle$$

Quer dizer, numa codificação em, por exemplo, SQL, teremos que, após testar se k se encontra na tabela obtida executando

$$\texttt{SELECT ALL } AccNr \texttt{ FROM } \rho\texttt{;}$$

fazer executar

$$\texttt{INSERT INTO } \rho \texttt{ VALUES } (k,t)\texttt{;}$$

Note-se que — como seria de esperar — a execução desta última instrução apenas não é segura, pois violará o invariante (8.85) caso k não seja chave de uma conta já aberta e com saldo conhecido.

Simulação de uma Operação sobre Árvores de Decisão

De todas as operações sobre o tipo $DecTree$ (*cf.* secção 8.7.4) vamos seleccionar a seguinte,

$$\begin{array}{l} decide : DecTree \times Answer \to DecTree \\ decide(\sigma, a) \stackrel{\mathrm{def}}{=} \begin{array}[t]{ll} let & m = dom(\pi_2(\sigma)) \\ in & \left\{ \begin{array}{lll} \sigma & \Leftarrow & a \notin m \\ (\pi_2(\sigma))(a) & \Leftarrow & a \in m \end{array} \right. \end{array} \end{array} \tag{9.15}$$

que especifica a acção de escolher uma dada resposta a disponível no menu da raiz de uma árvore de decisão σ e selecicionar a sub-árvore correspondente (se a for uma resposta válida).

Vamos querer calcular a simulação de $decide$ ao nível 4 da reificação calculada na secção 8.7.4, isto é, $DecTree_4$ (8.120). Começamos por construir o correspondente diagrama (comutativo) de reificação:

$$\begin{array}{ccc} DecTree \times Answer & \xrightarrow{decide} & DecTree \\ {\scriptstyle f \times 1_{Answer}} \uparrow & & \uparrow {\scriptstyle f} \\ DecTree_4 \times Answer & \xrightarrow[decide_4]{} & DecTree_4 \end{array}$$

que conduz à equação

$$f(decide_4(k, \langle t, t' \rangle, a)) = decide(f(k, \langle t, t' \rangle), a) \tag{9.16}$$

onde $decide_4$ é encarada como uma "incógnita" e f é a função de abstracção (8.124). Substituindo (9.15) em (9.16) obtemos

$$\begin{array}{rcl} f(decide_4(k, \langle t, t' \rangle, a)) & = & decide(f(k, \langle t, t' \rangle), a) \\ & = & \begin{array}[t]{ll} let & m = dom(\pi_2(f(k, \langle t, t' \rangle))) \\ in & \left\{ \begin{array}{lll} f(k, \langle t, t' \rangle) & \Leftarrow & a \notin m \\ (\pi_2(f(k, \langle t, t' \rangle)))(a) & \Leftarrow & a \in m \end{array} \right. \end{array} \end{array}$$

No processo de cálculo que se segue, assumiremos as definições (1.54) e (1.55) dos habituais operadores relacionais de *projecção/ selecção*. Ora é fácil mostrar que

$$dom(\pi_2(f(k, \langle t, t' \rangle))) = proj(2, sel(\langle 1, k \rangle, t'))$$

e que

$$(\pi_2(f(k, \langle t, t' \rangle)))(a) = \begin{array}[t]{ll} let & t'' = sel(\langle 1, k \rangle, t') \\ & t''' = sel(\langle 2, a \rangle, t'') \\ & k' = the(proj(3, t''')) \\ in & f(k', \langle t, t' \rangle) \end{array}$$

Então,

$$
\begin{array}{l}
f(decide_4(k,\langle t,t'\rangle,a)) = \\
\qquad \begin{array}{ll}
let & m = proj(2, sel(\langle 1,k\rangle,t')) \\
in & f\left(\left\{ \begin{array}{lll}
\langle k,\langle t,t'\rangle\rangle & & \Leftarrow \quad a \notin m \\
let & k' = the(proj(3, sel(\langle 2,a\rangle, sel(\langle 1,k\rangle,t')))) & \Leftarrow \quad a \in m \\
in & \langle k',\langle t,t'\rangle\rangle &
\end{array}\right.\right)
\end{array}
\end{array}
\tag{9.17}
$$

A remoção de f de ambos os membros da equação (9.17) conduz-nos a

$$
\begin{array}{l}
decide_4(k,\langle t,t'\rangle,a) \stackrel{\mathrm{def}}{=} \\
\qquad \begin{array}{ll}
let & m = proj(2, sel(\langle 1,k\rangle,t')) \\
 & k' = \left\{ \begin{array}{ll}
k & \Leftarrow \quad a \notin m \\
the(proj(3, sel(\langle 2,a\rangle, sel(\langle 1,k\rangle,t')))) & \Leftarrow \quad a \in m
\end{array}\right. \\
in & \langle k',\langle t,t'\rangle\rangle
\end{array}
\end{array}
$$

que implementa a esperada operação de "manipulação de apontadores" no contexto da programação em ambiente relacional. Notar que $decide_4$ não é solução única de (9.16) porque f (8.124) não é injectiva. Outras soluções válidas podiam aproveitar a operação para fazer 'garbage collection' sobre t e t' por remoção de todas as entradas directamente acessíveis de k, que não serão mais revisitadas, *cf.* o invariante (8.127).

Exercício 9.5 O conjunto dos números inteiros ($\mathbb{Z}_0$) é refinável em $\mathbb{N}_0 + \mathbb{N}_0$ (onde a etiqueta do co-produto "representa" o sinal, positivo ou negativo) sob a função de abstracção $f = [\,id, sym\,]$, onde se usa a construção funcional

$$
\begin{array}{rcl}
[\,f,g\,] & : & A + B \longrightarrow C \\
{[\,f,g\,](x)} & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcl} x = i_1(a) & \Rightarrow & f(a) \\ x = i_2(b) & \Rightarrow & g(b) \end{array}\right.
\end{array}
$$

e onde $sym(n) \stackrel{\mathrm{def}}{=} -n$.

Considere, neste contexto, o diagrama de refinamento de função que calcula o quadrado de um inteiro, $sq(z) \stackrel{\mathrm{def}}{=} z^2$, onde α é a variável:

$$
\begin{tikzcd}
\mathbb{Z}_0 \arrow[r, "sq"] & \mathbb{Z}_0 \\
\mathbb{N}_0 + \mathbb{N}_0 \arrow[u, "{[\,id,sym\,]}"] \arrow[r, "\alpha"'] & \mathbb{N}_0 + \mathbb{N}_0 \arrow[u, "{[\,id,sym\,]}"']
\end{tikzcd}
$$

1. Justifique os seguintes passos do cálculo:

$$
\begin{array}{rcll}
([\,id,sym\,]) \circ \alpha & = & sq \circ ([\,id,sym\,]) & (9.18) \\
 & = & [\,sq, (sq \circ sym)\,] & (9.19) \\
 & = & [\,sq, sq\,] & (9.20) \\
 & = & id \circ [\,sq, sq\,] & (9.21) \\
 & = & [\,id, sym\,] \circ i_1 \circ [\,sq, sq\,] & (9.22)
\end{array}
$$

Nota: baseie-se, entre outros, nos factos (1.36), (1.31) e (1.32).

2. Complete o processo de cálculo por forma a obter, finalmente

$$\alpha(x) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lcl} x = i_1(a) & \Rightarrow & i_1(a^2) \\ x = i_2(b) & \Rightarrow & i_1(b^2) \end{array} \right.$$

□

Exercício 9.6 Especifique ao nível de $DecTree$ as operações de *inicialização* de uma árvore de decisão, interrogação de qual a pergunta corrente e interrogação de qual o menu de respostas disponíveis para essa pergunta.

Calcule as respectivas simulações ao nível de $DecTree_4$.

□

9.3 Cálculo de Simulações Envolvendo Invariantes 'Ad Hoc'

Vimos na secção 8.8 como se podem adicionar invariantes 'ad hoc' a uma espécie de dados com o intuito de obter eficiência algorítmica. É agora a altura de ver tais invariantes a participar no processo de cálculo de simulações nessas condições.

O nosso exemplo será o das tabelas de 'hashing', já iniciado na secção 8.8.2. O conjunto de operações que pretendemos simular é o seguinte:

$$\begin{array}{rcl} initialDb & : & \longrightarrow Database \\ initialDb & \stackrel{\mathrm{def}}{=} & \emptyset \\ \\ insert & : & A \times Database \longrightarrow Database \\ insert(a,\sigma) & \stackrel{\mathrm{def}}{=} & \sigma \cup \{a\} \\ \\ find & : & A \times Database \longrightarrow 2 \\ find(a,\sigma) & \stackrel{\mathrm{def}}{=} & (a \in \sigma) \\ \\ remove & : & A \times Database \longrightarrow Database \\ remove(a,\sigma) & \stackrel{\mathrm{def}}{=} & \sigma - \{a\} \end{array} \tag{9.23}$$

para

$$Database \cong 2^A \tag{9.24}$$

A função de abstracção é — relembra-se — dada por (8.149).

A Operação $find$

O diagrama de refinamento para $find$ é:

$$\begin{tikzcd}
A \times 2^A \arrow[r, "find"] & 2 \\
A \times ((2^A)^n)_{\Phi_2} \arrow[u, "1_A \times f_1 \circ f_2"] \arrow[r, "find_2"'] & 2 \arrow[u, "1_2"']
\end{tikzcd}$$

tendo-se a seguinte equação a resolver em ordem a $find_2$:

$$find_2(a,t) = find(a, f_1(f_2(t)))$$

Teremos, sucessivamente,

$$\begin{aligned}
find_2(a,t) &= find(a, f_1(f_2(t))) \\
&= a \in f_1(f_2(t)) \\
&= a \in \bigcup_{i \in n} t(i) \\
&= \bigvee_{i \in n} a \in t(i) \\
&= \exists i \in n : a \in t(i)
\end{aligned}$$

Entrando agora em consideração com o invariante Φ_3 (8.148), teremos:

$$\begin{aligned}
\bigvee_{i \in n} a \in t(i) &= a \in t(H(a)) \vee \bigvee_{i \in n \wedge i \neq H(a)} a \in t(i) \\
&= (a \in t(H(a)) \vee \bigvee_{i \in n \wedge i \neq H(a)} F \\
&= a \in t(H(a)) \\
&= find(a, t(H(a)))
\end{aligned}$$

Em suma:

$$find_2(a,t) = find(a, t(H(a)))$$

i.é $find$ corresponde a $belongs$ sobre o 'bucket' a que a pertence, indexado pelo índice $H(a)$, como esperávamos. Como a cardinalidade de $t(H(a))$ é menor do que a de $\bigcup_{i \in n} t(i)$ [3], o ganho em eficiência é óbvio.

[3] O que só não acontece se H fôr uma função constante, uma óbvia "má" escolha para função de 'hashing'.

A Operação $insert$

Diagrama de refinamento:

$$\begin{array}{ccc} A \times 2^A & \xrightarrow{insert} & 2^A \\ {\scriptstyle 1_A \times f_1 \circ f_2}\uparrow & & \uparrow{\scriptstyle f_1 \circ f_2} \\ A \times ((2^A)^n)_{\Phi_2} & \xrightarrow[insert_2]{} & ((2^A)^n)_{\Phi_2} \end{array}$$

Equação de refinamento:

$$f_1(f_2(insert_2(a,t))) = insert(a, f_1(f_2(t)))$$

Raciocínio:

$$\begin{array}{rcl} f_1(f_2(insert_2(a,t))) & = & insert(a, f_1(f_2(t))) \\ & = & \{a\} \cup f_1(f_2(t)) \\ & = & \{a\} \cup \bigcup_{i \in n} t(i) \\ & = & \{a\} \cup t(H(a)) \cup \bigcup_{i \in n - \{H(a)\}} t(i) \\ & = & t'(H(a)) \cup \bigcup_{i \in n - \{H(a)\}} t(i) \end{array}$$

onde t' é a função "singular":

$$t' = \left(\begin{array}{c} H(a) \\ \{a\} \cup t(H(a)) \end{array} \right)$$

Então,

$$\begin{array}{rcl} t'(H(a)) \cup \bigcup_{i \in n - \{H(a)\}} t(i) & = & \bigcup_{i \in n} \left\{ \begin{array}{ccc} i = H(a) & \Rightarrow & t'(i) \\ \neg(i = H(a)) & \Rightarrow & t(i) \end{array} \right. \\ & = & \bigcup_{i \in n} \left\{ \begin{array}{ccc} i \in dom(t') & \Rightarrow & t'(i) \\ \neg(i \in dom(t')) & \Rightarrow & t(i) \end{array} \right. \\ & = & \bigcup_{i \in n} (t \dagger t')(i) \\ & = & f_1(f_2(t \dagger t')) \end{array}$$

Em suma,

$$f_1(f_2(insert_2(a,t))) = f_1(f_2(t \dagger \left(\begin{array}{c} H(a) \\ \{a\} \cup t(H(a)) \end{array} \right)))$$

ou, removendo $f_1 \circ f_2$ de ambos os lados da igualdade,

$$insert_2(a,t) \quad = \quad t \dagger \left(\begin{array}{c} H(a) \\ insert(a, t(H(a)) \end{array} \right)$$

Pode observar-se que $insert_2$ não é nada mais que $insert$ confinada ao 'collision bucket' que é relevante, obtendo eficiência dada a sua menor cardinalidade.

A Operação $initialDb$

Diagrama de refinamento:

$$\begin{tikzcd} {} \arrow[r, "initialDb"] & 2^A \\ {} \arrow[u] \arrow[r, "initialDb_2"'] & ((2^A)^n)_{\Phi_2} \arrow[u, "f_1 \circ f_2"'] \end{tikzcd}$$

Equação de refinamento:

$$f_1(f_2(initialDb_2)) = initialDb$$

Raciocínio: uma vez que

$$\bigcup_{i \in n} initialDb_2(i) \quad = \quad \emptyset$$

é imediato derivar

$$initialDb_2 \quad = \quad \left(\begin{array}{c} i \\ \emptyset \end{array} \right)_{i \in n}$$

Obtém-se assim a convencional inicialização da tabela, tipo "ciclo for" ($i = 1, n$) prefixando cada 'bucket' no conjunto vazio.

A Operação $remove$

Diagrama de refinamento:

$$\begin{tikzcd} A \times 2^A \arrow[r, "remove"] & 2^A \\ A \times ((2^A)^n)_{\Phi_2} \arrow[u, "1_A \times f_1 \circ f_2"] \arrow[r, "remove_2"'] & ((2^A)^n)_{\Phi_2} \arrow[u, "f_1 \circ f_2"'] \end{tikzcd}$$

Equação de refinamento:

$$f_1(f_2(remove_2(a,t))) = remove(a, f_1(f_2(t)))$$

Raciocínio:

$$\begin{array}{rcl}
f_1(f_2(remove_2(a,t))) & = & remove(a, f_1(f_2(t))) \\
& = & f_1(f_2(t)) - \{a\} \\
& = & \bigcup_{i \in n} t(i) - \{a\} \\
& = & (\bigcup_{i \in n} t(i)) \cap \overline{\{a\}} \\
& = & \bigcup_{i \in n} (t(i) \cap \overline{\{a\}}) \\
& = & \bigcup_{i \in n} (t(i) - \{a\}) \\
& = & (t(H(a)) - \{a\}) \cup \bigcup_{i \in n - \{H(a)\}} (t(i) - \{a\}) \\
& = & t'(H(a)) \cup \bigcup_{i \in n - \{H(a)\}} \underbrace{t(i) - \{a\}}_{(*)}
\end{array}$$

onde t' é a função "singular":

$$t' = \left(\begin{array}{c} H(a) \\ t(H(a)) - \{a\} \end{array} \right)$$

É altura de tirar partido do invariante Φ_2 (8.144), do qual inferimos

$$\begin{array}{c}
\forall i \in n : (\forall a \in t(i) : i = H(a)) \\
\updownarrow \\
\forall i \in n, a \in A : a \in t(i) \Rightarrow i = H(a) \\
\updownarrow \\
\forall i \in n, a \in A : i \neq H(a) \Rightarrow a \notin t(i) \\
\updownarrow \\
\forall i \in n, a \in A : i \neq H(a) \Rightarrow t(i) - \{a\} = t(i)
\end{array}$$

que é suficiente para reduzir $(*)$ acima a $t(i)$. A raciocínio subsequente é similar ao de $insert$, conduzindo a

$$f_1(f_2(remove_2(a,t))) = f_1(f_2(t \dagger \left(\begin{array}{c} H(a) \\ t(H(a)) - \{a\} \cup t(H(a)) \end{array} \right)))$$

i.é (removendo $f_1 \circ f_2$ de ambos os lados da igualdade):

$$remove_2(a,t) \quad = \quad t \dagger \begin{pmatrix} H(a) \\ remove(a, t(H(a)) \end{pmatrix}$$

Tal como em $insert_2$, mais uma vez se observa que $remove_2$ é nada mais do que $remove$ confinada ao relevante 'collision bucket'. Mas, ao invés de $insert_2$, $remove_2$ foi obtida recorrendo explicitamente ao invariante 'ad hoc'. Fica assim patente o papel de tal invariante no processo de obtenção de eficiência algorítmica.

9.4 Desrecursivação Algorítmica

Quando atrás se parou na expressão (9.6) como resultado de um processo de cálculo de uma simulação ($belongs$) estávamos apenas preocupados em obter uma versão "computável" da referida simulação.

É possível prosseguir no eixo algorítmico no sentido de se vir a obter uma codificação não-recursiva de $belongs$. Seja *beloop* uma função que satisfaça

$$beloop(x, y, b) = b \vee belongs(x, y) \tag{9.25}$$

Antes de mais, é imediato que

$$beloop(x, y, F) = belongs(x, y) \tag{9.26}$$

pois a estrutura $\langle\{V, F\}; \vee, F\rangle$ forma um monóide. Mais ainda, para $y = <>$,

$$\begin{array}{rcl} beloop(x, < >, b) & = & b \vee belongs(x, < >) \\ & = & b \vee F \\ & = & b \end{array} \tag{9.27}$$

assim como, para $y \neq < >$,

$$\begin{array}{rcl} beloop(x, y, b) & = & b \vee (x = head(y) \vee belongs(x, tail(y))) \\ & = & (b \vee (x = head(y)) \vee belongs(x, tail(y)) \\ & = & beloop(x, tail(y), (b \vee (x = head(y)))) \end{array} \tag{9.28}$$

cf. (9.26). Juntando (9.27) com (9.28), teremos

$$\begin{array}{l} beloop(x, y, b) \stackrel{\text{def}}{=} \\ \quad \left\{ \begin{array}{lcl} y = < > & \Rightarrow & b \\ y \neq < > & \Rightarrow & beloop(x, tail(y), b \vee (x = head(y))) \end{array} \right. \end{array} \tag{9.29}$$

Repare-se agora que o mesmo monóide tem a constante booleana T como elemento absorvente, *i.é* que

$$b \vee T = T \vee b = T$$

Logo, por (9.25),

$$beloop(x, y, T) = T$$

i.é

$$(b = T) \Rightarrow beloop(x, y, b) = b$$

Podemos tirar partido desta condição na seguinte elaboração de (9.29) por (B.2):

$$beloop(x, y, b) \stackrel{\text{def}}{=}$$
$$\left\{ \begin{array}{lll} y = < > & \Rightarrow & \left\{ \begin{array}{rcl} b & \Rightarrow & b \\ \neg(b) & \Rightarrow & b \end{array} \right. \\ y \neq < > & \Rightarrow & \left\{ \begin{array}{rcl} b & \Rightarrow & \\ \neg(b) & \Rightarrow & beloop(x, tail(y), b \vee (x = head(y))) \end{array} \right. \end{array} \right.$$

o que simplifica em

$$beloop(x, y, b) \stackrel{\text{def}}{=}$$
$$\left\{ \begin{array}{rcl} y = < > \wedge b & \Rightarrow & b \\ y = < > \wedge \neg b & \Rightarrow & b \\ y \neq < > \wedge b & \Rightarrow & b \\ y \neq < > \wedge \neg b & \Rightarrow & beloop(x, tail(y), b \vee (x = head(y))) \end{array} \right.$$

ou, finalmente

$$beloop(x, y, b) \stackrel{\text{def}}{=}$$
$$\left\{ \begin{array}{rcl} y = < > \vee b & \Rightarrow & b \\ y \neq < > \wedge \neg b & \Rightarrow & beloop(x, tail(y), (x = head(y))) \end{array} \right. \tag{9.30}$$

aplicando (B.3). Ora, juntando (9.26) com (9.30), obtemos a "semântica denotacional" de, respectivamente, a inicialização e o corpo do seguinte *ciclo-while*,

```
{  bool  found = 0;
   list  p;
   {       p = y;
           while   ((p != <>) && ¬ found)
                   { found = (x == head(p));        (9.31)
                     p = tail(p)
                   };
   }
}
```

codificado aqui numa notação procedimental *ad hoc*, "tipo C". A variável auxiliar `p` destina-se a poupar o parâmetro `y`. Usa-se ainda `found` em lugar de `b` por ser mais sugestivo. Fica assim bem evidente que variáveis como `found`, que parecem resultar de "toques de intuição" na programação 'ad hoc', afinal decorrem cientificamente, na programação formal, a partir das propriedades matemáticas dos operadores envolvidos. Veremos a seguir como este raciocínio é generalizável a ponto de abarcar uma grande classe de algoritmos.

9.4.1 Generalização

O processo de desrecursivação que foi apresentado na secção anterior não é nem mais nem menos do que uma instância de uma regra genérica para remoção de ("falsa") recursividade, que agora se discutirá na generalidade.

Nesse sentido, seja dada a função abstracta

$$\begin{array}{rcl} f & : & \ldots \times Y \times \ldots \longrightarrow M \\ f(\_, y, \_) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{rcl} p(\_, y, \_) & \Rightarrow & u \\ \neg(p(\_, y, \_)) & \Rightarrow & d(\_, y, \_) \theta f(\_, e(y), \_) \end{array} \right. \end{array} \tag{9.32}$$

cujos parâmetros algoritmicamente irrelevantes são omitidos, sendo os respectivos argumentos substituidos por "_", e onde ocorrem os seguintes operadores auxiliares,

$$\begin{array}{rcl} p & : & \ldots \times Y \times \ldots \longrightarrow 2 \\ e & : & Y \longrightarrow Y \\ d & : & \ldots \times Y \times \ldots \longrightarrow M \\ \theta & : & M \times M \longrightarrow M \\ u & : & \longrightarrow M \end{array}$$

onde $\langle M; \theta, u \rangle$ formam um monóide.

Não é difícil ver $belongs$ como caso particular de (9.32), para as substituições

$f(\_, y, \_)$	$belongs(x, y)$
$p(\_, y, \_)$	$y = <>$
u	F
$d(\_, y, \_)$	$d = head(y)$
θ	$\vee$
$e(y)$	$tail(y)$

Na prática, existe um sem número de definições recursivas que caem, tal como $belongs$, no esquema (9.32), por exemplo o factorial de y,

$$y! = \left\{ \begin{array}{rcl} y = 0 & \Rightarrow & 1 \\ \neg(y = 0) & \Rightarrow & y \times (y - 1)! \end{array} \right.$$

$multp$ (7.9), $comp$ (9.10), *etc. etc.*Assim, se deduzirmos uma regra para desrecursivar (9.32), poderemos aplicá-la a uma vasta gama de simulações recursivas [4].

Generalizando então o que acima se fez em relação a $belongs$, começaremos por introduzir a função

$$floop(\_, y, \_, r) \quad \stackrel{\mathrm{def}}{=} \quad r\theta f(\_, y, \_) \tag{9.33}$$

que, pelas propriedades do monóide $\langle M; \theta, u\rangle$, obedecerá à propriedade:

$$f(\_, y, \_) \quad = \quad floop(\_, y, \_, u) \tag{9.34}$$

De (9.33) obtém-se sucessivamente, por substituição, pela regra (B.1) e pelas propriedades do monóide $\langle M; \theta, u\rangle$:

$$\begin{aligned}
floop(_, y, _, r) &= r\theta f(_, y, _) \\
&= r\theta \left\{ \begin{array}{rcl} p(_, y, _) & \Rightarrow & u \\ \neg(p(_, y, _)) & \Rightarrow & d(_, y, _)\theta f(_, e(y), _) \end{array} \right. \\
&= \left\{ \begin{array}{rcl} p(_, y, _) & \Rightarrow & r\theta u \\ \neg p(_, y, _) & \Rightarrow & r\theta(d(_, y, _)\theta f(_, e(y), _)) \end{array} \right. \\
&= \left\{ \begin{array}{rcl} p(_, y, _) & \Rightarrow & r \\ \neg p(_, y, _) & \Rightarrow & \underbrace{(r\theta d(_, y, _))\theta f(_, e(y), _)}_{A} \end{array} \right.
\end{aligned}$$

Mas, por instanciação de (9.33), tem-se

$$A = floop(\_, e(y), \_, r\theta d(\_, y, \_))$$

Em suma, teremos

$$floop(\_, y, \_, r) \quad = \quad \left\{ \begin{array}{rcl} p(\_, y, \_) & \Rightarrow & r \\ \neg p(\_, y, \_) & \Rightarrow & floop(\_, e(y), \_, r\theta d(\_, y, \_)) \end{array} \right.$$

que, conjuntamente com a "inicialização" (9.34), condiz com a semântica do seguinte *ciclo-while*:

```
{  M      r = u;
   Y      y' = y;
   while  (¬ p(_,y',_))
          { r = r θ d(_,y',_);
            y' = e(y')
          };
}
```

[4] O padrão algorítmico (9.32) é conhecido na literatura pela designação de *esquema de recursividade linear monádica*. Trata-se apenas de um entre os vários esquemas falsamente recursivos que se conhecem.

No caso de θ admitir um valor absorvente em M, *i.é* um $a \in M$ tal que

$$a\theta m = m\theta a = a$$

é ainda possível especializar mais $floop$,

$$\begin{aligned} floop(\_, y, \_, r) &= \begin{cases} p(\_, y, \_) & \Rightarrow \quad r \\ \neg p(\_, y, \_) & \Rightarrow \quad \begin{cases} r = a & \Rightarrow \quad a \\ r \neq a & \Rightarrow \quad floop(\_, e(y), \_, r\theta d(\_, y, \_)) \end{cases} \end{cases} \\ &= \begin{cases} p(\_, y, \_) \vee r = a & \Rightarrow \quad r \\ \neg p(\_, y, \_) \wedge r \neq a & \Rightarrow \quad floop(\_, e(y), \_, r\theta d(\_, y, \_)) \end{cases} \end{aligned}$$

conduzindo ao *ciclo-while*:

```
{  M       r = u;
   Y       y' = y;
   while   (¬ p(_,y',_) && r!= a)
           { r = r θ d(_,y',_);
             y' = e(y')
           };
}
```

do qual $belongs$ (9.31) é uma instância.

Exercício 9.7 À luz do exposto nesta secção, calcule versões não recursivas de $y!$ e de $comp$ (9.10) e codifique-as em "pseudo"-C.
□

Exercício 9.8 Considere a seguinte função

$$\begin{array}{rcl} g & : & Y \longrightarrow M \\ g(y) & \stackrel{\text{def}}{=} & \begin{cases} p(y) & \Rightarrow \quad v \\ \neg(p(y)) & \Rightarrow \quad d(y)\,\theta\, g(e(y)) \end{cases} \end{array}$$

onde θ é associativa e v é um qualquer elemento de M.

1. Mostre que, para todo o $y \in Y$, a igualdade

$$g(y) \quad = \quad f(y)\,\theta\, v \tag{9.35}$$

se verifica, onde

$$\begin{array}{rcl} f & : & Y \longrightarrow M \\ f(y) & \stackrel{\text{def}}{=} & \begin{cases} p(y) & \Rightarrow \quad u \\ \neg(p(y)) & \Rightarrow \quad d(y)\,\theta\, f(e(y)) \end{cases} \end{array}$$

e u é elemento neutro de θ.

2. Mostre que, desde que θ seja também comutativa, então g é concretizável algoritmicamente no seguinte ciclo-`while`:

```
{   M r = v;
    Y yy = y;
    while (¬ p(yy))
        {   r = r θ d(yy);
            yy = e(yy);
        };
}
```

registado na notação "pseudo-C" usada nesta disciplina. **Sugestão**: parta da definição da função $floop$ (9.33).

3. No caso de v ser o elemento absorvente de θ, que pode dizer do comportamento do ciclo-`while` acima? Justifique formalmente.

□

Exercício 9.9 Qual o operador sobre sequências que simula o filtro

$$filter(S) \stackrel{\mathrm{def}}{=} \{f(x) \mid x \in S\}$$

no refinamento

$$\begin{tikzcd}
2^A \arrow[r, "filter"] & 2^B \\
A^\star \arrow[u, "elems"] \arrow[r, "?"'] & B^\star \arrow[u, "elems"']
\end{tikzcd}$$

onde $f : A \longrightarrow B$? Deduza esse operador, ou conjecture-o e prove a respectiva simulação.
□

9.4.2 Cálculo Avançado sobre Esquema Linear Monádico

Veremos, para terminarmos este capítulo, dois exemplos de concretização algorítmica orientados ao *esquema linear monádico* e que nos mostram duas perspctivas complementares do cálculo de programas.

No primeiro, trata-se da concretização algorítmica do fecho transitivo de um grafo acíclico. Pretende-se com este exemplo mostrar como o cálculo de uma desrecursivação pode e deve basear-se em propriedades da própria função que estamos a desrecursivar. No segundo, mostra-se não só como é possível reduzir o "grau de recursividade" (polinomialidade) de uma função (algorítmo), mas também como a pré-condição de uma função pode participar no processo transformacional, acabando por se fundir com o corpo da própria função.

Concretização Algorítmica do Fecho Transitivo de um Grafo Acíclico

Seja $g \in 2^{A \times A}$ um grafo acíclico (*e.g.* uma hierarquia de classes), sobre o qual faz sentido calcular os antecessores de um nó x:

$$ants(g,x) \stackrel{\text{def}}{=} \{\pi_1(p) \mid\ p \in g \wedge \pi_2(p) = x\}$$

O fecho por antecessores imediatos e transitivos é especificado por

$$ants^+(g,S) \quad \stackrel{\text{def}}{=} \quad \bigcup_{x \in S} \{x\} \cup ants^+(g, ants(g,x)) \tag{9.36}$$

Comecemos por desenvolver $\bigcup$ em (9.36):

$$ants^+(g,S) =$$
$$\left\{ \begin{array}{rcl} S = \emptyset & \Rightarrow & \emptyset \\ \neg(S = \emptyset) & \Rightarrow & let \quad x \in S \\ & & in \quad \underbrace{(\{x\} \cup ants^+(g, ants(g,x))) \cup ants^+(g, S - \{x\})}_{A} \end{array} \right.$$

Reparemos na seguinte propriedade distributiva do fecho $ants^+$ em relação à operação de acumulação ($\cup$), cuja dedução por (9.36) é imediata:

$$ants^+(g, S \cup R) \quad = \quad ants^+(g,S) \cup ants^+(g,R) \tag{9.37}$$

Aplicando esta propriedade logo após a associatividade de $\cup$:

$$\begin{array}{rcl} A & = & (\{x\} \cup ants^+(g, ants(g,x))) \cup ants^+(g, S - \{x\}) \\ & = & \{x\} \cup (ants^+(g, ants(g,x)) \cup ants^+(g, S - \{x\})) \\ & = & \{x\} \cup ants^+(g, ants(g,x) \cup (S - \{x\})) \end{array}$$

Obtemos assim a versão *linear monádica* para $ants^+$, de onde já podemos remover a recursividade conforme exemplos anteriores:

$$ants^+(g,S) \quad \stackrel{\text{def}}{=} \quad ants^+loop(g, S, \emptyset)$$

onde

$$ants^+loop(g,S,R) \stackrel{\text{def}}{=}$$
$$\left\{ \begin{array}{rcl} S = \emptyset & \Rightarrow & R \\ \neg(S = \emptyset) & \Rightarrow & let \quad x \in S \\ & & in \quad ants^+loop(g, \underbrace{ants(g,x) \cup (S - \{x\})}_{B}, R \cup \{x\}) \end{array} \right.$$

Como o grafo é acíclico, tem-se

$$x \notin ants(g, x) \tag{9.38}$$

Logo:

$$\begin{aligned} B &= ants(g, x) \cup (S - \{x\}) \\ &= (ants(g, x) \cup S) - \{x\} \end{aligned}$$

Em suma: obtemos o clássico ciclo em que S contém os nós ainda por visitar, R contém os nós já visitados (e, no fim do ciclo, o resultado de toda as visitas) e x é o elemento que é "marcado" como acabado de visitar:

```
R = vazio;
while S != vazio
    { x = next(S);
      R = addElem(R,x);
      S = union(S,ants(g,x));
      S = remElem(S,x)
     };
output(R);
```

NB: (sobre o termo "marcar" acima) — o par $\langle R, S \rangle$ é do tipo $2^A \times 2^A$, *i.é*,

$$\begin{aligned} 2^A \times 2^A &\cong (2^A)^2 \\ &\cong 2^{A \times 2} \end{aligned}$$

i.é, podemos ter um único S onde os elementos estão emparelhados com uma marca booleana (2) a indicar *visitado* = 0 *ou* 1.

Desrecursivação Bi-linear

Para terminarmos este capítulo, vamos fazer ainda mais um exercício de desrecursivação, este de uma função com pré-condição sobre uma estrutura algorítmica bilinear afectada de um invariante. Em particular, veremos como este participa no processo transformacional, que acaba por fundir o corpo da função com a sua pré-condição.

Na secção 8.7.1 foi referida a estrutura *árvore binária de procura*, que se entende normalmente como a definição que se segue (relembrar (8.97), aqui apenas enriquecida com selectores):

$$\begin{aligned} BinTree &\cong 1 + BinNode \\ BinNode &\cong \begin{array}{ll} LT : BinTree & \times \\ K : Key & \times \\ D : Data & \times \\ RT : BinTree & \end{array} \end{aligned}$$

sujeita ao seguinte invariante de (bi-)ordenação:

$$\begin{array}{l} invnd \quad : \quad BinTree \to Bool \\ \quad invnd(t) \stackrel{\text{def}}{=} \\ \quad\quad \left\{ \begin{array}{lcll} t = NIL & \Rightarrow & V & \\ \neg(t = NIL) & \Rightarrow & let & t = \langle lt, k, d, rt \rangle \\ & & in & (\forall lk \in collkeys(lt) . lk < k) \wedge \\ & & & (\forall rk \in collkeys(rt) . k < rk) \wedge \\ & & & invnd(lt) \wedge invnd(rt) \end{array} \right. \end{array} \tag{9.39}$$

para

$$\begin{array}{l} collkeys \quad : \quad BinTree \to 2^{Key} \\ \quad collkeys(t) \stackrel{\text{def}}{=} \\ \quad\quad \left\{ \begin{array}{lcll} t = NIL & \Rightarrow & \{\} & \\ \neg(t = NIL) & \Rightarrow & collkeys(LT(t)) & \cup \\ & & \{K(t)\} & \cup \\ & & collkeys(RT(t)) & \end{array} \right. \end{array} \tag{9.40}$$

A operação a ser realizada é a seguinte:

$$\begin{array}{l} findb \quad : \quad BinTree \times Key \to Data \\ \quad findb(t, k) \stackrel{\text{def}}{=} \\ \quad\quad \{ \quad k \in collkeys(t) \Rightarrow \\ \quad\quad\quad \begin{array}{ll} let & lt = LT(t) \\ & mk = K(t) \\ & md = D(t) \\ & rt = RT(t) \\ in & \left\{ \begin{array}{lcl} k = mk & \Rightarrow & md \\ \neg(k = mk) & \Rightarrow & \left\{ \begin{array}{lcl} k < mk & \Rightarrow & findb(lt, k) \\ \neg(k < mk) & \Rightarrow & findb(rt, k) \end{array} \right. \end{array} \right. \end{array} \end{array} \tag{9.41}$$

A função $findb$ é, pois, parcial. Reparemos como a sua pré-condição ($k \in collkeys(t)$) elimina a necessidade de testar se $t = NIL$. Como estamos interessados em realizar e codificar a função na sua totalidade, começaremos por "totalizar" $findb$ embebendo a pré-condição na sua definição e escolhendo um valor $error$ (mensagem de erro) tal que $error \notin Data$. Cria-se assim a sua versão $findbt$, total:

$$\begin{array}{lcl} findbt & : & BinTree \times Key \to Data \cup \{error\} \\ findbt(t, k) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} k \in collkeys(t) & \Rightarrow & findb(t) \\ \neg(k \in collkeys(t)) & \Rightarrow & error \end{array} \right. \end{array} \tag{9.42}$$

Prosseguiremos "desenrolando" a expressão de teste $k \in collkeys(t)$ via (9.40):

$$k \in collkeys(t) \quad = \quad \begin{cases} t = NIL & \Rightarrow \quad k \in \{\} \\ \neg(t = NIL) & \Rightarrow \quad \begin{array}{ll} k \in collkeys(LT(t)) & \vee \\ k = K(t) \vee & \vee \\ k \in collkeys(RT(t)) & \end{array} \end{cases}$$

Como $k \in \{\} \Leftrightarrow F$, re-escrevemos (9.42) em:

$$findbt(t,k) =$$
$$\begin{cases} (\begin{cases} t = NIL & \Rightarrow \quad F \\ \neg(t = NIL) & \Rightarrow \quad \begin{array}{ll} k \in collkeys(LT(t)) & \vee \\ k = K(t) \vee & \vee \\ k \in collkeys(RT(t)) & \end{array} \end{cases}) & \Rightarrow \quad findb(t,k) \\ \neg((\begin{cases} t = NIL & \Rightarrow \quad F \\ \neg(t = NIL) & \Rightarrow \quad \begin{array}{ll} k \in collkeys(LT(t)) & \vee \\ k = K(t) \vee & \vee \\ k \in collkeys(RT(t)) & \end{array} \end{cases})) & \Rightarrow \quad error \end{cases}$$

e, aplicando (B.4),

$$findbt(t,k) =$$
$$\begin{cases} t = NIL & \Rightarrow \quad error \\ \neg(t = NIL) & \Rightarrow \quad \begin{cases} \begin{array}{ll} k \in collkeys(LT(t)) & \vee \\ k = K(t) \vee & \vee \\ k \in collkeys(RT(t)) & \end{array} & \Rightarrow \quad findb(t,k) \\ \neg(\begin{array}{ll} k \in collkeys(LT(t)) & \vee \\ k = K(t) \vee & \vee \\ k \in collkeys(RT(t)) & \end{array}) & \Rightarrow \quad error \end{cases} \end{cases} \tag{9.43}$$

Usando a associatividade e comutatividade da disjunção e aplicando a regra (B.6), o predicado (9.43) re-escreve-se em

$$\begin{array}{l} k = K(t) \vee \\ k \in collkeys(LT(t)) \vee \\ k \in collkeys(RT(t))) \end{array} \quad \Leftrightarrow$$
$$\begin{cases} k = K(t) & \Rightarrow \quad V \\ \neg(k = K(t)) & \Rightarrow \quad \begin{cases} k \in collkeys(LT(t)) & \Rightarrow \quad V \\ \neg(k \in collkeys(LT(t))) & \Rightarrow \quad k \in collkeys(RT(t)) \end{cases} \end{cases}$$

Substituindo e simplificando, obtemos

$$findbt(t,k) =$$

$$A = \left\{ \begin{array}{lll} t = NIL & \Rightarrow & error \\ \neg(t = NIL) & \Rightarrow & \left\{ \begin{array}{lll} k = K(t) & \Rightarrow & findb(t,k) \\ \neg(k = K(t)) & \Rightarrow & A \end{array} \right. \end{array} \right.$$

$$\left\{ \begin{array}{lll} k \in collkeys(LT(t)) & \Rightarrow & findb(t,k) \\ \neg(k \in collkeys(LT(t))) & \Rightarrow & \left\{ \begin{array}{lll} k \in collkeys(RT(t)) & \Rightarrow & findb(t,k) \\ \neg(k \in collkeys(RT(t))) & \Rightarrow & error \end{array} \right. \end{array} \right. \quad (9.44)$$

Ora o invariante sobre $BinTree$ (9.39) e a definição de $findb$ (9.41) garantem os factos seguintes, para quaisquer $t \in BinNode$ e $k \in Key$:

$$\begin{array}{rcl} k = K(t) & \Rightarrow & findb(t,k) = D(t) \\ k \in collkeys(LT(t)) & \Rightarrow & k < K(t) \\ & \Rightarrow & findb(t,k) = findb(LT(t),k) \end{array} \quad (9.45)$$

que, por sua vez, tornam possíveis três substituições em (9.44), conduzindo a:

$$\begin{array}{rcl} findbt(t,k) & = & \left\{ \begin{array}{lll} t = NIL & \Rightarrow & error \\ \neg(t = NIL) & \Rightarrow & \left\{ \begin{array}{lll} k = K(t) & \Rightarrow & D(t) \\ \neg(k = K(t)) & \Rightarrow & A \end{array} \right. \end{array} \right. \\ A & = & \left\{ \begin{array}{lll} k \in collkeys(LT(t)) & \Rightarrow & findb(LT(t),k) \\ \neg(k \in collkeys(LT(t))) & \Rightarrow & B \end{array} \right. \\ B & = & \left\{ \begin{array}{lll} k \in collkeys(RT(t)) & \Rightarrow & findb(RT(t),k) \\ \neg(k \in collkeys(RT(t))) & \Rightarrow & error \end{array} \right. \end{array} \quad (9.46)$$

'Folding' B (9.46) de acordo com (9.42),

$$\left\{ \begin{array}{lll} k \in collkeys(RT(t)) & \Rightarrow & findb(RT(t),k) \\ \neg(k \in collkeys(RT(t))) & \Rightarrow & error \end{array} \right. = findbt(RT(t),k)$$

somos conduzidos a

$$\begin{array}{rcl} findbt(t,k) & = & \left\{ \begin{array}{lll} t = NIL & \Rightarrow & error \\ \neg(t = NIL) & \Rightarrow & \left\{ \begin{array}{lll} k = K(t) & \Rightarrow & D(t) \\ \neg(k = K(t)) & \Rightarrow & A \end{array} \right. \end{array} \right. \\ A & = & \left\{ \begin{array}{lll} k \in collkeys(LT(t)) & \Rightarrow & findb(LT(t),k) \\ \neg(k \in collkeys(LT(t))) & \Rightarrow & findbt(RT(t),k) \end{array} \right. \end{array} \quad (9.47)$$

O passo final consiste em remover a ocorrência que resta de $collkeys$ em (9.46). A estratégia para esse passo consiste em "enfraquecer" a condição

$$k \in collkeys(LT(t))$$

em (9.47) para

$$k < K(t)$$

cf. (9.45) [5]. Aumenta assim o conjunto de valores para os quais a condição de teste de (9.47) vale V, que assim incluirá os elementos de

$$X = \{k \in Key \mid \; k < K(t)\} - collkeys(LT(t))$$

Como, para qualquer $k \in X$,

$$k \neq K(t) \wedge k \notin collkeys(LT(t)) \wedge k \notin collkeys(RT(t))$$

teremos

$$findbt(RT(t), k) = error$$

cf. (9.44). Portanto, (9.47) pode ser substituída por

$$\left\{ \begin{array}{lcl} k < K(t) & \Rightarrow & \left\{ \begin{array}{lcl} k \in collkeys(LT(t)) & \Rightarrow & findb(LT(t), k) \\ \neg(k \in collkeys(LT(t))) & \Rightarrow & error \end{array} \right. \\ \neg(k < K(t)) & \Rightarrow & findbt(RT(t), k) \end{array} \right. \tag{9.48}$$

Fazendo 'folding' de (9.48) via (9.42), (9.47) será finalmente re-escrita em

$$findbt(t, k) =$$
$$\left\{ \begin{array}{lcl} t = NIL & \Rightarrow & error \\ \neg(t = NIL) & \Rightarrow & \left\{ \begin{array}{lcl} k = K(t) & \Rightarrow & D(t) \\ \neg(k = K(t)) & \Rightarrow & \left\{ \begin{array}{lcl} k < K(t) & \Rightarrow & findbt(LT(t), k) \\ \neg(k < K(t)) & \Rightarrow & findbt(RT(t), k) \end{array} \right. \end{array} \right. \end{array} \right.$$

ou, aplicando a regra (B.1),

$$findbt(t, k) =$$
$$\left\{ \begin{array}{lcl} t = NIL & \Rightarrow & error \\ \neg(t = NIL) & \Rightarrow & \left\{ \begin{array}{lcl} k = K(t) & \Rightarrow & D(t) \\ \neg(k = K(t)) & \Rightarrow & findbt(\left\{ \begin{array}{lcl} k < K(t) & \Rightarrow & LT(t) \\ \neg(k < K(t)) & \Rightarrow & RT(t) \end{array} \right. , k) \end{array} \right. \end{array} \right. \tag{9.49}$$

começando a pôr em evidência a falsa recursividade.

Para se obter a solução puramente iterativa ('tail-recursion') usa-se um resultado auxiliar dado em apêndice (B.11) que permite converter (9.49) em

$$findbt(t, k) \stackrel{\text{def}}{=}$$
$$\left\{ \begin{array}{lcl} t = NIL & \Rightarrow & error \\ \neg(t = NIL) & \Rightarrow & D(loop(t, k)) \end{array} \right.$$
$$loop(t, k) = \tag{9.50}$$
$$\left\{ \begin{array}{lcl} k = K(t) & \Rightarrow & t \\ \neg(k = K(t)) & \Rightarrow & \left\{ \begin{array}{lcl} t = NIL & \Rightarrow & error \\ \neg(t = NIL) & \Rightarrow & loop(\left\{ \begin{array}{lcl} k < K(t) & \Rightarrow & LT(t) \\ \neg(k < K(t)) & \Rightarrow & RT(t) \end{array} \right. , k) \end{array} \right. \end{array} \right. \tag{9.51}$$

[5] Dado um predicado p, por "enfraquecer p" designa-se a substituição de p por um q tal que $p \Rightarrow q$.

Em suma, obteve-se (por transformações) uma definição de $findbt$ que é directamente codificável num *ciclo-while*. É imediato codificar (9.51) sob a forma de instruções de uma linguagem imperativa. Por exemplo, o seguinte fragmento de programa PASCAL, derivado de (9.51):

```
TYPE Key = ....;
     Data = ...;
     BinTree = ^BinNode
     BinNode = RECORD
                 K: Key;
                 D: Data;
                 LT,RT: BinTree
               END;

VAR tree: BinTree;

PROCEDURE findbin (k: Key; VAR d: Data);
VAR p: BinTree; q: Key;
BEGIN p := tree;
      IF p = nil THEN terminate-error
                 ELSE q := p.K;
      WHILE q <> k DO
        BEGIN IF k < q THEN p := p^.LT
                       ELSE p := p^.RT;
              IF p = nil THEN terminate-error
                         ELSE q := p.K
        END;
      d := D(q)
  END (* findbin *);
```

é exactamente a codificação proposta por Fielding [Fie80] (ressalvadas leves diferenças na escolha dos identificadores) que é aí trabalhada em estilo 'invent-and-verify' usando regras de prova da metodologia VDM.

9.4.3 Síntese

A síntese de algoritmos a partir das suas especificações recorre quase sempre a técnicas de desrecursivação. De facto, muitas expressões matemáticas usadas em especificação formal "encerram em si" processos algorítmicos que, uma vez chegados à fase de implementação, importa explicitar.

Um dos casos mais interessantes neste aspecto tem a ver com as seguintes expressões que definem conjuntos por compreensão,

$$\{f(x) \mid \ x \in X \wedge p(x)\}$$

(a conhecida fórmula de "abstracção de Zermelo-Frænkl") dada uma função $f : A \to B$, um subconjunto finito $X \subseteq A$ e $p : A \to \{V, F\}$ um predicado de "filtragem" sobre A; e sequências por compreensão,

$$<f(x) \mid x \leftarrow L \wedge p(x)>$$

para os mesmos f e p e $L \in A^*$. Expressões como estas são extremamente vulgares em especificação formal, a primeira por especificar filtragem de informação (*e.g.* X pode ser uma tabela relacional, f pode exprimir a projecção de alguns dos seus atributos e p pode exprimir um predicado de selecção, *cf.* SQL) e a segundo por especificar processamento de 'streams'.

Como refinamos construções tão básicas como estas?

É sabido que $\{f(x) \mid \ x \in X \wedge p(x)\}$ designa exactamente o resultado da aplicação da seguinte função φ a X:

$$\begin{array}{l} \varphi : 2^A \quad \to \quad 2^B \\ \quad \varphi(y) \stackrel{\text{def}}{=} \\ \qquad \left\{ \begin{array}{lll} y = \emptyset & \Rightarrow & \emptyset \\ y \neq \emptyset & \Rightarrow & \begin{array}{ll} let & e \in y \\ & a = \left\{ \begin{array}{lll} p(e) & \Rightarrow & \{f(e)\} \\ \neg p(e) & \Rightarrow & \emptyset \end{array} \right. \\ in & a \cup \varphi(y - \{e\}) \end{array} \end{array} \right. \end{array}$$

o mesmo acontecendo entre $<f(x) \mid x \leftarrow L \wedge p(x)>$ e $\phi(L)$ para

$$\phi(y) \stackrel{\text{def}}{=} \left\{ \begin{array}{lll} y = <> & \Rightarrow & <> \\ y \neq <> & \Rightarrow & \begin{array}{ll} let & x = head(y) \\ & y' = tail(y) \\ in & \left\{ \begin{array}{lll} p(x) & \Rightarrow & cons(f(x), \phi(y')) \\ \neg(p(x)) & \Rightarrow & \phi(y') \end{array} \right. \end{array} \end{array} \right.$$

Reparemos que

$$\left\{ \begin{array}{lll} p(x) & \Rightarrow & cons(f(x), \phi(y')) \\ \neg(p(x)) & \Rightarrow & \phi(y') \end{array} \right.$$

é equivalente a

$$\left\{ \begin{array}{lll} p(x) & \Rightarrow & <f(x)> \\ \neg(p(x)) & \Rightarrow & <> \end{array} \right. \frown phi(y')$$

Assim, tanto φ como ϕ são instâncias do esquema linear monádico (9.32) e não será difícil mostrar que estão nas condições de serem desrecursivadas pelo resultado genérico da secção 9.4.1. A título de exemplo, para ϕ teremos:

```
{  B* r = <>;
   A* y' = y;
   A x;
   while (y' != <>)
       { x = head(y');
         y'= tail(y');
         if p(x) r = conc(r,f(x));
       }
}
```

ou, cada vez mais em concreto, codificando a função em termos de `stdin` e `stdout` e assumindo as funções de 'i/o' dedicadas `getA` e `putB`,

```
{
   A x;
   while ((x = getA(stdin)) != EOF)
         { if p(x) putB(f(x)); }
}
```

etc.

Exercício 9.10 Desenvolva o raciocínio anterior para a construção φ.
$\square$

Como nota final, é de referir que o problema da desrecursivação, que dissemos pertencer ao eixo do refinamento algorítmico (*cf.* Figura 7.2) se pode reduzir um caso particular de simulação em que todas as funções de abstracção são identidades.

9.5 Exercícios

Exercício 9.11 No contexto do conhecido refinamento de conjuntos por sequências ($2^A \trianglelefteq_{elems} A^\star$), calcule a simulação do cálculo do segmento inicial de um número natural, isto é, resolva em ordem a x o seguinte diagrama de refinamento:

$$\begin{tikzcd} I\!N_0 \arrow[r, "\lambda n.\overline{n}"] & 2^{I\!N} \\ I\!N_0 \arrow[u, "id"] \arrow[r, "x"'] & I\!N^\star \arrow[u, "elems"'] \end{tikzcd}$$

Será o resultado do seu cálculo único? Justifique.

□

Exercício 9.12 Justifique cuidadosamente o raciocínio que nos permitiu passar de (9.13) para (9.14).

□

Exercício 9.13 A função

$$f(n, r) \stackrel{\mathrm{def}}{=} 2^{id \times (\lambda x . x \times n)}(r)$$

implementa, a nível relacional, uma operação que conhece sobre multiconjuntos. Identifique-a e reconstitua o respectivo processo de cálculo.

□

Exercício 9.14 O seguinte ciclo-`while`,

```
{   M r = u;
    Y y' = y;
    while (¬ p(y'))
        {   r = r θ d(y');
            y' = e(y');
        };
    r = r θ v;
}
```

registado na notação "pseudo-C" usada neste texto, foi proposto como concretização algorítmica da seguinte função

$$\begin{array}{rcl} f & : & Y \longrightarrow M \\ f(y) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{rcl} p(y) & \Rightarrow & v \\ \neg(p(y)) & \Rightarrow & d(y)\ \theta\ f(e(y)) \end{array} \right. \end{array}$$

onde θ é associativa e tem u como elemento neutro.

Discuta a correcção dessa concretização algorítmica, justificando formalmente.

□

Exercício 9.15 Relembre o modelo $BAMS$ da secção 7.5 cujo refinamento em $BAMSR$ (relacional) foi calculado na secção 8.5.1.

Calcule a implementação relacional, ao nível $BAMSR$, do operador de *fecho* de uma conta, *i.é*, resolva em ordem a ω o seguinte diagrama de refinamento:

$$\begin{tikzcd} AccNr & \times & BAMS \arrow[r, "remAcc"] & BAMS \\ AccNr \arrow[u, "id"] & \times & BAMSR \arrow[u, "f"'] \arrow[r, "\omega"'] & BAMSR \arrow[u, "f"'] \end{tikzcd}$$

onde

$$remAcc(k, \sigma) \stackrel{\rm def}{=} \sigma \setminus \{k\} \tag{9.52}$$

Sugestão: Poder-lhe-á ser útil recorrer ao seguinte operador relacional de *eliminação* de tuplos,

$$del(\langle i, a\rangle, t) \stackrel{\rm def}{=} \{r \in t \mid \; a \neq \pi_i(r)\}$$

que é, afinal, o oposto de sel (1.55).
□

Exercício 9.16 No contexto do exercício anterior, suponha que o banco, que já tem o relacional a funcionar, pretende uma nova operação que faça imprimir automaticamente uma carta a enviar a todos os titulares cujas contas têm saldo inferior a um dado valor q.

1. Especifique um novo operador sobre $BAMS$ que selecione o conjunto desses titulares para uma qualquer $q \in Amount$.
2. Calcule a implementação relacional desse operador ao nível $BAMSR$.

□

Exercício 9.17 No contexto dos exercícios anteriores, suponha que o Gabinete de Relações Públicas (GRP) de uma universidade pretende um sistema de informação para catalogação de recortes de jornais (notícias que referem a Universidade) que permanentemente estão a ser selecionados dos jornais pelo GRP e arquivados em pastas.

Suponha que, após ter chegado ao seguinte modelo de dados para a base:

$$\begin{array}{rcll}
GRP & \cong & \#Cota \rightharpoonup (2^{Assunto} \times Dados) & \\
Dados & \cong & T : Texto & \times \quad \textit{/*título da notícia */} \\
 & & J : Jornal & \times \quad \textit{/*jornal em que apareceu a notícia */} \\
 & & D : Data & \times \quad \textit{/*data do jornal */} \\
 & & P : I\!N & \times \quad \textit{/*página do jornal */} \\
 & & D : \#Doss & \times \quad \textit{/*dossier onde foi arquivado o recorte */}
\end{array}$$

(onde $Texto$, $Jornal$, *etc.*são tipos a definir apropriadamente) você tem ainda que especificar e implementar operações para os seguintes propósitos:

- Obter os assuntos por que está catalogado um dado recorte (9.53)
- Recatalogar um recorte acrescentando-lhe mais um assunto (9.54)
- Inserir um novo recorte (ainda sem classificação) sabidos os seus dados e a respectiva cota (9.55)
- Obter as cotas de todos os recortes que versam um dado assunto (9.56)
- Obter os dados de um recorte dada a sua cota (9.57)
- Eliminar um dado recorte (9.58)

1. Faça a analogia formal entre GRP e $BAMS$, identificando um modelo mais abstracto do qual tanto $BAMS$ como GRP são especializações.

2. Suponha que já tem $BAMS$ implementado em ambiente relacional ($BAMSR$). Pretende-se agora implementar GRP re-utilizando, ao máximo, as implementações já disponíveis de operações sobre $BAMS$, nomeadamente $remAcc$ (9.52), $addAccHolder$ (9.11) e algumas outras cuja semântica se especifica a seguir:

$$\begin{array}{rcl} initBAMS & : & \rightarrow BAMS \\ initBAMS & \stackrel{\rm def}{=} & \left(\quad \right) \end{array}$$

$$\begin{array}{rcl} newAcc & : & AccNr \times AccHolder \times BAMS \rightarrow BAMS \\ newAcc & \stackrel{\rm def}{=} & \lambda(k,t,\sigma). \\ & & \left\{ \begin{array}{ccl} k \in dom(\sigma) & \Rightarrow & \sigma \\ \neg(k \in dom(\sigma)) & \Rightarrow & \sigma \cup \left(\begin{array}{c} k \\ \langle \{t\}, 0 \rangle \end{array} \right) \end{array} \right. \end{array}$$

$$\begin{array}{rcl} depIntoAcc & : & AccNr \times Amount \times BAMS \rightarrow BAMS \\ depIntoAcc & \stackrel{\rm def}{=} & \lambda(k,q,\sigma). \\ & & \left\{ \begin{array}{cccl} k \in dom(\sigma) & \Rightarrow & let & x = \sigma(k) \\ & & & t_k = \pi_1(x) \\ & & & q_k = \pi_2(x) \\ & & in & \sigma \dagger \left(\begin{array}{c} k \\ \langle t_k, q_k + q \rangle \end{array} \right) \\ \neg(k \in dom(\sigma)) & \Rightarrow & \sigma & \end{array} \right. \end{array}$$

$$\begin{array}{rcl} drawFromAcc & : & AccNr \times Amount \times BAMS \rightarrow BAMS \\ drawFromAcc & \stackrel{\rm def}{=} & \lambda(k,q,\sigma). \\ & & \left\{ \begin{array}{cccl} k \in dom(\sigma) & \Rightarrow & let & x = \sigma(k) \\ & & & t_k = \pi_1(x) \\ & & & q_k = \pi_2(x) \\ & & in & \left\{ \begin{array}{ccl} q \leq q_k & \Rightarrow & \sigma \dagger \left(\begin{array}{c} k \\ \langle t_k, q_k - q \rangle \end{array} \right) \\ \neg(q \leq q_k) & \Rightarrow & \sigma \end{array} \right. \\ \neg(k \in dom(\sigma)) & \Rightarrow & \sigma & \end{array} \right. \end{array}$$

$$\begin{array}{rcl} getAccHols & : & AccNr \times BAMS \rightarrow 2^{AccHolder} \\ getAccHols & \stackrel{\rm def}{=} & \lambda(k,\sigma). \\ & & \left\{ \begin{array}{ccl} k \in dom(\sigma) & \Rightarrow & \pi_1(\sigma(k)) \\ \neg(k \in dom(\sigma)) & \Rightarrow & \emptyset \end{array} \right. \end{array}$$

$$\begin{array}{rcl} allAccOf & : & AccHolder \times BAMS \rightarrow 2^{AccNr} \\ allAccOf & \stackrel{\rm def}{=} & \lambda(t,\sigma).\{k \in dom(\sigma) \mid \ t \in \pi_1(\sigma(k))\} \end{array}$$

$$\begin{array}{rcl} getAccBal & : & AccNr \times BAMS \rightarrow Amount \\ getAccBal & \stackrel{\rm def}{=} & \lambda(k,\sigma). \left\{ \ k \in dom(\sigma) \quad \Rightarrow \quad \pi_2(\sigma(k)) \right. \end{array}$$

Na 2.ª coluna da tabela seguinte indique operações de (9.53) a (9.58) sobre GRP que se podem implementar por simples reutilização das correspondentes operações sobre $BAMS$ (na 1.ª coluna):

$BAMS$	GRP
$initBAMS$	
$newAcc$	
$depIntoAcc$	
$drawFromAcc$	
$getAccHols$	
$allAccOf$	
$getAccBal$	

(**NB:** para cada operação não-reutilizável indique a razão.)

3. Selecione uma das operações (9.53) a (9.58) que ficaram de fora da tabela acima. Especifique-a e calcule a respectiva implementação relacional.

□

Exercício 9.18 Numa determinada implementação de SQL as tabelas relacionais ($2^{A_1 \times \ldots \times A_n}$) encontram-se implementadas sobre ficheiros simples, *i.é* como sequências em $(A_1 \times \ldots \times A_n)^\star$.

Calcule a simulação sobre ficheiros (sequências) do operador relacional de selecção:

$$sel : (\textstyle\sum_{i=1}^{n} A_i) \times 2^{A_1 \times \ldots \times A_n} \longrightarrow 2^{A_1 \times \ldots \times A_n}$$
$$sel(\langle i, a\rangle, t) \stackrel{\text{def}}{=} \{r \in t \mid a = \pi_i(r)\}$$

□

Exercício 9.19 Uma estrutura de informação da classe das listas é a chamada *lista de interesse*. Trata-se de uma lista em que o último elemento sobre o qual se manifestou interesse (por consulta ou inserção) "salta" automaticamente para cabeça da lista. A lista pode assim considerar-se ordenada pela ordem decrescente de "interesse" manifestada até ao momento pelos seus elementos, o que lhe confere um carácter adaptativo.

Teremos então, para um dado domínio de dados não-vazio A:

$$IList \quad \cong \quad A^\star$$

1. Calcule uma concretização algorítmica não recursiva da seguinte operação de inserção em $IList$,

$$ilInsert : IList \times A \to IList$$
$$ilInsert(l, a) \stackrel{\text{def}}{=} cons(a, filter(l, a))$$

em que se recorre à seguinte função auxiliar:

$$filter(l, a) \stackrel{\text{def}}{=}$$
$$\left\{ \begin{array}{lcl} l = <> & \Rightarrow & l \\ \neg(l = <>) & \Rightarrow & \left\{ \begin{array}{lcl} a = head(l) & \Rightarrow & filter(tail(l), a) \\ \neg(a = head(l)) & \Rightarrow & cons(head(l), filter(tail(l), a)) \end{array} \right. \end{array} \right.$$

2. Estará a seguinte especificação da operação de consulta

$$\begin{aligned} &ilFind : IList \times A \to 2 \\ &ilFind(l, a) \stackrel{\mathrm{def}}{=} a \in elems(l) \end{aligned}$$

   coerente com a descrição de *lista de interesse* que acima se deu? Justifique informalmente.

3. Decorre das operações que manipulam $IList$ que esta estrutura possui um invariante associado. Formule-o. Será que esse invariante pode ser explorado na optimização da função $ilInsert$? Justifique.

□

Exercício 9.20 Na interface com o utilizador de serviços sobre bases de dados torna-se inúmeras vezes necessário fazer passar para a camada de apresentação resultados estruturados de operações da camada computacional. Um caso típico é a visualização de resultados que são funções parciais finitas,

$$A \rightharpoonup B$$

e que se podem mostrar no écran sob a forma de tabelas, mapas, folhas de cálculo *etc.* Linguagens como o Visual C/C++ são particularmente adequadas para o efeito, recebendo essas funções da camada computacional sob a forma de listas ligadas de pares:

```
typedef struct L {
      A key;
      B data;
      struct L *next;
};
```

o que faz sentido tendo em conta um refinamento bem conhecido:

$$\begin{array}{lll} A \rightharpoonup B & \trianglelefteq^{fdp}_{mkf} & 2^{A \times B} \\ & \trianglelefteq_{elems} & (A \times B)^\star \\ & \trianglelefteq_{g_{(8.7)}} & L \end{array}$$

para

$$L \cong 1 + (A \times B) \times L$$

e onde $g_{(8.7)}$ é a função g do exercício 8.7.
Interessa pois ter ao dispôr uma pequena biblioteca de funções C/C++ que implementem a álgebra das funções finitas.

1. Complete o seguinte diagrama de refinamento referente a um operador dessa álgebra:

$$\begin{tikzcd} A \rightharpoonup B \arrow[r, "dom"] & ? \\ L \arrow[u, "mkf \circ elems \circ g_{(8.7)}"] \arrow[r, "dom'"'] & ? \arrow[u, "elems \circ g_{(8.7)}"'] \end{tikzcd}$$

2. Calcule uma versão não recursiva de dom'.
3. Indique, justificando, qual é a operação dessa álgebra que é simulada, ao mesmo nível de refinamento, pela função que se segue:

$$\begin{array}{l} \ldots : \ldots \times L \to \ldots \\ \phi(a,l) \stackrel{\text{def}}{=} \{a \in dom'(l) \Rightarrow \begin{array}[t]{ll} let & \begin{array}{l} x = \pi_2(l) \\ p = \pi_1(x) \\ x' = \pi_2(x) \end{array} \\ in & \left\{ \begin{array}{lcl} a = \pi_1(p) & \Rightarrow & \pi_2(p) \\ \neg(a = \pi_1(p)) & \Rightarrow & \phi(a,x') \end{array} \right. \end{array} \end{array}$$

□

Exercício 9.21 Recorde do Exercício 2.53 a definição da função seguinte:

$$\begin{array}{l} subl : A^\star \times I\!N \times I\!N_0 \to A^\star \\ subl(l,i,n) \stackrel{\text{def}}{=} \\ \quad \left\{ \begin{array}{lcl} n = 0 \vee l = <> & \Rightarrow & <> \\ n > 0 \wedge l \neq <> & \Rightarrow & \begin{array}[t]{ll} let & t = tail(l) \\ in & \left\{ \begin{array}{lcl} i = 1 & \Rightarrow & <head(l)> \frown subl(t,i,n-1) \\ i > 1 & \Rightarrow & subl(t,i-1,n) \end{array} \right. \end{array} \end{array} \right. \end{array}$$

que extrai, de uma sequência l, a sua subsequência de n elementos (ou de tantos quantos fôr possível extrair) que começa na i-ésima posição.

Calcule uma implementação iterativa de $subl$ e escreva-a sob a forma de um ciclo-`while` na notação “pseudo-C” usada neste texto.

(**Sugestão**: comece por transformar $subl$ numa instância do esquema linear monádico.)

□

Exercício 9.22 O seguinte modelo de dados,

$$\begin{array}{rcl} DPP & \cong & \#Equipment \rightharpoonup Structure \\ Structure & \cong & \#Unit \rightharpoonup Quantity \\ \#Unit & \cong & \#Component + \#Equipment \\ Quantity & \cong & I\!N \\ Parts & \cong & \#Component \rightharpoonup Quantity \end{array}$$

(onde $\#Equipment$ e $\#Component$ são espécies atómicas) é uma versão simplificada daquele que foi assunto da secção 8.9.1, suposto sujeito a um invariante que, como se viu, garante a correcta definição de árvores de produção finitas.

Uma das operações mais interessantes que se definiram sobre DPP é aquela que calcula a *explosão* de um equipamento no seu total de peças envolvidas,

$$\begin{array}{rcl} explode & : & \#Equipment \times DPP \longrightarrow Parts \\ explode(\#e,\sigma) & \stackrel{\text{def}}{=} & \begin{array}[t]{ll} let & f = apply(\sigma,\#e) \\ in & \bigoplus_{x \in dom(f)} \left\{ \begin{array}{lcl} x = \langle 1,k \rangle & \Rightarrow & \left(\begin{array}{c} k \\ f(x) \end{array} \right) \\ x = \langle 2,k \rangle & \Rightarrow & f(x) \otimes explode(k,\sigma) \end{array} \right. \end{array} \end{array}$$

onde $\oplus$ e $\otimes$ designam operações sobre multiconjuntos que deve conhecer e $apply$ é a seguinte função auxiliar:

$$\begin{array}{rcl} apply & : & DPP \times \#Equipment \longrightarrow Structure \\ apply(\sigma, k) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{lcl} k \in dom(\sigma) & \Rightarrow & \sigma(k) \\ k \notin dom(\sigma) & \Rightarrow & \left(\quad\right) \end{array} \right. \end{array}$$

1. Mostre que o facto seguinte se verifica,

$$explode(k, \sigma) \quad = \quad aux(apply(\sigma, k), \sigma)$$

onde aux é a seguinte função:

$aux : Structure \times DPP \longrightarrow Parts$

$aux(f, \sigma) \stackrel{\mathrm{def}}{=}$

$$\left\{ \begin{array}{lcl} f = \left(\quad\right) & \Rightarrow & \left(\quad\right) \\ f \neq \left(\quad\right) & \Rightarrow & let \quad x \in dom(f) \\ & & in \quad \left(\left\{ \begin{array}{lcl} x = \langle 1, k\rangle & \Rightarrow & \left(\begin{array}{c} k \\ f(x) \end{array} \right) \\ x = \langle 2, k\rangle & \Rightarrow & f(x) \otimes aux(apply(\sigma, k), \sigma) \end{array} \right. \right) \oplus aux(f \setminus \{x\}, \sigma) \end{array} \right.$$

2. Assuma como verdadeiros os factos seguintes sobre aux,

$$n \otimes aux(f, \sigma) \quad = \quad aux(n \otimes f, \sigma) \tag{9.59}$$

$$aux(f_1, \sigma) \oplus aux(f_2, \sigma) \quad = \quad aux(f_1 \oplus f_2, \sigma) \tag{9.60}$$

e mostre que eles são suficientes para obter a seguinte concretização não-recursiva de $aux(f, \sigma)$, expressa aqui em notação "pseudo-C":

```
{   Parts r = ();
    Structure ff = f;
    Quantity n ;
    Unit x;
    while (ff != ())
        {   x = choice(dom(ff)) ;
            n = ff(x);
            ff = ff\{x};
            if x == (1,k) r = r ⊕ ( k n ) ;
            if x == (2,k) ff = ff ⊕ (n ⊗ apply(σ,k));

        }
}
```

□

Exercício 9.23 Recorde o modelo de dados $ActPlan$ que é assunto do Exercício 2.70 e do qual se pode deduzir a seguinte implementação relacional,

$$ActPlan_6 \quad = \quad 2^{\#Act \times Description \times Span} \times 2^{\#Act \times \#Res \times I\!N} \times 2^{\#Act \times \#Act}$$

calculada após os 6 passos de refinamento que se seguem:

$$
\begin{array}{lll}
ActPlan & \cong & \#Act \rightharpoonup STR \times \mathbb{N} \times (\#Res \rightharpoonup \mathbb{N}) \times 2^{\#Act} \\
 & \cong_1 & \#Act \rightharpoonup ((STR \times \mathbb{N}) \times (\#Res \rightharpoonup \mathbb{N})) \times (\#Act \rightharpoonup 1) \\
 & \trianglelefteq_2 & (\#Act \rightharpoonup ((STR \times \mathbb{N}) \times (\#Res \rightharpoonup \mathbb{N}))) \times (\#Act \times \#Act \rightharpoonup 1) \\
 & \trianglelefteq_3 & ((\#Act \rightharpoonup STR \times \mathbb{N}) \times (\#Act \times \#Res \rightharpoonup \mathbb{N})) \times 2^{\#Act \times \#Act} \\
 & \cong_4 & (\#Act \rightharpoonup STR \times \mathbb{N}) \times (\#Act \times \#Res \rightharpoonup \mathbb{N}) \times 2^{\#Act \times \#Act} \\
 & \trianglelefteq_5 & 2^{\#Act \times (STR \times \mathbb{N})} \times 2^{(\#Act \times \#Res) \times \mathbb{N}} \times 2^{\#Act \times \#Act} \\
 & \cong_6 & 2^{\#Act \times STR \times \mathbb{N}} \times 2^{\#Act \times \#Res \times \mathbb{N}} \times 2^{\#Act \times \#Act}
\end{array}
$$

isto é, tal que:

$\cong_1$	$\{\ f_1 = \#Act \rightharpoonup \langle \pi_1 \circ \pi_1 \circ \pi_1, \pi_2 \circ \pi_1 \circ \pi_1, \pi_2 \circ \pi_1, dom \circ \pi_2 \rangle$	*cf.* (8.64) e (8.23)
$\trianglelefteq_2$	$\begin{cases} f_2 = \bowtie \\ \phi_2 = dpi \end{cases}$	*cf.* (8.72)
$\trianglelefteq_3$	$\begin{cases} f_3 = \bowtie \times g \\ \phi_3 = dpi \times \lambda r.V \end{cases}$	*cf.* (8.72)
$\cong_4$	$\{\ f_4 = \langle \langle \pi_1, \pi_2 \rangle, \pi_3 \rangle$	*cf.* (8.23)
$\trianglelefteq_5$	$\begin{cases} f_5 = mkf \times mkf \times id \\ \phi_5 = fdp \times fdp \times \lambda r.V \end{cases}$	*cf.* (8.9)
$\cong_6$	$\{\ f_6 = 2^{\langle \pi_1, \langle \pi_2, \pi_3 \rangle \rangle} \times 2^{\langle \langle \pi_1, \pi_2 \rangle, \pi_3 \rangle} \times id$	*cf.* (8.23)

onde

$$
g(r) \quad = \quad \left(\begin{array}{c} p \\ NIL \end{array} \right)_{p \in r}
$$

1. Simplifique a expressão

$$
f \quad \stackrel{\text{def}}{=} \quad \bigcirc_{i=1}^{6} f_i
$$

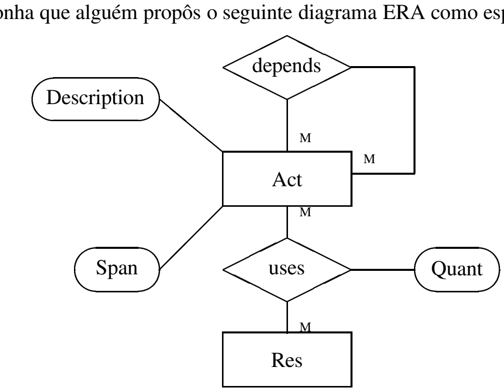
que exprime a função de abstracção global.

2. Calcule a simulação de $bestStart$ sobre $ActPlan_6$, onde $bestStart$ é a função:

$$
bestStart \quad : \quad \#Act \times ActPlan \longrightarrow \mathbb{N}_0
$$

$$
\begin{array}{lll}
bestStart(a, db) & \stackrel{\text{def}}{=} & let \quad D = \pi_4(db(a)) \\
 & & in \quad \left\{ \begin{array}{lll} D = \{\} & \Rightarrow & 0 \\ D \neq \{\} & \Rightarrow & \underset{aa \in D \cap dom(db)}{MAX} \quad bestStart(aa, db) + \pi_2(db(aa)) \end{array} \right.
\end{array}
$$

3. Suponha que alguém propôs o seguinte diagrama ERA como especificação de $Actplan$:



Com base na semântica em SETS que estudou para diagramas deste tipo, compare (formalmente) o modelo implícito neste diagrama com o modelo $ActPlan$ acima proposto.

□

9.6 Notas Bibliográficas

O cálculo 'fold/unfold' foi um tópico de intensa actividade de investigação nos anos 70. Ver [BD77, Dar82, Bp82] para as principais omissões da sua apresentação neste capítulo.

A aplicação explícita do cálculo 'fold/unfold' a um problema de refinamento algorítmico aparece pela primeira vez bem caracterizado no trabalho pioneiro de Darlington [Dar82, Dar84]. A estratégia aí proposta é desenvolvida em [Oli85] (a principal fonte para as secções 9.1, 9.2 e 9.4.2) com vista a converter em cálculo o tradicional desenvolvimento por 'invent and verify'. Por exemplo, a solução (9.9) é curiosamente semelhante a uma função $intersect$ que é primeiro proposta e depois verificada por indução sobre conjuntos na p.145 de [Jon80].

Ver [Oli85, Oli94] para mais informação sobre a aplicação das técnicas clássicas da "transformação de programas" ao refinamento algorítmico. Na referência [Oli82] é apresentado um "catálogo" de regras de desrecursivação do tipo das que foram estudadas neste capítulo.

Mas a principal omissão é, sem dúvida, a do chamado *Cálculo de Oxford*, recentemente desenvolvido por Caroll Morgan (a partir do chamado cálculo de *transformadores de predicados* de Dijkstra [Dij76]) e que o leitor interessado encontrará descrito num conhecido livro de texto [Mor90]. Como se mostra em [Oli92], este cálculo é ortogonal em relação a SETS. De facto, uma vez calculado o invariante de abstracção em em SETS, poderemos usar o Cálculo de Oxford para completar o refinamento algorítmico.

Bibliografia

[AKM81] M. A. Arbib, A. Kfoury, and R. N. Mool. *A Basis for Theoretical Computer Science*. Texts and Monographs in Computer Science. Springer-Verlag, 1981. D. Gries (Ed.).

[Bak80] J. W. Bakker. *Mathematical Theory of Program Correctness*. Series in Computer Science. Prentice-Hall International, 1980. C. A. R. Hoare.

[BD77] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *JACM*, 24(1):44–67, January 1977.

[BJ82] D. Bjorner and C. B. Jones. *Formal Specification and Software Development*. Series in Computer Science. Prentice-Hall International, 1982. C. A. R. Hoare.

[BL84] R. Burstall and B. Lampson. A kernel language for abstract data types and modules. *LNCS*, 173:1–50, 1984.

[Bp82] F. L. Bauer and H. Wössner. *Algorithmic Language and Program Development*. Springer-Verlag, 1982.

[Bur84] R. Burstall. Programming with modules as typed functional programming. In *Proc. Int. Conf. on* 5^{th} *Generation Computing Systems, Tokyo*, Nov. 1984.

[BWP84] M. Broy, M. Wirsing, and C. Pair. A systematic study of models of abstract data types. *TCS*, 33:139–174, 1984.

[Cos89] J. F. Costa. *Teoria Algébrica de Processos Animados*. I.S.T. de Lisboa, 1989. Tese de Mestrado.

[Dar82] J. Darlington. Program transformation. In *Funct. Prog. and Its Applications: An Advanced Course*. Cambridge Univ. Press, 1982.

[Dar84] J. Darlington. *The Design of Efficient Data Representations*, chapter 7, pages 139–156. Macmillan, London, 1984.

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer-Verlag, New-York, 1985.

[E.S89] ROSE Consortium E.S.F. The rose subproject. In *Workshop on REUSE of Software Development Component*, Berlin, 25–26 October 1989. Joint Organisation CEC-ESF.

[Fie80] E. Fielding. The specification of abstract mappings and their implementation as B^+-trees. Technical Report PRG-18, Oxford University, September 1980.

[Flo67] R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19, pages 19–32. American Mathematical Society, 1967. Proc. Symposia in Applied Mathematics.

[GH78] J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

[Gor79] M. J. C. Gordon. *The Denotational Description of Programming Language: An Introduction*. Springer Verlag, 1979.

[GR83] A. Goldberg and D. Robson. *Smalltalk'80: the Language and Its Implementation*. Addison-Wesley, 1983.

[GTW78] J. Goguen, J. W. Thatcher, and E. G. Wagner. *Initial Algebra Approach to the Specification, Correctness and Implementation of Algebraic Data Types*, volume IV, chapter ?, pages ?–? Prentice-Hall, 1978.

[GTWA77] J. Goguen, J. W. Thatcher, E. G. Wagner, and Wright J. B. (ADJ). Initial algebra semantics and continuous algebras. *JACM*, 24(1):68–95, 1977.

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

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *CACM*, 12,10:576–580, 583, October 1969.

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

[HS78] E. Horowitz and S. Sahni. *Fundamentals of Data Structures*. Computer Software Engineering Series. Pitman, 1978. E. Horowitz (Ed.).

[JG86] G. Jones and M. Goldsmith. *Programming in OCCAM*. Series in Computer Science. Prentice-Hall International, 1986. C. A. R. Hoare.

[Jon80] C. B. Jones. *Software Development — A Rigorous Approach*. Series in Computer Science. Prentice-Hall International, 1980. C. A. R. Hoare.

[Jon86] C. B. Jones. *Systematic Software Development Using* VDM. Series in Computer Science. Prentice-Hall International, 1986. C. A. R. Hoare.

[Jou92] I. S. Jourdan. Reificação de tipos abstractos de dados: Uma abordagem matemática. Master's thesis, University of Coimbra, 1992. (In Portuguese).

[Knu74] D. Knuth. Structured programming with goto statements. *ACM Comp. Surveys*, 6(4), Dec. 1974.

[MA86] E. G. Manes and M. A. Arbib. *Algebraic Approaches to Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 1986. D. Gries, series editor.

[Mac71] S. MacLane. *Categories for the Working Mathematician*. Springer-Verlag, New-York, 1971.

[Man74] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, New-York, 1974.

[McC63] J. McCarthy. Towards a mathematical science of computation. In C. M. Popplewell, editor, *Proc. of* IFIP *62*, pages 21–28, Amsterdam-London, 1963. North-Holland Pub. Company.

[Mil78] R. Milner. A type polymorphism in programming. *JCSS*, 17:348–375, 1978.

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

[Mor90] C. Morgan. *Programming from Specification*. Series in Computer Science. Prentice-Hall International, 1990. C. A. R. Hoare, series editor.

[OC93] J. N. Oliveira and A. M. Cruz. Formal calculi applied to software component knowledge elicitation. Technical Report C19-WP2D, DI/INESC, December 1993. IMI Contract *C.1.9. Sviluppo di Metodologie, Sistemi e Servizi Innovativi in Rete*.

[Oli82] J. N. Oliveira. Using FP as a tool for program transformation and proving in a formal dataflow system. Technical Report Dataflow Group-JNO3/82, Department of Computer Science, University of Manchester, August 1982.

[Oli85] J. N. Oliveira. The transformational paradigm as a means of smoothing abrupt softwar e design steps. Technical Report CCES-JNO:R2/85, U. Minho, December 1985.

[Oli90] J. N. Oliveira. A reification calculus for model-oriented software specification. *Formal Aspects of Computing*, 2(1):1–23, April 1990.

[Oli92] J. N. Oliveira. Software reification using the SETS calculus. In *Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK*, pages 140–171. Springer-Verlag, 8–10 January 1992. (Invited paper).

[Oli93] J. N. Oliveira. Components & compositionality in SOUR, 25 Februrary 1993. SOUR Meeting, INESC Lisbon.

[Oli94] J. N. Oliveira. Hash tables — a case study in $\trianglelefteq$-calculation. Technical Report DI/INESC 94-12-1, DI/INESC, U. Minho, December 1994.

[Par80] D. Park. On the semantics of fair parallelism. *LNCS*, 80:504–526, 1980.

[Rod93] C. J. Rodrigues. Sobre o desenvolvimento formal de bases de dados. Master's thesis, University of Minho, 1993. (In Portuguese).

[SFSE87] A. Sernadas, J. Fiadeiro, C. Sernadas, and H.-D. Ehrich. Abstract object types: A temporal perspective. In B. Banieqbal A. Pnueli, H. Barringer, editor, *Colloquium on Temporal Logic and Specification*. Springer-Verlag, 1987?

[Spi89] J. M. Spivey. *The Z Notation — A Reference Manual*. Series in Computer Science. Prentice-Hall International, 1989. C. A. R. Hoare.

[Sto81] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. The MIT Press, Cambridge Mass., 1981.

[Wan79] M. Wand. Final algebraic semantics and data type extensions. *JCSS*, 19:27–44, 1979.

[Wir76] N. Wirth. *Algorithms + Data Structures = Programs*. Prentice-Hall, 1976.

[Wol88] M. Wolczko. Semantics of object-oriented languages. Technical Report UMCS-88-6-1, U. Manchester, 1988.

Apêndice A

Tábua de Propriedades Básicas

Este anexo apresenta um repertório de propriedades elementares da teoria dos conjuntos invocadas no presente texto. O anexo não se pretende ser nem exaustivo nem minimal. As propriedades apresentam-se organizadas segundo as classes dos operadores básicos que suportam a especificação formal construtiva proposta ao longo do texto.

Usa-se a simbologia tradicional. Supoem-se universalmente quantificadas todas as variáveis livres.

A.1 Cálculo Básico de Condições

$$\frac{}{p \wedge q \Rightarrow p} \tag{A.1}$$

$$\frac{p \Rightarrow q}{\neg q \Rightarrow \neg q} \tag{A.2}$$

$$\frac{V \Rightarrow p}{p} \tag{A.3}$$

$$\frac{F \Rightarrow p}{V} \tag{A.4}$$

$$\frac{\forall x \in A : p(x) \wedge \forall x \in A : q(x)}{\forall x \in A : p(x) \wedge q(x)} \tag{A.5}$$

$$\frac{\forall x \in A : p(x), S \subseteq A}{\forall x \in S : p(x)} \tag{A.6}$$

$$\neg(p \Rightarrow q) \quad \leftrightarrow \quad p \wedge \neg q \tag{A.7}$$
$$p \Rightarrow q \quad \leftrightarrow \quad \neg p \vee q \tag{A.8}$$
$$\neg(\forall x \in A : p(x)) \quad \leftrightarrow \quad \exists x \in A : \neg p(x) \tag{A.9}$$

A.1.1 Condições sobre Conjuntos

$$A \subseteq B \quad \leftrightarrow \quad \forall x \in A : x \in B \tag{A.10}$$
$$a \in A \cup B \quad \leftrightarrow \quad a \in A \vee a \in B \tag{A.11}$$
$$a \in A \cap B \quad \leftrightarrow \quad a \in A \wedge a \in B \tag{A.12}$$
$$a \in \{x \in A \mid \ p(x)\} \quad \leftrightarrow \quad a \in A \wedge p(a) \tag{A.13}$$
$$a \notin A \quad \leftrightarrow \quad \{a\} \cap A = \emptyset \tag{A.14}$$
$$A \cap B = \emptyset \quad \leftrightarrow \quad A \subseteq \overline{B} \tag{A.15}$$
$$A \subseteq B \wedge A \subseteq C \quad \leftrightarrow \quad A \subseteq B \cap C \tag{A.16}$$

A.2 Igualdades Elementares de Conjuntos

$$A - B \quad = \quad A \cap \overline{B} \tag{A.17}$$
$$A \quad = \quad \{a \mid \ a \in A\} \tag{A.18}$$
$$A \quad = \quad \bigcup_{a \in A} \{a\} \tag{A.19}$$
$$\bigcup_{i \in I} \{the(A_i)\} \quad = \quad \bigcup_{i \in I} A_i \tag{A.20}$$
$$|\emptyset| \quad = \quad 0 \tag{A.21}$$
$$|A \times B| \quad = \quad |A| \times |B| \tag{A.22}$$
$$|A \cup B| \quad \leq \quad |A| \cup |B| \tag{A.23}$$
$$A \cap A = A \quad , \quad A \cup A = A \tag{A.24}$$
$$A \cap B = B \cap A \quad , \quad A \cup B = B \cup A \tag{A.25}$$
$$A \cap (B \cap C) = (A \cap B) \cap C \quad , \quad A \cup (B \cup C) = (A \cup B) \cup C \tag{A.26}$$
$$A \cap (B \cup C) \quad = \quad (A \cap B) \cup (A \cap C) \tag{A.27}$$
$$A \cup (B \cap C) \quad = \quad (A \cup B) \cap (A \cup C) \tag{A.28}$$
$$A - A \quad = \quad \emptyset \tag{A.29}$$
$$A - (B - C) \quad = \quad A - (B \cup C) \tag{A.30}$$

A.3 Funções Parciais Finitas

Para quaisquer funções parciais finitas f, g e h, tem-se [1]:

$$f \cup g = g \cup f \tag{A.31}$$
$$f \cup f = f \tag{A.32}$$
$$f \dagger f = f \tag{A.33}$$
$$f \cup (g \cup h) = (f \cup g) \cup h \tag{A.34}$$
$$f \dagger (g \dagger h) = (f \dagger g) \dagger h \tag{A.35}$$
$$f = f \cup (\quad) \tag{A.36}$$
$$f = (\quad) \cup f \tag{A.37}$$
$$f = f \dagger (\quad) \tag{A.38}$$
$$f = (\quad) \dagger f \tag{A.39}$$
$$dom(f \cup g) = dom(f) \cup dom(g) \tag{A.40}$$
$$dom(f \dagger g) = dom(f) \cup dom(g) \tag{A.41}$$
$$rng(f \dagger g) = rng(f) \cup rng(g) \tag{A.42}$$
$$rng(f \cup g) = rng(f) \cup rng(g) \tag{A.43}$$
$$(f \cup g) \setminus S = (f \setminus S) \cup (g \setminus S) \tag{A.44}$$
$$(f \dagger g) \setminus S = (f \setminus S) \dagger (g \setminus S) \tag{A.45}$$
$$(f \setminus S) \setminus R = f \setminus (S \cup R) \tag{A.46}$$
$$f \circ (g \setminus S) = (f \circ g) \setminus S \tag{A.47}$$
$$f \mid (S \cup R) = (f \mid S) \cup (f \mid R) \tag{A.48}$$
$$(f \mid R) \mid S = (f \mid S) \mid R \tag{A.49}$$
$$(f \mid R) \setminus R = f \mid \emptyset \tag{A.50}$$
$$f \setminus S \cup f \mid S = f \tag{A.51}$$
$$(f \setminus R) \mid S = f \mid (S - R) \tag{A.52}$$
$$S \cap dom(f) = \emptyset \Rightarrow f \setminus S = f \tag{A.53}$$
$$dom(f \setminus S) = dom(f) - S \tag{A.54}$$
$$dom(f \mid S) = dom(f) \cap S \tag{A.55}$$
$$f[dom(f)] = rng(f) \tag{A.56}$$
$$f \dagger g = f \setminus dom(g) \cup g \tag{A.57}$$
$$f \mid dom(f) = f \tag{A.58}$$
$$(f \dagger g) \mid S = (f \mid S) \dagger (g \mid S) \tag{A.59}$$

[1] Sempre que ocorre o operador de *união* de funções ($\cup$) supõe-se, adicionalmente, que os respectivos operandos são funções coerentes, *cf.* Definição 1.5.

$$f = f \setminus S \cup f \mid S \tag{A.60}$$
$$f \setminus dom(f) = (\quad) \tag{A.61}$$
$$f \setminus (S \cap R) = f \setminus S \cup f \setminus R \tag{A.62}$$
$$(\quad) \setminus S = (\quad) \tag{A.63}$$
$$(\quad) \mid S = (\quad) \tag{A.64}$$

A.4 Sequências

Sejam s, r, t sequências em $A^\star$, $f : A \longrightarrow B$ e $g : B \longrightarrow C$ aplicações, e a um qualquer elemento de A. Tem-se:

$$head(cons(a, s)) = a \tag{A.65}$$
$$tail(cons(a, s)) = s \tag{A.66}$$
$$s \frown <> = s \tag{A.67}$$
$$<> \frown s = s \tag{A.68}$$
$$s \frown (r \frown t) = (s \frown r) \frown t \tag{A.69}$$
$$g^* \circ f^* = (g \circ f)^* \tag{A.70}$$
$$f^*(s \frown r) = f^*(s) \frown f^*(r) \tag{A.71}$$
$$<a> \frown s = cons(a, s) \tag{A.72}$$
$$length(<>) = 0 \tag{A.73}$$
$$length(cons(a, s)) = 1 + length(s) \tag{A.74}$$
$$length(s \frown r) = length(s) + length(r) \tag{A.75}$$
$$length \circ f^* = length \tag{A.76}$$
$$elems(<>) = \emptyset \tag{A.77}$$
$$elems(cons(a, s)) = \{a\} \cup elems(s) \tag{A.78}$$
$$elems(s \frown r) = elems(s) \cup elems(r) \tag{A.79}$$
$$elems \circ f^* = 2^f \circ elems \tag{A.80}$$
$$elems(<f(a) \mid a \leftarrow s \wedge p(a)>) \subseteq f[elems(s)] \tag{A.81}$$

Para $s \neq <>$ e $1 \leq i \leq length(s)$, tem-se

$$tail(s)(i) = s(i + 1) \tag{A.82}$$
$$tail(s \frown r) = tail(s) \frown r \tag{A.83}$$
$$head(s \frown r) = head(s) \tag{A.84}$$
$$tail \circ f^* = f^* \circ tail \tag{A.85}$$
$$elems(tail(s)) \subseteq elems(s) \tag{A.86}$$

$$length(tail(s)) = length(s) - 1 \tag{A.87}$$
$$cons(head(s), tail(s)) = s \tag{A.88}$$

Apêndice B

Algumas Leis do Cálculo Funcional

A notação funcional que é usada neste texto para exprimir funções — definidas por expressões funcionais condicionais, eventualmente recursivas — goza de propriedades básicas que são conhecidas, no seu conjunto, pela designação de *cálculo 'Fold/Unfold'* (designação que se explicará mais à frente).

Esse cálculo, estudado extensivamente nas duas últimas décadas, será aqui apenas brevemente exposto, partindo de alguns conceitos básicos que a seguir se apresentam. Sem perda de generalidade, exprimir-nos-emos em termos de funções unárias. A extensão coerente dessas funções com argumentos extra não põe problemas especiais.

Como se estudou na secção 3.4.2, o valor de uma expressão matematicamente indefinida será representado por um símbolo especial, $\perp$, que define matematicamente a noção de "cálculo abortado" ou de "computação que não termina". $\perp$ é, portanto, "o pior" valor que uma função pode debitar como resultado. Como então vimos, esta ideia pode ser formalizada por uma ordem parcial $\leq$ tal que:

$$\perp \leq x$$

para qualquer valor x (é claro que $\perp \leq \perp$). Tem-se ainda que, para quaisquer valores a, b e qualquer função f,

$$\begin{array}{rcl} f(\perp) & = & \perp \\ \left\{ \begin{array}{rcl} \perp & \Rightarrow & a \\ \neg(\perp) & \Rightarrow & b \end{array} \right. & = & \perp \\ \left\{ \begin{array}{rcl} V & \Rightarrow & a \\ \neg(V) & \Rightarrow & \perp \end{array} \right. & = & a \end{array}$$

$$\left\{ \begin{array}{rcl} F & \Rightarrow & \perp \\ \neg(F) & \Rightarrow & b \end{array} \right. = b$$

Na base do cálculo funcional estão algumas leis que se passam a enunciar.

B.1 Distributividade Aplicação/Condição

$$f(x, \left\{ \begin{array}{rcl} p & \Rightarrow & q \\ \neg p & \Rightarrow & r \end{array} \right.) = \left\{ \begin{array}{rcl} p & \Rightarrow & f(x,q) \\ \neg p & \Rightarrow & f(x,r) \end{array} \right. \tag{B.1}$$

B.2 Manipulação de Condições

$$q = \left\{ \begin{array}{rcl} p & \Rightarrow & q \\ \neg p & \Rightarrow & q \end{array} \right. \tag{B.2}$$

$$\left\{ \begin{array}{rcl} p & \Rightarrow & q \\ r & \Rightarrow & q \\ t & \Rightarrow & s \end{array} \right. = \left\{ \begin{array}{rcl} (p \vee r) & \Rightarrow & q \\ t & \Rightarrow & s \end{array} \right. \tag{B.3}$$

$$\left\{ \begin{array}{rcl} (\left\{ \begin{array}{rcl} p & \Rightarrow & q \\ \neg(p) & \Rightarrow & r \end{array} \right.) & \Rightarrow & s \\ \neg((\left\{ \begin{array}{rcl} p & \Rightarrow & q \\ \neg(p) & \Rightarrow & r \end{array} \right.)) & \Rightarrow & t \end{array} \right. = \left\{ \begin{array}{rcl} p & \Rightarrow & \left\{ \begin{array}{rcl} q & \Rightarrow & s \\ \neg(q) & \Rightarrow & t \end{array} \right. \\ \neg(p) & \Rightarrow & \left\{ \begin{array}{rcl} r & \Rightarrow & s \\ \neg(r) & \Rightarrow & t \end{array} \right. \end{array} \right. \tag{B.4}$$

$$\left\{ \begin{array}{rcl} p & \Rightarrow & \left\{ \begin{array}{rcl} q & \Rightarrow & a \\ \neg(q) & \Rightarrow & b \end{array} \right. \\ \neg(p) & \Rightarrow & x \end{array} \right. = \left\{ \begin{array}{rcl} (p \wedge q) & \Rightarrow & a \\ (p \wedge \neg q) & \Rightarrow & b \\ \neg p & \Rightarrow & x \end{array} \right. \tag{B.5}$$

B.3 Aumento de Definição

Sempre que uma função g é mais definida que uma outra f, isto é, que

$$f(x) = g(x) \vee f(x) = \perp$$

— relembrar (3.48) — então g pode sempre substituir f em qualquer expressão ϕ em que f participe, *i.é*

$$\phi(f(x)) \leq \phi(g(x))$$

A ilustração mais típica deste tipo de substituição tem a ver com as seguintes leis, que permitirão substituir as conectivas lógicas $\vee$ e $\wedge$ por expressões condicionais "mais latas":

$$p \vee q \quad \leq \quad \left\{ \begin{array}{ccc} p & \Rightarrow & V \\ \neg p & \Rightarrow & q \end{array} \right. \tag{B.6}$$

$$p \wedge q \quad \leq \quad \left\{ \begin{array}{ccc} p & \Rightarrow & q \\ \neg p & \Rightarrow & F \end{array} \right.$$

habitualmente designadas "$\vee$ preguiçoso" e "$\wedge$ preguiçoso".

B.4 'Fold/Unfold'

Dada a definição de uma função recursiva f, com padrão genérico

$$f(x) \stackrel{\text{def}}{=} \left\{ \begin{array}{ccc} p(x) & \Rightarrow & g(x) \\ \neg(p(x)) & \Rightarrow & h(f(e(x))) \end{array} \right.$$

sempre que f é aplicada a um dado argumento a numa dada expressão ϕ, *i.é*

$$\phi(f(a))$$

chama-se fazer o 'unfold' (=*desenrolar*) de f o processo de substituição de f pela sua regra de definição, *i.é*, $\phi(f(a))$ pode ser transformado em

$$\phi(\left\{ \begin{array}{ccc} p(a) & \Rightarrow & g(a) \\ \neg(p(a)) & \Rightarrow & h(f(e(a))) \end{array} \right.)$$

o que normalmente permite ainda obter a expressão

$$\left\{ \begin{array}{ccc} p(a) & \Rightarrow & \phi(g(a)) \\ \neg(p(a)) & \Rightarrow & \phi(h(f(e(a)))) \end{array} \right. \tag{B.7}$$

por aplicação da lei distributiva (B.1).

Reparemos que a transformação por 'unfold' corresponde a ler a regra genérica

$$\phi(f(x)) = \phi(\left\{ \begin{array}{ccc} p(x) & \Rightarrow & g(x) \\ \neg(p(x)) & \Rightarrow & h(f(e(x))) \end{array} \right.)$$

da esquerda para a direita. A leitura no sentido inverso, da direita para a esquerda, que permitirá simplificar (B.7) para $\phi(f(a))$, é de extrema utilidade no cálculo e designa-se por 'fold' (=*enrolar*) de uma expressão através de uma definição recursiva [1].

[1] De facto, deve escrever-se $\leq$ onde, na igualdade acima, se escreveu $=$, indicando que o 'fold' de uma expressão pode ser inseguro, isto é, converter a expressão em valor indefinido. Teremos o cuidado de evitar essas situações, aliás muito pouco frequentes.

B.5 Outros Resultados

Considere-se o seguinte esquema abstracto,

$$f(x) = \begin{cases} p(x) & \Rightarrow & e(x) \\ \neg(p(x)) & \Rightarrow & \begin{cases} q(x) & \Rightarrow & d(x) \\ \neg(q(x)) & \Rightarrow & f(r(x)) \end{cases} \end{cases} \tag{B.8}$$

onde r é da forma

$$r(x) = \begin{cases} c(x) & \Rightarrow & a(x) \\ \neg(c(x)) & \Rightarrow & b(x) \end{cases}$$

e onde e é uma *função de erro* com propriedades de "função absorvente" com respeito à composição, *i.é*

$$w(e(x)) = e(x) \tag{B.9}$$

verifica-se. Teremos então, introduzindo uma função auxiliar w em (B.8):

$$\begin{aligned} f(x) &= \begin{cases} p(x) & \Rightarrow & e(x) \\ \neg(p(x)) & \Rightarrow & w(x) \end{cases} \\ w(x) &= \begin{cases} q(x) & \Rightarrow & d(x) \\ \neg(q(x)) & \Rightarrow & f(r(x)) \end{cases} \end{aligned} \tag{B.10}$$

onde, 'unfolding' f em (B.10),

$$w(x) = \begin{cases} q(x) & \Rightarrow & d(x) \\ \neg(q(x)) & \Rightarrow & \underbrace{(\lambda x. \begin{cases} p(x) & \Rightarrow & e(x) \\ \neg(p(x)) & \Rightarrow & w(x) \end{cases})(r(x))}_{A} \end{cases}$$

Considerando agora (B.9), teremos

$$\begin{aligned} A &= (\lambda x. \begin{cases} p(x) & \Rightarrow & w(e(x)) \\ \neg(p(x)) & \Rightarrow & w(x) \end{cases})(r(x)) \\ &= (\lambda x. w(\begin{cases} p(x) & \Rightarrow & e(x) \\ \neg(p(x)) & \Rightarrow & x \end{cases}))(r(x)) \\ &= w((\lambda x. \begin{cases} p(x) & \Rightarrow & e(x) \\ \neg(p(x)) & \Rightarrow & x \end{cases})(r(x))) \end{aligned}$$

i.é otemos uma solução para w (B.10) que é iterativa e directamente convertível em *ciclo*-`while`:

$$w(x) = \begin{cases} q(x) & \Rightarrow & d(x) \\ \neg(q(x)) & \Rightarrow & w((\lambda x. \begin{cases} p(x) & \Rightarrow & e(x) \\ \neg(p(x)) & \Rightarrow & x \end{cases})(r(x))) \end{cases} \tag{B.11}$$

Apêndice C

Tábua de Funções e Invariantes em SETS

Apresenta-se neste anexo uma relação de funções de abstracção e invariantes associados as leis do cálculo SETS. O anexo não pretende de forma nenhuma ser exaustivo. Cada função ou invariante indica, em índice, a lei a que se refere. Por exemplo, $f_{(8.31)}$ e $\phi_{(8.31)}$ são, respectivamente, a função de abstracção e o invariante referentes à lei (8.31). Segue-se a ordem pela qual essas leis ocorrem no texto. Definem-se também as inversas de algumas funções de abstracção bijectivas. As funções e predicados básicos empregues são listados no fim do apêndice (secção C.3).

C.1 Funções de Abstracção

$$f_{(8.9)} = mkf \tag{C.1}$$

$$f_{(8.24)} = \pi_1 \tag{C.2}$$

$$\tag{C.3}$$

$$f_{(8.31)} = \lambda\sigma.\langle\sigma(1), \ldots, \sigma(n)\rangle \tag{C.4}$$

$$f_{(8.32)} = \lambda x.x \tag{C.5}$$

$$f_{(8.37)} = \lambda\langle x, y\rangle.\begin{pmatrix} a \\ \langle x(a), y(a)\rangle \end{pmatrix}_{a\in A} \tag{C.6}$$

$$f_{(8.38)} = uncurry \tag{C.7}$$

$$f_{(8.58)} = \lambda\sigma.\left(\begin{pmatrix} a \\ \sigma(a,c) \end{pmatrix}_{\langle a,c\rangle \in dom(\sigma)}^{c} \right)_{c \in C} \tag{C.8}$$

$$f_{(8.58)}{}^{-1} = \lambda\sigma.\bigcup_{c\in C} \begin{pmatrix} \langle c,a\rangle \\ \sigma(c)(a) \end{pmatrix}_{a \in dom(\sigma)(c)} \tag{C.9}$$

$$f_{(8.59)} = \pi_1 \tag{C.10}$$

$$f_{(8.60)} = \pi_2 \tag{C.11}$$

$$f_{(8.64)} = dom \tag{C.12}$$

$$f_{(8.69)} = \bowtie \tag{C.13}$$

$$f_{(8.70)} = \lambda\sigma.\begin{pmatrix} \langle a,b\rangle \\ (\sigma(a))(b) \end{pmatrix}_{a\in dom(\sigma)\wedge b\in dom(\sigma(a))} \tag{C.14}$$

$$f_{(8.72)} = \underset{n}{\bowtie} \tag{C.15}$$

$$f_{(8.75)} = \overset{+}{\bowtie} \tag{C.16}$$

$$f_{(8.132)} = elems \tag{C.17}$$

$$f_{(8.172)} = \lambda\sigma.\begin{pmatrix} a \\ \langle b, \sigma(\langle a,b\rangle)\rangle \end{pmatrix}_{\langle a,b\rangle \in dom(\sigma)} \tag{C.18}$$

$$f_{(8.172)}^{-1} = \lambda\sigma.\begin{pmatrix} \langle a, \pi_1(\sigma(a))\rangle \\ \pi_2(\sigma(a)) \end{pmatrix}_{a\in dom(\sigma)} \tag{C.19}$$

C.2 Invariantes Induzidos

$$\phi_{(8.9)} = fdp \tag{C.20}$$

$$\phi_{(8.69)} = eqd \tag{C.21}$$

$$\phi_{(8.70)} = \lambda\sigma.\left(\quad\right) \notin rng(\sigma) \tag{C.22}$$

$$\phi_{(8.72)} = dpi \tag{C.23}$$

$$\phi_{(8.75)} = djd \tag{C.24}$$

$$\phi_{(8.172)} = fdp \circ dom \tag{C.25}$$

C.3 Funções e Predicados Básicos

$$\overset{+}{\bowtie}(\sigma,\tau) \stackrel{\text{def}}{=} i_1 \circ \sigma \cup i_2 \circ \tau \tag{C.26}$$

$$djd(\sigma,\tau) \stackrel{\text{def}}{=} dom(\sigma) \cap dom(\tau) = \emptyset \tag{C.27}$$

$$dpi(\sigma,\tau) \stackrel{\text{def}}{=} \pi_1[dom(\tau)] \subseteq dom(\sigma) \tag{C.28}$$

$$eqd(\langle\sigma,\tau\rangle) \stackrel{\text{def}}{=} dom(\sigma) = dom(\tau) \tag{C.29}$$

$$elems(b) \stackrel{\text{def}}{=} \{b(i) \mid\ i \in length(b)\} \tag{C.30}$$

$$fdp(r) \stackrel{\text{def}}{=} \forall t, t' \in r : \pi_1(t) = \pi_1(t') \Rightarrow \pi_2(t) = \pi_2(t') \tag{C.31}$$

$$i_1(a) \stackrel{\text{def}}{=} \langle 1, a\rangle \tag{C.32}$$

$$i_2(a) \stackrel{\text{def}}{=} \langle 2, a\rangle \tag{C.33}$$

$$\sigma \bowtie \tau \stackrel{\text{def}}{=} \begin{pmatrix} a \\ \langle\sigma(a), \tau(a)\rangle \end{pmatrix}_{a \in dom(\sigma)} \tag{C.34}$$

$$mkf(\rho) \stackrel{\text{def}}{=} \begin{pmatrix} a \\ the(\{b \in B \mid\ a\rho b\}) \end{pmatrix}_{a \in \pi_1[\rho]} \tag{C.35}$$

$$\bowtie_n (\sigma,\tau) \stackrel{\text{def}}{=} \left(\begin{array}{c} a \\ \langle\sigma(a), \begin{pmatrix} b \\ \tau(a,b) \end{pmatrix}_{\langle a,b\rangle \in sel(a,\tau)} \rangle \end{array} \right)_{a \in dom(\sigma)} \tag{C.36}$$

$$sel(a,\tau) \stackrel{\text{def}}{=} \{\langle a', b\rangle \in dom(\tau) \mid\ a' = a\} \tag{C.37}$$

$$uncurry(t) \stackrel{\text{def}}{=} \lambda(b,c).t(b)(c) \tag{C.38}$$

Apêndice D

Soluções de Alguns Exercícios

Apresentam-se neste anexo resoluções de exercícios deste texto. Chama-se à atenção para o facto de muitos exercícios terem resoluções alternativas. Procurou-se documentar os vários estilos de resolução possíveis, desde a indução estrutural à dedução sistemática suportada por métodos como o 'fold/unfold'. À medida que os exercícios se tornam rotineiros relaxa-se um pouco a justificação exaustiva dos passos dados (leitor deverá identificar em cada caso as leis cuja referência se omite).

D.1 Exercícios do Capítulo 1

Resolução 1.4: Para a gramática dada propõe-se a seguinte assinatura:

$$\Sigma_G = \left\{ \begin{array}{rcl} pop & : & \langle \mathit{pilha} \rangle \to \langle \mathit{pilha} \rangle \\ push & : & \langle \mathit{elem} \rangle \times \langle \mathit{pilha} \rangle \to \langle \mathit{pilha} \rangle \\ empty & : & \to \langle \mathit{pilha} \rangle \\ top & : & \langle \mathit{pilha} \rangle \to \langle \mathit{elem} \rangle \\ true & : & \to \langle \mathit{bool} \rangle \\ false & : & \to \langle \mathit{bool} \rangle \\ empty? & : & \langle \mathit{pilha} \rangle \to \langle \mathit{bool} \rangle \end{array} \right.$$

□

Resolução 1.6: A gramática proposta é equivalente a:

$$\begin{array}{rcl} \langle A \rangle & ::= & \mathtt{a}\ \langle B \rangle \mid \mathtt{b}\ \langle A \rangle\ \mathtt{a}\ \langle C \rangle \\ \langle C \rangle & ::= & \epsilon \mid \langle B \rangle \langle C \rangle \end{array}$$

$$
\begin{array}{lcl}
\langle B\rangle & ::= & \texttt{a b} \mid \texttt{a}\,\langle D\rangle\,\texttt{c} \\
\langle D\rangle & ::= & \langle A\rangle \mid \langle A\rangle\langle D\rangle
\end{array}
$$

(relembre-se (1.7) e (1.8) e o Exercício 1.5 da pág. 13) o que conduz à assinatura:

$$
\Sigma_G = \left\{
\begin{array}{lcl}
\sigma_1 & : & \langle B\rangle \rightarrow \langle A\rangle \\
\sigma_2 & : & \langle A\rangle \times \langle C\rangle \rightarrow \langle A\rangle \\
\sigma_3 & : & \rightarrow \langle C\rangle \\
\sigma_4 & : & \langle B\rangle \times \langle C\rangle \rightarrow \langle C\rangle \\
\sigma_5 & : & \rightarrow \langle B\rangle \\
\sigma_6 & : & \langle D\rangle \rightarrow \langle B\rangle \\
\sigma_7 & : & \langle A\rangle \rightarrow \langle D\rangle \\
\sigma_8 & : & \langle A\rangle \times \langle D\rangle \rightarrow \langle D\rangle
\end{array}
\right.
$$

□

Resolução 1.12: Ter-se-á, sucessivamente [1]:

- O facto (1.37) verifica-se:

$$
\begin{array}{rcl}
((f \circ h) \times (g \circ i))(\langle a, b\rangle) & = & ((f \times g) \circ (h \times i))(\langle a, b\rangle) \\
 & \updownarrow & \text{por (1.14) e (1.19)} \\
\langle (f \circ h)(a), (g \circ i)(b)\rangle & = & (f \times g)((h \times i)(\langle a, b\rangle)) \\
 & \updownarrow & \text{por (1.14) e (1.19)} \\
\langle f(h(a)), g(i(b))\rangle & = & (f \times g)(\langle h(a), i(b)\rangle) \\
 & \updownarrow & \text{por (1.19)} \\
\langle f(h(a)), g(i(b))\rangle & = & \langle f(h(a)), g(i(b))\rangle \\
 & \updownarrow & \text{propriedade reflexiva da igualdade} \\
 & V &
\end{array}
$$

- O facto (1.38) verifica-se:

$$
\begin{array}{rcl}
1_{A \times B}(\langle a, b\rangle) & = & (1_A \times 1_B)(\langle a, b\rangle) \\
 & \updownarrow & \text{def. de } 1_X \text{ e (1.19)} \\
\langle a, b\rangle & = & \langle 1_A(a), 1_B(b)\rangle \\
 & \updownarrow & \text{def. de } 1_X \\
\langle a, b\rangle & = & \langle a, b\rangle \\
 & \updownarrow & \text{prop. refl. da igualdade} \\
 & V &
\end{array}
$$

[1] Deixa-se ao leitor a tarefa de justificar os passos não documentados.

- O facto (1.39) verifica-se:

$$
\begin{array}{rcl}
(\langle f,g\rangle \circ h)(x) & = & (\langle f\circ h, g\circ h\rangle)(x) \\
 & \updownarrow & \text{por (1.14) e (1.20)} \\
\langle f,g\rangle(h(x)) & = & \langle (f\circ h)(x), (g\circ h)(x)\rangle \\
 & \updownarrow & \text{por (1.20) e (1.14)} \\
\langle f(h(x)), g(h(x))\rangle & = & \langle f(h(x)), g(h(x))\rangle \\
 & \updownarrow & \text{prop. refl. da igualdade} \\
 & V &
\end{array}
$$

- O facto (1.40) verifica-se:

$$
\begin{array}{rcl}
(\pi_1 \circ \langle f,g\rangle)(x) & = & f(x) \\
 & \updownarrow & \text{por (1.14)} \\
\pi_1(\langle f,g\rangle(x)) & = & f(x) \\
 & \updownarrow & \text{por (1.20)} \\
\pi_1(\langle f(x), g(x)\rangle) & = & f(x) \\
 & \updownarrow & \text{def. de } \pi_1 \\
f(x) & = & f(x) \\
 & \updownarrow & \text{prop. refl. da igualdade} \\
 & V &
\end{array}
$$

- O facto (1.41) verifica-se, sendo a prova análoga à anterior.
- O facto (1.42) verifica-se:

$$
\begin{array}{rcl}
((f\circ h)+(g\circ i))(x) & = & ((f+g)\circ(h+i))(x) \\
 & \updownarrow & \text{por (1.14)} \\
((f\circ h)+(g\circ i))(x) & = & (f+g)((h+i)(x)) \\
 & \updownarrow & \text{por (1.29)} \\
\left\{\begin{array}{lcl} x=i_1(a) & \Rightarrow & i_1((f\circ h)(a)) \\ x=i_2(b) & \Rightarrow & i_2((g\circ i)(b)) \end{array}\right. & = & (f+g)(\left\{\begin{array}{lcl} x=i_1(a) & \Rightarrow & i_1(h(a)) \\ x=i_2(b) & \Rightarrow & i_2(i(b)) \end{array}\right.) \\
 & \updownarrow & \text{por (1.14) e (B.1)} \\
\left\{\begin{array}{lcl} x=i_1(a) & \Rightarrow & i_1(f(h(a))) \\ x=i_2(b) & \Rightarrow & i_2(g(i(b))) \end{array}\right. & = & \left\{\begin{array}{lcl} x=i_1(a) & \Rightarrow & (f+g)(i_1(h(a))) \\ x=i_2(b) & \Rightarrow & (f+g)(i_2(i(b))) \end{array}\right. \\
 & \updownarrow & \text{por (1.29)} \\
\left\{\begin{array}{lcl} x=i_1(a) & \Rightarrow & i_1(f(h(a))) \\ x=i_2(b) & \Rightarrow & i_2(g(i(b))) \end{array}\right. & = & \left\{\begin{array}{lcl} x=i_1(a) & \Rightarrow & i_1(f(h(a))) \\ x=i_2(b) & \Rightarrow & i_2(g(i(b))) \end{array}\right. \\
 & \updownarrow & \text{prop. refl. da igualdade} \\
 & V &
\end{array}
$$

- O facto (1.43) verifica-se:

$$
\begin{array}{rcl}
1_{A+B}(x) & = & (1_A + 1_B)(x) \\
& \updownarrow & \text{def. de } 1_X \text{ e por (1.29)} \\
x & = & \left\{ \begin{array}{lcl} x = i_1(a) & \Rightarrow & i_1(1_A(a)) \\ x = i_2(b) & \Rightarrow & i_2(1_B(b)) \end{array} \right. \\
& \updownarrow & \text{def. de } 1_X \\
x & = & \left\{ \begin{array}{lcl} x = i_1(a) & \Rightarrow & i_1(a) \\ x = i_2(b) & \Rightarrow & i_2(b) \end{array} \right. \\
& \updownarrow & \text{subst. de iguais por iguais} \\
x & = & \left\{ \begin{array}{lcl} x = i_1(a) & \Rightarrow & x \\ x = i_2(b) & \Rightarrow & x \end{array} \right. \\
& \updownarrow & \text{variante de (B.2)} \\
x & = & x \\
& \updownarrow & \text{prop. refl. da igualdade} \\
& V &
\end{array}
$$

- O facto (1.44) verifica-se e decorre de

$$\pi_1 \circ (j \times i) \circ \langle f, h \rangle = j \circ f \tag{D.1}$$

$$\pi_2 \circ (j \times i) \circ \langle f, h \rangle = i \circ h \tag{D.2}$$

Aplicando a construção de funções a ambos os membros de (D.1) e (D.2) ter-se-á:

$$
\begin{array}{rcl}
\pi_1 \circ (j \times i) \circ \langle f, h \rangle & = & j \circ f \\
\pi_2 \circ (j \times i) \circ \langle f, h \rangle & = & i \circ h \\
\hline
\langle \pi_1 \circ (j \times i) \circ \langle f, h \rangle, \pi_2 \circ (j \times i) \circ \langle f, h \rangle \rangle & = & \langle j \circ f, i \circ h \rangle \\
& \updownarrow & \\
\langle \pi_1, \pi_2 \rangle \circ (j \times i) \circ \langle f, h \rangle & = & \langle j \circ f, i \circ h \rangle \\
& \updownarrow & \\
id \circ (j \times i) \circ \langle f, h \rangle & = & \langle j \circ f, i \circ h \rangle \\
& \updownarrow & \\
(j \times i) \circ \langle f, h \rangle & = & \langle j \circ f, i \circ h \rangle
\end{array}
$$

(deixa-se ao leitor a tarefa de justificar cada passo do raciocínio acima). Alternativamente, bastaria recorrer a (1.19) e (1.20):

$$
\begin{array}{rcl}
\langle (j \times i) \circ \langle f, h \rangle \rangle (a) & = & (j \times i)(\langle f, h \rangle (a)) \\
& = & (j \times i)(\langle f(a), h(a) \rangle) \\
& = & \langle j(f(a)), i(h(a)) \rangle \\
& = & \langle (j \circ f)(a), (i \circ h)(a) \rangle \\
& = & \langle j \circ f, i \circ h \rangle (a)
\end{array}
$$

como queríamos demonstrar. (Deixa-se ao leitor a tarefa de justificar cada passo do raciocínio acima).

□

Resolução 1.13: Ter-se-á, sucessivamente [2]:

- O facto (1.56) verifica-se:

$$
\begin{array}{rcl}
2^{f\circ g}(x) & = & (2^f \circ 2^g)(x) \\
& \updownarrow & \text{por (1.50)} \\
\{(f\circ g)(a) \mid\ a \in x\} & = & (2^f \circ 2^g)(x) \\
& \updownarrow & \text{por (1.14)} \\
\{f(g(a)) \mid\ a \in x\} & = & 2^f(2^g(x)) \\
& \updownarrow & \text{por (1.50)} \\
\{f(g(a)) \mid\ a \in x\} & = & 2^f(\{g(a) \mid\ a \in x\}) \\
& \updownarrow & \text{por (1.50)} \\
\{f(g(a)) \mid\ a \in x\} & = & \{f(b) \mid\ b \in \{g(a) \mid\ a \in x\}\} \\
& \updownarrow & \text{subst. de iguais por iguais} \\
\{f(g(a)) \mid\ a \in x\} & = & \{f(b) \mid\ b \in \{y \mid\ a \in x \wedge y = g(a)\}\} \\
& \updownarrow & \text{por (A.13)} \\
\{f(g(a)) \mid\ a \in x\} & = & \{f(b) \mid\ a \in x \wedge b = g(a)\} \\
& \updownarrow & \text{subst. de iguais por iguais} \\
\{f(g(a)) \mid\ a \in x\} & = & \{f(g(a)) \mid\ a \in x\} \\
& \updownarrow & \text{prop. refl. da igualdade} \\
& V &
\end{array}
$$

- O facto (1.57) verifica-se:

$$
\begin{array}{rcl}
2^f(\emptyset) & \underset{\text{por (1.50)}}{=} & \{f(a) \mid\ a \in \emptyset\} \\
& = & \{f(a) \mid\ F\} \\
& = & \emptyset
\end{array}
$$

- O facto (1.58) verifica-se:

$$
\begin{array}{rcl}
2^f(x \cup y) & \underset{\text{por (1.50)}}{=} & \{f(a) \mid\ a \in x \cup y\} \\
& \underset{\text{por (A.11)}}{=} & \{f(a) \mid\ a \in x \vee a \in y\} \\
& \underset{\text{por } \{x \mid\ p(x) \vee q(x)\} = \{x \mid\ p(x)\} \cup \{x \mid\ q(x)\}}{=} & \{f(a) \mid\ a \in x\} \cup \{f(a) \mid\ a \in y\}
\end{array}
$$

---

[2] Deixa-se ao leitor a tarefa de justificar os passos não documentados.

$$\underset{\text{por (1.50)}}{=} \quad 2^f(x) \cup 2^f(y)$$

- O facto (1.59) verifica-se:

$$\begin{aligned}
2^f(x \cap y) \quad & \underset{\text{por (1.50)}}{=} \quad \{f(a) \mid a \in x \cap y\} \\
& \underset{\text{por (A.12)}}{=} \quad \{f(a) \mid a \in x \wedge a \in y\} \\
& \underset{\text{dual de passo equivalente acima}}{=} \quad \{f(a) \mid a \in x\} \cap \{f(a) \mid a \in y\} \\
& \underset{\text{por (1.50)}}{=} \quad 2^f(x) \cap 2^f(y)
\end{aligned}$$

- O facto (1.60) verifica-se:

$$\begin{aligned}
(r \circ s)^{-1} \quad & \underset{\text{por (1.52) e (1.51)}}{=} \quad \{\langle \pi_2(q), \pi_1(p) \rangle \mid p \in r \wedge q \in s \wedge \pi_2(p) = \pi_1(q)\} \\
& \underset{\text{por def. proj. e (1.51)}}{=} \quad \{\langle \pi_1(q'), \pi_2(p') \rangle \mid p' \in r^{-1} \wedge q' \in s^{-1} \wedge \pi_1(p') = \pi_2(q')\} \\
& \underset{\text{por mudança de variáveis}}{=} \quad \{\langle \pi_1(p), \pi_2(q) \rangle \mid q \in r^{-1} \wedge p \in s^{-1} \wedge \pi_2(p) = \pi_1(q)\} \\
& \underset{\text{por (1.52)}}{=} \quad s^{-1} \circ r^{-1}
\end{aligned}$$

- O facto (1.61) verifica-se:

$$\begin{aligned}
r \circ (s \cup t) \quad & \underset{\text{por (1.52)}}{=} \quad \{\langle \pi_1(q), \pi_2(p) \rangle \mid p \in r \wedge q \in (s \cup t) \wedge \pi_2(p) = \pi_1(q)\} \\
& \underset{\text{por (A.11) e cálculo de predicados elementar}}{=} \quad \{\langle \pi_1(q), \pi_2(p) \rangle \mid p \in r \wedge q \in s \wedge \pi_2(p) = \pi_1(q) \vee p \in r \wedge q \in t \wedge \pi_2(p) = \pi_1(q)\} \\
& \underset{\text{por (A.11)}}{=} \quad \{\langle \pi_1(q), \pi_2(p) \rangle \mid p \in r \wedge q \in s \wedge \pi_2(p) = \pi_1(q)\} \cup \{\langle \pi_1(q), \pi_2(p) \rangle \mid p \in r \wedge q \in t \wedge \pi_2(p) = \pi_1(q)\} \\
& \underset{\text{por (1.52)}}{=} \quad (r \circ s) \cup (r \circ t)
\end{aligned}$$

- O facto (1.62) verifica-se:

$$\begin{aligned}
r \circ \emptyset \quad & \underset{\text{por (1.52)}}{=} \quad \{\langle \pi_1(q), \pi_2(p) \rangle \mid p \in r \wedge q \in \emptyset \wedge \pi_2(p) = \pi_1(q)\} \\
& \underset{\text{pois } x \in \emptyset = F}{=} \quad \{\langle \pi_1(q), \pi_2(p) \rangle \mid p \in r \wedge F \wedge \pi_2(p) = \pi_1(q)\} \\
& \underset{\text{pois } p \wedge F = F}{=} \quad \{\langle \pi_1(q), \pi_2(p) \rangle \mid F\}
\end{aligned}$$

$$= \quad \emptyset$$
pois $\{x \mid \ F\} = \emptyset$

$\square$

Resolução 1.17: Ter-se-á, sucessivamente:

- O facto (1.72) verifica-se:

$$\begin{array}{rcl}
(A \rightharpoonup (f \circ g))(\sigma) & = & ((A \rightharpoonup f) \circ (A \rightharpoonup g))(\sigma) \\
& \updownarrow & \text{por (1.14)} \\
\left(\begin{array}{c} a \\ f(g(\sigma(a))) \end{array}\right)_{a \in dom(\sigma)} & = & (A \rightharpoonup f)((A \rightharpoonup g)(\sigma)) \\
& \updownarrow & \text{por (1.14) e def. de } A \rightharpoonup f \\
\left(\begin{array}{c} a \\ f(g(\sigma(a))) \end{array}\right)_{a \in dom(\sigma)} & = & (A \rightharpoonup f)(\left(\begin{array}{c} a \\ g(\sigma(a)) \end{array}\right)_{a \in dom(\sigma)}) \\
& \updownarrow & \text{por def. de } A \rightharpoonup f \\
\left(\begin{array}{c} a \\ f(g(\sigma(a))) \end{array}\right)_{a \in dom(\sigma)} & = & \left(\begin{array}{c} a \\ f(g(\sigma(a))) \end{array}\right)_{a \in dom(\sigma)} \\
& \updownarrow & \text{prop. refl. da igualdade} \\
& V &
\end{array}$$

- O facto (1.73) verifica-se, para f injectiva:

$$\begin{array}{rcl}
(dom \circ (f \rightharpoonup A))(\sigma) & = & (2^f \circ dom)(\sigma) \\
& \updownarrow & \text{por (1.14)} \\
dom((f \rightharpoonup A)(\sigma)) & = & 2^f(dom(\sigma)) \\
& \updownarrow & \text{por (1.70)} \\
dom(\left(\begin{array}{c} f(a) \\ \sigma(a) \end{array}\right)_{a \in dom(\sigma)}) & = & 2^f(dom(\sigma)) \\
& \updownarrow & \text{por definição de domínio} \\
\{f(a) \mid \ a \in dom(\sigma)\} & = & 2^f(dom(\sigma)) \\
& \updownarrow & \text{por (1.50)} \\
2^f(dom(\sigma)) & = & 2^f(dom(\sigma)) \\
& \updownarrow & \text{prop. refl. da igualdade} \\
& V &
\end{array}$$

- O facto (1.74) verifica-se:

$$(rng \circ (A \rightharpoonup f))(\sigma) \quad = \quad (2^f \circ rng)(\sigma)$$

$$
\begin{array}{rcl}
 & \updownarrow & \text{por (1.14)} \\
rng((A \rightharpoonup f)(\sigma)) & = & 2^f(rng(\sigma)) \\
 & \updownarrow & \text{por (1.50) e def. de } A \rightharpoonup f \\
rng(\left(\begin{array}{c} a \\ f(\sigma(a)) \end{array}\right)_{a \in dom(\sigma)}) & = & \{f(x) \mid \ x \in rng(\sigma)\} \\
 & \updownarrow & \text{por def. de } rng \\
\{f(\sigma(a)) \mid \ a \in dom(\sigma)\} & = & \{f(x) \mid \ x \in \{\sigma(a) \mid \ a \in dom(\sigma)\}\} \\
 & \updownarrow & \text{prop. teoria de conjuntos} \\
\{f(\sigma(a)) \mid \ a \in dom(\sigma)\} & = & \{f(\sigma(a)) \mid \ a \in dom(\sigma)\} \\
 & \updownarrow & \text{prop. refl. da igualdade} \\
 & V &
\end{array}
$$

- O facto (1.75) verifica-se:

$$
\begin{array}{rcl}
(dom \circ (A \rightharpoonup f))(\sigma) & = & dom(\sigma) \\
 & \updownarrow & \text{por (1.14)} \\
dom((A \rightharpoonup f)(\sigma)) & = & dom(\sigma) \\
 & \updownarrow & \text{por def. de } A \rightharpoonup f \\
dom(\left(\begin{array}{c} a \\ f(\sigma(a)) \end{array}\right)_{a \in dom(\sigma)}) & = & dom(\sigma) \\
 & \updownarrow & \text{por def. de } dom \\
\{a \mid \ a \in dom(\sigma)\} & = & dom(\sigma) \\
 & \updownarrow & \text{por (A.18)} \\
 & V &
\end{array}
$$

- O facto (1.76) verifica-se:

$$
\begin{array}{rcl}
(rng \circ (i \rightharpoonup B))(\sigma) & = & rng(\sigma) \\
 & \updownarrow & \text{por (1.70)} \\
rng(\left(\begin{array}{c} i(a) \\ \sigma(a) \end{array}\right)_{a \in dom(\sigma)}) & = & rng(\sigma) \\
 & \updownarrow & \text{por def. de } rng \\
\{\sigma(a) \mid \ a \in dom(\sigma)\} & = & rng(\sigma) \\
 & \updownarrow & \text{por def. de } rng \\
 & V &
\end{array}
$$

- O facto (1.77) verifica-se:

$$
(A \rightharpoonup f)(\sigma \mid S) \quad = \quad ((A \rightharpoonup f)(\sigma)) \mid S
$$

$$
\begin{array}{rcl}
 & \updownarrow & \text{por def. de } A \rightharpoonup f \text{ e (1.66)} \\
(A \rightharpoonup f)(\begin{pmatrix} a \\ \sigma(a) \end{pmatrix}_{a \in dom(\sigma)\cap S}) & = & (\begin{pmatrix} a \\ f(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)}) \mid S \\
 & \updownarrow & \text{por def. de } A \rightharpoonup f \text{ e (1.66)} \\
\begin{pmatrix} a \\ f(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)\cap S} & = & \begin{pmatrix} a \\ f(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)\cap S} \\
 & \updownarrow & \text{prop. refl. da igualdade} \\
 & V &
\end{array}
$$

- O facto (1.78) verifica-se:

$$
\begin{array}{rl}
(A \rightharpoonup f)(\sigma_1 \cup \sigma_2) \underset{\text{por def. de } A \rightharpoonup f}{=} & \begin{pmatrix} a \\ f(\sigma_1 \cup \sigma_2(a)) \end{pmatrix}_{a \in dom(\sigma_1 \cup \sigma_2)} \\
\underset{\text{por (A.40)}}{=} & \begin{pmatrix} a \\ f(\sigma_1 \cup \sigma_2(a)) \end{pmatrix}_{a \in dom(\sigma_1)\cup dom(\sigma_2)} \\
\underset{\text{por prop. da def. de compreensão}}{=} & \begin{pmatrix} a \\ f(\sigma_1 \cup \sigma_2(a)) \end{pmatrix}_{a \in dom(\sigma_1)} \cup \begin{pmatrix} a \\ f(\sigma_1 \cup \sigma_2(a)) \end{pmatrix}_{a \in dom(\sigma_2)} \\
\underset{\text{pela Def. 1.6}}{=} & \begin{pmatrix} a \\ f(\sigma_1(a)) \end{pmatrix}_{a \in dom(\sigma_1)} \cup \begin{pmatrix} a \\ f(\sigma_2(a)) \end{pmatrix}_{a \in dom(\sigma_2)} \\
\underset{\text{por def. de } A \rightharpoonup f}{=} & (A \rightharpoonup f)(\sigma_1) \cup (A \rightharpoonup f)(\sigma_2)
\end{array}
$$

- O facto (1.79) verifica-se:

$$
\begin{array}{rcl}
((f \rightharpoonup 1_D) \circ (1_A \rightharpoonup g))(\sigma) & = & (f \rightharpoonup g)(\sigma) \\
 & \updownarrow & \text{por (1.14) e (1.68)} \\
(f \rightharpoonup 1_D)((1_A \rightharpoonup g)(\sigma)) & = & \begin{pmatrix} f(a) \\ g(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)} \\
 & \updownarrow & \text{por (1.68)} \\
(f \rightharpoonup 1_D)(\begin{pmatrix} a \\ g(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)}) & = & \begin{pmatrix} f(a) \\ g(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)} \\
 & \updownarrow & \text{por (1.68)} \\
\begin{pmatrix} f(a) \\ g(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)}) & = & \begin{pmatrix} f(a) \\ g(\sigma(a)) \end{pmatrix}_{a \in dom(\sigma)} \\
 & \updownarrow & \\
 & V &
\end{array}
$$

- O facto (1.80) verifica-se, como se demonstra a seguir.

O lado esquerdo de (1.80) re-escreve em

$$
\begin{array}{rcl}
(A \rightharpoonup f)(\sigma) \dagger (A \rightharpoonup f)(\tau) & \underset{\text{por (A.57)}}{=} & ((A \rightharpoonup f)(\sigma)) \setminus dom((A \rightharpoonup f)(\tau)) \cup (A \rightharpoonup f)(\tau) \\
& \underset{\text{por (1.75)}}{=} & ((A \rightharpoonup f)(\sigma)) \setminus dom(\tau) \cup (A \rightharpoonup f)(\tau) \\
& = & (A \rightharpoonup f)(\sigma \setminus dom(\tau)) \cup (A \rightharpoonup f)(\tau)
\end{array}
$$

(estude o passo injustificado).
O lado direito de (1.80) re-escreve em

$$
\begin{array}{rcl}
(A \rightharpoonup f)(\sigma \dagger \tau) & \underset{\text{por (A.57)}}{=} & (A \rightharpoonup f)(\sigma \setminus dom(\tau) \cup \tau) \\
& \underset{\text{por (1.78)}}{=} & (A \rightharpoonup f)(\sigma \setminus dom(\tau)) \cup (A \rightharpoonup f)(\tau)
\end{array}
$$

Logo, os lados de (1.80) coincidem, como queríamos mostrar.

- O facto (1.81) verifica-se:

$$
\begin{array}{rcl}
A \rightharpoonup (1_B)(\sigma) & = & 1_{A \rightharpoonup B}(\sigma) \\
& \updownarrow & \text{por (1.69) e (1.13)} \\
\left(\begin{array}{c} a \\ 1_B(\sigma(a)) \end{array}\right)_{a \in dom(\sigma)} & = & \sigma \\
& \updownarrow & \text{por (1.13)} \\
\left(\begin{array}{c} a \\ \sigma(a) \end{array}\right)_{a \in dom(\sigma)} & = & \sigma \\
& \updownarrow & \\
\sigma & = & \sigma \\
& \updownarrow & \\
& V &
\end{array}
$$

$\square$

Resolução 1.20:

$$
\begin{array}{rcl}
\mathcal{A} & : & \Sigma \longrightarrow Sets \\
\mathcal{A}(Tabela) & \stackrel{\text{def}}{=} & 2^{\mathcal{A}(Tuplo)} \\
\mathcal{A}(Tuplo) & \stackrel{\text{def}}{=} & \mathcal{A}(AtrId) \rightharpoonup \mathcal{A}(Valor) \\
\mathcal{A}(AtrIds) & \stackrel{\text{def}}{=} & 2^{\mathcal{A}(AtrId)} \\
\mathcal{A}(Histogama) & \stackrel{\text{def}}{=} & \mathcal{A}(Valor) \rightharpoonup I\!\!N
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{A}(Valores) & \stackrel{\text{def}}{=} & 2^{\mathcal{A}(Valor)} \\
\mathcal{A}(Nat) & \stackrel{\text{def}}{=} & I\!N_0 \\
\mathcal{A}(valores)(a,r) & \stackrel{\text{def}}{=} & \{t(a) \mid\ t \in r \wedge a \in dom(t)\} \\
\mathcal{A}(atributos)(r) & \stackrel{\text{def}}{=} & \bigcup_{t \in r} dom(t) \\
\mathcal{A}(contagem)(a,v,r) & \stackrel{\text{def}}{=} & card(\{t \mid\ t \in r \wedge a \in dom(t) \wedge t(a) = v\}) \\
\mathcal{A}(histograma)(a,r) & \stackrel{\text{def}}{=} & \left(\begin{array}{c} v \\ \mathcal{A}(contagem)(a,v,r) \end{array} \right)_{v \in \mathcal{A}(valores)(a,r)}
\end{array}
$$

NB: Não é necessário especificar modelos para as espécies $Valor$ e $AtrId$, pois o modelo nada presume sobre elas (*cf.* parametrização). □

Resolução 1.32: Propõe-se:

$$
\begin{array}{rcl}
refsTo & : & WWW \times Ref \longrightarrow 2^{Ref} \\
refsTo(\sigma, r) & \stackrel{\text{def}}{=} & \{r' \in dom(\sigma) \mid\ \langle 2, r \rangle \in elems(\sigma(r'))\}
\end{array}
$$

□

Resolução 1.34:

1. Propõe-se:

$$
\begin{array}{rcl}
nomes & : & PlanoEstudos \to 2^{Nome} \\
nomes(pe) & \stackrel{\text{def}}{=} & \begin{array}{ll} let & ob = \{d \mid\ d \in pe \wedge \text{is-}Obrigat(d)\} \\ & op = pe - ob \\ in & 2^N(ob) \cup \bigcup_{d \in op} 2^N(D(d)) \end{array}
\end{array}
$$

2. Propõe-se:

$$
\begin{array}{rcl}
areas & : & PlanoEstudos \to 2^{ACien} \\
areas(pe) & \stackrel{\text{def}}{=} & \begin{array}{ll} let & ob = \{d \mid\ d \in pe \wedge \text{is-}Obrigat(d)\} \\ & op = pe - ob \\ in & (\bigcup_{d \in op} 2^{AC}(D(d))) - 2^{AC}(ob) \end{array}
\end{array}
$$

□

Resolução 1.35: Ter-se-á [3]:

[3] Deixa-se ao leitor a tarefa de justificar passos por documentar.

(A.70) — esta igualdade converte-se em

$$\begin{array}{rcl}
(g^* \circ f^*)(s) & = & ((g \circ f)^*)(s) \\
& \updownarrow & (1.14) \\
g^*(f^*(s)) & = & ((g \circ f)^*)(s) \\
& \updownarrow & (1.84) \\
g^*(f^*(s)) & = & <(g \circ f)(a) \mid a \leftarrow s> \\
& \updownarrow & (1.84) \\
g^*(<f(a) \mid a \leftarrow s>) & = & <g(f(a)) \mid a \leftarrow s> \\
& \updownarrow & (1.84) \\
<g(b) \mid b \leftarrow <f(a) \mid a \leftarrow s>> & = & <g(f(a)) \mid a \leftarrow s> \\
& \updownarrow & \\
& V &
\end{array}$$

(A.76) — esta igualdade converte-se em

$$\begin{array}{rcl}
(length \circ f^*)(s) & = & length(s) \\
& \updownarrow & (1.14) \\
length(f^*(s)) & = & length(s) \\
& \updownarrow & (1.84) \\
length(<f(a) \mid a \leftarrow s>) & = & length(s) \\
& \updownarrow & \\
& V &
\end{array}$$

(A.80) — esta igualdade converte-se em

$$\begin{array}{rcl}
(elems \circ f^*)(s) & = & (2^f \circ elems)(s) \\
& \updownarrow & \\
elems(f^*(s)) & = & (2^f \circ elems)(s) \\
& \updownarrow & (1.84) \\
elems(<f(a) \mid a \leftarrow s>) & = & 2^f(elems(s)) \\
& \updownarrow & (1.50) \\
elems(<f(a) \mid a \leftarrow s>) & = & \{f(a) \mid \ a \in elems(s)\} \\
& \updownarrow & \\
& V &
\end{array}$$

(A.85) — esta igualdade converte-se em

$$\begin{array}{rcl}
(tail \circ f^*)(s) & = & (f^* \circ tail)(s) \\
& \updownarrow & (1.14)
\end{array}$$

$$
\begin{array}{rcl}
tail(f^*(s)) & = & f^*(tail(s)) \\
& \updownarrow & (1.84) \\
tail(<f(a) \mid a \leftarrow s>) & = & <f(a) \mid a \leftarrow tail(s)> \\
& \updownarrow & \\
& V &
\end{array}
$$

$\Box$

Resolução 1.37: Tem-se $\mathsf{G}(X) = 2^X$ e $\mathsf{F}(X) = X \rightharpoonup C$, no diagrama da esquerda (para f injectiva) e $x = rng$ no da direita. Logo os factos são:

$$
\begin{array}{rcl}
dom \circ (f \rightharpoonup C) & = & 2^f \circ dom \\
rng \circ (C \rightharpoonup f) & = & 2^f \circ rng
\end{array}
$$

que coincidem com (1.73) e (1.75), respectivamente. $\Box$

Resolução 1.38:

1. Basta-nos mostrar que a expressão

$$x \dagger \underbrace{y \dagger \left(\begin{array}{c} a \\ x(a) + y(a) \end{array} \right)_{a \in dom(x|(dom(y)))}}_{z}$$

designa exactamente o mesmo que o lado direito de (1.102). Repare-se em primeiro lugar que

$$dom(x \mid (dom(y))) = dom(x) \cap dom(y)$$

Recorrendo ao facto (A.57), ter-se-á a sub-expressão z equivalente a

$$y \setminus (dom(x)) \cup \left(\begin{array}{c} a \\ x(a) + y(a) \end{array} \right)_{a \in dom(x) \cap dom(y)}$$

(O leitor reparará facilmente que o facto

$$X - (Y \cap X) \quad = \quad X - Y \tag{D.3}$$

foi também útil neste raciocínio). Apliquemos agora (A.57) a $x \dagger z$ e obteremos $x \setminus dom(z) \cup z$. Falta apenas desenvolver $dom(z)$:

$$
\begin{array}{rcl}
dom(z) & \underset{\text{por (A.40) e (A.54)}}{=} & (dom(y) - dom(x)) \cup (dom(x) \cap dom(y)) \\
& \underset{\text{por (A.17)}}{=} & (dom(y) \cap \overline{dom(x)}) \cup (dom(x) \cap dom(y))
\end{array}
$$

$$\begin{array}{cl} \underset{\text{por (A.27)}}{=} & dom(y) \cap (\overline{dom(x)} \cup dom(x)) \\ \underset{\text{def. de } \overline{A}}{=} & dom(y) \end{array}$$

Juntando tudo ter-se-á, de facto

$$(x \setminus dom(y)) \cup (y \setminus dom(x)) \cup \left(\begin{array}{c} a \\ x(a) + y(a) \end{array} \right)_{a \in dom(x) \cap dom(y)}$$

tal como em (1.102).

2. Propõe-se:

$$\begin{array}{rcl} \otimes & : & I\!N \times MSet(A) \to MSet(A) \\ n \otimes \sigma & \stackrel{\text{def}}{=} & \left(\begin{array}{c} a \\ n \times \sigma(a) \end{array} \right)_{a \in dom(\sigma)} \end{array}$$

Prova da propriedade proposta (deixa-se ao leitor a tarefa de justificar os passos dados, que são elementares):

$$\begin{array}{rcl} n \otimes (m \otimes \sigma) & = & n \otimes (\left(\begin{array}{c} a \\ m \times \sigma(a) \end{array} \right)_{a \in dom(\sigma)}) \\ & = & \left(\begin{array}{c} a \\ n \times (m \times \sigma(a)) \end{array} \right)_{a \in dom(\sigma)} \\ & = & \left(\begin{array}{c} a \\ (n \times m) \times \sigma(a)) \end{array} \right)_{a \in dom(\sigma)} \\ & = & (n \times m) \otimes \sigma \end{array}$$

□

Resolução 1.39: Propõe-se

$$\begin{array}{rcl} \Cap & : & MSet(A) \times MSet(A) \longrightarrow MSet(A) \\ x \Cap y & \stackrel{\text{def}}{=} & \left(\begin{array}{c} a \\ inf(x(a), y(a)) \end{array} \right)_{a \in dom(x) \cap dom(y)} \end{array}$$

onde

$$\begin{array}{rcl} inf & : & I\!N \times I\!N \longrightarrow I\!N \\ inf(x, y) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} x \leq y & \Rightarrow & x \\ x > y & \Rightarrow & y \end{array} \right. \end{array}$$

□

Resolução 1.43: Veja-se o Exercício 1.45. □

Resolução 1.50: Propõe-se a seguinte especificação:

$$clear(sigma,k,d) \stackrel{\text{def}}{=} \left\{ \begin{array}{lll} k \notin dom(\sigma) & \Rightarrow & \sigma \\ k \in dom(\sigma) & \Rightarrow & \begin{array}{ll} let & x = \sigma(k) \\ & w = W(x) \\ & w_1 = \left(\begin{array}{c} n \\ C(w(n)) \end{array} \right)_{n \in dom(w) \wedge D(w(n)) \leq d} \\ & w_2 = w \setminus dom(w_1) \\ & c = C(x) \\ & c_1 = \{x \in c \mid \ D(x) \leq d\} \\ & c_2 = c - c1 \\ & b = B(x) + \sum_{k \in dom(w_1)} w_1(k) - \sum_{x \in c_1} C(x) \\ in & \sigma \dagger \left(\begin{array}{c} k \\ \langle H(x), b, w_2, c_2 \rangle \end{array} \right) \end{array} \end{array} \right.$$

□

D.2 Exercícios do Capítulo 2

Resolução 2.6:

1. Vamos definir os isomorfismos pedidos da direita para a esquerda. Ter-se-á:

(2.37) $A \times B \cong B \times A$

$$\langle \pi_2, \pi_1 \rangle$$

(2.38) $A \times (B \times C) \cong (A \times B) \times C$

$$\langle \pi_1 \circ \pi_1, \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle$$

(2.39) $A \times 1 \cong A$

$$\lambda a \cdot \langle a, NIL \rangle$$

(2.40) $A + B \cong B + A$

$$[\ i_2, i_1\ ]$$

cf. (1.30). Repare-se como esta lei é a "dual" de (2.37), substituindo projecções por injecções e a construção pela alternativa.

(2.41) $A + (B + C) \cong (A + B) + C$

$$[\,[\,i_1, i_2 \circ i_1\,], i_2 \circ i_2\,]$$

itso é:

$$\lambda x \cdot \begin{cases} x = i_1(y) & \Rightarrow \begin{cases} y = i_1(z) & \Rightarrow & i_1(z) \\ y = i_2(z) & \Rightarrow & i_2(i_1(z)) \end{cases} \\ x = i_2(y) & \Rightarrow \quad i_2(i_2(y)) \end{cases}$$

Repare-se que esta lei é a "dual" de (2.38).

(2.42) $A + 0 \cong A$

$$\lambda a \cdot \langle 1, a \rangle$$

(2.43) $A \times 0 \cong 0$

A função identidade no conjunto vazio.

(Repare-se que $A \times 0 = 0$.)

(2.44) $A \times (B + C) \cong (A \times B) + (A \times C)$

$$[\,(1_A \times i_1), (1_A \times i_2)\,]$$

isto é:

$$\lambda x \cdot \begin{cases} x = i_1(y) & \Rightarrow & \langle \pi_1(y), i_1(\pi_2(y)) \rangle \\ x = i_2(y) & \Rightarrow & \langle \pi_1(y), i_2(\pi_2(y)) \rangle \end{cases} \tag{D.4}$$

(2.46) $\underbrace{A \times \ldots \times A}_{n} \cong A^n$

$$\lambda \sigma . \langle \sigma(1), \ldots, \sigma(n) \rangle$$

(2.47) $\underbrace{A + \ldots + A}_{n} \cong n \times A$

$$\lambda x . x$$

2. Trata-se de (2.47) seguida de (2.39):

$$\begin{aligned} \underbrace{1 + 1}_{2} &\cong 2 \times 1 \\ &\cong 2 \end{aligned}$$

□

Resolução 2.15: As primeiras reticências completam-se com:

$$\ldots \quad = \quad \begin{cases} fs(i) = (\quad) & \Rightarrow & fs \setminus \{i\} \\ \neg(fs(i) = (\quad)) & \Rightarrow & fs \end{cases}$$

As segundas reticências completam-se com:

$$\ldots \quad = \quad let \quad \begin{array}{l} i = head(p) \\ t = tail(p) \end{array} \quad in \quad \begin{cases} i \notin dom(fs) & \Rightarrow \quad fs \\ i \in dom(fs) & \Rightarrow \quad \begin{cases} \text{is-}File(fs(i)) & \Rightarrow \quad fs \\ \text{is-}Dir(fs(i)) & \Rightarrow \quad fs \dagger \begin{pmatrix} i \\ rmdir(fs(i), t) \end{pmatrix} \end{cases} \end{cases}$$

□

Resolução 2.22: Ter-se-á:

$$\begin{array}{rcl} X & \cong & Y \times B \\ & \updownarrow & \\ X & \cong & (1 + A \times Y) \times B \\ & \updownarrow & \\ X & \cong & 1 \times B + A \times Y \times B \\ & \updownarrow & \\ X & \cong & B + A \times \underbrace{Y \times B}_{X} \\ & \updownarrow & \\ X & \cong & B + A \times X \end{array}$$

(Cada passo deve ser justificado com as respectivas leis de SETS.) □

Resolução 2.24: Da inversão do PASCAL dado resulta:

$$\begin{cases} Exp & \cong & (1 + Node) \times \mathbb{Z}_0 \\ Node & \cong & (1 + BinOps) \times (1 + Node) \\ BinOps & \cong & BinOp \times Exp \\ BinOp & \cong & 4 \end{cases}$$

isto é,

$$\begin{cases} Exp & \cong & (1 + Node) \times \mathbb{Z}_0 \\ Node & \cong & (1 + 4 \times Exp) \times (1 + Node) \end{cases}$$

Ora da gramática dada obtem-se

$$\begin{array}{rcl} Exp & \cong & \mathbb{Z}_0 + Exp + 4 \times Exp^2 \\ & \updownarrow & \\ Exp & \cong & \mathbb{Z}_0 + Exp \times (1 + 4 \times Exp) \end{array}$$

que se converte em

$$\left\{\begin{array}{rcl} Exp & \cong & Y \times \mathbb{Z}_0 \\ Y & \cong & 1 + \underbrace{(1 + 4 \times Exp) \times Y}_{Node} \end{array}\right. \tag{D.5}$$

de acordo com o resultado do Exercício 2.22. Introduzindo $Node$ teremos, ainda:

$$\left\{\begin{array}{rcl} Exp & \cong & Y \times \mathbb{Z}_0 \\ Node & \cong & (1 + 4 \times Exp) \times Y \\ Y & \cong & 1 + Node \end{array}\right.$$

Removendo Y teremos, finalmente,

$$\left\{\begin{array}{rcl} Exp & \cong & (1 + Node) \times \mathbb{Z}_0 \\ Node & \cong & (1 + 4 \times Exp) \times (1 + Node) \end{array}\right.$$

que coincide com (D.5), como queríamos. $\square$

Resolução 2.26: Prova por indução sobre $\mathbb{N}_0$:

1. *Caso de Base*: ($k = 0$). Tem-se, neste caso

$$\begin{array}{rcl} f(0) & = & 0^2 \\ & \updownarrow & \text{expansão de } f \text{ para } k = 0 \\ 0 & = & 0 \\ & \updownarrow & \text{prop. refl. da igualdade} \\ & V & \end{array}$$

2. *Salto indutivo*: ($k > 0$)

   (a) *Hipótese de indução*:

$$f(k-1) = (k-1)^2$$

   (b) *Passo*: Ter-se-á:

$$\begin{array}{rcl} f(k) & = & k^2 \\ & \updownarrow & \text{expansão de } f \text{ para } k > 0 \\ impar(k) + f(k-1) & = & k^2 \\ & \updownarrow & \text{expansão de } impar \\ 2k - 1 + f(k-1) & = & k^2 \\ & \updownarrow & \text{hipótese de indução} \\ 2k - 1 + (k-1)^2 & = & k^2 \\ & \updownarrow & \text{expansão do binónio} \\ 2k - 1 + k^2 - 2k + 1 & = & k^2 \end{array}$$

$$
\begin{array}{rcl}
 & \updownarrow & \text{cancelamento de simétricos} \\
k^2 & = & k^2 \\
 & \updownarrow & \text{prop. refl. da igualdade} \\
 & V &
\end{array}
$$

□

Resolução 2.27: Prova por indução sobre 2^X:

1. *Caso de Base*: $(x = \emptyset)$. Tem-se, neste caso

$$
\begin{array}{rcl}
elems(list(\emptyset)) & = & \emptyset \\
 & \updownarrow & \text{expansão de } list \text{ para } x = \emptyset \\
elems(<>) & = & \emptyset \\
 & \updownarrow & \text{por (A.77)} \\
\emptyset & = & \emptyset \\
 & \updownarrow & \text{prop. refl. da igualdade} \\
 & V &
\end{array}
$$

2. *Salto indutivo*: $(x \supset \emptyset)$
   (a) *Hipótese de indução*:

$$\forall e \in x : elems(list(x - \{e\})) = x - \{e\}$$

   (b) *Passo*: Ter-se-á:

$$
\begin{array}{rcl}
elems(list(x)) & = & x \\
 & \updownarrow & \text{expansão de } list \text{ para } x \neq \emptyset \\
elems(cons(e, list(x - \{e\}))) & = & x \\
 & \updownarrow & \text{por (A.78)} \\
\{e\} \cup elems(list(x - \{e\}))) & = & x \\
 & \updownarrow & \text{hipótese de indução} \\
\{e\} \cup (x - \{e\}) & = & x \\
 & \updownarrow & \text{por (A.17), (A.28), } etc. \\
x & = & x \\
 & \updownarrow & \text{prop. refl. da igualdade} \\
 & V &
\end{array}
$$

□

Resolução 2.32:

1. O facto a demonstrar ou refutar é

$$\forall l \in I\!N^*, a \in I\!N : \phi(l) \Rightarrow \phi(ins(l,a))$$

O seu significado informal é o de garantir que uma lista l não tem elementos repetidos. Vamos tentar provar que ins preserva ϕ.
Prova por indução sobre $A^\star$:

(a) *Caso de Base*: ($l = < >$). Tem-se, neste caso

$$\begin{array}{rcl}
\phi(< >) & \Rightarrow & \phi(ins(< >, a)) \\
& \updownarrow & \\
length(< >) = card(elems(< >)) & \Rightarrow & length(ins(< >, a)) = card(elems(ins(< >, a))) \\
& \updownarrow & \\
0 = card(\emptyset) & \Rightarrow & length(<a>) = card(elems(<a>)) \\
& \updownarrow & \\
0 = 0 & \Rightarrow & 1 = card(\{a\}) \\
& \updownarrow & \\
V & \Rightarrow & 1 = 1 \\
& \updownarrow & \\
V & \Rightarrow & V \\
& \updownarrow & \\
& V &
\end{array}$$

(b) *Salto indutivo*: ($l \neq <>$)

i. *Hipótese de indução*:

$$\phi(tail(l)) \Rightarrow \phi(ins(tail(l), a))$$

ii. *Passo*: A demonstração a fazer é uma implicação de implicações,

$$(\phi(tail(l)) \Rightarrow \phi(ins(tail(l), a))) \quad \Rightarrow \quad (\phi(l) \Rightarrow \phi(ins(l, a)))$$

que se reduz a

$$\phi(ins(tail(l), a)) \wedge \phi(l) \quad \Rightarrow \quad \phi(ins(l, a)) \tag{D.6}$$

uma vez que $\phi(l)$ é mais forte que $\phi(tail(l))$ [4]. Mas repare-se que (D.6) é falso, isto é, que existem l e a tais que $\phi(ins(tail(l), a)) \wedge \phi(l) \wedge \neg\phi(ins(l, a))$ — faça-se, por exemplo, $l = <2, 1>$ e $a = 1$. Logo a prova fracassa e assim se mostra que ins não preserva o invariante ϕ.

[4] Veja-se o porquê desta simplificação, que é normalmente útil em demonstração de invariantes:

$$\begin{array}{c}
(a \Rightarrow b) \Rightarrow (c \Rightarrow d) \\
\updownarrow \\
((a \Rightarrow b) \wedge c) \Rightarrow d \\
\updownarrow
\end{array}$$

2. A prova anterior só fracassou porque l pode não estar estritamente ordendada. Por outro lado, ins parece inserir ordenadamente um elemento numa lista. Isto sugere que uma propriedade que ins preserva como invariante é

   $$\varphi(l) \stackrel{\text{def}}{=} \forall i, j \leq length(l) : i < j \Rightarrow l(i) < l(j)$$

   Repare-se que φ é, aliás, logicamente mais forte que ϕ.

□

Resolução 2.42: Vamos aplicar o Teorema de Kleene, para $f(x) = R \cup x^{-1}$. Estando garantida a continuidade de f (porquê?), ter-se-á (justifique os passos dados):

$$\begin{array}{rcl} f^0(\emptyset) & = & \emptyset \\ f^1(\emptyset) & = & R \\ f^2(\emptyset) & = & R \cup R^{-1} \\ f^3(\emptyset) & = & R \cup (R \cup R^{-1})^{-1} \\ & = & R \cup R^{-1} \cup (R^{-1})^{-1} \\ & = & R \cup R^{-1} \cup R \\ & = & R \cup R^{-1} \end{array}$$

Logo $\mu f = R \cup R^{-1}$ (*i.é*, o fecho simétrico de R).

Coincidirá $R \cup R^{-1}$ com o maior dos pontos-fixos de f? Em geral, não: é fácil ver que qualquer relação simétrica que contenha R é ponto fixo de f:

$$\begin{array}{rcl} f(S \cup S^{-1}) \text{ tal que } S \text{ inclui } R & = & R \cup (S \cup S^{-1})^{-1} \\ & = & R \cup S^{-1} \cup (S^{-1})^{-1} \\ & = & R \cup S^{-1} \cup S \\ & = & R \cup S \cup S^{-1} \\ & = & S \cup S^{-1} \end{array}$$

A maior de todas essas relações S é, obviamente, o supremo $P \times P$. □

Resolução 2.44: Para $R = \emptyset$ a equação

$$x = 1_P \cup R \cup x^{-1} \cup R \circ x$$

$$((\neg a \vee b) \wedge c) \Rightarrow d$$
$$\updownarrow$$
$$(\neg a \wedge c \vee b \wedge c) \Rightarrow d$$
$$\updownarrow$$
$$(b \wedge c) \Rightarrow d$$

O último passo só é válido quando a é logicamente mais fraco que c.

simplifica-se em

$$x = 1_P \cup x^{-1}$$

Estamos, pois, na situação do Exercício 2.42, para o caso $R = 1_P$. Ter-se-á, de imediato (justifique os passos dados):

$$\begin{array}{rcl}
\mu f & = & 1_P \cup (1_P)^{-1} \\
& = & 1_P \cup \{\langle q, p\rangle \mid \ (p,q) \in 1_P\} \\
& = & 1_P \cup \{\langle p, p\rangle \mid \ (p,p) \in 1_P\} \\
& = & 1_P \cup 1_P \\
& = & 1_P
\end{array}$$

Logo, para $R = \emptyset$, $\mu f = 1_P$.

Para mostrar que, para qualquer R, $P \times P$ é o maior de todos os pontos-fixos de f basta mostar que é ponto fixo (pois $P \times P$ é o limite universal superior do recticulado $2^{P\times P}$). Ter-se-á (justifique os passos dados):

$$\begin{array}{rcl}
f(P \times P) & = & 1_P \cup R \cup (P \times P)^{-1} \cup R \circ (P \times P) \\
& = & 1_P \cup R \cup (P \times P) \cup R \circ (P \times P) \\
& = & P \times P
\end{array}$$

Logo, é ponto fixo. □

Resolução 2.45:

1. f é contínua pelo facto de $\cup$ e $\lambda x.x \circ x$ serem funções monótonas (Teorema 2.3).
2. Continuando:

$$\begin{array}{rcl}
f^4(\emptyset) & = & R \cup (R \cup R^2 \cup R^3 \cup R^4)^2 \\
& = & \bigcup_{j=1}^{8} R^j \\
& \vdots & \\
f^i(\emptyset) & = & \bigcup_{j=1}^{2^{i-1}} R^j \\
& \vdots & \\
f^\infty(\emptyset) & = & \bigcup_{j=1}^{\infty} R^j
\end{array}$$

Logo $\mu f = R^+$, *i.é*, o fecho transitivo de R.

□

Resolução 2.48: Antes de mais, vamos simplificar a equação dada:

$$\begin{array}{rcl}
X & \cong & A \times X + (X \rightharpoonup 0) + B \times X \\
& \updownarrow & \text{por (2.108)} \\
X & \cong & A \times X + (0+1)^X + B \times X \\
& \updownarrow & \text{por (2.42) e (2.107)} \\
X & \cong & A \times X + 1 + B \times X \\
& \updownarrow & \text{por (2.40) e (2.41)} \\
X & \cong & 1 + A \times X + B \times X \\
& \updownarrow & \text{por (2.44)} \\
X & \cong & 1 + (A+B) \times X
\end{array}$$

É fácil de ver que, afinal, estamos perante uma instância de (2.29). Logo a menor solução da equação dada é $(A+B)^\star$. □

Resolução 2.49: Este exercício resolve-se mostrando que $pack$ é injectiva mas não é sobrejectiva, logo não é uma bijecção. □

Resolução 2.50: Faça-se, por exemplo, $A = 0$. Ter-se-á, então,

$$\begin{array}{rcl}
(0^\star)^2 & \cong & 2^\star \times 0^\star \\
& \updownarrow & \\
(1)^2 & \cong & 2^\star \times 1 \\
& \updownarrow & \\
1 & \cong & 2^\star \\
& \updownarrow & \\
& F &
\end{array}$$

onde cada passo deve ser justificado com as respectivas leis de SETS. □

Resolução 2.52: Aplicando o método proposto, ter-se-á:

$$\left\{\begin{array}{rcl}
Exp & \cong & Sinal \times Termo \times (AdiOp \times Termo)^* \\
Sinal & \cong & 1+1+1 \\
Termo & \cong & Factor \times (MulOp \times Factor)^* \\
Factor & \cong & 1+1+Exp \\
AdiOp & \cong & 1+1 \\
MulOp & \cong & 1+1
\end{array}\right.$$

$\updownarrow$ por (2.46) e três substituições

$$\left\{\begin{array}{rcl} Exp & \cong & 3 \times Termo \times (2 \times Termo)^* \\ Termo & \cong & Factor \times (2 \times Factor)^* \\ Factor & \cong & 2 + Exp \end{array}\right.$$

$\updownarrow$ por substituição de $Factor$ e (2.44)

$$\left\{\begin{array}{rcl} Exp & \cong & 3 \times Termo \times (2 \times Termo)^* \\ Termo & \cong & (2 + Exp) \times (4 + 2 \times Exp))^* \end{array}\right.$$

$\updownarrow$ por substituição de $Termo$

$$\left\{\begin{array}{rcl} Exp & \cong & 3 \times ((2 + Exp) \times (4 + 2 \times Exp))^*) \times (2 \times ((2 + Exp) \times (4 + 2 \times Exp))^*))^* \end{array}\right.$$

□

Resolução 2.53:

1. Tem-se: para $i = n = 2$:

$$\begin{array}{rcl} subl(<2,3,1,1,2>,2,2) & = & subl(<3,1,1,2>,1,2) \\ & = & <3> \frown subl(<1,1,2>,1,1) \\ & = & <3> \frown <1> \frown subl(<1,2>,1,0) \\ & = & <3> \frown <1> \frown <> \\ & = & <3,1> \end{array}$$

para $i = n = 3$:

$$\begin{array}{rcl} subl(<2,3,1,1,2>,3,3) & & subl(<3,1,1,2>,2,3) \\ & = & subl(<1,1,2>,1,3) \\ & = & <1> \frown subl(<1,2>,1,2) \\ & = & <1,1> \frown subl(<2>,1,1) \\ & = & <1,1,2> \frown subl(<>,1,0) \\ & = & <1,1,2> \frown <> \\ & = & <1,1,2> \end{array}$$

para $i = 4, n = 3$:

$$\begin{array}{rcl} subl(<2,3,1,1,2>,4,3) & = & subl(<3,1,1,2>,3,3) \\ & = & subl(<1,1,2>,2,3) \\ & = & subl(<1,2>,1,3) \\ & = & <1> \frown subl(<2>,1,2) \end{array}$$

$$\begin{array}{ll} = & <1,2> \frown subl(<>,1,1) \\ = & <1,2> \frown <> \\ = & <1,2> \end{array}$$

A intuição diz-nos que $subl$ extrai de uma sequência l a sua subsequência de n elementos (ou os que fôr possível extrair) que começa na i-esima posição.

2. O facto a provar é

$$\forall x \in A^\star : subl(x, 1, length(x)) = x$$

(= x é idêntica à sua própria sub-sequência de igual comprimento a começar na 1.ª posição).

Prova por indução sobre $A^\star$:

(a) *Caso de Base*: ($x = <>$). Tem-se, neste caso, trivialmente:

$$\begin{array}{rcl} subl(<>, 1, length(<>)) = <> & \Leftrightarrow & <> = <> \\ & \Leftrightarrow & V \end{array}$$

(b) *Salto indutivo*: ($x \neq <>$)

i. *Hipótese de indução*:

$$\forall x \in A^\star : subl(tail(x), 1, length(tail(x))) = tail(x)$$

ii. *Passo*: Teremos, sucessivamente:

$$\begin{array}{rcl} subl(x, 1, length(x)) & \underset{\text{pois } length(x) > 0}{=} & \begin{array}{ll} let & h = head(x) \\ & t = tail(x) \\ in & <h> \frown subl(t, 1, length(x) - 1) \end{array} \\ & \underset{\text{por (A.87)}}{=} & <head(x)> \frown subl(tail(x), 1, length(tail(x))) \\ & \underset{\text{pela H.indução}}{=} & <head(x)> \frown tail(x) \\ & = & x \end{array}$$

como se pretendia demonstrar.

□

Resolução 2.55:

1. Basta fazer $l = <>$. Ter-se-á então

$$\begin{array}{rcl} length(subl(<>, i, n)) & = & n \\ & \updownarrow & \\ length(<>) & = & n \end{array}$$

$$\begin{array}{rcl} & \updownarrow & \\ 0 & = & n \\ & \updownarrow & \\ & F & \end{array}$$

pois $n \geq 0$ em geral.

2. Só a situação em que $n > 0$ e $m > 0$ é que nos deve preocupar. De facto, para $n = 0$ ter-se-á, simplesmente,

$$\begin{array}{rcl} subl(l, i, 0+m) & = & subl(l, i, 0) \frown subl(l, i+0, m) \\ & \updownarrow & \\ subl(l, i, m) & = & <> \frown subl(l, i, m) \\ & \updownarrow & \\ subl(l, i, m) & = & subl(l, i, m) \\ & \updownarrow & \\ & V & \end{array}$$

e, para $m = 0$, simplesmente,

$$\begin{array}{rcl} subl(l, i, n+0) & = & subl(l, i, n) \frown subl(l, i+n, 0) \\ & \updownarrow & \\ subl(l, i, n) & = & subl(l, i, n) \frown <> \\ & \updownarrow & \\ & V & \end{array}$$

Na situação em que $n > 0$ e $m > 0$, o caso de base ($l = <>$) é imediato:

$$\begin{array}{rcl} subl(<>, i, n+m) & = & subl(<>, i, n) \frown subl(<>, i+n, m) \\ & \updownarrow & \\ <> & = & <> \frown <> \\ & \updownarrow & \\ <> & = & <> \\ & \updownarrow & \\ & V & \end{array}$$

No passo indutivo temos ($l \neq <>$) e a seguinte hipótese de indução,

$$subl(t, i, n+m) \quad = \quad subl(t, i, n) \frown subl(t, i+n, m)$$

para $t = tail(l)$. Temos que prever dois casos, $i = 1$ e $i > 1$. No primeiro, teremos (para $h = head(l)$):

$$\begin{array}{rcl} <\!h\!> \frown subl(t, i, n+m-1) & = & <\!h\!> \frown subl(t, i, n-1) \frown subl(l, i+n, m) \\ & \updownarrow & \\ <\!h\!> \frown subl(t, i, n+m-1) & = & <\!h\!> \frown subl(t, i, n-1) \frown subl(t, i+n-1, m) \end{array}$$

pois $i+n>1$. Pela hipótese de indução, para $n-1$ em lugar de n:

$$\begin{array}{rcl} <h> \frown subl(t,i,n+m-1) & = & <h> \frown subl(t,i,n-1) \frown subl(t,i+n-1,m) \\ & \updownarrow & \\ <h> \frown subl(t,i,n+m-1) & = & <h> \frown subl(t,i,n-1+m) \\ & \updownarrow & \\ & V & \end{array}$$

Falta apenas tratar o caso $i>1$ em que, também, se terá $i+n>1$:

$$\begin{array}{rcl} subl(l,i,n+m) & = & subl(l,i,n) \frown subl(l,i+n,m) \\ & \updownarrow & \\ subl(t,i-1,n+m) & = & subl(t,i-1,n) \frown subl(t,i+n-1,m) \\ & \updownarrow & \\ & V & \end{array}$$

pela própria hipótese de indução, para $i-1$ em lugar de i.

□

Resolução 2.56:

1. Propõe-se

$$msetTot(\sigma) \stackrel{\text{def}}{=} \sum_{a\in dom(\sigma)} \sigma(a)$$

mas repare-se que, por $+$ não ser idempotente, $\sum_{a\in dom(\sigma)} \sigma(a) \neq \sum rng(\sigma)$.

2. Propõe-se

$$msetAgl(\rho,\sigma) \stackrel{\text{def}}{=} \left(\begin{array}{c} b \\ msetTot(\rho \mid \{a\in dom(\sigma) \mid \ b=\sigma(a)\}) \end{array}\right)_{b\in rng(\sigma)}$$

— mas é preciso tomar consciência do comportmento desta função quando $dom(\sigma) \subset dom(\rho)\ldots$

3. Faça-se $\sigma=\rho$ acima (o que implica que o codomínio B terá que ser $I\!N$) e ter-se-á:

$$\begin{array}{rl} & \left(\begin{array}{c} b \\ msetTot(\rho \mid \{a\in dom(\rho) \mid \ b=\rho(a)\}) \end{array}\right)_{b\in rng(\rho)} \\ = & \left(\begin{array}{c} b \\ \sum_{a\in dom(\rho)\wedge b=\rho(a)} \rho(a) \end{array}\right)_{b\in rng(\rho)} \\ = & \left(\begin{array}{c} b \\ \sum_{a\in dom(\rho)\wedge b=\rho(a)} b \end{array}\right)_{b\in rng(\rho)} \end{array}$$

É fácil de ver que, se ρ for injectiva, se terá

$$\ldots \quad = \quad \begin{pmatrix} b \\ b \end{pmatrix}_{b \in rng(\rho)}$$

obtendo-se de facto a função identidade em $rng(\rho)$. Mas tal não acontecerá se ρ não for injectiva: em lugar da imagem b teremos $b \times card(\{a \in dom(\rho) \mid b = \rho(a)\})$. O facto proposto é, pois, falso em geral.

□

Resolução 2.57:

1. Tem-se, simplificando $total$:

$$total(\sigma) \stackrel{\text{def}}{=} \sum\{\pi_2(\sigma(k)) \mid k \in dom(\sigma)\}$$

No exemplo sugerido, tem-se $total(\sigma) = \sum\{10\} = 10$. O problema surge do facto de $+$ não ser, em $I\!N$, idempotente: $n + n \neq n$. Logo a repetição de quantias é relevante. Assim, sugere-se:

$$total(\sigma) \stackrel{\text{def}}{=} \sum{<}\pi_2(\sigma(k)) \mid k \leftarrow list(dom(\sigma)){>}$$

onde $list$ é a função que "lista um conjunto" referida na disciplina. A função proposta vem a ser, finalmente, a seguinte função recursiva:

$$total(\sigma) \stackrel{\text{def}}{=} \left\{ \begin{array}{lll} dom(\sigma) = \{\} & \Rightarrow & 0 \\ dom(\sigma) \neq \{\} & \Rightarrow & let \quad k \in dom(\sigma) \\ & & in \quad \pi_2(\sigma(k)) + total(\sigma \setminus \{k\}) \end{array} \right.$$

2. Propõe-se:

$$todosTit(\sigma) \stackrel{\text{def}}{=} \bigcup\{\pi_1(\sigma(k)) \mid k \in dom(\sigma)\}$$

(Repare-se que $\cup$ é idempotente…)

□

Resolução 2.58:

1. Não há "buracos" no plano de estudos:

$$\phi_1(p) \stackrel{\text{def}}{=} \exists n \in I\!N_0 : 2^{[\ A, A\ ]}(p) = \overline{n}$$

onde se emprega a construção (1.30) sobre operadores de selecção [5].

[5] A expressão $2^{[\ A,A\ ]}(p)$ — isto é $2^{[\ 1_A, 1_A\ ]}(p)$ — abrevia, pois, a expressão

$$\{\left\{ \begin{array}{lll} x = i_1(a) & \Rightarrow & A(a) \\ x = i_2(b) & \Rightarrow & A(b) \end{array} \right. \mid a \in p\}$$

2. O número de disciplinas por semestre não pode exceder 5:

$$\phi_2(p) \stackrel{\rm def}{=} \forall i \in 5, a \in 2 : card(aux(p,i,s)) \leq 5$$

onde

$$aux(p,i,s) \stackrel{\rm def}{=} \{d \in p \mid [\,A, A\,](d) = i \wedge [\,R, R\,](d) \in \{s, 3\}\}$$

(onde 3 é o regime de uma disciplina anual).

□

Resolução 2.59:

1. A relação de decomposição de equipamentos em sub-equipamentos deverá ser acíclica (de outra forma teremos equipamentos "infinitos").
Formalmente:

$$\phi(\sigma) \stackrel{\rm def}{=} \begin{array}{ll} let & x = (\#Equip \rightharpoonup dom)(\sigma) \\ & y = \left(\begin{array}{c} e \\ \{e' \mid \langle 2, e'\rangle \in x(e)\} \end{array}\right)_{e \in dom(x)} \\ in & acyclic(discollect(y)) \end{array}$$

onde $discollect$ e $acyclice$ são as funções que vêm nos Exercícios 2.75 e 2.28, respectivamente.

2. Propõe-se aqui uma versão total e não a parcial sugerida no 'template' dado:

$explode : \#Equip \times Equip \longrightarrow Comps$

$explode(e,\sigma) \stackrel{\rm def}{=}$

$$\left\{\begin{array}{lll} e \in dom(\sigma) & \Rightarrow & let \quad b = \sigma(e) \\ & & in \quad \bigoplus_{u \in dom(b)} \left\{\begin{array}{lll} u = \langle 1, c\rangle & \Rightarrow & \left(\begin{array}{c} c \\ b(u) \end{array}\right) \\ u = \langle 2, e'\rangle & \Rightarrow & b(u) \otimes explode(e', \sigma) \end{array}\right. \\ \neg(e \in dom(\sigma)) & \Rightarrow & (\quad) \end{array}\right.$$

□

Resolução 2.60:

1. Tem-se:

$$\begin{array}{rcl} HASH & \cong & AB^{TAM} \\ TAM & \cong & 100 \\ AB & \cong & 1 + arvbin \\ arvbin & \cong & \mathbb{Z} \times INF \times AB \times AB \\ INF & \cong & med \times med \end{array}$$

onde se pode reconhecer, como forma de armazenamento de *sinónimos*, a estrutura *árvore binária*:

$$AB \quad \cong \quad 1 + \mathbb{Z} \times INF \times AB \times AB$$

2. Seja $H : \mathbb{Z} \longrightarrow 100$ a função de 'hashing'. Ter-se-á:

$$\begin{array}{rcl} \phi & : & HASH \longrightarrow 2 \\ \phi(t) & = & \forall i \in TAM : \phi'(i, t(i)) \end{array}$$

onde

$$\begin{array}{rcl} \phi' & : & TAM \times AB \longrightarrow 2 \\ \phi'(i,x) & = & \left\{ \begin{array}{lcl} x = \langle 1, NIL \rangle & \Rightarrow & V \\ x = \langle 2, a \rangle & \Rightarrow & H(\pi_1(a)) = i \wedge \phi'(i, \pi_3(a)) \wedge \phi'(i, \pi_4(a)) \end{array} \right. \end{array}$$

□

Resolução 2.62:

1. Propõe-se:

$$total(\sigma) \stackrel{\text{def}}{=} \sum_{i \in dom(\sigma)} \begin{array}{ll} let & x = \sigma(i) \\ in & \left\{ \begin{array}{lcl} x = i_1(n) & \Rightarrow & n \\ x = i_2(\sigma') & \Rightarrow & total(sigma') \end{array} \right. \end{array}$$

2. O padrão abstracto de Orc é a equação recursiva

$$X \quad \cong \quad A \rightharpoonup (B + X)$$

— para $A = Item$ e $B = I\!N$ — que é exactamente o padrão abstracto de FS (2.94), para $A = Id$ e $B = File$. Recordando o Exercício 2.13, é fácil, por inspecção, fazer as associações seguintes:

- *rubricas* corresponde a $files$
- $cats$ corresponde a $dirs$
- $novaSubcat$ corresponde a $mkdir$.

□

Resolução 2.68:

1. Provar ou refutar o primeiro facto:

$$
\begin{array}{rcl}
 & & \text{Por (2.167) e sugestão dada} \\
A \rightharpoonup \phi(\sigma_1 \cup \sigma_2) & \Leftrightarrow & \displaystyle\bigwedge_{x \in dom(\sigma_1 \cup \sigma_2)} \phi(\sigma_1 \cup \sigma_2(x)) \\
 & & \text{por (A.40)} \\
 & \Leftrightarrow & \displaystyle\bigwedge_{x \in (dom(\sigma_1) \cup dom(\sigma_2))} \phi(\sigma_1 \cup \sigma_2(x)) \\
 & & \text{pela associatividade de } \wedge \text{, generalizada} \\
 & \Leftrightarrow & \displaystyle\bigwedge_{x \in dom(\sigma_1)} \phi(\sigma_1 \cup \sigma_2(x)) \wedge \bigwedge_{x \in dom(\sigma_2)} \phi(\sigma_1 \cup \sigma_2(x)) \\
 & & \text{pela definição 1.6} \\
 & \Leftrightarrow & \displaystyle\bigwedge_{x \in dom(\sigma_1)} \phi(\sigma_1(x)) \wedge \bigwedge_{x \in dom(\sigma_2)} \phi(\sigma_2(x)) \\
 & & \text{de volta a (2.167) e sugestão dada} \\
 & \Leftrightarrow & (A \rightharpoonup \phi(\sigma_1)) \wedge (A \rightharpoonup \phi(\sigma_2))
\end{array}
$$

   Fica provado.

2. Provar ou refutar o segundo facto:

$$
\begin{array}{rcl}
A \rightharpoonup \phi(\sigma) & \Rightarrow & A \rightharpoonup \phi(\sigma \setminus S) \\
 & \updownarrow & \text{por (2.167) e sugestão dada} \\
\displaystyle\bigwedge_{x \in dom(\sigma)} \phi(\sigma(x)) & \Rightarrow & \displaystyle\bigwedge_{x \in dom(\sigma \setminus S)} \phi(\sigma \setminus S(x)) \\
 & \updownarrow & \text{por (A.54)} \\
\displaystyle\bigwedge_{x \in dom(\sigma)} \phi(\sigma(x)) & \Rightarrow & \displaystyle\bigwedge_{x \in dom(\sigma) - S} \phi(\sigma(x)) \\
 & \updownarrow & \text{por (A.1), generalizada} \\
 & V &
\end{array}
$$

   Fica provado.

3. Provar ou refutar o terceiro facto: é imediata a prova a partir de (2.167) e (A.5), que é uma equivalência. Fica provado.

4. Provar ou refutar o quarto facto: note-se que, por (2.167), sugestão dada e generalização da conhecida lei de Morgan, se terá:

$$
\neg(A \rightharpoonup \phi) \quad \Leftrightarrow \quad \bigvee_{x \in dom(\sigma)} \neg(\phi(\sigma(x)))
$$

o que é manifestamente diferente de

$$\bigwedge_{x \in dom(\sigma)} \neg(\phi(\sigma(x))) \quad \Leftrightarrow \quad A \rightharpoonup (\neg\phi)$$

Fica refutado.

□

Resolução 2.69: Sabemos que

$$\sigma_1 \dagger \sigma_2 = \sigma_1 \setminus dom(\sigma_2) \cup \sigma_2$$

cf. (A.57). Logo, pelos factos invocados do Exercício 2.68, ter-se-á, sucessivamente:

$$\begin{array}{rcl} A \rightharpoonup \phi(\sigma_1 \dagger \sigma_2) & = & A \rightharpoonup \phi(\sigma_1 \setminus dom(\sigma_2) \cup \sigma_2) \\ & \Leftrightarrow & A \rightharpoonup \phi(\sigma_1 \setminus dom(\sigma_2)) \wedge A \rightharpoonup \phi(\sigma_2) \\ & \Leftarrow & A \rightharpoonup \phi(\sigma_1) \wedge A \rightharpoonup \phi(\sigma_2) \end{array}$$

QED. □

Resolução 2.70: Propõe-se

$$bestSchedule(db) \stackrel{\text{def}}{=} \left(\begin{array}{c} a \\ bestStart(a, db) \end{array} \right)_{a \in dom(db)}$$

onde — sob garantia de ser acíclico o grafo $\#Act \rightharpoonup \pi_4(db)$, recordar Exercício 2.59 — $bestStart$ é a função [6]

$$bestStart(a, db) \stackrel{\text{def}}{=} \begin{array}{ll} let & x = db(a) \\ & D = \pi_4(x) \\ in & \left\{ \begin{array}{lll} D = \{\} & \Rightarrow & 0 \\ D \neq \{\} & \Rightarrow & MAX_{aa \in D \cap dom(db)} bestStart(aa, db) + \pi_2(db(aa)) \end{array} \right. \end{array}$$

Note-se que esta especificação ignora deliberadamente actividades precedentes desconhecidas. O necessário invariante, impondo a propriedade acíclica acima referida, é semelhante ao proposto no Exercício 2.59. □

Resolução 2.75:

1. O raciocínio é o seguinte:

   (a) $\delta = \circ$ pois é o único operador binário;

[6] Ver mais tarde a segunda alínea do Exercício 9.23.

(b) daí resulta que os resultados de β e α devam ser relações;

(c) de $Key \rightharpoonup elems$ inferimos $b \in Key \rightharpoonup Author^\star$;

(d) logo $a \in 2^{Page \times 2^{Key}}$;

(e) do exposto resulta $\beta = merge$ e $\alpha = discollect$;

(f) falta apenas $\gamma = collect$.

2. Propõe-se:

$$\begin{array}{rcl} listByIndex & : & A^\star \times (A \rightharpoonup B) \longrightarrow (A \times B)^\star \\ listByIndex(l, \sigma) & \stackrel{\mathrm{def}}{=} & <\langle a, \sigma(a)\rangle \mid a \leftarrow l \wedge a \in dom(\sigma)> \end{array}$$

— mas repare-se no comportamento desta função para $dom(\sigma) \subset elems(l)$.

3. Formulação [7]:

$$\begin{array}{ll} (a) & mkAuthorIndex(a, (\quad)) = <> \\ (b) & k \notin dom(b) \Rightarrow mkAuthorIndex(a \cup \{\langle p, \{k\}\rangle\}, b) = mkAuthorIndex(a, b) \end{array}$$

Prova de (a): Repare-se que, por definição da construção $A \rightharpoonup f$, se tem $Key \rightharpoonup elems((\quad)) = (\quad)$ levando a $discollect((\quad)) = \emptyset$. Assim, $merge(a) \circ \emptyset = \emptyset$. Segue-se que $\emptyset^{-1} = \emptyset$ e $collect(\emptyset) = (\quad)$. Segue-se então que $(Str \rightharpoonup NatSort)((\quad)) = (\quad)$. Finalmente, $StrSort(dom((\quad))) = <>$, logo

$$listByIndex(<>, (\quad)) = <>$$

como queríamos.

Prova de (b): a ideia é mostrar que a relação gerada por δ é a mesma, quer para $mkAuthorIndex(a \cup \{\langle p, \{k\}\rangle\}, b)$, quer para $mkAuthorIndex(a, b)$.

Da definição de $merge$ decorre uma propriedade que vai ser útil nesta prova,

$$merge(r \cup s) \quad = \quad merge(r) \cup merge(s)$$

— recordar (2.85). Logo:

$$\begin{array}{rcl} merge(a \cup \{\langle p, \{k\}\rangle\}) & = & merge(a) \cup merge(\{\langle p, \{k\}\rangle\}) \\ & = & merge(a) \cup \{\langle p, k\rangle\} \end{array}$$

De (1.61) decorre, então:

$$(merge(a) \cup \{\langle p, \{k\}\rangle\}) \circ x \quad = \quad (merge(a) \circ x) \cup (\{\langle p, k\rangle\} \circ x)$$

onde x abrevia $discollect(Key \rightharpoonup elems(b))$. Mas

$$\begin{array}{rcl} \{\langle p, k\rangle\} \circ x & = & \{\langle p, \pi_2(q)\rangle \mid \; q \in x \wedge k = \pi_1(q)\} \\ & = & \emptyset \quad \text{isto se } k \notin 2^{\pi_1}(x) \end{array}$$

[7] Supõem-se as fórmulas universalmente quantificadas.

Ora

$$\begin{aligned} 2^{\pi_1}(x) &= dom(discollect(Key \rightharpoonup elems(b))) \\ &= dom(b) \end{aligned}$$

Logo $k \notin 2^{\pi_1}(x)$ é, afinal, o mesmo que $k \notin dom(b)$ o que faz parte da hipótese. Fica assim completa a prova.
QED.

□

Resolução 2.78:

1.

$$\begin{aligned} split(<2,5,4>,2) &= <subl(<2,5,4>,(j-1)\times 2+1,2) \mid j \leftarrow inseq(\left\{ \begin{array}{lll} rem(3,2)=0 & \Rightarrow & 1 \\ \neg(rem(3,2)=0) & \Rightarrow & 2 \end{array} \right.)> \\ &= <subl(<2,5,4>,(j-1)\times 2+1,2) \mid j \leftarrow inseq(2)> \\ &= <subl(<2,5,4>,(j-1)\times 2+1,2) \mid j \leftarrow <1,2>> \\ &= <subl(<2,5,4>,0\times 2+1,2), subl(<2,5,4>,1\times 2+1,2)> \\ &= <subl(<2,5,4>,1,2), subl(<2,5,4>,3,2)> \\ &= <<2,5>,<4>> \end{aligned}$$

Tem-se, de imediato, para X e Y ainda por determinar,

$$split \quad : \quad X \times I\!N \longrightarrow Y^\star$$

pois o resultado de $split$ é uma sequência e a função tem dois parâmetros, dos quais o segundo é passado como 3.º argumento a $subl$ ($I\!N_0$), ainda que o seu tipo tenha de ser reduzido a $I\!N$ para evitar a divisão $div(c,0)$ no corpo de $split$. Como l é 1.º argumento de $subl$, tem-se $X = A^\star$. Como o resultado de $subl$ é também do tipo $A^\star$, ter-se-á, finalmente:

$$split \quad : \quad A^\star \times I\!N_0 \longrightarrow (A^\star)^\star$$

2. $inseq(n)$ calcula a sequência que é a permutação crescente do segmento inicial $\overline{n}$. $split(l,n)$ realiza a partição de uma lista l na lista de sublistas suas de comprimento fixo n, excepto, possivelmente, a última.

3. O facto

$$elems(length^\star(split(l,n))) \quad = \quad \{n\}$$

só se verifica quando $length(l)$ é múltiplo de n. Basta fazer $l = < >$ para o contradizer:

$$elems(length^\star(split(< >,n)))$$

$$
\begin{aligned}
&= elems(length^\star(<<>\,|\, j \leftarrow <1>>)) \\
&= elems(length^\star(<<>>)) \\
&= elems(<0>) \\
&= \{0\} \\
&\neq \{n\}
\end{aligned}
$$

(pois $n > 0$)

□

D.3 Exercícios do Capítulo 3

Resolução 3.2: Partindo da definição de *alcance de* t,

$$[t] = \{\mathcal{W}^\rho(t) \mid \ \rho \in V_\mathcal{W}\}$$

(Definição 3.5), teremos, no nosso caso:

$$
\begin{aligned}
[1+x] &= \{\mathcal{W}^\rho(1+x) \mid \ \rho = \left(\begin{array}{c} x \\ t \end{array}\right) \wedge t \in W_{\Sigma,nat}\} \\
&= \{1+1, 1+(1+1), 1+(1+(1+1)), \ldots\}
\end{aligned}
$$

e

$$
\begin{aligned}
[x+1] &= \{\mathcal{W}^{\rho'}(x+1) \mid \ \rho' = \left(\begin{array}{c} x \\ t' \end{array}\right) \wedge t' \in W_{\Sigma,nat}\} \\
&= \{1+1, (1+1)+1, ((1+1)+1)+1, \ldots\}
\end{aligned}
$$

Para validarmos as afirmações basta calcularmos $[1+x] \cap [x+1]$:

$$[1+x] \cap [x+1] \quad = \quad \{1+t \mid \ t \in W_{\Sigma,nat} \wedge \exists t' \in W_{\Sigma,nat} : 1+t = t'+1\}$$

O sinal "=" na expressão anterior é o de igualdade sintática, a congruência da álgebra $\mathcal{W}$; a este nível o axioma dado é irrelevante (só o seria se a congruência fosse $\cong_E$). Assim, por exemplo, $(1+1)+1 \neq 1+(1+1)$ e só para $t = t' = 1$ é que a igualdade se verifica, isto é:

$$[1+x] \cap [x+1] \quad = \quad \{1+1\}$$

correspondendo a $\rho(x) = \rho'(x) = 1$.

Em suma, só a última afirmação é que é verdadeira. □

Resolução 3.8: Ter-se-á, necessariamente, $\mathsf{A}(f) = length$. Logo, $\mathsf{A}(\rho) = A^\star$ (para um dado conjunto A) e $\mathsf{A}(\mu) = I\!N_0$, reduzindo os nossos graus de liberdade a

$$\begin{cases} f : A^\star \to I\!N_0 \\ u : \to I\!N_0 \\ b : \to A^\star \\ a : I\!N_0 \times I\!N_0 \to I\!N_0 \\ d : \pi \times A^\star \to I\!N_0 \\ e : \pi \times A^\star \to A^\star \end{cases}$$

tal que

$$\begin{array}{rcl} length(b) & \equiv & u \\ length(e(z,x)) & \equiv & a(d(z,x), length(x)) \end{array}$$

(omitem-se os $\mathsf{A}(\ldots)$ para maior legibilidade).

A constante b terá que ser uma lista de As e u o seu comprimento: $b = <>$ e $u = 0$ são uma escolha natural (mas não única, claro). O construtor e é o único em A para listas de As. Se escolhermos $cons$ como sua possível instanciação, para $\mathsf{A}(\pi) = A$, veremos d e a implicitamente instanciados pelo segundo axioma, pois

$$length(cons(z,x)) \quad = \quad 1 + length(x)$$

Ter-se-á, portanto,

$$\begin{array}{rcl} a(y,x) & = & y + x \\ d(z,x) & = & 1 \end{array}$$

o que termina a nossa construção da álgebra que se pede. $\Box$

Resolução 3.20: No primeiro caso temos (o raciocínio decorre de propriedades elementares das operações $-$ e $+$ sobre $I\!N$):

$$\begin{array}{rcl} F(\lambda(n,m).n \times m) & = & \lambda(x,y).\begin{cases} x = 1 & \Rightarrow & y \\ x > 1 & \Rightarrow & y + (x-1) \times y \end{cases} \\ & = & \lambda(x,y).\begin{cases} x = 1 & \Rightarrow & y \\ x > 1 & \Rightarrow & y + x \times y - y \end{cases} \\ & = & \lambda(x,y).\begin{cases} x = 1 & \Rightarrow & y \\ x > 1 & \Rightarrow & y - y + x \times y \end{cases} \\ & = & \lambda(x,y).\begin{cases} x = 1 & \Rightarrow & y \\ x > 1 & \Rightarrow & 0 + x \times y \end{cases} \\ & = & \lambda(x,y).\begin{cases} x = 1 & \Rightarrow & y \\ x > 1 & \Rightarrow & x \times y \end{cases} \end{array}$$

$$\begin{array}{cl} = & \lambda(x,y).\left\{\begin{array}{lcl} x=1 & \Rightarrow & 1\times y \\ x>1 & \Rightarrow & x\times y \end{array}\right. \\ = & \lambda(x,y).\left\{\begin{array}{lcl} x=1 & \Rightarrow & x\times y \\ x>1 & \Rightarrow & x\times y \end{array}\right. \\ \underset{\text{por (B.2)}}{=} & \lambda(x,y).x\times y \end{array}$$

Logo a função dada é ponto fixo da funcional F.

No segundo caso temos:

$$\begin{array}{rcl} F(\lambda(n,m).n^m) & = & \lambda(x,y).\left\{\begin{array}{lcl} x=1 & \Rightarrow & y \\ x>1 & \Rightarrow & y+(x-1)^y \end{array}\right. \\ & = & \lambda(x,y).y+(x-1)^y \end{array}$$

sendo imediatamente óbvio que $\lambda(n,m).n^m$ não é ponto fixo: veja-se o caso $x=1$, em que $x^y=1$ e a função dá y como resultado. $\square$

Resolução 3.22: No primeiro caso, basta mostrar que $[\![\circledcirc(f \texttt{ and } f')]\!]$ é o mesmo significado que $[\![(\circledcirc f) \texttt{ and } (\circledcirc f')]\!]$. Teremos

$$\begin{array}{rcl} [\![\circledcirc(f \texttt{ and } f')]\!] & = & \lambda i.\left\{\begin{array}{lcl} i=0 & \Rightarrow & F \\ i>0 & \Rightarrow & [\![f \texttt{ and } f']\!](i-1) \end{array}\right. \\ & = & \lambda i.\left\{\begin{array}{lcl} i=0 & \Rightarrow & F \\ i>0 & \Rightarrow & [\![f]\!](i-1)\wedge[\![f']\!](i-1) \end{array}\right. \end{array}$$

quanto ao lado esquerdo, e

$$\begin{array}{rcl} [\![(\circledcirc f) \texttt{ and } (\circledcirc f')]\!] & = & \lambda i.[\![\circledcirc f]\!](i)\wedge[\![\circledcirc f']\!](i) \\ & = & \lambda i.(\left\{\begin{array}{lcl} i=0 & \Rightarrow & F \\ i>0 & \Rightarrow & [\![f]\!](i-1) \end{array}\right. \wedge \left\{\begin{array}{lcl} i=0 & \Rightarrow & F \\ i>0 & \Rightarrow & [\![f']\!](i-1) \end{array}\right. \\ & = & \lambda i.\left\{\begin{array}{lcl} i=0 & \Rightarrow & F\wedge F \\ i>0 & \Rightarrow & [\![f]\!](i-1)\wedge[\![f']\!](i-1) \end{array}\right. \\ & = & \lambda i.\left\{\begin{array}{lcl} i=0 & \Rightarrow & F \\ i>0 & \Rightarrow & [\![f]\!](i-1)\wedge[\![f']\!](i-1) \end{array}\right. \end{array}$$

quanto a lado direito. A igualdade é evidente.

Quanto a segundo axioma o processo resume-se a

$$\begin{array}{rcl} [\![\texttt{ not } (\bigcirc f)]\!] & = & \lambda i.\neg([\![\bigcirc f]\!](i)) \\ & = & \lambda i.\neg([\![f]\!](i+1)) \end{array}$$

e

$$\begin{array}{rcl} [\![\bigcirc(\texttt{ not } f)]\!] & = & \lambda i.[\![\texttt{ not } f]\!](i+1) \\ & = & \lambda i.(\lambda j.\neg([\![f]\!](j)))(i+1) \\ & = & \lambda i.\neg([\![f]\!](i+1)) \end{array}$$

chegando-se também a uma igualdade. □

D.4 Exercícios do Capítulo 4

Resolução 4.9:

1. Trata-se do axioma (4.17), como se prova facilmente:

$$
\begin{array}{rcl}
MOD(BinOp(empty)) & = & MOD(unit) \\
 & \updownarrow & \\
MOD(BinOp)(MOD(empty)) & = & I.\epsilon \\
 & \updownarrow & \\
MOD(BinOp)(\emptyset) & = & I.\epsilon \\
 & \updownarrow & \\
\left\{ \begin{array}{lcl} \emptyset = \emptyset & \Rightarrow & I.\epsilon \\ \neg(\emptyset = \emptyset) & \Rightarrow & \begin{array}{ll} let & \ldots \\ in & \ldots \end{array} \end{array} \right. & = & I.\epsilon \\
 & \updownarrow & \\
I.\epsilon & = & I.\epsilon \\
 & \updownarrow & \\
 & V &
\end{array}
$$

2. Comecemos por ver o que é necessário para provar (4.18):

$$
\begin{array}{rcl}
MOD(BinOp(inj(i))) & = & MOD(i) \\
 & \updownarrow & \\
MOD(BinOp)(\{i\}) & = & i \\
 & \updownarrow & \\
\begin{array}{ll} let & a \in \{i\} \\ in & MOD(binOp)(a, MOD(BinOp)(\{i\} - \{a\})) \end{array} & = & i \\
 & \updownarrow & \\
I.\theta(i, MOD(BinOp)(\emptyset)) & = & i \\
 & \updownarrow & \\
I.\theta(i, I.\epsilon) & = & i
\end{array}
$$

Logo, o I-axioma que é preciso para completar a prova acima é

$$\theta(i, \epsilon) \quad \equiv \quad i$$

(isto é, ϵ deve ser elemento neutro de θ).

Passemos agora a (4.19). Repare-se antes de mais que, sendo ϵ elemento neutro de θ, a definição de $BinOp$ tem a estrutura de f_φ dada em (2.80). Assim, tem-se:

$$BinOp(x) \quad = \quad I.\Theta_{a\in x} a$$

desde que θ seja associativa e comutativa:

$$\begin{array}{rcl} \theta(i, \theta(j,k)) & \equiv & \theta(\theta(i,j), k) \\ \theta(i,j) & \equiv & \theta(j,i) \end{array}$$

Passemos agora ao facto a provar,

$$MOD(BinOp(union(x,y))) \quad = \quad MOD(binOp(BinOp(x), BinOp(y)))$$

que se reduz a:

$$I.\Theta_{a\in x\cup y} \quad = \quad I.\theta(I.\Theta_{a\in x}, I.\Theta_{a\in y})$$

Ora é conhecido que este facto só se verifica se $I.\theta$ fôr idempotente, isto é, se

$$I.\theta(i,i) \quad \equiv \quad i$$

Em suma, há 4 axiomas a impôr a I:

$$\begin{array}{rcl} \theta(i,\epsilon) & \equiv & i \\ \theta(i, \theta(j,k)) & \equiv & \theta(\theta(i,j), k) \\ \theta(i,j) & \equiv & \theta(j,i) \\ \theta(i,i) & \equiv & i \end{array}$$

□

Resolução 4.10: Vejamos o que acontece em relação à espécie As:

$$\begin{array}{rcl} (MOD(\mathcal{T}_I))(As) & \cong & \mathcal{T}_J(As) \\ & \updownarrow & \\ 2^{\mathcal{T}_I(Item)} & \cong & 1 \\ & \updownarrow & \\ 2^1 & \cong & 1 \\ & \updownarrow & \\ 2 & \cong & 1 \\ & \updownarrow & \\ & F & \end{array}$$

Logo o isomorfismo não se verifica nesta espécie, o que é suficiente para mostrar que $MOD(\mathcal{T}_I)$ e $\mathcal{T}_J$ não são J-álgebras isomorfas. □

D.5 Exercícios do Capítulo 5

Resolução 5.5:

1. Pela definição de inv-Q e ϕ ficamos a saber que $Q = A \rightharpoonup B$. Logo f será uma função de B para B e g um qualquer observador de Q. Em suma:

$$\begin{array}{lcl} f & : & B \longrightarrow B \\ g & : & (A \rightharpoonup B) \longrightarrow C \end{array}$$

2. Temos

$$(A \rightharpoonup \phi)(\sigma) \quad \stackrel{\text{def}}{=} \quad \forall a \in dom(\sigma) : \phi(\sigma(a))$$

recordar o Exercício 2.68.

Se f preservar ϕ, isso significa que, para todo o $b \in B$, $\phi(b) \Rightarrow \phi(f(b))$, logo, por maioria de razão,

$$\forall a \in dom(\sigma) : \phi(\sigma(a)) \Rightarrow \phi(f(\sigma(a)))$$

Daqui infere-se:

$$(\forall a \in dom(\sigma) : \phi(\sigma(a))) \quad \Rightarrow \quad (\forall a \in dom(\sigma) : \phi(f(\sigma(a))))$$

que é a mesma coisa que

$$(A \rightharpoonup \phi(\sigma)) \quad \Rightarrow \quad (A \rightharpoonup (\phi \circ f)(\sigma))$$

ou a mesma coisa que

$$\text{inv-}Q(\sigma) \quad \Rightarrow \quad \text{inv-}Q((A \rightharpoonup f)(\sigma))$$

que é exactamente o argumento da preservação de inv-Q por E, pois

$$(A \rightharpoonup (\phi \circ f)(\sigma)) \quad = \quad (A \rightharpoonup \phi)(A \rightharpoonup f(\sigma))$$

como se pode facilmente demonstrar.

□

Resolução 5.6:

1. Designaremos por α e β os espaços a completar em E, isto é, temos:

$$\begin{array}{rcl} E & : & \alpha \rightarrow \beta \\ E(a, r') & \stackrel{\text{def}}{=} & \sigma' = g(\sigma, a) \wedge r' = f(\sigma, a) \end{array}$$

tendo-se, obrigatoriamente,

$$\begin{array}{lcl} f & : & Q \times \alpha \to \beta \\ g & : & Q \times \alpha \to Q \end{array}$$

Da definição de f infere-se, de imediato, $\beta = 2$ (Booleanos). Infere-se também que Q terá de ter a estrutura de uma função finita cujo domínio é do mesmo tipo que o contra-domínio de H, isto é, $\mathbb{N}$; ao aplicar H a a ficamos a saber que α é A; infere-se ainda que $\sigma(n)$ é um conjunto a que a pode pertencer; sendo a do tipo A, ter-se-á, em suma:

$$E : A \to 2$$

e

$$Q \cong \mathbb{N} \rightharpoonup 2^A$$

A definição de g confirma esta tipificação.

2. Abstratamente, trata-se de uma operação de arquivo de $a \in A$ numa família de conjuntos de A indexada por $\mathbb{N}$, sendo o índice do conjunto a albergar a identificado pelo cálculo de $H(a)$.

   Podemos pensar numa infinidade de instâncias deste objecto computacional, por exemplo, A um tipo de dados $Aluno$, $H(a)$ o cálculo da média final de curso de a, e Q a estrutura de uma base de dados que arquiva alunos em classes de equivalência indexadas por nota final. Mas todas essas instâncias caem na classe de uma conhecida estrutura computacional, a *tabela de 'hashing'*, onde H é a função de 'hashing' (relembrar figura 1.8 da pág. 48 e respectivo exercício).

3. O facto a provar simplifica em

$$\text{inv-}Q(\sigma) \quad \Rightarrow \quad \text{inv-}Q(g(\sigma, a))$$

   Por (A.5), inv-$Q(\sigma)$ desdobra-se em duas quantificações,

$$\text{inv-}Q_1(\sigma) \wedge \text{inv-}Q_2(\sigma)$$

   onde:

$$\begin{array}{lcl} \text{inv-}Q_1(\sigma) & \stackrel{\text{def}}{=} & \forall n \in dom(\sigma) : \emptyset \neq \sigma(n) \\ \text{inv-}Q_2(\sigma) & \stackrel{\text{def}}{=} & \forall n \in dom(\sigma) : (\forall i \in \sigma(n) : H(i) = n) \end{array}$$

   O facto a provar pode ser demonstrado através de duas provas mais simples,

$$\begin{array}{rcll} \text{inv-}Q_1(\sigma) & \Rightarrow & \text{inv-}Q_1(g(\sigma, a)) & (a) \\ \text{inv-}Q_2(\sigma) & \Rightarrow & \text{inv-}Q_2(g(\sigma, a)) & (b) \\ \hline \underbrace{\text{inv-}Q_1(\sigma) \wedge \text{inv-}Q_2(\sigma)}_{\text{inv-}Q(\sigma)} & \Rightarrow & \underbrace{\text{inv-}Q_1(g(\sigma, a)) \wedge \text{inv-}Q_2(g(\sigma, a))}_{\text{inv-}Q(g(\sigma,a))} & \end{array}$$

   1

Prova de (a):

$$
\begin{array}{lcl}
\text{inv-}Q_1(\sigma) & \Rightarrow & \text{inv-}Q_1(\sigma \dagger \begin{pmatrix} H(a) \\ \{a\} \cup \ldots \end{pmatrix}) \\
& \updownarrow & \\
\text{inv-}Q_1(\sigma) & \Rightarrow & \text{inv-}Q_1(\sigma \setminus \{H(a)\}) \wedge (\emptyset \neq \{a\} \cup \ldots) \\
& \updownarrow & \\
\text{inv-}Q_1(\sigma) & \Rightarrow & \emptyset \neq \{a\} \cup \ldots \\
& \updownarrow & \\
\text{inv-}Q_1(\sigma) & \Rightarrow & V \\
& \updownarrow & \\
& V &
\end{array}
$$

(**NB**: deixa-se ao leitor a tarefa de justificar os passos dados).

Prova de (b) — segue o mesmo esquema da anterior:

$$
\begin{array}{lcl}
\text{inv-}Q_2(\sigma) & \Rightarrow & \text{inv-}Q_2(\sigma \dagger \begin{pmatrix} H(a) \\ \{a\} \cup \ldots \end{pmatrix}) \\
& \updownarrow & \\
\text{inv-}Q_2(\sigma) & \Rightarrow & \text{inv-}Q_2(\sigma \setminus \{H(a)\}) \wedge (\forall i \in \{a\} \cup \ldots : H(i) = H(a)) \\
& \updownarrow & \\
\text{inv-}Q_2(\sigma) & \Rightarrow & \forall i \in \{a\} \cup \ldots : H(i) = H(a) \\
& \updownarrow & \\
\text{inv-}Q_2(\sigma) & \Rightarrow & H(a) = H(a) \wedge \forall i \in \left\{ \begin{array}{lcl} H(a) \in dom(\sigma) & \Rightarrow & \sigma(H(a)) \\ H(a) \notin dom(\sigma) & \Rightarrow & \emptyset \end{array} \right. : H(i) = H(a) \\
& \updownarrow & \\
\text{inv-}Q_2(\sigma) & \Rightarrow & \left\{ \begin{array}{lcl} H(a) \in dom(\sigma) & \Rightarrow & \forall i \in \sigma(H(a)) : H(i) = H(a) \\ H(a) \notin dom(\sigma) & \Rightarrow & V \end{array} \right. \\
& \updownarrow & \\
& V &
\end{array}
$$

(**NB**: deixa-se ao leitor a tarefa de justificar os passos dados).

□

Resolução 5.7:

1.

$$
\begin{array}{lcl}
NULL\_INV & : & IdL \longrightarrow 2^{InvNr} \\
NULL\_INV(id, s') & \stackrel{\text{def}}{=} & \left\{ \begin{array}{lcl} id \in dom(\sigma) & \Rightarrow & \begin{array}{ll} let & x = \pi_2(\sigma(id)) \\ in & \sigma' = \sigma \wedge s' = \overline{Max(dom(x))} - dom(x) \end{array} \end{array} \right.
\end{array}
$$

2.

$$\begin{array}{rcl} SALES\_BY\_DATE & : & Date \longrightarrow (IdA \rightharpoonup I\!N) \\ SALES\_BY\_DATE(d, f') & \stackrel{\text{def}}{=} & let \quad x = puncurry(IdL \rightharpoonup \pi_2(\sigma)) \\ & & \qquad y = \left(\begin{array}{c} k \\ \pi_2(x(k)) \end{array} \right)_{k \in dom(x) \wedge d = \pi_1(x(k))} \\ & & in \quad \sigma' = \sigma \wedge f' = \bigoplus_{k \in dom(y)} y(k) \end{array}$$

onde

$$\begin{array}{rcl} puncurry & : & (A \rightharpoonup (B \rightharpoonup C)) \longrightarrow ((A \times B) \rightharpoonup C) \\ puncurry(\sigma) & \stackrel{\text{def}}{=} & \left(\begin{array}{c} \langle a, b \rangle \\ (\sigma(a))(b) \end{array} \right)_{(a \in dom(\sigma)) \wedge (b \in dom(\sigma(a)))} \end{array}$$

□

D.6 Exercícios do Capítulo 7

Resolução 7.1: O facto a provar é

$$\forall b \in 2^A : \exists s \in A^\star : elems(s) = b$$

Prova por indução sobre 2^A:

1. *Caso de Base*: $(b = \{\})$. Neste caso, o facto a provar reduz-se a

$$\exists s \in A^\star : \{\} = elems(s)$$

   Basta escolher $s = <>$.

2. *Salto indutivo*: $(b \supset \{\})$

   (a) *Hipótese de indução*: Para cada $e \in b$, tem-se

$$\exists r \in A^\star : b - \{e\} = elems(r)$$

   (b) *Passo*: Queremos inferir

$$\exists s \in A^\star : b = elems(s)$$

   a partir da hipótese de indução. Basta escolher $s = <e> \frown r$, pois

$$\begin{array}{rcl} elems(s) = b & \Leftrightarrow & elems(<e> \frown r) = b \\ & \Leftrightarrow & \{e\} \cup elems(r) = b \\ & \Leftrightarrow & \{e\} \cup (b - \{e\}) = b \\ & \Leftrightarrow & b = b \\ & \Leftrightarrow & V \end{array}$$

□

Resolução 7.2: O facto a provar é

$$\forall a \in A, s \in A^\star : belongs(a,s) = a \in elems(s)$$

Prova por indução sobre $A^\star$:

1. *Caso de Base*: ($s = <>$). Tem-se, neste caso

$$\begin{array}{rcl} belongs(a,<>) = a \in elems(<>) & \Leftrightarrow & F = a \in \{\} \\ & \Leftrightarrow & F = F \\ & \Leftrightarrow & V \end{array}$$

2. *Salto indutivo*: ($s \neq <>$)

   (a) *Hipótese de indução*:

$$\forall a \in A : belongs(a, tail(s)) = a \in elems(tail(s))$$

   (b) *Passo*: Teremos, sucessivamente:

$$\begin{array}{rl} belongs(a,s) & \underset{\text{por 'unfolding'}}{=} \ \begin{array}{ll} let & h = head(s) \\ in & \left\{ \begin{array}{rcl} a = h & \Rightarrow & V \\ \neg(a = h) & \Rightarrow & belongs(a, tail(s)) \end{array} \right. \end{array} \\ & \underset{\text{pela H.indução}}{=} \ \begin{array}{ll} let & h = head(s) \\ in & \left\{ \begin{array}{rcl} a = h & \Rightarrow & V \\ \neg(a = h) & \Rightarrow & a \in elems(tail(s)) \end{array} \right. \end{array} \\ & \underset{\text{por (B.6)}}{=} \ \begin{array}{ll} let & h = head(s) \\ in & (a = h) \vee a \in elems(tail(s)) \end{array} \\ & \underset{\text{por substituição de } h}{=} \ (a = head(s)) \vee a \in elems(tail(s)) \\ & \underset{\text{por (D.7)}}{=} \ a \in \{head(s)\} \vee a \in elems(tail(s)) \\ & \underset{\text{por (A.78)}}{=} \ a \in (\{head(s)\} \cup elems(tail(s))) \\ & \underset{\text{por (A.79)}}{=} \ a \in elems(cons(head(s), tail(s)) \\ & \underset{\text{por (D.8)}}{=} \ a \in elems(s) \end{array}$$

como se pretendia demonstrar. Na demonstração recorreu-se aos factos seguintes, tomados como básicos:

$$a \in \{b\} \quad \Leftrightarrow \quad a = b \tag{D.7}$$

$$s \neq < > \quad \Rightarrow \quad cons(head(s), tail(s)) = s \tag{D.8}$$

□

Resolução 7.3: Prova por indução sobre l:

1. *Caso de Base*: ($l = <>$). Tem-se, neste caso

$$\begin{array}{rcl} elems(<> \frown r)) = elems(<>) \cup elems(r) & \Leftrightarrow & elems(r) = \{\} \cup elems(r) \\ & \Leftrightarrow & elems(r) = elems(r) \\ & \Leftrightarrow & V \end{array}$$

2. *Salto indutivo*: ($l \neq <>$)

   (a) *Hipótese de indução*:

$$elems(tail(l) \frown r) = elems(tail(l)) \cup elems(r)$$

   (b) *Passo*: Teremos, sucessivamente:

$$\begin{array}{cl} & elems(l \frown r) \\ \underset{\text{‘unfolding’ e (A.67)}}{=} & elems(\left\{ \begin{array}{lcl} l = <> & \Rightarrow & r \\ \neg(l = <>) & \Rightarrow & cons(head(l), tail(l) \frown r) \end{array} \right.) \\ \underset{\text{por (B.1) e (A.78)}}{=} & \left\{ \begin{array}{lcl} l = <> & \Rightarrow & elems(r) \\ \neg(l = <>) & \Rightarrow & \{head(l)\} \cup elems(tail(l) \frown r) \end{array} \right. \\ \underset{\text{por (A.79)}}{=} & \left\{ \begin{array}{lcl} l = <> & \Rightarrow & elems(r) \\ \neg(l = <>) & \Rightarrow & \{head(l)\} \cup (elems(tail(l)) \cup elems(r)) \end{array} \right. \\ \underset{\text{por (A.26)}}{=} & \left\{ \begin{array}{lcl} l = <> & \Rightarrow & elems(r) \\ \neg(l = <>) & \Rightarrow & (\{head(l)\} \cup elems(tail(l))) \cup elems(r) \end{array} \right. \\ \underset{\text{por (A.78)}}{=} & \left\{ \begin{array}{lcl} l = <> & \Rightarrow & \{\} \cup elems(r) \\ \neg(l = <>) & \Rightarrow & elems(cons(head(l), tail(l)) \cup elems(r) \end{array} \right. \\ \underset{\text{por (D.8) e (A.77)}}{=} & \left\{ \begin{array}{lcl} l = <> & \Rightarrow & elems(l) \cup elems(r) \\ \neg(l = <>) & \Rightarrow & elems(l) \cup elems(r) \end{array} \right. \\ \underset{\text{por (B.2)}}{=} & elems(l) \cup elems(r) \end{array}$$

como se pretendia demonstrar.

O problema é a repetição de elementos comuns a l e r, conduzindo a listas arbitrariamente longas. □

Resolução 7.4: Vamos definir a operação de filtragem pretendida como se segue:

$$filtra(l) \stackrel{\text{def}}{=} \begin{cases} l = <> & \Rightarrow & <> \\ \neg(l = <>) & \Rightarrow & let \quad h = head(l) \\ & & \qquad\quad t = tail(l) \\ & & in \quad \begin{cases} h \in elems(t) & \Rightarrow & t \\ \neg(h \in elems(t)) & \Rightarrow & cons(h, filtra(t)) \end{cases} \end{cases}$$

O facto que vamos pretender provar é o seguinte:

$$\forall l \in A^\star : elems(filtra(l)) = elems(l)$$

Prova por indução sobre $A^\star$:

1. *Caso de Base*: ($l = <>$). Tem-se, neste caso

$$\begin{aligned} elems(filtra(<>)) = elems(<>) \quad & \Leftrightarrow \quad elems(<>) = elems(<>) \\ & \Leftrightarrow \quad V \end{aligned}$$

2. *Salto indutivo*: ($l \neq <>$)

   (a) *Hipótese de indução*:

$$elems(filtra(tail(l))) \quad = \quad elems(tail(l)) \tag{D.9}$$

   (b) *Passo*: Seja $h = head(l)$ e $t = tail(l)$. Há dois casos a considerar, conforme $h \in elems(t)$ se verifica ou não. No primeiro, tem-se que $\{h\} \subseteq elems(t)$. Logo, $\{h\} \cup elems(t) = elems(t)$, ou seja, $elems(cons(h, t)) = elems(t)$, ou seja, $elems(l) = elems(t)$, ou seja, finalmente:

$$elems(filtra(l)) = elems(l)$$

   No segundo caso, ter-se-á $filtra(l) = cons(h, filtra(t))$ e, assim,

$$\begin{aligned} elems(filtra(l)) \quad & = \quad elems(cons(h, filtra(t))) \\ & \underset{\text{por (A.78)}}{=} \quad \{h\} \cup elems(filtra(t))) \\ & \underset{\text{por (D.9)}}{=} \quad \{h\} \cup elems(t) \\ & \underset{\text{por (A.78)}}{=} \quad elems(cons(h, t)) \\ & \underset{\text{por (D.8)}}{=} \quad elems(l) \end{aligned}$$

   como se pretende demonstrar.

□

Resolução 7.5: Modelo cujo estado é dado por

$$\begin{array}{rcl}\Sigma & \cong & top : (\{0\} \cup \overline{max}) \times \\ & & stack : (\overline{max} \rightharpoonup Elem)\end{array}$$

e oferecendo os eventos seguintes:

$$\begin{array}{rcl}TOP & : & \to Elem \\ TOP(e') & \stackrel{\text{def}}{=} & \sigma' = \sigma \wedge \begin{array}{ll} let & s = stack(\sigma) \\ & t = top(\sigma) \\ in & e' = s(t) \end{array}\end{array}$$

$$\begin{array}{rcl}INIT & : & \to \\ INIT() & \stackrel{\text{def}}{=} & \sigma' = \langle 0, stack(\sigma)\rangle\end{array}$$

$$\begin{array}{rcl}POP & : & \to Elem \\ POP(e') & \stackrel{\text{def}}{=} & \begin{array}{ll} let & s = stack(\sigma) \\ & t = top(\sigma) \\ in & e' = s(t) \wedge \sigma' = \langle t-1, s\rangle \end{array}\end{array}$$

$$\begin{array}{rcl}PUSH & : & Elem \to \\ PUSH(e) & \stackrel{\text{def}}{=} & \begin{array}{ll} let & s = stack(\sigma) \\ & t = top(\sigma) \\ in & \sigma' = \langle t+1, s \dagger \begin{pmatrix} t+1 \\ e \end{pmatrix} \rangle \end{array}\end{array}$$

$$\begin{array}{rcl}EMPTY & : & \to Bool \\ EMPTY(b') & \stackrel{\text{def}}{=} & \sigma' = \sigma \wedge b' = (top(\sigma) = 0)\end{array}$$

É óbvio que este modelo, decalcado do código, exibe a insegurança desse próprio código no tocante aos eventos $PUSH$, TOP e POP (ausência das necessárias pre-condições — descubra quais). □

D.7 Exercícios do Capítulo 8

Resolução 8.1:
a) Prova de

$$A \preceq_{1_A} A$$

(1) $A \preceq_{1_A} A \stackrel{\text{def}}{=} \exists_{f:A\longrightarrow A} \cdot\ f$ é sobrejectiva.

(2) Ora $1_A : A \longrightarrow A$ e 1_A é sobrejectiva. Prova:

(2.1) $1_A(a) \quad \stackrel{\text{def}}{=} \quad a$

(2.2) $\forall_{a\in A}.\ \exists_{a'\in A}.\ a = 1_A(a')$

(2.3) Pela definição de 1_A, a' existe e é igual a a.

(2.4) Logo 1_A é sobrejectiva.

b) Prova de

$$(A \preceq_f B \quad \wedge B \preceq_g C) \quad \Rightarrow \quad A \preceq_{f\circ g} C$$

(1) Considere que se verifica $(A \preceq_f B \quad \wedge B \preceq_g C)$

(2) Então verifica-se $A \preceq_f B \quad \stackrel{\text{def}}{=} \quad \exists_{f:B\longrightarrow A} \cdot\ f$ é sobrejectiva.

(3) Verifica-se também $B \preceq_g C \quad \stackrel{\text{def}}{=} \quad \exists_{g:C\longrightarrow B} \cdot\ g$ é sobrejectiva.

(4) Por definição, f é sobrejectiva sse $\forall_{a\in A} \cdot\ \exists_{b\in B} \cdot\ a = f(b)$

(5) Por definição, g é sobrejectiva sse $\forall_{b\in B} \cdot\ \exists_{c\in C} \cdot\ b = g(c)$

(6) Queremos provar que $A \preceq_{f\circ g} C$, isto é, provar que $f \circ g : C \longrightarrow A$ e que $f \circ g$ é sobrejectiva.

(7) Ora, $f : B \longrightarrow A$ e $g : C \longrightarrow B$, logo, pela definicao de composição de funções, $f \circ g : C \longrightarrow A$.

(8) Como por hipótese, f e g são sobrejectivas, $f \circ g$ também é uma função sobrejectiva, como demonstraremos a seguir:

(8.1) $\forall_{a\in A} \cdot (\ \exists_{b\in B} \cdot\ a = f(b))$

(8.2) $\forall_{b\in B} \cdot (\ \exists_{c\in C} \cdot\ b = g(c))$

(8.3) $\forall_{a\in A} \cdot (\ \exists_{b\in B} \cdot\ a = f(b) \wedge\ (\ \exists_{c\in C} \cdot\ b = g(c))$

(8.4) $\forall_{a\in A} \cdot (\ \exists_{b\in B} \cdot (\ \exists_{c\in C} . a = f(b)\ \wedge\ b = g(c)))$

(8.5) $\forall_{a\in A} \cdot (\ \exists_{c\in C} \cdot\ a = f(g(c)))$

(8.6) $\forall_{a\in A} \cdot (\ \exists_{c\in C} \cdot\ a = f \circ g(c))$

c) Prova de

$$(A \preceq_f B \quad \wedge B \preceq_g A) \quad \Rightarrow \quad A \cong B$$

(1) $A \cong B$ sse $\exists_{f:A\longrightarrow B} \cdot f$ é sobrejectiva e injectiva ou $\exists_{g:B\longrightarrow A} \cdot g$ é sobrejectiva e injectiva.

(2) Se em **b)** fizermos $C = A$, podemos concluir que $f \circ g$ e $g \circ f$ existem e são sobrejectivas. Logo f e g são injectivas.

□

Resolução 8.2: Exemplos de representação da sequência $<a, b, c, >$, onde $A = \{a, b, c, d, e, \cdots\}$.

	Exemplo	Modelo de dados sugerido
2		
	$<a, b, c>$	A^*
2		
(1)	$\{a, b, c\}$	2^A
2		
(2)	$\begin{pmatrix} 7 & 10 & 99 \\ a & b & c \end{pmatrix}$	$I\!N \rightharpoonup A$
2		
(3)	$\begin{pmatrix} a & b & c \\ 1 & 2 & 3 \end{pmatrix}$	$(A \rightharpoonup I\!N)_{inv3}$
2		
(4)	$\begin{pmatrix} 1 & 2 & 3 \\ a & b & c \end{pmatrix}$	$(I\!N \rightharpoonup A)_{inv4}$
2		

(1) 2^A não é adequado para representar A^*, pois apesar de todos os elementos de A^* serem representáveis, elementos diferentes de A^* são identificados em 2^A (logo perde-se a capacidade de recuperação).
Exemplo: $<a, b, c>$, $<c, b, a>$, $<b, b, a, c>$, têm todas a mesma representação em 2^A: $\{a, b, c\}$.

(2) $I\!N \rightharpoonup A$ é adequado para representar A^* desde que se convencione um "critério" de ordenação em $I\!N$, *e.g.* a ordem $a < b$. A função de recuperação poderá ser, assim,

$$f(\sigma) \stackrel{\rm def}{=} \begin{cases} \sigma = (\ ) & \Rightarrow \ <> \\ \neg(\sigma = (\ )) & \Rightarrow \ let \quad m = min(dom(\sigma)) \\ & \qquad in \quad <\sigma(m)> \frown f(\sigma \setminus \{m\}) \end{cases} \tag{D.10}$$

onde min calcula o menor inteiro dentro de um conjunto não vazio de inteiros:

$$min(S) \stackrel{\rm def}{=} \begin{cases} card(S) = 1 & \Rightarrow \ the(S) \\ \neg(card(S) = 1) & \Rightarrow \ let \quad x \in dom(\sigma) \\ & \qquad\qquad m = min(S - \{x\}) \\ & \qquad in \quad \begin{cases} x \leq m & \Rightarrow \ x \\ \neg(x \leq m) & \Rightarrow \ m \end{cases} \end{cases}$$

(3) $A \rightharpoonup I\!N$ sujeito ao invariante

$$inv3(f) \stackrel{\rm def}{=} card(rang(f)) = max(rang(f))$$

para

$$max(c) \stackrel{\text{def}}{=} \begin{cases} c = \{\} & \Rightarrow \quad 0 \\ \neg(c = \{\}) & \Rightarrow \quad let \quad \begin{array}{l} e = choice(c) \\ c' = c - \{e\} \\ m' = max(c') \end{array} \\ & \qquad\quad in \quad \begin{cases} e > m' & \Rightarrow \quad e \\ \neg(e > m') & \Rightarrow \quad m' \end{cases} \end{cases}$$

— não é adequado para representar A^*, pois existem elementos de A^* não representáveis em $A \rightharpoonup \mathbb{N}$, isto é, uma eventual função de "recuperação" não seria sobrejectiva o que contraria a definição.
Exemplo: $<a, b, b, c>$ não é representável em $A \rightharpoonup \mathbb{N}$.

(4) $A^* \trianglelefteq_{fr}^{inv4} \mathbb{N} \rightharpoonup A$, onde

$$inv4(f) \stackrel{\text{def}}{=} card(dom(f)) = max(dom(f))$$

para as funções

$$\begin{array}{lcl} fr & : & (\mathbb{N} \rightharpoonup A) \longrightarrow A^* \\ fr(f) & \stackrel{\text{def}}{=} & \begin{cases} f = (\ ) & \Rightarrow \quad <> \\ \neg(f = (\ )) & \Rightarrow \quad <f(1)> \frown fr(shl(f)) \end{cases} \\ & & \\ shl & : & (\mathbb{N} \rightharpoonup A) \longrightarrow (\mathbb{N} \rightharpoonup A) \\ shl(f) & \stackrel{\text{def}}{=} & \begin{pmatrix} i-1 \\ f(i) \end{pmatrix}_{i \in dom(f) \wedge i \neq 1} \end{array}$$

□

Resolução 8.3: Queremos provar

$$2^A \trianglelefteq_{dom} A \rightharpoonup \mathbb{N} \trianglelefteq_{mkms} A^*$$

o que equivale a provar dois factos:

(1) $2^A \trianglelefteq_{dom} A \rightharpoonup \mathbb{N}$
Verifica-se, pois $dom : (A \rightharpoonup \mathbb{N}) \longrightarrow 2^A$ é sobrejectiva, isto é,

$$\forall C \in 2^A : \exists f \in (A \rightharpoonup \mathbb{N}) : c = dom(f)$$

já que, para todo o subconjunto C de A, é sempre possível encontrar uma função $f \in A \rightharpoonup \mathbb{N}$ que o tenha como domínio, por exemplo $\begin{pmatrix} a \\ 1 \end{pmatrix}_{a \in C}$.

(2) $A \rightharpoonup I\!N \trianglelefteq_{mkms} A^*$, onde

$$\begin{array}{rcl} mkms & : & A^* \longrightarrow (A \rightharpoonup I\!N) \\ mkms(l) & \stackrel{\mathrm{def}}{=} & \left(\begin{array}{c} a \\ length(<e \mid e \leftarrow l \wedge e = a>) \end{array} \right)_{a \in elems(l)} \end{array}$$

Teremos de provar que $mkms$ é sobrejectiva, isto é,

$$\forall f \in (A \rightharpoonup I\!N) : \exists l \in A^* : f = mkms(l)$$

Podemos fazê-lo construtivamente, associando, a cada f, a sequência $r(f)$ seguinte:

$$r(f) = Conc(<<a \mid i \in \overline{f(a)}> \mid a \in dom(f)>)$$

onde $Conc(< l_1, \ldots, l_n >) = l_1 \frown \cdots \frown l_n$ e a construção $<\ldots \mid i \in \ldots>$ deve ser convenientemente interpretada (porquê?). Por exemplo, a $f = \left(\; \right)$ associa-se $<>$, a $f = \left(\begin{array}{cc} a & b \\ 2 & 3 \end{array} \right)$ associa-se a lista $<a, a, b, b, b>$, *etc.* Falta apenas mostrar que $f = mkms(r(f))$. Sendo fácil mostrar que $elems(r(f)) = dom(f)$, bastará provar, para cada $a \in dom(f)$, que $length(<e \mid e \leftarrow r(f) \wedge e = a> = f(a)$ se verifica. Ora é fácil de mostrar que $<e \mid e \leftarrow r(f) \wedge e = a> = <a \mid i \in \overline{f(a)}>$, de onde se deduz $length<a \mid i \in \overline{f(a)}> = f(a)$, como se pretendia.

□

Resolução 8.4: Há 4 factos a validar:

(1) $A = B \Rightarrow A \cong B$
Óbvio.

(2) $A \cong B \Rightarrow A \preceq B$
Decorre do facto de $\cong$ ser fecho antissimétrico de $\preceq$.

(3) $A \subseteq B \Rightarrow A \preceq B$
De $A \subseteq B$ decorre que o cardinal de A é inferior ao de B, logo $A \preceq B$.

(4) $A \preceq B \Rightarrow A \trianglelefteq B$
Faça-se $S = B$ em (8.10).

□

Resolução 8.5:

(8.33) $A^0 \cong 1$

$$\lambda a \cdot \left(\; \right)$$

(8.34) $A^{B+C} \cong A^B \times A^C$

$$\lambda \langle x, y \rangle \cdot \left(\begin{array}{c} \langle 1, b \rangle \\ x(b) \end{array} \right)_{b \in B} \cup \left(\begin{array}{c} \langle 2, c \rangle \\ y(c) \end{array} \right)_{c \in C}$$

(8.35) $A^1 \cong A$

$$\lambda a \cdot \begin{pmatrix} NIL \\ a \end{pmatrix}$$

(8.36) $1^A \cong 1$

$$\lambda x \cdot \begin{pmatrix} a \\ NIL \end{pmatrix}_{a \in A}$$

(8.37) $(B \times C)^A \cong B^A \times C^A$

$$\lambda \langle x, y \rangle . \begin{pmatrix} a \\ \langle x(a), y(a) \rangle \end{pmatrix}_{a \in A}$$

□

Resolução 8.6: Partindo de

$$\begin{array}{rcl} curry & : & C^{A \times B} \longrightarrow (C^B)^A \\ \sigma & \rightsquigarrow & \lambda a \cdot (\lambda b \cdot \sigma(a, b)) \end{array}$$

$$\begin{array}{rcl} uncurry & : & (C^B)^A \longrightarrow C^{A \times B} \\ \sigma & \rightsquigarrow & \lambda \langle a, b \rangle \cdot (\sigma(a))(b) \end{array}$$

queremos provar que: $curry = uncurry^{-1}$, isto é, provar que: (a) $curry \circ uncurry = 1_{(C^B)^A}$; e que (b) $uncurry \circ curry = 1_{C^{A \times B}}$.

(a)

$$\begin{array}{rcl} curry \circ uncurry(\sigma) & = & curry(uncurry(\sigma)) \\ & = & curry(\lambda \langle a, b \rangle \cdot (\sigma(a))(b)) \\ & = & \lambda a \cdot (\lambda b \cdot (\lambda \langle a, b \rangle \cdot (\sigma(a))(b)) \langle a, b \rangle) \\ & = & \lambda a \cdot (\lambda b \cdot (\sigma(a))(b)) \\ & = & 1_{(C^B)^A}(\sigma) \\ & = & \sigma \end{array}$$

(b)

$$\begin{array}{rcl} uncurry \circ curry(\sigma) & = & uncurry(curry(\sigma)) \\ & = & uncurry(\lambda a \cdot (\lambda b \cdot \sigma(a, b))) \\ & = & \lambda \langle a, b \rangle \cdot (((\lambda a \cdot (\lambda b \cdot \sigma(a, b)))(a))(b)) \\ & = & \lambda \langle a, b \rangle \cdot ((\lambda b \cdot \sigma(a, b))(b)) \\ & = & \lambda \langle a, b \rangle \cdot \sigma(a, b) \\ & = & 1_{C^{A \times B}}(\sigma) \\ & = & \sigma \end{array}$$

□

Resolução 8.7: Partindo de

$$elems(x) \stackrel{\mathrm{def}}{=} \begin{cases} (x = <>) \Rightarrow \{\} \\ (x \neq <>) \Rightarrow \{head(x)\} \cup elems(tail(x)) \end{cases}$$

e de

$$g(x) \stackrel{\mathrm{def}}{=} \begin{cases} (x = \langle 1, NIL\rangle) \Rightarrow <> \\ (x = \langle 2, \langle a, l\rangle\rangle) \Rightarrow cons(a, g(l)) \end{cases}$$

vamos calcular $f = elems \circ g$:

$$\begin{array}{cl} & f(x) \\ = & (elems \circ g)(x) \\ \underset{\text{`unfolding' de } g}{=} & elems(\begin{cases} (x = \langle 1, NIL\rangle) & \Rightarrow & <> \\ (x = \langle 2, \langle a, l\rangle\rangle) & \Rightarrow & cons(a, g(l)) \end{cases}) \\ \underset{\text{(B.1)}}{=} & \begin{cases} (x = \langle 1, NIL\rangle) & \Rightarrow & elems(<>) \\ (x = \langle 2, \langle a, l\rangle\rangle) & \Rightarrow & elems(cons(a, g(l))) \end{cases} \\ \underset{\text{def. de } elems}{=} & \begin{cases} (x = \langle 1, NIL\rangle) & \Rightarrow & \{\} \\ (x = \langle 2, \langle a, l\rangle\rangle) & \Rightarrow & \{head(cons(a, g(l)))\} \cup elems(tail(cons(a, g(l)))) \end{cases} \\ \underset{\text{(A.65) e (A.66)}}{=} & \begin{cases} (x = \langle 1, NIL\rangle) & \Rightarrow & \{\} \\ (x = \langle 2, \langle a, l\rangle\rangle) & \Rightarrow & \{a\} \cup elems(g(l)) \end{cases} \\ \underset{\text{`folding' via } f}{=} & \begin{cases} (x = \langle 1, NIL\rangle) & \Rightarrow & \{\} \\ (x = \langle 2, \langle a, l\rangle\rangle) & \Rightarrow & \{a\} \cup f(l) \end{cases} \end{array}$$

Em suma, obteve-se:

$$f(x) \quad \stackrel{\mathrm{def}}{=} \quad \begin{cases} (x = \langle 1, NIL\rangle) & \Rightarrow & \{\} \\ (x = \langle 2, \langle a, x'\rangle\rangle) & \Rightarrow & \{a\} \cup f(x') \end{cases} \tag{D.11}$$

□

Resolução 8.8: Defina-se

$$discollect(\sigma) \stackrel{\mathrm{def}}{=} \{\langle a, b\rangle \mid \ a \in dom(\sigma) \wedge b \in \sigma(a)\}$$

Então:

$$collect(discollect(\sigma) \quad = \quad \begin{pmatrix} a \\ \{x \in B \mid \ a \; discollect(\sigma) \; x\} \end{pmatrix}_{a \in \pi_1[discollect(\sigma)]}$$

$$
\begin{aligned}
&= \left(\begin{array}{c} a \\ \{x \in B \mid\ a \in dom(\sigma) \wedge x \in \sigma(a)\} \end{array} \right)_{a \in dom(\sigma)} \\
&= \sigma
\end{aligned}
$$

e

$$
\begin{aligned}
discollect(collect(\rho) &= \{\langle a,b\rangle \mid\ a \in dom(collect(\rho)) \wedge b \in collect(\rho)(a)\} \\
&= \{\langle a,b\rangle \mid\ a \in \pi_1[\rho] \wedge b \in \{x \in B \mid\ a\rho x\}\} \\
&= \{\langle a,b\rangle \mid\ a \in \pi_1[\rho] \wedge a\rho b\} \\
&= \rho
\end{aligned}
$$

□

Resolução 8.10: Provar $(B \rightharpoonup C)^A \cong (A \times B) \rightharpoonup C$:

$$
\begin{aligned}
(B \rightharpoonup C)^A &\cong (8.39) \quad ((C+1)^B)^A \\
&\cong (8.38) \quad (C+1)^{A \times B} \\
&\cong (8.39) \quad (A \times B) \rightharpoonup C
\end{aligned}
$$

NB: poderá adicionalmente ser verificado o seguinte isomorfismo:

$$
\begin{array}{ccc}
(A \times B) \rightharpoonup C & \longrightarrow & (B \rightharpoonup C)^A \\
\sigma & \rightsquigarrow & \lambda a.(\left(\begin{array}{c} b \\ \sigma(a,b) \end{array} \right)_{\langle a,b\rangle \in dom(\sigma)})
\end{array}
\tag{D.12}
$$

□

Resolução 8.11: Considere a seguinte sequência de isomorfismos:

$$
\begin{aligned}
(A \rightharpoonup B) \rightharpoonup C &\cong (8.39) \quad (C+1)^{A \rightharpoonup B} \\
&\cong (8.39) \quad (C+1)^{((B+1)^A)} \\
&\cong (8.38) \quad (C+1)^{A \times (B+1)} \\
&\cong (8.29) \quad (C+1)^{(A \times B)+(A \times 1)} \\
&\cong (8.24) \quad (C+1)^{(A \times B)+A} \\
&\cong (8.34) \quad (C+1)^{A \times B} \times (C+1)^A \\
&\cong (8.39) \quad ((A \times B) \rightharpoonup C) \times (A \rightharpoonup C)
\end{aligned}
$$

No terceiro passo a lei (8.38) está mal referida, pois $(C^A)^B \neq C^{(A^B)}$. □

Resolução 8.13:

$$\begin{array}{rcl} |A \rightharpoonup (B \rightharpoonup C)_+| & = & (|(B \rightharpoonup C)_+| + 1)^{|A|} \\ & = & ((|B \rightharpoonup C| - 1) + 1)^{|A|} \\ & = & |B \rightharpoonup C|^{|A|} \\ & = & ((|C| + 1)^{|B|})^{|A|} \\ & = & (|C| + 1)^{|B| \times |A|} \\ & = & (|C| + 1)^{|B \times A|} \\ & = & |(B \times A) \rightharpoonup C| \\ & = & |(A \times B) \rightharpoonup C| \end{array}$$

Se se fizer $C = 1$, ter-se-á:

$$A \rightharpoonup (B \rightharpoonup 1)_+ \cong (A \times B) \rightharpoonup 1 \quad \Leftrightarrow \quad A \rightharpoonup 2_+^B \cong 2^{A \times B}$$

isto é, obtém-se a lei (8.55) como caso particular. $\square$

Resolução 8.14: A função

$$\sigma \bowtie \tau \quad \stackrel{\mathrm{def}}{=} \quad \left(\begin{array}{c} a \\ \langle \sigma(a), \tau(a) \rangle \end{array} \right)_{a \in dom(\sigma)}$$

será injectiva sobre eqd, sse

$$(\sigma' \bowtie \tau' = \sigma'' \bowtie \tau'') \Rightarrow (\langle \sigma', \tau' \rangle = \langle \sigma'', \tau'' \rangle)$$

sempre que $dom(\sigma') = dom(\tau')$ e $dom(\sigma'') = dom(\tau'')$. Ora

$$\left(\begin{array}{c} a' \\ \langle \sigma'(a'), \tau'(a') \rangle \end{array} \right)_{a' \in dom(\sigma')} = \left(\begin{array}{c} a'' \\ \langle \sigma''(a''), \tau''(a'') \rangle \end{array} \right)_{a'' \in dom(\sigma'')}$$

sse $dom(\sigma') = dom(\sigma'')$ e $\forall a \in dom(\sigma') : (\sigma'(a) = \sigma''(a)) \wedge (\tau'(a) = \tau''(a))$. Logo $\sigma' = \sigma'' \wedge \tau' = \tau''$, o que implica $\langle \sigma', \tau' \rangle = \langle \sigma'', \tau'' \rangle$. $\square$

Resolução 8.15: O facto é verdadeiro e indica que toda a sequência de pares é representável por um par de sequências do mesmo comprimento, adequadamente tipadas. Trata-se de uma lei análoga a (8.65), podendo a sua dedução ser feita de forma semelhante:

$$\begin{array}{rcl} (A \times B)^\star & \stackrel{\mathrm{def}}{=} & \bigcup_{n \geq 0} (A \times B)^n \\ & \cong & \bigcup_{n \geq 0} A^n \times B^n \\ & = & \{\langle l_1, l_2 \rangle \mid \ l_1 \in A^n \wedge l_2 \in B^n \wedge n \geq 0\} \\ & = & \{\langle l_1, l_2 \rangle \mid \ l_1 \in A^\star \wedge l_2 \in B^\star \wedge length(l_1) = length(l_2)\} \\ & = & (A^\star \times B^\star)_{\lambda(l_1, l_2).length(l_1) = length(l_2)} \end{array}$$

Logo,

$$\phi \stackrel{\text{def}}{=} \lambda(l,r).length(l) = length(r)$$

A função de abstracção que se propõe é a adaptação de $\bowtie$ a sequências, *i.é*:

$$f \stackrel{\text{def}}{=} \lambda(l,r).\begin{cases} l = <> & \Rightarrow \quad <> \\ l \neq <> & \Rightarrow \quad <\langle head(l), head(r)\rangle> \frown f(tail(l), tail(r)) \end{cases}$$

Alternativamente, pode fazer-se a dedução de f e ϕ a partir de a alínea (4) do exercício 8.2:

$$\begin{array}{rll} (A \times B)^\star & \trianglelefteq_{fr}^{inv4} & I\!N \rightharpoonup (A \times B) \\ & \trianglelefteq_{\bowtie}^{eqd} & (I\!N \rightharpoonup A) \times (I\!N \rightharpoonup B) \\ & \cong_{f_3} & A^\star \times B^\star \end{array}$$

em cuja resolução fr e $inv4$ são definidos, Sendo f_3 a inversa de fr, deixa-se como exercício a prossecução deste raciocínio. $\Box$

Resolução 8.16: Classe de funções finitas que $discollect$ abstrai na relação vazia:

$$\begin{array}{rl} discollect(\sigma) = \emptyset & \Leftrightarrow \\ \{\langle a,b\rangle \mid \ a \in dom(\sigma) \wedge b \in \sigma(a)\} = \emptyset & \Leftrightarrow \\ a \in dom(\sigma) \wedge b \in \sigma(a) = F & \Leftrightarrow \\ a \notin dom(\sigma) \vee b \notin \sigma(a) & \Leftrightarrow \\ \sigma = \begin{pmatrix} \ \end{pmatrix} \vee \forall a \in dom(\sigma) : \sigma(a) = \emptyset & \end{array}$$

A função

$$\begin{array}{rcl} collect & : & 2^{A\times B} \longrightarrow A \rightharpoonup 2^B \\ collect(\sigma) & \stackrel{\text{def}}{=} & \begin{pmatrix} a \\ \{x \in B \mid \ a\sigma x\} \end{pmatrix}_{a \in \pi_1[\sigma]} \end{array}$$

não é uma função sobrejectiva, pois, por exemplo, para $f = \begin{pmatrix} a \\ \{\} \end{pmatrix}$ em $A \rightharpoonup 2^B$, não existe nenhum elemento $r \in 2^{A\times B}$ tal que $r = collect(r)$. $\Box$

Resolução 8.18: Teremos:

$$\begin{array}{rcl} BAMS & \cong & AccNr \rightharpoonup (2^{AccHolder} \times Amount) \\ & (8.22) \cong_1 & AccNr \rightharpoonup (Amount \times 2^{AccHolder}) \\ & (8.64) \cong_2 & AccNr \rightharpoonup (Amount \times (AccHolder \rightharpoonup 1)) \\ & (8.72) \trianglelefteq_3 & (AccNr \rightharpoonup Amount) \times ((AccNr \times AccHolder) \rightharpoonup 1)) \\ & (8.64) \cong_4 & (AccNr \rightharpoonup Amount) \times (2^{AccNr \times AccHolder}) \\ & (8.9) \trianglelefteq_5 & 2^{AccNr \times Amount} \times 2^{AccNr \times AccHolder} \end{array}$$

onde:

$$
\begin{array}{ll}
\cong_1 & \left\{ \begin{array}{l} f_1 = AccNr \rightharpoonup (\lambda \langle a, b \rangle \cdot \langle b, a \rangle) \\ \phi_1 = \lambda x \cdot V \end{array} \right. \\
\cong_2 & \left\{ \begin{array}{l} f_2 = AccNr \rightharpoonup (id \times dom) \\ \phi_2 = \lambda x \cdot V \end{array} \right. \\
\trianglelefteq_3 & \left\{ \begin{array}{l} f_3 = \bowtie_n \\ \phi_3 = dpi \end{array} \right. \\
\cong_4 & \left\{ \begin{array}{l} f_4 = id \times (\lambda r \cdot \left(\begin{array}{c} p \\ NIL \end{array} \right)_{p \in r}) \\ \phi_4 = \lambda x \cdot V \end{array} \right. \\
\trianglelefteq_5 & \left\{ \begin{array}{l} f_5 = mkf \times id \\ \phi_5 = fdp \times (\lambda x \cdot V) \end{array} \right.
\end{array}
$$

Cálculo do invariante final:

$$
\begin{array}{rcl}
\phi(\langle \rho, \gamma \rangle) & = & \phi_5(\langle \rho, \gamma \rangle) \wedge \\
& & \phi_4(f_5(\langle \rho, \gamma \rangle)) \wedge \\
& & \phi_3(f_4(f_5(\langle \rho, \gamma \rangle))) \wedge \\
& & \phi_2(f_3(f_4(f_5(\langle \rho, \gamma \rangle)))) \wedge \\
& & \phi_1(f_2(f_3(f_4(f_5(\langle \rho, \gamma \rangle))))) \\
& = & \phi_5(\langle \rho, \gamma \rangle) \wedge \phi_3(f_4(f_5(\langle \rho, \gamma \rangle))) \\
& = & fdp(\rho) \wedge \phi_3(f_4(\langle mkf(\rho), \gamma \rangle)) \\
& = & fdp(\rho) \wedge dpi(\langle mkf(\rho), \left(\begin{array}{c} p \\ NIL \end{array} \right)_{p \in \gamma} \rangle) \\
& = & fdp(\rho) \wedge \pi_1[\gamma] \subseteq dom(mkf(\rho)) \\
& = & fdp(\rho) \wedge \pi_1[\gamma] \subseteq \pi_1[\rho]
\end{array}
$$

Em suma, a igualdade presente em (8.85) é relaxada à inclusão, como se previa informalmente. □

Resolução 8.19: No que se segue, designaremos por h a função de abstracção implícita no resultado do exercício 8.13, *i.é*

$$
h(\sigma) \stackrel{\text{def}}{=} \left(\begin{array}{c} a \\ \left(\begin{array}{c} b \\ \sigma(a, b) \end{array} \right)_{\langle a, b \rangle \in dom(\sigma)} \end{array} \right)_{a \in 2^{\pi_1}(dom(\sigma))}
$$

por h' o isomorfismo (D.12), por i a função (D.4) e por j a função

$$
j = \lambda \langle a, b, c \rangle . \langle \langle a, b \rangle, c \rangle \tag{D.13}
$$

Teremos, então:

$$
\begin{array}{rcl}
DPP & \cong & ((\#Comp + \#Equip) \rightharpoonup I\!N) \times \\
& & (\#Comp \rightharpoonup I\!N) \times \\
& & (\#Equip \rightharpoonup ((\#Comp + \#Equip) \rightharpoonup I\!N)_{+}) \\
& \cong_1 & ((\#Comp \rightharpoonup I\!N) \times (\#Equip \rightharpoonup I\!N)) \times \\
& & (\#Comp \rightharpoonup I\!N) \times \\
& & ((\#Equip \times (\#Comp + \#Equip)) \rightharpoonup I\!N)) \\
& \cong_2 & ((\#Comp \rightharpoonup I\!N) \times (\#Equip \rightharpoonup I\!N)) \times \\
& & (\#Comp \rightharpoonup I\!N) \times \\
& & ((\#Equip \times \#Comp) + (\#Equip \times \#Equip)) \rightharpoonup I\!N)) \\
& \cong_3 & ((\#Comp \rightharpoonup I\!N) \times (\#Equip \rightharpoonup I\!N)) \times \\
& & (\#Comp \rightharpoonup I\!N) \times \\
& & (((\#Equip \times \#Comp) \rightharpoonup I\!N) \times ((\#Equip \times \#Equip) \rightharpoonup I\!N)) \\
& \cong_4 & ((\#Comp \rightharpoonup I\!N) \times (\#Comp \rightharpoonup I\!N)) \times \\
& & (\#Equip \rightharpoonup I\!N) \times \\
& & ((\#Equip \times \#Comp) \rightharpoonup I\!N) \times \\
& & ((\#Equip \times \#Equip) \rightharpoonup I\!N) \\
& \trianglelefteq_5 & (\#Comp \rightharpoonup I\!N)^2 \times \\
& & 2^{\#Equip \times I\!N} \times \\
& & 2^{(\#Equip \times \#Comp) \times I\!N} \times \\
& & 2^{(\#Equip \times \#Equip) \times I\!N} \\
& \trianglelefteq_6 & ((2 \times \#Comp) \rightharpoonup I\!N) \times \\
& & 2^{\#Equip \times I\!N} \times \\
& & 2^{\#Equip \times \#Comp \times I\!N} \times \\
& & 2^{\#Equip \times \#Equip \times I\!N} \\
& \trianglelefteq_7 & 2^{2 \times \#Comp \times I\!N} \times \\
& & 2^{\#Equip \times I\!N} \times \\
& & 2^{\#Equip \times \#Comp \times I\!N} \times \\
& & 2^{\#Equip \times \#Equip \times I\!N}
\end{array}
$$

Tem-se ainda (referindo apenas os invariantes relevantes, *i.é* não universalmente verdadeiros):

$$
\begin{array}{ll}
\cong_1 & \{ \; f_1 = f_{(8.63)} \times id \times h \\
\cong_2 & \{ \; f_2 = id \times id \times (i \rightharpoonup I\!N) \\
\cong_3 & \{ \; f_3 = id \times id \times f_{(8.63)} \\
\cong_4 & \{ \; f_4 = \lambda \langle \langle \rho_1, \rho_2 \rangle, \rho_3, \rho_4, \rho_5 \rangle \cdot \langle \langle \rho_1, \rho_3 \rangle, \rho_2, \langle \rho_4, \rho_5 \rangle \rangle
\end{array}
$$

$$\trianglelefteq_5 \quad \left\{ \begin{array}{l} f_5 = id \times mkf \times mkf \times mkf \\ \phi_5 = (\lambda x \cdot V) \times fdp \times fdp \times fdp \end{array} \right.$$

$$\cong_6 \quad \left\{ \; f_6 = h' \times id \times 2^j \times 2^j \right.$$

$$\trianglelefteq_7 \quad \left\{ \begin{array}{l} f_7 = (mkf \circ 2^j) \times id \times id \times id \\ \phi_7 = (fdp \circ 2^j) \times \Pi_{i=1}^{3}(\lambda x \cdot V) \end{array} \right.$$

de que resulta a seguinte função de abstracção global,

$$\begin{array}{rcl} f & = & f_1 \circ f_2 \circ f_3 \circ f_4 \circ f_5 \circ f_6 \circ f_7 \\ & = & (f_{(8.63)} \times id \times (h \circ (i \rightharpoonup I\!N) \circ f_{(8.63)})) \circ f_4 \circ ((h' \circ mkf \circ 2^j) \times id \times mkf \circ 2^j \times mkf \circ 2^j) \end{array}$$

e o seguinte invariante induzido:

$$\phi \quad = \quad (fdp \circ 2^j) \times fdp \times (fdp \circ 2^j) \times (fdp \circ 2^j)$$

que poderão ser ainda mais trabalhados, se necessário. Por exemplo, o termo $fdp \circ 2^j$ conduzirá a:

$$\begin{array}{rcl} (fdp \circ 2^j)(r) & = & fdp(\{\langle\langle a, b\rangle, c\rangle \mid \ \langle a, b, c\rangle \in r\}) \\ & = & fdp(\{\langle\langle \pi_1(t), \pi_2(t)\rangle, \pi_3(t)\rangle \mid \ t \in r\}) \\ & = & \forall t, t' \in r : \pi_1(t) = \pi_1(t') \wedge \pi_2(t) = \pi_2(t') \Rightarrow \pi_3(t) = \pi_3(t') \end{array}$$

etc.

□

Resolução 8.20: Relembre-se a resolução do exercício 8.2:

$$\begin{array}{rcl} A^* & \trianglelefteq_1 & I\!N \rightharpoonup A \\ & \trianglelefteq_2 & 2^{I\!N \times A} \end{array}$$

tendo-se:

$$\trianglelefteq_1 \quad \left\{ \begin{array}{l} f_1 = fr \\ \phi_1 = inv1 \end{array} \right.$$

$$\trianglelefteq_2 \quad \left\{ \begin{array}{l} f_2 = mkf \\ \phi_2 = fdp \end{array} \right.$$

Síntese do invariante induzido a nível relacional:

$$\begin{array}{rcl} \phi(\sigma) & = & \phi_2(\sigma) \wedge \phi_1(f_2(\sigma)) \\ & = & fdp(\sigma) \wedge inv1(mkf(\sigma)) \\ & = & fdp(\sigma) \wedge (card(dom(mkf(\sigma))) = max(dom(mkf(\sigma)))) \\ & = & fdp(\sigma) \wedge (card(\pi_1[\sigma]) = max(\pi_1[\sigma])) \end{array}$$

□

Resolução 8.21: Deduz-se que

$$f \quad = \quad f_1 \circ f_2 \circ f_3$$

onde

$$\begin{array}{rcl} f_1 & = & A \rightharpoonup \langle dom \circ \pi_2, \pi_1 \rangle \\ f_2 & = & \bowtie \\ f_3 & = & mkf \times g \end{array}$$

Começando por f_3, mkf identifica-se como função de abstracção da lei (8.9). Quanto a g, para funcionar como função de abstracção o seu tipo terá que ser $2^X \longrightarrow (X \rightharpoonup 1)$, para um qualquer X, já que tem de ser sobrejectiva. Identifica-se aqui, pois, a lei (8.64). Assim,

$$(A \rightharpoonup B) \times (X \rightharpoonup 1) \quad \trianglelefteq_3 \quad 2^{A \times B} \times 2^X$$

Passando a f_2, $\bowtie$ identifica-se como função de abstracção da lei (8.72) mas, para que f_2 "encaixe" em f_3 é preciso que X seja um par $A \times Y$, para um dado Y. Ter-se-á então:

$$\begin{array}{rcl} A \rightharpoonup B \times (Y \rightharpoonup 1) & \trianglelefteq_2 & (A \rightharpoonup B) \times (A \times Y \rightharpoonup 1) \\ & \trianglelefteq_3 & 2^{A \times B} \times 2^{A \times Y} \end{array}$$

Finalmente, em f_1 apenas temos que inspeccionar $\langle dom \circ \pi_2, \pi_1 \rangle$, que é afinal o mesmo que $\langle dom \circ \pi_2, id \circ \pi_1 \rangle$, sugerindo a aplicação da lei (1.44):

$$\langle dom \circ \pi_2, id \circ \pi_1 \rangle \quad = \quad (dom \times id) \circ \langle \pi_2, \pi_1 \rangle$$

Logo,

$$\begin{array}{rcl} A \rightharpoonup \langle dom \circ \pi_2, id \circ \pi_1 \rangle & = & A \rightharpoonup ((dom \times id) \circ \langle \pi_2, \pi_1 \rangle) \\ & = & (A \rightharpoonup (dom \times id)) \circ (A \rightharpoonup \langle \pi_2, \pi_1 \rangle) \end{array}$$

Há, pois, dois passos de refinamento aqui: no primeiro, a $\langle \pi_2, \pi_1 \rangle$ associa-se a lei (8.22), recordar o exercício 8.5. No nosso contexto, teremos:

$$A \rightharpoonup (Y \rightharpoonup 1) \times B \quad \trianglelefteq_{1.2} \quad A \rightharpoonup B \times (Y \rightharpoonup 1)$$

no segundo, dom aplica $Y \rightharpoonup 1$ em 2^Y — de novo a lei (8.64), agora em sentido inverso — e id aplica B em B: No nosso contexto, teremos:

$$A \rightharpoonup 2^Y \times B \quad \trianglelefteq_{1.1} \quad A \rightharpoonup (Y \rightharpoonup 1) \times B$$

Pondo tudo junto:

$$\begin{array}{rcl} A \rightharpoonup 2^Y \times B & \cong_{1.1} & A \rightharpoonup (Y \rightharpoonup 1) \times B \\ & \cong_{1.2} & A \rightharpoonup B \times (Y \rightharpoonup 1) \\ & \trianglelefteq_2 & (A \rightharpoonup B) \times (A \times Y \rightharpoonup 1) \\ & \trianglelefteq_3 & 2^{A \times B} \times 2^{A \times Y} \end{array}$$

— recordar o exercício 8.18 e sua resolução — para

$$\begin{array}{ll} \cong_{1.1} & \textit{cf.}\ (8.64) \\ \cong_{1.2} & \textit{cf.}\ (8.22) \\ \trianglelefteq_2 & \textit{cf.}\ (8.72) \\ \trianglelefteq_3 & \textit{cf.}\ (8.9)\ \text{e}\ (8.64) \end{array}$$

□

Resolução 8.22: Podemos ignorar o invariante ϕ se re-definirmos $\Sigma = A \rightharpoonup B^+$.

Vamos inferir o invariante a associar a Σ' e compará-lo com o ϕ' proposto. Sabemos, pelo exercício 8.2, que

$$B^* \quad \preceq_f \quad I\!N \rightharpoonup B$$

onde f é dada por (D.10). Logo, ter-se á

$$A \rightharpoonup B^* \quad \preceq_{A \rightharpoonup f} \quad A \rightharpoonup (I\!N \rightharpoonup B)$$

isto é,

$$\begin{array}{rll} A \rightharpoonup B^+ & \preceq_1 & A \rightharpoonup (I\!N \rightharpoonup B)_+ \\ & \cong_2 & (A \times I\!N) \rightharpoonup B \\ & \trianglelefteq_3 & 2^{(A \times I\!N) \times B} \\ & \cong_4 & 2^{A \times I\!N \times B} \end{array}$$

Note-se que para o cálculo do invariante apenas interessam os passos 3 e 4:

$$\begin{array}{ll} \trianglelefteq_3 & \left\{ \begin{array}{l} f_3 = mkf \\ \phi_3 = fdp \end{array} \right. \\ \cong_4 & \left\{ \; f_4 = 2^j \right. \end{array}$$

onde j é a função (D.13). O invariante resultante vem então a ser $fdp \circ 2^j$ que, como se viu no final do exercício 8.19, corresponde a:

$$\phi'' \quad = \quad \forall t, t' \in r : \pi_1(t) = \pi_1(t') \wedge \pi_2(t) = \pi_2(t') \Rightarrow \pi_3(t) = \pi_3(t')$$

Repare-se que o padrão deste predicado,

$$p \wedge q \Rightarrow r$$

quando comparado com o de ϕ',

$$p \Rightarrow \neg q$$

mostra ser

$$\phi'' = \phi' \vee r$$

Logo ϕ' é mais restrito que ϕ''. □

Resolução 8.30:

1. De acordo com (8.157) teremos, para o diagrama (a)

$$\begin{array}{l}((\#Disciplina \rightharpoonup \#Departamento \times Regente)\times \\ (\#Departamento \rightharpoonup Nome)\times \\ (\#Disciplina \rightharpoonup Nome))_\gamma\end{array}$$

   e para o diagrama (b)

$$\begin{array}{l}((\#Disciplina \rightharpoonup \#Departamento \times 1)\times \\ (\#Departamento \rightharpoonup Nome)\times \\ (\#Disciplina \rightharpoonup Nome \times Regente))_\gamma\end{array}$$

   ambos os modelos sujeitos ao mesmo invariante:

$$\gamma(\rho,\sigma,\sigma') \quad \stackrel{\rm def}{=} \quad dom(\rho) = dom(\sigma') \wedge 2^{\pi_1}(rng(\rho)) \subseteq dom(\sigma)$$

   É fácil agregar ρ e σ' em ambos os modelos, reconhecendo em $dom(\rho) = dom(\sigma')$ o invariante eqd (8.66) associado à lei (8.69). Logo, o modelo de (a) é afinal o refinamento de

$$\begin{array}{l}((\#Disciplina \rightharpoonup \#Departamento \times Regente \times Nome)\times \\ (\#Departamento \rightharpoonup Nome))_{\gamma'}\end{array}$$

   e o modelo de (b) o refinamento de

$$\begin{array}{l}((\#Disciplina \rightharpoonup \#Departamento \times Nome \times Regente)\times \\ (\#Departamento \rightharpoonup Nome))_{\gamma'}\end{array}$$

   (eliminando também 1, elemento neutro de $\times$) sujeitos a um invariante mais leve:

$$\gamma(\rho,\sigma) \quad \stackrel{\rm def}{=} \quad 2^{\pi_1}(rng(\rho)) \subseteq dom(\sigma)$$

   Por (8.22) os dois modelos são isomorfos e, portanto, indistinguíveis sob o ponto de vista de especificação. Não vale a pena discutir — é a ajuda que lhes deve dar!

2. Conversão de (b) para (a):

$$\begin{array}{lll} f(\rho,\sigma,\sigma') \quad \stackrel{\rm def}{=} & let & \sigma'_1 = \#Disciplina \rightharpoonup \pi_1(\sigma') \\ & & \sigma'_2 = \#Disciplina \rightharpoonup \pi_2(\sigma') \\ & & \rho_1 = \#Disciplina \rightharpoonup \pi_1(\rho) \\ & in & \langle \rho_1 \bowtie \sigma'_2, \sigma, \sigma'_1 \rangle \end{array}$$

□

Resolução 8.33: No Exercício 8.15 referem-se dois desses factos, o que se aí propõe e a lei (8.65) de que se partiu. As funções de representação correspondentes são, respectivamente,

$$f^{-1} \quad \stackrel{\rm def}{=} \quad \langle {\pi_1}^\star, {\pi_2}^\star \rangle$$

e

$$\bowtie^{-1} \stackrel{\mathrm{def}}{=} \langle A \rightharpoonup \pi_1, A \rightharpoonup \pi_2 \rangle$$

Outras duas leis nesta situação são (8.9) — propondo-se

$$mkf^{-1}(\sigma) \stackrel{\mathrm{def}}{=} \{\langle a, \sigma(a)\rangle \mid a \in dom(\sigma)\}$$

— e (8.72), propondo-se

$$\underset{n}{\bowtie}^{-1} \stackrel{\mathrm{def}}{=} \langle A \rightharpoonup \pi_1, uncurry \circ (A \rightharpoonup \pi_2)\rangle$$

onde

$$uncurry(t) \stackrel{\mathrm{def}}{=} \begin{pmatrix} \langle b, c\rangle \\ t(b)(c) \end{pmatrix}_{b \in dom(t) \wedge c \in dom(t(b))}$$

□

Resolução 8.34: Propõe-se

$$f(\langle x, y\rangle) \stackrel{\mathrm{def}}{=} \langle boolgate(x, y, V), boolgate(x, y, F)\rangle$$

para a função auxiliar

$$boolgate(x, y, b) \stackrel{\mathrm{def}}{=} \begin{cases} x = <> \wedge\, y = <> & \Rightarrow\ <> \\ x \neq <> \wedge\, y \neq <> & \Rightarrow\ (\begin{cases} head(x) = b & \Rightarrow\ head(y) \\ \neg(head(x) = b) & \Rightarrow\ <> \end{cases}) \frown boolgate(tail(x), tail(y), b) \end{cases}$$

— recordar Exercício 2.11.

A função de abstracção f tem duas razões para não ser injectiva. Primeiro, a possível desigualdade de comprimentos de x e y, eliminável através de um invariante

$$\phi(\langle x, y\rangle) \stackrel{\mathrm{def}}{=} length(x) = length(y)$$

Em segundo lugar, dadas duas sequências t e f, o seu entrelaçamento numa só sequ?encia y (dado por x quando $\langle t, f\rangle = f(\langle x, y\rangle)$) é múltiplo. Por exemplo, o par $\langle <a>, <a>\rangle$ será representável tanto por $\langle <1, 0>, <a, a>\rangle$ como por $\langle <0, 1>, <a, a>\rangle$. □

Resolução 8.37:

1. Pelo exercício sugerido tem-se

$$\begin{array}{rcl} 2^B & \trianglelefteq_{elems} & B^\star \\ & \trianglelefteq_{g_{Ex.8.7}} & L \end{array}$$

(onde $L \cong 1 + B \times L$) tendo sido no mesmo exercício calculada a função de abstracção composta $f = elems \circ g_{Ex.8.7}$, que doravante designaremos por $f_{(D.11)}$, veja-se (D.11).

Assim:

$$\begin{array}{rll} A \rightharpoonup B^\star & \cong_{A \rightharpoonup g_{Ex.8.7}} & A \rightharpoonup L \\ & \trianglelefteq_{mkf} & 2^{A \times L} \\ & \trianglelefteq_{elems} & (A \times L)^\star \\ & \trianglelefteq_{g_{Ex.8.7}} & X \ \ onde \ \ X \cong 1 + (A \times L) \times X \end{array}$$

Em suma, temos:

$$\left\{ \begin{array}{rcl} PQueue(A, B) & \trianglelefteq & X \\ X & \cong & 1 + (A \times L) \times X \\ L & \cong & 1 + B \times L \end{array} \right.$$

Codificando em C, teremos qualquer coisa como:

```
struct X {
   A key;
   L *queue;
   struct X *next;
};
struct L {
   B first;
   struct L *rest;
};
```

2. Queremos calcular

$$f = (A \rightharpoonup g_{Ex.8.7}) \circ \underbrace{mkf \circ f_{(D.11)}}_{h}$$

Comecemos por h:

$$\begin{array}{rcl} h(x) & = & mkf(f_{(D.11)}(x)) \\ & = & \left\{ \begin{array}{rcl} x = \langle 1, NIL \rangle & \Rightarrow & mkf(\{\}) \\ x = \langle 2, \langle \langle a, l \rangle, x' \rangle \rangle & \Rightarrow & mkf(\{\langle a, l \rangle\} \cup f_{(D.11)}(x')) \end{array} \right. \end{array}$$

Repare-se, pela definição de mkf, que:

$$\begin{array}{rcl} mkf(\{\}) & = & \left(\quad \right) \\ mkf(\rho \cup \gamma) & = & mkf(\rho) \cup mkf(\gamma) \\ mkf(\{\langle a, b \rangle\}) & = & \left(\begin{array}{c} a \\ b \end{array} \right) \end{array}$$

Logo,

$$h(x) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{ccc} x = \langle 1, NIL\rangle & \Rightarrow & \left(\ \right) \\ x = \langle 2, \langle\langle a, b\rangle, x'\rangle\rangle & \Rightarrow & \left(\begin{array}{c} a \\ b \end{array}\right) \cup h(x') \end{array} \right. \tag{D.14}$$

Finalmente,

$$\begin{array}{rcl} f(x) & = & (A \rightharpoonup g_{Ex.8.7})(h(x)) \\ & = & \left\{ \begin{array}{ccc} x = \langle 1, NIL\rangle & \Rightarrow & \left(\ \right) \\ x = \langle 2, \langle\langle a, l\rangle, x'\rangle\rangle & \Rightarrow & \left(\begin{array}{c} a \\ g_{Ex.8.7}(l) \end{array}\right) \cup \underbrace{(A \rightharpoonup g_{Ex.8.7})(h(x'))}_{f(x')} \end{array} \right. \end{array}$$

pois

$$\begin{array}{rcl} (A \rightharpoonup f)(\left(\ \right)) & = & \left(\ \right) \\ (A \rightharpoonup f)(\sigma \cup \sigma') & = & (A \rightharpoonup f)(\sigma) \cup (A \rightharpoonup f)(\sigma') \end{array}$$

Em suma:

$$f(x) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{ccc} x = \langle 1, NIL\rangle & \Rightarrow & \left(\ \right) \\ x = \langle 2, \langle\langle a, l\rangle, x'\rangle\rangle & \Rightarrow & \left(\begin{array}{c} a \\ g_{Ex.8.7}(l) \end{array}\right) \cup f(x') \end{array} \right.$$

□

Resolução 8.38: Basta aplicar (8.61) para $A = 1$. Com alguma "re-escrita" teremos:

$$A \trianglelefteq_{\pi_2}^{\lambda x.\text{is-}A(x)} 1 + A$$

ou seja, exclui-se o valor NIL para o apontador em causa. □

Resolução 8.40:

1. X é a implementação, *cf.*:

$$\begin{array}{rcl} Y & = & Id \rightharpoonup (Page \times 2^{LnkId \times Id}) \\ & \cong & Id \rightharpoonup (Page \times (LnkId \times Id \rightharpoonup 1)) \\ & \trianglelefteq_{\bowtie}^{dpi} & (Id \rightharpoonup Page) \times (Id \times LnkId \times Id \rightharpoonup 1) \\ & \cong & (Id \rightharpoonup Page) \times 2^{Id \times LnkId \times Id} \\ & = & X \end{array}$$

Só há lugar a invariante a partir do 2.º passo:

$$\begin{array}{rcl}
\phi(\sigma,\rho) & = & \phi_2(f_3(\sigma,\rho)) \\
 & = & dpi((id \times \lambda S. \left(\begin{array}{c} x \\ NIL \end{array}\right)_{x \in S})(\sigma,\rho)) \\
 & = & dpi(\sigma, \left(\begin{array}{c} x \\ NIL \end{array}\right)_{x \in \rho}) \\
 & = & \pi_1[\rho] \subseteq dom(\sigma)
\end{array}$$

□

D.8 Exercícios do Capítulo 9

Resolução 9.1: Teremos, sucessivamente [8],

$$\begin{array}{l}
elems(x(l_1,l_2)) = elems(l_1) \cup elems(l_2) \\
\quad = \quad (\left\{\begin{array}{rcl} l_1 = <> & \Rightarrow & \emptyset \\ \neg(l_1 = <>) & \Rightarrow & \{head(l_1)\} \cup elems(tail(l_1)) \end{array}\right.) \cup elems(l_2) \\
\quad = \quad \left\{\begin{array}{rcl} l_1 = <> & \Rightarrow & elems(l_2) \\ \neg(l_1 = <>) & \Rightarrow & (\{head(l_1)\} \cup elems(tail(l_1))) \cup elems(l_2) \end{array}\right. \\
\quad = \quad \left\{\begin{array}{rcl} l_1 = <> & \Rightarrow & elems(l_2) \\ \neg(l_1 = <>) & \Rightarrow & elems(tail(l_1)) \cup (\{head(l_1)\} \cup elems(l_2)) \end{array}\right. \\
\quad = \quad \left\{\begin{array}{rcl} l_1 = <> & \Rightarrow & elems(l_2) \\ \neg(l_1 = <>) & \Rightarrow & elems(tail(l_1)) \cup \left\{\begin{array}{rcl} head(l_1) \in elems(l_2) & \Rightarrow & elems(l_2) \\ \neg(head(l_1) \in elems(l_2)) & \Rightarrow & \{head(l_1)\} \cup elems(l_2) \end{array}\right. \end{array}\right. \\
\quad = \quad \left\{\begin{array}{rcl} l_1 = <> & \Rightarrow & elems(l_2) \\ \neg(l_1 = <>) & \Rightarrow & \left\{\begin{array}{rcl} belongs(head(l_1),l_2) & \Rightarrow & elems(tail(l_1)) \cup elems(l_2) \\ \neg(belongs(head(l_1),l_2)) & \Rightarrow & elems(tail(l_1)) \cup \{head(l_1)\} \cup elems(l_2) \end{array}\right. \end{array}\right. \\
\quad = \quad \left\{\begin{array}{rcl} l_1 = <> & \Rightarrow & elems(l_2) \\ \neg(l_1 = <>) & \Rightarrow & \left\{\begin{array}{rcl} belongs(head(l_1),l_2) & \Rightarrow & elems(x(tail(l_1),l_2)) \\ \neg(belongs(head(l_1),l_2)) & \Rightarrow & elems(<head(l_1)> \frown x(tail(l_1),l_2)) \end{array}\right. \end{array}\right. \\
\quad = \quad elems(\left\{\begin{array}{rcl} l_1 = <> & \Rightarrow & l_2 \\ \neg(l_1 = <>) & \Rightarrow & \left\{\begin{array}{rcl} belongs(head(l_1),l_2) & \Rightarrow & x(tail(l_1),l_2) \\ \neg(belongs(head(l_1),l_2)) & \Rightarrow & <head(l_1)> \frown x(tail(l_1),l_2) \end{array}\right. \end{array}\right.)
\end{array}$$

[8] Justifique cada passo do raciocínio.

Em suma, obtemos:

$$x(l_1, l_2) \stackrel{\text{def}}{=}$$

$$\left\{\begin{array}{lcl} l_1 = < > & \Rightarrow & l_2 \\ \neg(l_1 = < >) & \Rightarrow & \left\{\begin{array}{lcl} belongs(head(l_1), l_2) & \Rightarrow & < > \\ \neg(belongs(head(l_1), l_2)) & \Rightarrow & <head(l_1)> \end{array}\right. \frown x(tail(l_1), l_2) \end{array}\right.$$

□

Resolução 9.2: Anotem-se as seguintes propriedades de $card$:

$$card(\emptyset) = 0 \tag{D.15}$$

$$card(\{a\}) = 1 \tag{D.16}$$

$$A \cap B = \emptyset \Rightarrow card(A \cup B) = card(A) + card(B) \tag{D.17}$$

Tem-se a seguinte equação de refinamento:

$$comp(s) = card(elems(s))$$

que vamos resolver em ordem a $comp$:

$$comp(s)$$

$$= \quad card(elems(s))$$

$$\underset{\text{‘unfolding’ de } elems}{=} \quad card(\left\{\begin{array}{lcl} s = < > & \Rightarrow & \emptyset \\ s \neq < > & \Rightarrow & \{head(s)\} \cup elems(tail(s)) \end{array}\right.)$$

$$\underset{\text{(B.1)}}{=} \quad \left\{\begin{array}{lcl} s = < > & \Rightarrow & card(\emptyset) \\ s \neq < > & \Rightarrow & card(\{head(s)\} \cup elems(tail(s))) \end{array}\right.$$

$$\underset{\text{(D.15) e (D.17)}}{=} \quad \left\{\begin{array}{lcl} s = < > & \Rightarrow & 0 \\ s \neq < > & \Rightarrow & \left\{\begin{array}{lcl} \{head(s)\} \cap elems(tail(s)) = \emptyset & \Rightarrow & card(\{head(s)\} + card(elems(tail(s))) \\ \neg(\{head(s)\} \cap elems(tail(s)) = \emptyset) & \Rightarrow & card(\{head(s)\} \cup elems(tail(s))) \end{array}\right. \end{array}\right.$$

$$\underset{\text{(D.16) e } \{head(s)\} \subseteq elems(tail(s))}{=} \quad \left\{\begin{array}{lcl} s = < > & \Rightarrow & 0 \\ s \neq < > & \Rightarrow & \left\{\begin{array}{lcl} \{head(s)\} \cap elems(tail(s)) = \emptyset & \Rightarrow & 1 + card(elems(tail(s))) \\ \neg(\{head(s)\} \cap elems(tail(s)) = \emptyset) & \Rightarrow & card(elems(tail(s))) \end{array}\right. \end{array}\right.$$

$$\underset{\text{introdução de } let}{=} \quad \left\{\begin{array}{lcl} s = < > & \Rightarrow & 0 \\ s \neq < > & \Rightarrow & \begin{array}{ll} let & h = head(s) \\ & t = tail(s) \\ in & \left\{\begin{array}{lcl} h \in elems(t) & \Rightarrow & card(elems(t)) \\ \neg(h \in elems(t)) & \Rightarrow & 1 + card(elems(t)) \end{array}\right. \end{array} \end{array}\right.$$

$$\underset{\text{‘folding’ por } comp}{=} \quad \left\{\begin{array}{lcl} s = < > & \Rightarrow & 0 \\ s \neq < > & \Rightarrow & \begin{array}{ll} let & h = head(s) \\ & t = tail(s) \\ in & \left\{\begin{array}{lcl} h \in elems(t) & \Rightarrow & comp(t) + 0 \\ \neg(h \in elems(t)) & \Rightarrow & comp(t) + 1 \end{array}\right. \end{array} \end{array}\right.$$

$$\underset{\text{(B.1)}}{=} \left\{ \begin{array}{lll} s = <> & \Rightarrow & 0 \\ s \neq <> & \Rightarrow & \begin{array}{ll} let & h = head(s) \\ & t = tail(s) \\ in & comp(t) + \left\{ \begin{array}{lll} belongs(h,t) & \Rightarrow & 0 \\ \neg(belongs(h,t)) & \Rightarrow & 1 \end{array} \right. \end{array} \end{array} \right.$$

Em suma, calculamos (9.10), como se pretendia. □

Resolução 9.3:

1. Justificação dos passos dados: (1) 'unfold' de $elems$; (2) regra (B.1) e $\{\} - B = \{\}$; (3) distributividade de $-$ (A.17) por $\cup$; (4) 'folding' via equação de partida.
2. Prosseguindo (sugere-se que o leitor justifique os novos passos):

$$\ldots = \left\{ \begin{array}{lll} l = <> & \Rightarrow & \emptyset \\ l \neq <> & \Rightarrow & (\{head(l)\} - elems(r)) \cup elems(dif(tail(l), r)) \end{array} \right.$$

$$= \left\{ \begin{array}{lll} l = <> & \Rightarrow & \emptyset \\ l \neq <> & \Rightarrow & (\left\{ \begin{array}{lll} head(l) \in elems(r) & \Rightarrow & \emptyset \\ \neg(head(l) \in elems(r)) & \Rightarrow & \{head(l)\} \end{array} \right.) \cup elems(dif(tail(l), r)) \end{array} \right.$$

$$= \left\{ \begin{array}{lll} l = <> & \Rightarrow & \emptyset \\ l \neq <> & \Rightarrow & \left\{ \begin{array}{lll} head(l) \in elems(r) & \Rightarrow & elems(dif(tail(l), r)) \\ \neg(head(l) \in elems(r)) & \Rightarrow & \{head(l)\} \cup elems(dif(tail(l), r)) \end{array} \right. \end{array} \right.$$

$$= elems(\left\{ \begin{array}{lll} l = <> & \Rightarrow & <> \\ l \neq <> & \Rightarrow & \left\{ \begin{array}{lll} belongs(head(l), r) & \Rightarrow & dif(tail(l), r) \\ \neg(belongs(head(l), r)) & \Rightarrow & cons(head(l), dif(tail(l), r)) \end{array} \right. \end{array} \right.)$$

obtendo-se finalmente, por eliminação de $elems$ de ambos os lados da equação, a definição:

$$dif(l, r) \stackrel{\text{def}}{=} \left\{ \begin{array}{lll} l = <> & \Rightarrow & <> \\ l \neq <> & \Rightarrow & \left\{ \begin{array}{lll} belongs(head(l), r) & \Rightarrow & dif(tail(l), r) \\ \neg(belongs(head(l), r)) & \Rightarrow & cons(head(l), dif(tail(l), r)) \end{array} \right. \end{array} \right.$$

□

Resolução 9.5:

1. Justificações:
   - Passo de (9.18) para (9.19) — facto (1.36) e posterior contracção de $sq \circ id$ em sq (identidade da composição).
   - Passo de (9.19) para (9.20) — Propriedade em $\mathbb{Z}_0$: $(-x)^2 = x^2$
   - Passo de (9.20) para (9.21) — Introdução de id (identidade da composição).

- Passo de (9.21) para (9.22) — facto (1.31), para $f = id$ e $g = sym$.

2. Tendo-se obtido, na alínea anterior,

$$([\, id, sym \,]) \circ \alpha \quad = \quad [\, id, sym \,] \circ i_1 \circ [\, sq, sq \,]$$

obtem-se agora, por eleminação da função de abstracção:

$$\alpha \quad \stackrel{\mathrm{def}}{=} \quad i_1 \circ [\, sq, sq \,]$$

isto é,

$$\begin{array}{rcl}
\alpha(x) & \stackrel{\mathrm{def}}{=} & i_1([\, sq, sq \,](x)) \\
& = & i_1(\left\{ \begin{array}{lcl} x = i_1(a) & \Rightarrow & sq(a) \\ x = i_2(b) & \Rightarrow & sq(b) \end{array} \right.) \\
& = & \left\{ \begin{array}{lcl} x = i_1(a) & \Rightarrow & i_1(sq(a)) \\ x = i_2(b) & \Rightarrow & i_1(sq(b)) \end{array} \right. \\
& = & \left\{ \begin{array}{lcl} x = i_1(a) & \Rightarrow & i_1(a^2) \\ x = i_2(b) & \Rightarrow & i_1(b^2) \end{array} \right.
\end{array}$$

como queríamos. (Cada passo deve ser justificado.)

□

Resolução 9.7: $y!$ instancia (9.32) da forma seguinte:

$f(\_, y, \_)$	$y!$
$p(\_, y, \_)$	$y = 0$
u	1
$d(\_, y, \_)$	y
θ	$\times$
$e(y)$	$y - 1$

Como não há elemento absorvente de $\times$ em $I\!N$, ter-se-á, de imediato:

```
{   nat     r = 1;
    nat0    y' = y;
    while   (y' != 0)
            { r = r × y';
              y' = y' - 1
            };
}
```

comp instancia (9.32) da forma seguinte:

$f(\_, y, \_)$	$comp(y)$
$p(\_, y, \_)$	$y = < >$
u	0
$d(\_, y, \_)$	$\begin{cases} belongs(head(y), tail(y)) & \Rightarrow 0 \\ \neg(belongs(head(y), tail(y))) & \Rightarrow 1 \end{cases}$
θ	$+$
$e(y)$	$tail(y)$

Como não há elemento absorvente de $+$ em $I\!N_0$, ter-se-á, após algumas simplificações algorítmicas óbvias [9]:

```
{  nat0   r = 0;
   list   y' = y;
   while  (y' != < >)
          { A h = head(y');
            y' = tail(y');
            if ¬(belongs(h,y')) { r = r + 1 };
          };
}
```

A integração com (9.31) pode também fazer-se, conduzindo a qualquer coisa como:

```
{  nat0   r = 0;
   list   y' = y;
   while  (y' != < >)
          { A h = head(y');
            y' = tail(y');
            list p = y';
            bool found = 0;
            while ((p != < >) && (found==0))
                   { found = (h == head(p));
                     p = tail(p);
                   };
            if (found == 0) { r = r + 1 };
          };
}
```

□

Resolução 9.8:

[9] **NB:** Em rigor, tais simplificações deveriam ser justificadas em face da semântica da linguagem algorítmica em causa.

1. Desenvolvendo o facto a provar, teremos:

$$
\begin{array}{rcl}
g(y) & = & f(y)\,\theta\, v \\
 & = & (\left\{\begin{array}{rcl} p(y) & \Rightarrow & u \\ \neg(p(y)) & \Rightarrow & d(y)\,\theta\, f(e(y)) \end{array}\right.)\,\theta\, v \\
 & = & \left\{\begin{array}{rcl} p(y) & \Rightarrow & u\,\theta\, v \\ \neg(p(y)) & \Rightarrow & d(y)\,\theta\,(f(e(y)))\,\theta\, v \end{array}\right. \\
 & = & \left\{\begin{array}{rcl} p(y) & \Rightarrow & v \\ \neg(p(y)) & \Rightarrow & d(y)\,\theta\,\underbrace{(f(e(y))\,\theta\, v)}_{g(e(y))} \end{array}\right. \\
 & = & \left\{\begin{array}{rcl} p(y) & \Rightarrow & v \\ \neg(p(y)) & \Rightarrow & d(y)\,\theta\, g(e(y)) \end{array}\right.
\end{array}
$$

(Cada passo deve ser justificado.)

2. Partindo da alínea anterior, sabendo que θ é comutativa e aproveitando a sugestão dada, teremos:

$$
\begin{array}{rcl}
g(y) & = & f(y)\,\theta\, v \\
 & = & v\,\theta\, f(y) \\
 & = & floop(y, v)
\end{array}
$$

Daqui se deduz que a única coisa que se altera á a inicialização do ciclo (v em lugar de u). Logo o código dado está correcto.

3. No caso de v ser o elemento absorvente de θ teremos

$$
\begin{array}{rcl}
g(y) & = & f(y)\,\theta\, v \\
 & = & v
\end{array}
$$

Logo o ciclo é "inútil", pois tem a semântica da sua inicialização. Ou seja, o código dado pode ser reduzido a

```
{   M r = v;
}
```

□

Resolução 9.9: Vamos deduzi-lo a partir da equação de refinamento

$$elems(?(l)) = filter(elems(l))$$

que é, afinal, mais simples:

$$elems\circ ? = 2^f \circ elems$$

O cálculo é imediato:

$$\begin{array}{rcl} elems \circ ? & = & 2^f \circ elems \\ & \underset{\text{(A.80)}}{=} & elems \circ f^\star \end{array}$$

Removendo $elems$ de ambos os lados da igualdade obtida ter-se-á:

$$? \stackrel{\text{def}}{=} f^\star$$

isto é,

$$?(l) \stackrel{\text{def}}{=} <f(x) \mid x \leftarrow l>$$

□

Resolução 9.11: Equação de refinamento:

$$\begin{array}{rcl} (\lambda n.\overline{n}) \circ id(n) & = & elems(x(n)) \\ & \updownarrow & \\ \overline{n} & = & elems(x(n)) \end{array}$$

Cálculo:

$$\begin{array}{rcl} \overline{n} & = & elems(x(n)) \\ & \updownarrow & \\ \left\{ \begin{array}{lcl} n = 0 & \Rightarrow & \{\} \\ n > 0 & \Rightarrow & \{n\} \cup \overline{n-1} \end{array} \right. & = & elems(x(n)) \\ & \updownarrow & \\ \left\{ \begin{array}{lcl} n = 0 & \Rightarrow & elems(< >) \\ n > 0 & \Rightarrow & elems(<n>) \cup elems(x(n-1)) \end{array} \right. & = & elems(x(n)) \\ & \updownarrow & \\ \left\{ \begin{array}{lcl} n = 0 & \Rightarrow & elems(< >) \\ n > 0 & \Rightarrow & elems(x(n-1)) \cup elems(<n>) \end{array} \right. & = & elems(x(n)) \\ & \updownarrow & \\ \left\{ \begin{array}{lcl} n = 0 & \Rightarrow & elems(< >) \\ n > 0 & \Rightarrow & elems(x(n-1) \frown <n>) \end{array} \right. & = & elems(x(n)) \\ & \updownarrow & \\ elems(\left\{ \begin{array}{lcl} n = 0 & \Rightarrow & < > \\ n > 0 & \Rightarrow & x(n-1) \frown <n> \end{array} \right.) & = & elems(x(n)) \end{array}$$

de onde se extrai:

$$x(n) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lll} n = 0 & \Rightarrow & <> \\ n > 0 & \Rightarrow & x(n-1) \frown <n> \end{array} \right.$$

(Cada passo deve ser justificado com as respectivas leis de SETS.)

Naturalmente, x não é única — por exemplo, omitindo o passo acima em que se trocou, por comutatividade, a ordem dos argumentos de uma reunião de conjuntos, obter-se-ia

$$x(n) \stackrel{\mathrm{def}}{=} \left\{ \begin{array}{lll} n = 0 & \Rightarrow & <> \\ n > 0 & \Rightarrow & <n> \frown x(n-1) \end{array} \right.$$

(sequência descendente, neste caso). □

Resolução 9.12: Vai ser útil o facto seguinte, válido para qualquer função finita σ e conjunto S:

$$\sigma = (\sigma \setminus S) \cup (\sigma \mid S) \tag{D.18}$$

Ora repare-se que (9.13) re-escreve, sucessivamente, em

$$\begin{array}{rcl} (9.13) & = & f(\langle \rho, \gamma \rangle) \dagger (f(\langle \rho \cup \{\langle k, t \rangle\}, \gamma \rangle) \mid \{k\}) \\ & = & (f(\langle \rho, \gamma \rangle)) \setminus \{k\} \cup (f(\langle \rho \cup \{\langle k, t \rangle\}, \gamma \rangle) \mid \{k\}) \\ & = & (f(\langle \rho \cup \{\langle k, t \rangle\}, \gamma \rangle)) \setminus \{k\} \cup (f(\langle \rho \cup \{\langle k, t \rangle\}, \gamma \rangle) \mid \{k\}) \\ & \underset{(D.18)}{=} & (9.14) \end{array}$$

□

Resolução 9.15: Antes de mais, precisamos do seguinte resultado, para qualquer X:

$$f(\langle \rho, \gamma \rangle) \setminus X = f(\langle \underbrace{\{t \in \rho \mid \ \pi_1(t) \notin X\}}_{A}, \underbrace{\{t \in \gamma \mid \ \pi_1(t) \notin X\}}_{B} \rangle) \tag{D.19}$$

Demonstração: Repare-se que $collect(A) = collect(\rho) \setminus X$ e que $mkf(B) = mkf(\gamma) \setminus X$. Logo:

$$\begin{array}{rcl} f(\langle A, B \rangle) & = & f(\langle collect(\rho) \setminus X, mkf(\gamma) \setminus X \rangle) \\ & = & f(\langle \rho, \gamma \rangle) \setminus X \end{array}$$

como queríamos demonstrar. □

Considere-se agora a equação de refinamento:

$$f(\omega(k, \langle \rho, \gamma \rangle) = f(\langle \rho, \gamma \rangle) \setminus \{k\}$$

Ter-se-á:

$$\begin{array}{rcl} f(\omega(k,\langle\rho,\gamma\rangle)) & = & f(\langle\rho,\gamma\rangle)\setminus\{k\} \\ & \underset{(D.19)}{=} & f(\langle\{t\in\rho \mid\ \pi_1(t)\neq k\},\{t\in\gamma \mid\ \pi_1(t)\neq k\}\rangle) \\ & = & f(\langle del(\langle 1,k\rangle,\rho), del(\langle 1,k\rangle,\gamma)\rangle) \end{array}$$

isto é,

$$\omega(k,\langle\rho,\gamma\rangle) \quad \stackrel{\mathrm{def}}{=} \quad \langle del(\langle 1,k\rangle,\rho), del(\langle 1,k\rangle,\gamma)\rangle$$

Codificando informalmente em SQL, para $\rho \in 2^{N:AccNr\times T:AccHolder}$ e $\gamma \in 2^{N:AccNr\times Q:Amount}$:

```
DELETE FROM ρ WHERE N = k
DELETE FROM γ WHERE N = k
```

□

Resolução 9.16:

1. Novo operador:

$$o(\sigma,q) \stackrel{\mathrm{def}}{=} \bigcup_{n\in dom(\sigma)\wedge\pi_2(\sigma(n))<q} \pi_1(\sigma(n))$$

   (nota: $x\in A\wedge p(x)$ abrevia $x\in\{a\in A \mid\ p(a)\}$)

2. Queremos calcular a simulação $o(f(\langle\rho,\gamma\rangle),q)$, onde f é a função de abstracção calculada em §8.5.1:

$$o(f(\rho,\gamma),q) = \bigcup_{n\in\underbrace{dom(f(\rho,\gamma))}_{A}\wedge\underbrace{\pi_2(f(\rho,\gamma)(n))}_{B}<q} \underbrace{\pi_1(f(\rho,\gamma)(n))}_{C}$$

   Reparemos antes de mais que:

$$\begin{array}{rcl} A & = & dom(collect(\rho)) \\ & = & \pi_1[\rho] \\ & = & \pi_1[\gamma] \quad \text{/*}\textit{por eqd no invariante}\text{ */} \\ & = & \{\pi_1(x) \mid\ x\in\gamma\} \\ B & = & \pi_2(f(\rho,\gamma)(n)) \\ & = & mkf(\gamma)(n) \\ & = & the(\{\pi_2(t) \mid\ t\in\gamma\wedge\pi_1(t)=n\}) \\ C & = & \pi_1(f(\rho,\gamma)(n)) \\ & = & collect(\rho)(n) \quad \text{/*}\textit{(8.56)}\text{ */} \\ & = & \{\pi_2(t) \mid\ t\in\rho\wedge\pi_1(t)=n\} \end{array}$$

Teremos então:

$$
\begin{aligned}
& o(f(\rho,\gamma),q) && \text{(D.20)}\\
= & \bigcup_{n\in\{\pi_1(x)\mid\ x\in\gamma\}\wedge the(\{\pi_2(t)\mid\ t\in\gamma\wedge\pi_1(t)=n\})<q} \{\pi_2(t)\mid\ t\in\rho\wedge\pi_1(t)=n\} && \text{(D.21)}\\
= & \bigcup_{x\in\gamma\wedge the(\{\pi_2(t)\mid\ t\in\gamma\wedge\pi_1(t)=\pi_1(x)\})<q} \{\pi_2(t)\mid\ t\in\rho\wedge\pi_1(t)=\pi_1(x)\} && \text{(D.22)}\\
= & \bigcup_{x\in\gamma\wedge\pi_2(x)<q} \{\pi_2(t)\mid\ t\in\rho\wedge\pi_1(t)=\pi_1(x)\} && \text{(D.23)}\\
= & \bigcup_{x\in\gamma}\{\pi_2(t)\mid\ t\in\rho\wedge\pi_1(t)=\pi_1(x)\wedge\pi_2(x)<q\} && \text{(D.24)}\\
= & \{\pi_2(t)\mid\ t\in\rho\wedge\bigvee_{x\in\gamma}(\pi_1(t)=\pi_1(x)\wedge\pi_2(x)<q)\} && \text{(D.25)}\\
= & \pi_2[\{t\in\rho\mid\ \exists x\in\gamma:(\pi_1(t)=\pi_1(x)\wedge\pi_2(x)<q)\}] && \text{(D.26)}
\end{aligned}
$$

Para além de a fdp no invariante, recorreu-se ao seguintes factos básicos, que não carecem de demonstração:

$$
\begin{aligned}
\{a\in\{f(x)\mid\ x\in A\wedge p(x)\}\mid\ q(a)\} &= \{f(x)\mid\ x\in A\wedge p(x)\wedge q(a)\} && \text{(D.27)}\\
\Theta_{a\in\{f(x)\mid\ x\in A\wedge p(a)\}}g(a) &= \Theta_{x\in A\wedge p(a)}g(f(x)) && \text{(D.28)}\\
\{f(x)\mid\ \bigvee_{i\in I}p_i(x)\} &= \bigcup_{i\in I}\{f(x)\mid\ p_i(x)\} && \text{(D.29)}\\
\bigvee_{x\in X}p(x) &= \exists x\in X:p(x) && \text{(D.30)}
\end{aligned}
$$

(para X finito).

Codificando informalmente em SQL, para $\rho\in 2^{N:AccNr\times T:AccHolder}$ e $\gamma\in 2^{N:AccNr\times Q:Amount}$:

```
SELECT ρ.T FROM ρ,γ
      WHERE ρ.N = γ.N AND γ.Q < q
```

□

Resolução 9.19:

1. Relembrando

$$
\begin{aligned}
cons(a,l) &= \langle a\rangle \frown l\\
l &= \langle\rangle \frown l
\end{aligned}
$$

teremos:

$$
\begin{aligned}
cons(head(l), filter(tail(l),a)) &= \langle head(l)\rangle \frown filter(tail(l),a)\\
filter(tail(l),a) &= \langle\rangle \frown filter(tail(l),a)
\end{aligned}
$$

o que permite pôr $filter(tail(l), a)$ em evidência e reconhecer em $filter$ o esquema monádico:

$$\underbrace{\left\{\begin{array}{lcl} a = head(l) & \Rightarrow & <> \\ \neg(a = head(l)) & \Rightarrow & <head(l)> \end{array}\right.}_{d(l)} \underbrace{\frown}_{\theta} filter(\underbrace{tail(l)}_{e(l)}, a)$$

e deduzir:

$$filter(l, a) \quad = \quad filterit(l, a, <>)$$

para

$$filterit(l, a, r) \stackrel{\text{def}}{=}$$
$$\left\{\begin{array}{lcl} l = <> & \Rightarrow & r \\ \neg(l = <>) & \Rightarrow & filterit(tail(l), a, \left\{\begin{array}{lcl} a = head(l) & \Rightarrow & r \\ \neg(a = head(l)) & \Rightarrow & r \frown <head(l)> \end{array}\right. \end{array}\right.$$

2. Não, porque a consulta de uma lista de interesse tem o efeito lateral de alterar a lista consultada fazendo saltar o elemento de interesse para o topo, caso exista. Logo, $ilFind$ não tem semântica funcional e terá de ser especificada como um evento, num modelo com estado $\sigma \in IList$.

3. Não há elementos repetidos:

$$\phi(l) \stackrel{\text{def}}{=} (elems(l)) = length(l)$$

o que implica que

$$head(l) \notin elems(tail(l))$$

Tem-se ainda que

$$a \notin elems(l) \Rightarrow filter(l, a) = l$$

(carece de prova, mas não é pedida). Assim, para $l \neq <>$ e $a = head(l)$, tem-se que $a = head(l)$ implica $a \notin elems(tail(l))$, que por sua vez implica que $filter(tail(l), a) = tail(l)$. Logo, esse ramo de $filter$ simplifica para:

$$a = head(l) \Rightarrow tail(l)$$

□

Resolução 9.20:

1. O "?" superior é substituído por 2^A. O "?" inferior é substituído por X, para $X \cong 1 + A \times X$, *cf.* exercício 8.37.

2. Repare-se que $elems \circ g_{Ex.8.7}$ se encontra definida em (D.11), que designaremos por $f_{(D.11)}$, e que $mkf \circ elems \circ g_{Ex.8.7}$ é a função h calculada em (D.14), que designaremos por $h_{(D.14)}$. Teremos então a seguinte equação de refinamento:

$$\underbrace{elems(g_{Ex.8.7}(dom'(x))}_{f_{(D.11)}(dom'(x))} = dom(\underbrace{mkf(elems(g_{Ex.8.7}(x)))}_{h_{(D.14)}(x)})$$

Desenvolvimento de $dom(h_{(D.14)}(x))$:

$$\begin{array}{ll}
 & dom(h_{(D.14)}(x)) \\
= & dom(\left\{\begin{array}{lll} x = \langle 1, NIL\rangle & \Rightarrow & \left(\ \right) \\ x = \langle 2, \langle t, x'\rangle\rangle & \Rightarrow & \left(\begin{array}{c}\pi_1(t)\\ \pi_2(t)\end{array}\right) \cup h_{(D.14)}(x') \end{array}\right.) \\
= & \left\{\begin{array}{lll} x = \langle 1, NIL\rangle & \Rightarrow & \underbrace{\emptyset}_{f_{(D.11)}(\langle 1, NIL\rangle)} \\ x = \langle 2, \langle t, x'\rangle\rangle & \Rightarrow & \{\pi_1(t)\} \cup \underbrace{dom(h_{(D.14)}(x'))}_{f_{(D.11)}(dom'(x'))} \end{array}\right. \\
= & \left\{\begin{array}{lll} x = \langle 1, NIL\rangle & \Rightarrow & f_{(D.11)}(\langle 1, NIL\rangle) \\ x = \langle 2, \langle t, x'\rangle\rangle & \Rightarrow & \underbrace{\{\pi_1(t)\} \cup f_{(D.11)}(dom'(x'))}_{f_{(D.11)}(\langle 2, \langle \pi_1(t), dom'(x')\rangle\rangle)} \end{array}\right. \\
= & f_{(D.11)}(\left\{\begin{array}{lll} x = \langle 1, NIL\rangle & \Rightarrow & \langle 1, NIL\rangle \\ x = \langle 2, \langle t, x'\rangle\rangle & \Rightarrow & \langle 2, \langle \pi_1(t), dom'(x')\rangle\rangle \end{array}\right.)
\end{array}$$

Cancelando $f_{(D.11)}$ em ambos os membros do resultado acima obtemos, finalmente:

$$dom'(x) \stackrel{\mathrm{def}}{=} \left\{\begin{array}{lll} x = \langle 1, NIL\rangle & \Rightarrow & \langle 1, NIL\rangle \\ x = \langle 2, \langle t, x'\rangle\rangle & \Rightarrow & \langle 2, \langle \pi_1(t), dom'(x')\rangle\rangle \end{array}\right.$$

Versão não-recursiva:

$$\begin{array}{rcl}
dom'(x) & = & domloop(x, \langle 1, NIL\rangle) \\
domloop(x, r) & \stackrel{\mathrm{def}}{=} & \left\{\begin{array}{lll} x = \langle 1, NIL\rangle & \Rightarrow & r \\ x = \langle 2, \langle t, x'\rangle\rangle & \Rightarrow & domloop(x', \langle 2, \langle \pi_1(t), r\rangle\rangle \end{array}\right.
\end{array}$$

3. Tem-se, por inferência, que a assinatura de ϕ é

$$\phi : A \times L \to B$$

A operação simulada é a própria aplicação (parcial) de funções a argumentos:

$$ap(\sigma, a) \stackrel{\mathrm{def}}{=} \left\{\ a \in dom(\sigma) \quad \Rightarrow \quad \sigma(a)\right.$$

A justificação procede do próprio cálculo de ϕ como refinamento de ap. Comecemos por expandir $h_{(D.14)}$:

$$ap(h_{(D.14)}(x), a) \quad = \quad \begin{cases} x = \langle 1, NIL \rangle & \Rightarrow \quad \left(\quad \right)(a) \\ x = \langle 2, \langle t, x' \rangle \rangle & \Rightarrow \quad \underbrace{ap(\begin{pmatrix} \pi_1(t) \\ \pi_2(t) \end{pmatrix} \cup h_{(D.14)}(x'), a)}_{X} \end{cases}$$

Desenvolvimento de X:

$$\begin{aligned} X \quad & = \quad \begin{cases} a \in \{\pi_1(t)\} & \Rightarrow \quad ap(\begin{pmatrix} \pi_1(t) \\ \pi_2(t) \end{pmatrix}, a) \\ a \in dom(h_{(D.14)}(x')) & \Rightarrow \quad \underbrace{ap(h_{(D.14)}(x'), a)}_{\phi(x', a)} \end{cases} \\ & = \quad \begin{cases} a = \pi_1(t) & \Rightarrow \quad \pi_2(t) \\ \neg(a = \pi_1(t)) & \Rightarrow \quad \phi(x', a) \end{cases} \end{aligned}$$

Então:

$$\phi(x, a) \quad \stackrel{\text{def}}{=} \quad \begin{cases} x = \langle 1, NIL \rangle & \Rightarrow \quad \perp \\ x = \langle 2, \langle t, x' \rangle \rangle & \Rightarrow \quad \begin{cases} a = \pi_1(t) & \Rightarrow \quad \pi_2(t) \\ \neg(a = \pi_1(t)) & \Rightarrow \quad \phi(x', a) \end{cases} \end{cases}$$

Teremos finalmente que introduzir uma pre-condição a ϕ para eliminar o caso em que ϕ daria $\perp$ como resultado. Obtém-se então a correspondente função parcial:

$$\phi(x, a) \quad \stackrel{\text{def}}{=} \quad \begin{cases} x \neq \langle 1, NIL \rangle \quad \Rightarrow & let \quad x = \langle 2, \langle t, x' \rangle \rangle \\ & in \quad \begin{cases} a = \pi_1(t) & \Rightarrow \quad \pi_2(t) \\ \neg(a = \pi_1(t)) & \Rightarrow \quad \phi(x', a) \end{cases} \end{cases}$$

□

Resolução 9.21: Repare-se antes de mais [10] que

$$\begin{cases} i = 1 & \Rightarrow \quad <head(l)> \frown subl(t, i, n-1) \\ i > 1 & \Rightarrow \quad subl(t, i-1, n) \end{cases}$$

é o mesmo que

$$\begin{cases} i = 1 & \Rightarrow \quad <head(l)> \frown subl(t, i, n-1) \\ i > 1 & \Rightarrow \quad <> \frown subl(t, i-1, n) \end{cases}$$

e, logo, que

$$\begin{cases} i = 1 & \Rightarrow \quad <head(l)> \\ i > 1 & \Rightarrow \quad <> \end{cases} \frown \begin{cases} i = 1 & \Rightarrow \quad subl(t, i, n-1) \\ i > 1 & \Rightarrow \quad subl(t, i-1, n) \end{cases}$$

[10] Justifique cada passo do raciocínio que se segue.

ou ainda, estendendo o raciocínio:

$$\left\{\begin{array}{lcl} i=1 & \Rightarrow & <head(l)> \\ i>1 & \Rightarrow & <> \end{array}\right. \frown subl(t, \left\{\begin{array}{lcl} i=1 & \Rightarrow & i \\ i>1 & \Rightarrow & i-1 \end{array}\right., \left\{\begin{array}{lcl} i=1 & \Rightarrow & n-1 \\ i>1 & \Rightarrow & n \end{array}\right.)$$

Logo, estamos perante (9.32), para as substituições

$f(\_, y, \_)$	$subl(y, i, n)$
$p(\_, y, \_)$	$n = 0 \vee y = <>$
u	$< >$
$d(\_, y, \_)$	$\left\{\begin{array}{lcl} i=1 & \Rightarrow & <head(y)> \\ i>1 & \Rightarrow & <> \end{array}\right.$
θ	$\frown$
$\_, e(y), \_$	$tail(y), \left\{\begin{array}{lcl} i=1 & \Rightarrow & i \\ i>1 & \Rightarrow & i-1 \end{array}\right., \left\{\begin{array}{lcl} i=1 & \Rightarrow & n-1 \\ i>1 & \Rightarrow & n \end{array}\right.$

o que conduz, em notação “pseudo-C” bastante livre, a

```
{   A* p =y;
    int j =i;
    int m =n;
    A* r =<>;
    while ((m > 0) && (p != <>))
       {   A h = head(p);
           p = tail(p) ;
           if (j == 1)
              {   m--;
                  r = r ⌢ <h>;
              }
           else j--;
       }
}
```

isto é, ao ciclo-`while` pretendido. □

Resolução 9.22:

1. Antes de mais, repare-se que $explode(\#e, \sigma)) = f_{\oplus}(apply(\sigma, \#e), \sigma)$, onde

$$f_{\oplus}(f, \sigma) \stackrel{\text{def}}{=} \left\{\begin{array}{lcl} f = \left(\;\right) & \Rightarrow & \left(\;\right) \\ f \neq \left(\;\right) & \Rightarrow & \begin{array}{ll} let & x \in dom(f) \\ & y = \left\{\begin{array}{lcl} x = \langle 1, k\rangle & \Rightarrow & \left(\begin{array}{c} k \\ f(x) \end{array}\right) \\ x = \langle 2, k\rangle & \Rightarrow & f(x) \otimes explode(k, \sigma) \end{array}\right. \\ in & y \oplus f_{\oplus}(f \setminus \{x\}, \sigma) \end{array} \end{array}\right.$$

é a função que resulta da conversão do *esquema de redução* $\bigoplus_{x\in\ldots}\ldots$ — recordar (2.80) e (2.81), para $\varphi = \oplus$. A substituição de $explode(k,\sigma)$ por $f_\oplus(apply(\sigma,k),\sigma)$ na definição de $f_\oplus$ é legítima e mostra-nos que, afinal, a função aux proposta no exercício coincide com $f_\oplus$. Logo o facto proposto é verdadeiro.

2. Do primeiro facto assumido como verdadeiro infere-se

$$f(x) \otimes aux(apply(\sigma,k),\sigma) = aux(f(x) \otimes apply(\sigma,k),\sigma)$$

o que pode ser aproveitado para re-escrever o corpo da definição de aux em

$$\left\{\begin{array}{lcl} x = \langle 1,k\rangle & \Rightarrow & \left(\begin{array}{c} k \\ f(x) \end{array}\right) \oplus aux(f \setminus \{x\},\sigma) \\ x = \langle 2,k\rangle & \Rightarrow & (aux(f(x) \otimes apply(\sigma,k),\sigma)) \oplus aux(f \setminus \{x\},\sigma) \end{array}\right.$$

Aplicando agora o segundo facto assumido como verdadeiro teremos o mesmo fragmento re-escrito em

$$\left\{\begin{array}{lcl} x = \langle 1,k\rangle & \Rightarrow & \left(\begin{array}{c} k \\ f(x) \end{array}\right) \oplus aux(f \setminus \{x\},\sigma) \\ x = \langle 2,k\rangle & \Rightarrow & aux(f(x) \otimes apply(\sigma,k) \oplus f \setminus \{x\},\sigma) \end{array}\right.$$

o que vem a dar em

$$\left\{\begin{array}{lcl} x = \langle 1,k\rangle & \Rightarrow & \left(\begin{array}{c} k \\ f(x) \end{array}\right) \\ x = \langle 2,k\rangle & \Rightarrow & (\quad) \end{array}\right. \oplus \underbrace{\left\{\begin{array}{lcl} x = \langle 1,k\rangle & \Rightarrow & aux(f \setminus \{x\},\sigma) \\ x = \langle 2,k\rangle & \Rightarrow & aux(f(x) \otimes apply(\sigma,k) \oplus f \setminus \{x\},\sigma) \end{array}\right.}_{X}$$

explorando o facto de $(\quad)$ ser o elemento neutro de $\oplus$, e de acordo com a regra seguinte, que estende (B.1):

$$f(\left\{\begin{array}{lcl} p & \Rightarrow & q \\ \neg(p) & \Rightarrow & r \end{array}\right., \left\{\begin{array}{lcl} p & \Rightarrow & s \\ \neg(p) & \Rightarrow & t \end{array}\right.) = \left\{\begin{array}{lcl} p & \Rightarrow & f(q,s) \\ \neg(p) & \Rightarrow & f(r,t) \end{array}\right.$$

Prosseguindo, X re-escreve em

$$aux(f \setminus \{x\} \oplus \left\{\begin{array}{lcl} x = \langle 1,k\rangle & \Rightarrow & (\quad) \\ x = \langle 2,k\rangle & \Rightarrow & f(x) \otimes apply(\sigma,k) \end{array}\right., \sigma)$$

de novo explorando o elemento neutro de $\oplus$ e por (B.1). Verificamos assim que aux

instancia o esquema linear monádico (9.32) da forma seguinte:

$f(\_, y, \_)$	$aux(y, \sigma)$
$p(\_, y, \_)$	$y = (\ \ )$
u	$(\ \ )$
$d(\_, y, \_)$	$\left\{\begin{array}{lcl} x = \langle 1, k\rangle & \Rightarrow & \left(\begin{array}{c} k \\ y(x)\end{array}\right) \\ x = \langle 2, k\rangle & \Rightarrow & (\ \ )\end{array}\right.$ /*para $x \in dom(y)$ */
θ	$\oplus$
$e(y)$	$\left\{\begin{array}{lcl} x = \langle 1, k\rangle & \Rightarrow & (\ \ ) \\ x = \langle 2, k\rangle & \Rightarrow & y(x) \otimes apply(\sigma, k)\end{array}\right. \oplus y \setminus x$

Como não há elemento absorvente de $\oplus$ em $Parts$, ter-se-á de imediato, em "pseudo-C",

```
{  Parts       r = ();
   Structure   y' = y;
   while       (y' != ())
               { Unit x = choice(dom(y'));
                 Quantity n = y'(x);
```
$$r = r \oplus \left\{\begin{array}{lcl} x = \langle 1, k\rangle & \Rightarrow & \left(\begin{array}{c} k \\ n\end{array}\right) \\ x = \langle 2, k\rangle & \Rightarrow & (\ \ )\end{array}\right. ;$$
$$y' = y' \setminus \{x\} \oplus \left\{\begin{array}{lcl} x = \langle 1, k\rangle & \Rightarrow & (\ \ ) \\ x = \langle 2, k\rangle & \Rightarrow & n \otimes apply(\sigma, k)\end{array}\right. ;$$
```
               };
}
```

Após simplificações algorítmicas baseadas no elemento neutro de $\oplus$ e semântica da construção `if ...then ...else` [11] obter-se-á:

$$
\begin{array}{lll}
\{ & \texttt{Parts} & \texttt{r = ();} \\
 & \texttt{Structure} & \texttt{y' = y;} \\
 & \texttt{while} & \texttt{(y' != ())} \\
 & & \{ \ \texttt{Unit x = choice(dom(y'));} \\
 & & \quad \texttt{Quantity n = y'(x);} \\
 & & \quad \texttt{y' = y'} \ \backslash\{x\} \ \texttt{;} \\
 & & \quad \texttt{r = r} \oplus \left\{ \begin{array}{lll} x = \langle 1, k\rangle & \Rightarrow & \left(\begin{array}{c} k \\ n \end{array} \right) \\ x = \langle 2, k\rangle & \Rightarrow & (\quad) \end{array} \right. \quad \texttt{;} \\
 & & \quad \texttt{y' = y'} \oplus \left\{ \begin{array}{lll} x = \langle 1, k\rangle & \Rightarrow & (\quad) \\ x = \langle 2, k\rangle & \Rightarrow & n \otimes apply(\sigma, k) \end{array} \right. \quad \texttt{;} \\
 & & \}\texttt{;} \\
\} & &
\end{array}
$$

e finalmente o código proposto no exercício. Note-se que as primeiras três instruções do ciclo while se podem converter numa só operação de `get` à função finita (futura tabela).

□

[11] Mais uma vez se chama a atenção para o facto de, em rigor, tais simplificações carecerem de justificação em face da semântica da linguagem algorítmica em causa.