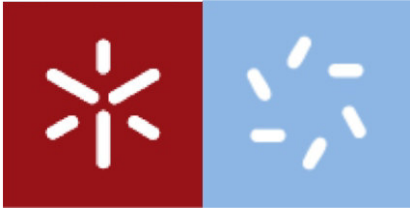
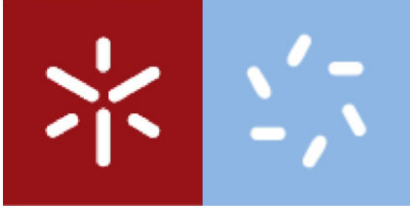


Universidade do Minho
Escola de Ciências

Xavier Gomes Pinho

A Máquina Abstracta Categorial

dezembro de 2014



Universidade do Minho
Escola de Ciências

Xavier Gomes Pinho

A Máquina Abstracta Categorial

Dissertação de Mestrado
Mestrado em Matemática e computação

Trabalho realizado sob a orientação da
Professor Doutor José Carlos Espirito Santo
e
Professor Doutor José Carlos Bacelar Almeida

dezembro de 2014

Agradecimentos

Este trabalho foi realizado sob a orientação do Professor Doutor José Carlos Espírito Santo e do Professor Doutor José Carlos Bacelar Almeida. A eles agradeço todo o apoio, disponibilidade, direcção e paciência com os meus progressos por vezes lentos ao longo do último ano. Mais particularmente, gostaria de agradecer ao Professor Doutor José Carlos Espírito Santo por me cativar para esta área das ciências da computação, e ao Professor Doutor José Carlos Bacelar Almeida pela sugestão que viria a desbloquear o caminho para a formalização do operador de ponto-fixo.

Xavier Pinho

Braga, 29 de Outubro de 2014

Resumo

Desde o trabalho seminal de P. J. Landin nos anos 1960 que se têm utilizado máquinas abstractas para a implementação de linguagens de programação. Nos anos 1980 foi estabelecida uma correspondência entre o Cálculo λ e uma variante da Lógica Combinatória inspirada em categorias cartesianas fechadas. Desta relação surgiu uma máquina abstracta para a implementação do Cálculo λ — a Máquina Abstracta Categorial, ou CAM. Historicamente, esta nova máquina abstracta esteve na origem da linguagem de programação funcional Caml.

Nesta dissertação explicamos como surgiu a CAM, exemplificamos o seu uso, demonstramos a sua correcção, e fazemos um enquadramento histórico à sua volta. Complementamos a literatura de referência ao formalizarmos e demonstrarmos correctas três extensões: constantes, condicionais e operador de ponto-fixo.

Abstract

Since the seminal work of P.J. Landin in the 1960s, abstract machines have been used for the implementation of programming languages. In the 1980s a correspondence has been established between λ-calculus and a variant of Combinatory Logic inspired by cartesian closed categories. From this relationship, an abstract machine has arised for the implementation of λ-calculus — the Categorical Abstract Machine, or CAM. Historically, this new abstract machine led to the functional programming language Caml.

In this dissertation it is explained how the CAM emerged, it is illustrated its usage, it is demonstrated its correctness, and it is made an historical insersion. The reference literature is complemented by formalising and demonstrating the correctness of three extensions: constants, conditionals and fixed-point operator.

Conteúdo

Introdução

Neste trabalho debruçamo-nos sobre uma máquina abstracta para a implementação de linguagens de programação funcionais, partindo da teoria matemática na sua génese e terminando na demonstração da sua correcção, pelo meio apresentando um exemplo completo da sua utilização na compilação de uma linguagem de programação funcional.

A programação funcional é um paradigma de programação que gira em torno de expressões cuja simplificação conduz ao valor final (ou *output*), bem ao estilo da simplificação de expressões aritméticas em matemática. Central a este paradigma é o conceito de função enquanto regra de correspondência. De facto, numa linguagem de programação funcional encontramos essencialmente declarações de funções e aplicações destas a argumentos.

Coincidentemente, existe em matemática uma teoria lidando precisamente com funções enquanto regras de correspondência: o Cálculo λ (Church, 1941). Um dos pioneiros a observar esta relação entre programas funcionais e o Cálculo λ foi P. J. Landin (Landin, 1964). A partir daí, tornou-se claro que, para implementar uma linguagem de programação funcional, basta implementar o Cálculo λ.

Porém, a implementação directa do Cálculo λ levanta problemas de eficiência. No Cálculo λ, a explicação do efeito da aplicação de uma função a um argumento — a regra β — é dada através de uma operação de substituição *sem captura de variáveis*. Para se evitar o problema da captura de variáveis, recorre-se a uma renomeação prévia das variáveis da expressão à qual vamos aplicar a substituição. Ora, este artifício de renomeação é potencialmente caro a nível computacional (Peyton Jones, 1987).

Várias abordagens foram sendo apresentadas para se contornar esta dificuldade:

- Landin (Landin, 1964) idealizou uma *máquina abstracta* — um modelo de alto nível de uma máquina concreta —, a SECD, com um compartimento à parte (chamado *ambiente*) no qual se encontravam os valores das variáveis, a ser actualizado de cada vez que uma operação de substituição devesse ocorrer. Durante o processo de execução da SECD, sempre que fosse necessário utilizar o valor de uma variável, consultava-se o ambiente.

- De Bruijn (de Bruijn, 1972) sugeriu a utilização de índices no lugar de variáveis, permitindo com isso evitar de antemão as situações nas quais era preciso renomear previamente as variáveis para se evitar o problema da captura de variáveis.

- Turner (Turner, 1979) sugeriu um sistema de reescrita no qual não constam, de todo, variáveis, baseado numa equivalência entre o Cálculo λ e a Lógica Combinatória.

A Lógica Combinatória (Curry and Feyes, 1958) é um sistema formal cuja sintaxe gira em torno de um número reduzido de constantes e da aplicação destas umas às outras, e que tem subjacente a filosofia de que as variáveis são secundárias. Muito curiosamente, é possível traduzir coerentemente os objectos do Cálculo λ em objectos da Lógica Combinatória e vice-versa de forma a observar-se uma equivalência entre as duas teorias, vide (Hindley and Seldin, 2008, Chapter 9), (Hankin, 1994, Section 4.2) ou (Sørensen and Urzyczyn, 2006, Section 5.4), por exemplo.

Outra equivalência envolvendo o Cálculo λ viria a ser descoberta por J. Lambek (Lambek and Scott, 1986, Chapter 10–11), desta vez envolvendo a Teoria de Categorias. Sabendo desta equivalência, P.-L. Curien introduziu uma variante da Lógica Combinatória, agora designada Lógica Combinatória Categorial, que provou também equivalente ao Cálculo λ (Curien, 1986). A Lógica Combinatória Categorial, tal como a Lógica Combinatória (Clássica), dispensa as variáveis; mas, curiosamente, a tradução de objectos do Cálculo λ

em objectos desta Lógica é feita com base na ideia de índices proposta por de Bruijn.

Perante a Lógica Combinatória Categorial, G. Cousineau deduziu uma outra máquina abstracta para a implementação do Cálculo λ, a *Máquina Abstracta Categorial* ou CAM (do inglês "Categorical Abstract Machine") (Cousineau, 1996). A CAM é, à semelhança da SECD de Landin, uma máquina de ambiente. De facto, estas duas máquinas diferem essencialmente apenas na forma como realizam uma salvaguarda de valores, podendo argumentar-se que CAM é a mais simples (Cousineau et al., 1987). Outra vantagem da CAM sobre a SECD é ser possível extrair algumas optimizações da Lógica Combinatória Categorial directamente. Historicamente, a CAM está na origem da linguagem de programação funcional Caml, que depois evoluiu na actual linguagem OCaml (Cousineau, 1996).

Sumário O conteúdo principal deste trabalho encontra-se nos capítulos segundo, terceiro e quarto. No segundo, apresentamos a Lógica Combinatória Categorial e vemos a sua equivalência com o Cálculo λ. No terceiro, apresentamos a Máquina Abstracta Categorial e apresentamos um exemplo completo da sua utilização para a implementação de uma linguagem de programação funcional minimal. No quarto, demonstramos a correcção da Máquina Abstracta Categorial.

Quanto aos restantes capítulos: no primeiro, recordamos o Cálculo λ, a sua variante com índices de de Bruijn e a Lógica Combinatória (Clássica); no quinto, fazemos um enquadramento histórico em redor da Máquina Abstracta Categorial; no sexto, encerramos.

Capítulo 1

Preliminares

Neste capítulo recordamos alguns resultados e definições básicas a respeito do Cálculo λ, do Cálculo λ com índices de de Bruijn e da Lógica Combinatória.

1.1 Cálculo λ

O Cálculo λ é um sistema formal que gira em torno do conceito de função enquanto regra de correspondência. Foi inventado nos anos 1930 por Alonzo Church (Church, 1941) e desde então ocupa um lugar central em Ciências da Computação, contribuindo para uma formalização da noção de computabilidade (Barendregt, 1997), servindo de modelo para linguagens de programação funcionais (Landin, 1964), entre outros.

Sintaxe

Definição 1.1. Seja $\mathcal{V}$ um conjunto numerável (de *variáveis*) cujos elementos denotamos pelas meta-variáveis x, y, z, eventualmente etiquetadas. Define-se o conjunto dos *termos-*λ, Λ, indutivamente por:

1. Todas as variáveis são termos-λ, isto é, $\mathcal{V} \subseteq \Lambda$;
2. Se M e N estão em Λ, então a *aplicação* (MN) está em Λ;
3. Se M está em Λ e x é uma variável, então a *abstracção* $(\lambda x M)$ está em Λ.

Denotamos pelas meta-variáveis M, N, P, Q, eventualmente etiquetadas, termos-λ.

Terminologia 1.2.

1. A respeito de uma aplicação (MN), dizemos que M ocupa a *posição de função* e que N ocupa a *posição de argumento*;

2. Numa abstracção (λxM), dizemos que M é o *corpo* (ou o *âmbito*) da abstracção. Também dizemos que x é o *parâmetro formal* da abstracção.

Notação 1.3. Adoptam-se as seguintes convenções de escrita:

1. Omitem-se os parêntesis mais externos de termos-λ;

2. Escrevemos $M_1M_2M_3$ no lugar de $(M_1M_2)M_3$. Mais geralmente, a aplicação associa à esquerda;

3. Escrevemos $\lambda x_1x_2.M$ no lugar de $\lambda x_1(\lambda x_2M)$. Mais geralmente, admitimos escrever $\lambda x_1x_2 \dots x_n.M$ no lugar de $\lambda x_1(\lambda x_2(\dots(\lambda x_nM))\dots)$. Por hábito de escrita, também admitimos escrever $\lambda x.M$ no lugar de λxM.

O símbolo λ numa abstracção λxM funciona como um quantificador lógico actuando sobre a variável x. Posto isto, uma ocorrência de uma variável x diz-se *livre* se não ocorre no âmbito de uma abstracção $\lambda x.M$, e diz-se *ligada* caso contrário. Para todo o termo-λ M, denota-se por $FV(M)$ (do inglês "free variables") o conjunto das variáveis que ocorrem livres em M, e por $BV(M)$ (do inglês "bound variables") o conjunto das variáveis que ocorrem ligadas em M.

Definição 1.4. Seja $M \in \Lambda$. Definem-se $FV(M)$ e $BV(M)$ recursivamente por:

1. $FV(x) = \{x\}$;
2. $FV(MN) = FV(M) \cup FV(N)$;
3. $FV(\lambda xM) = FV(M)\backslash\{x\}$.

1. $BV(x) = \varnothing$;
2. $BV(MN) = BV(M) \cup BV(N)$;
3. $BV(\lambda xM) = \{x\} \cup BV(M)$.

Se $FV(M) = \varnothing$, dizemos que M é *fechado* (ou um *combinador*).

Define-se em seguida a operação que permite substituir ocorrências de variáveis livres por termos-λ.

Definição 1.5. Sejam $M, N \in \Lambda$ e $x \in \mathcal{V}$. Define-se a *substituição de x por N em M sem captura de variáveis*, que denotamos como $M[x \coloneqq N]$, recursivamente por:

1. $x[x \coloneqq N] = N$;
2. $y[x \coloneqq N] = y$, se $y \neq x$;
3. $(M_1 M_2)[x \coloneqq N] = (M_1[x \coloneqq N] M_2[x \coloneqq N])$;
4. $(\lambda x M)[x \coloneqq N] = \lambda x M$;
5. $(\lambda y M)[x \coloneqq N] = \lambda y M[x \coloneqq N]$, se $x \notin FV(M)$ ou $y \notin FV(N)$;
6. $(\lambda y M)[x \coloneqq N] = \lambda z (M[y \coloneqq z])[x \coloneqq N]$, se $x \in FV(M)$, $y \in FV(N)$ e z é uma variável tal que $z \notin FV(MN)$.

A cláusula 6 na definição 1.5 é o que previne a substituição de incorrer no fenómeno lógico conhecido como *captura de variáveis*. Observe-se o que aconteceria se $(\lambda y M)[x \coloneqq N]$ fosse definido como $\lambda y M[x \coloneqq N]$: as ocorrências livres de y em N estariam agora no âmbito de uma abstracção cujo parâmetro formal é y, donde y passaria a ocorrer ligado — i.e. y seria *capturado* pelo quantificador λy. O artifício da cláusula 6 consiste numa renomeação prévia da variável y por uma outra que não ocorra livre em N.

Reduções e conversões

Definição 1.6. Seja $\mathcal{R}$ uma relação binária sobre Λ. Diz-se que $\mathcal{R}$ é *compatível* se (para todo M, N, P, Q, x):

1a. $(MN)\mathcal{R}(PN)$, sempre que $M\mathcal{R}P$;

1b. $(MN)\mathcal{R}(MQ)$, sempre que $N\mathcal{R}Q$;

2. $(\lambda x M)\mathcal{R}(\lambda x N)$, sempre que $M\mathcal{R}N$.

Definição 1.7. Uma *noção de redução* sobre Λ é simplesmente uma relação binária sobre Λ.

Terminologia 1.8. Seja $\mathcal{R}$ uma noção de redução sobre Λ e $M, N \in \Lambda$ tais que $M\mathcal{R}N$. Dizemos que M é um *redex-$\mathcal{R}$* (do inglês "reducible expression") e que N é o seu *contractum*.

Definição 1.9. Seja $\mathcal{R}$ uma noção de redução sobre Λ. Denotamos por:

1. $\to_{\mathcal{R}}$ o fecho compatível de $\mathcal{R}$, designado como *um passo de redução* $\mathcal{R}$;

2. $\to_{\mathcal{R}}^{+}$ o fecho compatível e transitivo de $\mathcal{R}$, designado como *um ou mais passos de redução* $\mathcal{R}$;

3. $\to_{\mathcal{R}}^{*}$ o fecho compatível, reflexivo e transitivo de $\mathcal{R}$, designado como *redução* $\mathcal{R}$;

4. $=_{\mathcal{R}}$ o fecho compatível e de equivalência de $\mathcal{R}$, designado como *conversão* $\mathcal{R}$.

Observação 1.10. Desta definição é imediata a seguinte hierarquia:

$$\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}}^{+} \subseteq \to_{\mathcal{R}}^{*} \subseteq =_{\mathcal{R}} .$$

Lema 1.11. *Seja $\mathcal{R}$ uma noção de redução sobre Λ. Quer $\to_{\mathcal{R}}^{*}$ quer $=_{\mathcal{R}}$ são relações compatíveis.*

Demonstração. Por indução sobre $M \to_{\mathcal{R}}^{*} N$ e $M =_{\mathcal{R}} N$ respectivamente. A demonstração é uma aplicação rotineira da H.I., pelo que a omitimos. □

Definição 1.12. Seja $M \in \Lambda$ e $\mathcal{R}$ uma noção de redução sobre Λ. Diz-se que M é uma *forma $\mathcal{R}$ normal* se não existe $N \in \Lambda$ tal que $M \to_{\mathcal{R}} N$.

Terminamos esta subsecção com a apresentação das noções de redução α, β e η. A conversão α relaciona termos-λ que diferem apenas na escolha de variáveis ligadas. A conversão β captura o efeito intuitivo da aplicação

de uma função a um argumento, tratando-se da noção de redução central de todo o Cálculo λ. A conversão η captura a noção de igualdade extensional, como em "$f = g$, se $f(x) = g(x)$ para todo x."[1]

Definição 1.13. Seja α a seguinte noção de redução sobre Λ:

(α) $\{(\lambda xM, \lambda yM[x := y]) \mid y \notin FV(M)\}$.

Exemplo 1.14. Tem-se

1. $\lambda xx =_\alpha \lambda yy$;
2. $\lambda x.xz =_\alpha \lambda y.yz$;
3. $\lambda xy.xy =_\alpha \lambda yx.yx$;
4. $\lambda x.xy \neq_\alpha \lambda x.xz$.

Em apresentações do Cálculo λ é consuetudinário identificar termos-λ módulo a conversão α, como se o conjunto Λ se tratasse na verdade do conjunto quociente $\Lambda/=_\alpha$ [2]. Esta perspectiva permite-nos, em particular, evitar a cláusula 6 da definição 1.5.

Observação 1.15. A propósito, iremos também utilizar neste trabalho uma variante da operação de substituição conhecida como *substituição paralela* (ou *simultânea*), que passamos a definir. Sejam $x_0, \ldots, x_n \in \mathcal{V}$ (notação: $\vec{x}$) e $N_0, \ldots, N_n \in \Lambda$ (notação: $\vec{N}$). Define-se a substituição paralela $\cdot[\vec{x} \coloneqq \vec{N}] : \Lambda \to \Lambda$ recursivamente como:

1. $x[\vec{x} \coloneqq \vec{N}] = N_i$, se $x = x_i$, para algum $0 \leqslant i \leqslant n$;
2. $y[\vec{x} \coloneqq \vec{N}] = y$, se $y \notin \vec{x}$;
3. $(M_1M_2)[\vec{x} \coloneqq \vec{N}] = M_1[\vec{x} \coloneqq \vec{N}]M_2[\vec{x} \coloneqq \vec{N}]$;
4. $(\lambda yP)[\vec{x} \coloneqq \vec{N}] = \lambda yP[\vec{x} \coloneqq \vec{N}]$.

Na alínea 4 pressupomos que $y \notin FV(N_i)$ para todo $0 \leqslant i \leqslant n$, de acordo com a convenção de identificar termos-λ módulo conversão α descrita no parágrafo anterior.

[1] Ver (Hankin, 1994, Section 2.4) ou (Hindley and Seldin, 2008, Chapter 7).

[2] Em (Sørensen and Urzyczyn, 2006, Chapter 1) esta construção dos termos-λ é realizada com detalhe.

Definição 1.16. Seja β a seguinte noção de redução sobre Λ:

(β) $\{((\lambda x M)N, M[x \coloneqq N])\}$.

Exemplo 1.17. Tem-se

1. $(\lambda xx)z \to_\beta x[x \coloneqq z] = z$;
2. $(\lambda x.xx)(\lambda zz) \to_\beta^* \lambda yy$;

O conjunto das formas β normais admite a seguinte caracterização.

Proposição 1.18. *O conjunto das formas β normais, $\mathcal{N} \subseteq \Lambda$, é dado indutivamente por:*

1. *$x \in \mathcal{N}$, para todo $x \in \mathcal{V}$;*
2. *$xM_0 \dots M_n \in \mathcal{N}$, para todo $x \in \mathcal{V}$ e $M_0, \dots, M_n \in \mathcal{N}$;*
3. *$\lambda x M \in \mathcal{N}$, para todo $x \in \mathcal{V}$ e $M \in \mathcal{N}$.*

Demonstração. Ver (Hindley and Seldin, 2008, Lemma 1.33). □

Definição 1.19. Seja η a seguinte noção de redução sobre Λ:

(η) $(\lambda x.Mx, M) \mid x \notin FV(M)\}$.

Exemplo 1.20. Tem-se

1. $\lambda x.yx \to_\eta y$;
2. $\lambda x.(\lambda xx)x \to_\eta \lambda xx$.

Também permitimos formar a redução/conversão $\beta\eta$, enquanto formada a partir da noção de redução $\beta \cup \eta$.

Proposição 1.21 (Ponto-fixo)**.** *Para todo $M \in \Lambda$, existe $M' \in \Lambda$ tal que $M' =_\beta MM'$.*

Demonstração. Seja $M \in \Lambda$. Tome-se $M' = YM$, onde $Y = \lambda f.HH$, com $H = \lambda x.f(xx)$. Tem-se

$$
\begin{aligned}
& YM \\
=_\beta \quad & (\lambda x.M(xx))(\lambda x.M(xx)) \\
=_\beta \quad & M((\lambda x.M(xx))(\lambda x.M(xx))) \\
=_\beta \quad & M(YM)
\end{aligned}
$$

□

O combinador Y da última demonstração[3] é chamado um combinador de *ponto-fixo* por satisfazer a propriedade

$$YM =_\beta M(YM), \text{ para todo } M \in \Lambda. \tag{1.1}$$

Existe uma infinidade de outros combinadores nestas condições (Hankin, 1994, p. 80).

Estratégias de redução

Num certo termo-λ podem ocorrer simultaneamente vários redexes de uma certa noção de redução $\mathcal{R}$. Nessas circunstâncias é natural considerarmos qual dos redexes possíveis optamos por reduzir primeiro. A esta escolha pré-determinada chamamos *estratégia de redução*. Formalmente, uma estratégia de redução é simplesmente uma restrição sobre uma relação de redução, tipicamente desrespeitando as condições de compatibilidade. No que se segue apenas nos interessamos por estratégias de redução para a relação β.

Associada a cada estratégia de redução está uma noção de forma normal (também dita de *valor*), potencialmente diferente da noção de forma β normal. Apresentamos de seguida três estratégias de redução relevantes e as respectivas formas normais:

1. A estratégia *call-by-name*, cujas formas normais dizem-se *weak head*;
2. A estratégia *normal*, cujas formas normais coincidem com as formas β normais;

[3] Também conhecido como o combinador paradoxal de Curry (Hankin, 1994, p. 80).

3. A estratégia *call-by-value*, cujas formas normais dizem-se *weak*.

Antes disso, precisamos de introduzir a seguinte definição.

Definição 1.22. Um redex M_1 diz-se à *esquerda* de um redex M_2 conforme o símbolo λ no redex M_1 ocorre à esquerda do símbolo λ do redex M_2. Um redex diz-se mais *exterior* se não está contido em nenhum outro redex. Reciprocamente, um redex diz-se mais *interior* se não contém outros redexes.

Notação 1.23. Nas definições seguintes empregamos a notação

$$\frac{P_1 \quad P_2 \quad \ldots \quad P_n}{Q}$$

como abreviatura para a expressão "Q, se P_1, P_2, ..., P_n."

Definição 1.24. (Sestoft, 2002, Section 7.1)] A estratégia de redução *call-by-name* escolhe primeiro o redex mais exterior e mais à esquerda fora do âmbito de abstracções. Mais formalmente, define-se a relação $\twoheadrightarrow_{\text{bn}}$ indutivamente por:

1. $x \twoheadrightarrow_{\text{bn}} x$;
2. $(\lambda x M) \twoheadrightarrow_{\text{bn}} (\lambda x M)$;
3. $\dfrac{M_1 \twoheadrightarrow_{\text{bn}} (\lambda x M) \qquad M[x \coloneqq M_2] \twoheadrightarrow_{\text{bn}} M'}{(M_1 M_2) \twoheadrightarrow_{\text{bn}} M'}$;
4. $\dfrac{M_1 \twoheadrightarrow_{\text{bn}} M_1' \neq (\lambda x M)}{(M_1 M_2) \twoheadrightarrow_{\text{bn}} (M_1' M_2)}$.

As formas normais da relação $\twoheadrightarrow_{\text{bn}}$ dizem-se *weak head* e são da forma:

$$W ::= \lambda x M \mid x M_0 \ldots M_n.$$

Definição 1.25 ((Sestoft, 2002, Section 7.2))**.** A estratégia de redução *normal* (ou *leftmost outermost*) (notação: $\twoheadrightarrow_{\text{no}}$) escolhe primeiro o redex mais à esquerda e mais exterior. Define-se indutivamente através de:

1. $x \twoheadrightarrow_{\text{no}} x$;

2. $\dfrac{M \twoheadrightarrow_{\text{no}} M'}{(\lambda x M) \twoheadrightarrow_{\text{no}} (\lambda x M')}$;

3. $\dfrac{M_1 \twoheadrightarrow_{\text{bn}} (\lambda x M) \quad M[x \coloneqq M_2] \twoheadrightarrow_{\text{no}} M'}{(M_1 M_2) \twoheadrightarrow_{\text{no}} M'}$;

4. $\dfrac{M_1 \twoheadrightarrow_{\text{bn}} M_1' \neq (\lambda x M) \quad M_1' \twoheadrightarrow_{\text{no}} M_1'' \quad M_2 \twoheadrightarrow_{\text{no}} M_2'}{(M_1 M_2) \twoheadrightarrow_{\text{no}} (M_1'' M_2')}$.

As formas normais desta redução são as formas β normais.

Observação 1.26. O recurso à relação $\twoheadrightarrow_{\text{bn}}$ nesta definição serve para garantir que o corpo das abstracções não é reduzido, porque se fosse, deixaria de reduzir o redex mais exterior primeiro.

Definição 1.27 ((Sestoft, 2002, Section 7.3))**.** A estratégia de redução *call-by-value* escolhe primeiro o redex mais à esquerda e fora do âmbito de uma abstracção. Formalmente define-se $\twoheadrightarrow_{\text{bv}}$ indutivamente por:

1. $x \twoheadrightarrow_{\text{bv}} x$;

2. $(\lambda x M) \twoheadrightarrow_{\text{bv}} (\lambda x M)$;

3. $\dfrac{M_1 \twoheadrightarrow_{\text{bv}} (\lambda x M) \quad M_2 \twoheadrightarrow_{\text{bv}} M_2' \quad M[x \coloneqq M_2'] \twoheadrightarrow_{\text{bv}} M'}{(M_1 M_2) \twoheadrightarrow_{\text{bv}} M'}$;

4. $\dfrac{M_1 \twoheadrightarrow_{\text{bv}} M_1' \neq (\lambda x M) \quad M_2 \twoheadrightarrow_{\text{bv}} M_2'}{(M_1 M_2) \twoheadrightarrow_{\text{bv}} (M_1' M_2')}$.

As formas normais da relação $\twoheadrightarrow_{\text{bv}}$ dizem-se *weak* e são da forma:

$$W ::= \lambda x M \mid x W_0 \ldots W_n.$$

Observação 1.28. Note-se que as estratégias *call-by-name* e *call-by-value* não reduzem os corpos das abstracções.

Essencialmente, as estratégias *call-by-name* e *call-by-value* diferem no seguinte aspecto: a estratégia *call-by-value* reduz o argumento de uma função na totalidade antes de o substituir no corpo da função, enquanto que a estratégia *call-by-name* utiliza o argumento tal como está na dita substituição. Para além disto, a estratégia *call-by-value* também difere da estratégia *call-by-name* ao avaliar sempre o termo-λ em posição de argumento numa aplicação, cf. alínea 4 das respectivas definições.

1.2 Cálculo λ com índices de de Bruijn

O Cálculo λ com índices de de Bruijn (ou Cálculo λ de de Bruijn) [4], denotado por Cálculo λ^{DB}, é uma variante do Cálculo λ que incorpora desde logo a identificação de termos módulo a conversão α. Isto é conseguido substituindo na sintaxe dos termos-λ as variáveis por índices naturais. O papel destes índices será explicado à frente.

Definição 1.29. Define-se o conjunto dos *termos-λ de de Bruijn*, Λ^{DB}, indutivamente por:

1. O *índice* n está em Λ^{DB}, para todo $n \in \mathbb{N}_0$;
2. A *aplicação* (MN) está em Λ^{DB}, se M e N estão em Λ^{DB};
3. A *abstracção* (λM) está em Λ^{DB}, se M está em Λ^{DB}.

Num termo-λ de de Bruijn ocorrem índices em vez de variáveis e o símbolo λ não vem etiquetado com a variável que quantifica. A filosofia aqui consiste em utilizar os índices para identificar que símbolo λ quantifica essa ocorrência, mantendo assim a ligação entre as ocorrências de variáveis e o respectivo quantificador λ. O índice indica quantos níveis de abstracção necessitamos de "subir" (a partir dessa ocorrência) para encontrarmos o quantificador λ que o introduz. Mais precisamente, para um dado índice i seja k o total de λ's no alcance dos quais este índice se encontra. Essa ocorrência dir-se-á ligada se $i \leqslant k$, ou livre caso contrário.

A tradução de termos-λ em termos-λ de de Bruijn pode ser realizada através da seguinte função.

Definição 1.30. Seja $M \in \Lambda$ e sejam $x_0, \ldots, x_n \in \mathcal{V}$ (notação: $\vec{x}$) tais que $FV(M) \subseteq \vec{x}$. Define-se $M_{DB(\vec{x})}$ (abreviadamente M_{DB}) recursivamente por:

1. $x_{DB(\vec{x})} = i$, onde i é o menor índice em $\vec{x}$ tal que $x = x_i$, com $0 \leqslant i \leqslant n$.
2. $(MN)_{DB(\vec{x})} = M_{DB(\vec{x})} N_{DB(\vec{x})}$;

[4] Em honra do seu inventor, o matemático holandês Nicolaas Govert de Bruijn (de Bruijn, 1972).

3. $(\lambda x M)_{DB(\vec{x})} = \lambda(M_{DB(x,\vec{x})})$.

Exemplo 1.31. Quer $\lambda xy.x$, quer $\lambda yx.y$, reescrevem-se em notação de de Bruijn como $\lambda\lambda 2$.

Os termos-λ de de Bruijn permitem uma implementação mais mecânica da redução β.

Definição 1.32. Seja β a seguinte noção de redução:

(β) $\{((\lambda M)N, M[0 \coloneqq N])\}$.

Falta-nos definir $M[0 \coloneqq N]$. Nesta operação existe um jogo de manipulação de índices. Por um lado, vamos substituir todas as ocorrências livres de 0 por N, como de costume. Por outro, precisamos de *actualizar* todas as ocorrências livres em N para que se mantenham livres, i.e. não sejam capturadas pelo quantificador λ.

Definição 1.33. Sejam $M, N \in \Lambda$ e $m \in \mathbb{N}_0$. Define-se $M[m \coloneqq N]$ recursivamente por:

1. $n[m \coloneqq N] = \begin{cases} n & \text{se } n < m; \\ U_0^n(N) & \text{se } n = m \\ n-1 & \text{se } n > m \end{cases}$
2. $(M_1 M_2)[m \coloneqq N] = (M_1[m \coloneqq N] M_2[m \coloneqq N])$;
3. $(\lambda M)[m \coloneqq N] = \lambda(M[m+1 \coloneqq N])$,

onde (para todo $k, i \in \mathbb{N}_0$) $U_i^k : \Lambda^{DB} \to \Lambda^{DB}$ (dita *função de actualização*) é a função definida recursivamente por:

1. $U_i^k(n) = \begin{cases} n & \text{se } n < i \\ n+k & \text{se } n \geqslant i; \end{cases}$
2. $U_i^k(M_1 M_2) = U_i^k(M_1) U_i^k(M_2)$;
3. $U_i^k(\lambda M) = \lambda(U_{i+1}^k(M))$.

Proposição 1.34 ((Curien, 1993, Exercise 1.2.7.1)). *Sejam $M, N \in \Lambda$ e $x_0, \ldots, x_n \in \mathcal{V}$ (notação: $\vec{x}$) tais que $FV(MN) \subseteq \vec{x}$. Suponhamos que $((\lambda x.M)N)_{DB} = (\lambda P)Q$ para algum $P, Q \in \Lambda^{DB}$. Então*

$$(M[x \coloneqq N])_{DB} = P[0 \coloneqq Q].$$

1.3 Lógica Combinatória Clássica

A Lógica Combinatória é uma teoria equivalente (num sentido a precisar à frente) ao Cálculo λ iniciada nos anos 1920 por Moses Schönfinkel e posteriormente redescoberta e desenvolvida por Haskell Curry (Cardone and Hindley, 2006). Enquanto teoria equivalente ao Cálculo λ, é assinalável o facto de não comportar, de todo, a noção de variável ligada.

Definição 1.35. Seja $\mathcal{V}$ um conjunto numerável (de *variáveis*). O conjunto dos *termos combinatoriais*, CL (do inglês "Combinatory Logic"), é definido indutivamente por:

1. Todas as variáveis estão em CL;
2. As constantes I, K e S estão em CL;
3. A aplicação (FG) está em CL, se F e G estão em CL.

Denotamos pelas meta-variáveis F, G, H, eventualmente etiquetadas, termos combinatoriais.

Notação 1.36. Numa aplicação omitem-se os parêntesis mais externos. Para além disso, convenciona-se que a aplicação associa à esquerda, i.e. $F_1F_2F_3$ denota $(F_1F_2)F_3$.

Denotamos por $V(F)$ o conjunto das variáveis que ocorrem em F. Quando $V(F) = \varnothing$ dizemos que F é um *combinador*.

Definição 1.37. Define-se, para todo $F \in$ CL, $V(F)$ recursivamente por:

1. $V(x) = \{x\}$;

2. $V(\mathsf{I}) = V(\mathsf{K}) = V(\mathsf{S}) = \varnothing$;

3. $V(FG) = V(F) \cup V(G)$.

A operação de substituição de variáveis neste sistema consiste numa substituição cega de todas ocorrências de interesse, uma vez que não existem variáveis ligadas.

Definição 1.38. Sejam $F, G \in \mathrm{CL}$ e $x \in \mathcal{V}$. Define-se $F[x \coloneqq G]$ recursivamente por:

1. $x[x \coloneqq G] = G$;

2. $y[x \coloneqq G] = y$, se $x \neq y$;

3. $H[x \coloneqq G] = H$, para todo $H \in \{\mathsf{I}, \mathsf{K}, \mathsf{S}\}$;

4. $(H_1H_2)[x \coloneqq G] = H_1[x \coloneqq G]H_2[x \coloneqq G]$.

Definição 1.39. Seja $\mathcal{R}$ uma relação binária sobre CL. Dizemos que $\mathcal{R}$ é *compatível* se satisfaz (para todo F, G, H):

1. $(FH)\mathcal{R}(GH)$, sempre que $F\mathcal{R}G$;

2. $(HF)\mathcal{R}(HG)$, sempre que $F\mathcal{R}G$.

Fixada a definição de relação compatível sobre CL, podemos transportar imediatamente para este conjunto os conceitos de passo de redução ($\to_{\mathcal{R}}$), redução ($\to_{\mathcal{R}}^{*}$) e conversão ($=_{\mathcal{R}}$).

Definição 1.40. Seja w a seguinte noção de redução sobre CL:

(w) $\{(\mathsf{I}F, F) \mid F \in \mathrm{CL}\} \cup \{(\mathsf{K}FG, F) \mid F, G \in \mathrm{CL}\} \cup \{(\mathsf{S}FGH, FH(GH)) \mid F, G, H \in \mathrm{CL}\}$.

Exemplo 1.41. Tem-se

1. $\mathsf{I}(\mathsf{K}x) \to_w \mathsf{K}x$;
2. $\mathsf{K}(\mathsf{I}xy)z \to_w \mathsf{K}(xy)z \to_w xy$;
3. $\mathsf{SKK}x \to_w \mathsf{K}x(\mathsf{K}x) \to_w x$;
4. $\mathsf{SII}x \to_w \mathsf{I}x(\mathsf{I}x) \to_w xx$.

Vejamos agora a relação entre a Lógica Combinatória e o Cálculo λ.

A conversão w é facilmente simulada no Cálculo λ pela conversão β através da seguinte tradução.

Definição 1.42. Seja $\cdot_\lambda : \mathrm{CL} \to \Lambda$ a função definida recursivamente por:

1. $x_\lambda = x$;
2. $\mathsf{I}_\lambda = \lambda x x$;
3. $\mathsf{K}_\lambda = \lambda xy.x$;
4. $\mathsf{S}_\lambda = \lambda xyz.xz(yz)$.
5. $(FG)_\lambda = F_\lambda G_\lambda$.

Proposição 1.43 ((Sørensen and Urzyczyn, 2006, Proposition 5.4.3))**.** *Para todo $F, G \in CL$, se $F \to_w G$ então $F_\lambda \to_\beta^+ G_\lambda$.*

Demonstração. Por indução rotineira sobre $F \to_w G$. □

Corolário 1.43.1. *Para todo $F, G \in CL$, $F_\lambda =_\beta G_\lambda$, se $F =_w G$.*

Falta agora traduzir termos-λ em combinadores. A dificuldade reside na tradução de abstracções.

Definição 1.44. Seja $\cdot_{\mathrm{CL}} : \Lambda \to \mathrm{CL}$ a função definida recursivamente por:

1. $x_{\mathrm{CL}} = x$;
2. $(MN)_{\mathrm{CL}} = M_{\mathrm{CL}} N_{\mathrm{CL}}$;
3. $(\lambda x M)_{\mathrm{CL}} = \lambda^* x M_{\mathrm{CL}}$,

onde, para todo $x \in \mathcal{V}$ e $F \in \mathrm{CL}$, $\lambda^* x F$ é meta-termo definido por:

1. I, se $F = x$;
2. $\mathsf{K}F$, se $x \notin V(F)$;
3. $\mathsf{S}(\lambda^* x F_1)(\lambda^* x F_2)$, se $F = F_1 F_2$ e o caso anterior não se verificar.

Observação 1.45. Note-se que $FV(M) = V(M_{\mathrm{CL}})$. Logo, se M é fechado, M_{CL} é um combinador.

Esta definição de $\lambda^* xF$ é motivada pela seguinte proposição.

Proposição 1.46 ((Sørensen and Urzyczyn, 2006, Proposition 5.4.6))**.** *Para todo $F, G \in CL$ e $x \in \mathcal{V}$, verifica-se:*

1. $(\lambda^* xF)G \to_w^* F[x \coloneqq G]$;

2. $(\lambda^* xF)_\lambda \to_\beta^* \lambda x(F_\lambda)$.

Contudo, a conversão β não é simulável pela conversão w através desta tradução. Considere-se (Çağman and Hindley, 1998) $M = (\lambda y(\lambda xxy)z)$ e $N = (\lambda yzy)$. Então $M =_\beta N$, mas

$$M_{\mathrm{CL}} = \mathsf{S}(\mathsf{S}(\mathsf{K}(\mathsf{SI}))(\mathsf{S}(\mathsf{KK})\mathsf{I}))(\mathsf{K}z)$$

não é convertível em N_{CL} em virtude de já ser uma forma w normal. Fundamentalmente, a razão para este desnível prende-se com a não-admissibilidade da regra (ξ) em $=_w$ [5]:

$$(\xi)\frac{M = N}{\lambda^* xM = \lambda^* xN},$$

sendo por causa disto que a conversão w se diz *weak* [6].

Neste ponto podemos munir a conversão w de forma a que simule a conversão β ou encontrar uma conversão em Λ que simule a conversão w. A primeira alternativa está presente em (Hindley and Seldin, 2008, Chapter 9), e requer uma regra nova sob uma nova classe de termos (ditos *funcionais*). A segunda alternativa é elaborada em (Çağman and Hindley, 1998).

Neste trabalho estaremos interessados na conversão $\beta\eta$. Para isso vamos considerar a versão extensional da conversão w, a conversão *ext*, que já admite a regra (ξ).

[5] A regra (ξ) é satisfeita em $=_\beta$ por se tratar da condição 2 do fecho de compatibilidade, cf. 1.6.

[6] Ver (Sørensen and Urzyczyn, 2006, p. 117) ou (Hindley and Seldin, 2008, Warning 2.33 p. 30, Lemma 9.5 pp. 93–94).

Definição 1.47. Define-se a conversão *ext* como sendo a menor relação compatível e de equivalência satisfazendo:

1. Se $F =_w G$ então $F =_{ext} G$;
2. Se $Fx =_{ext} Gx$ para todo $x \notin V(FG)$, então $F =_{ext} G$.

Pode-se provar que a conversão *ext* é a variante extensional da conversão w, da mesma forma que a conversão $\beta\eta$ é a variante extensional de β (Hindley and Seldin, 2008, p. 83).

Teorema 1.48 (Correspondência)**.** *Para todo $M, N \in \Lambda$ e $F, G \in CL$, tem-se:*

1. $(F_\lambda)_{CL} =_{ext} F$;
2. $(M_{CL})_\lambda =_{\beta\eta} M$;
3. $F =_{ext} G$ *se e só se* $F_\lambda =_{\beta\eta} G_\lambda$;
4. $M =_{\beta\eta} N$ *se e só se* $M_{CL} =_s N_{CL}$.

Demonstração. Ver (Hindley and Seldin, 2008, Theorem 9.17) ou (Sørensen and Urzyczyn, 2006, Proposition 5.5.3). □

Observação 1.49. Esta correspondência pode ser obtida de forma mais elegante através de uma noção de redução sobre CL, s (dita *strong*), cuja conversão induzida ($=_s$) é igual a $=_{ext}$ (Hindley and Seldin, 2008, Lemma 8.17 (d), p. 90). A redução $\to_s^*$ consiste na junção da regra (ξ) à redução $\to_w^*$ (Hindley and Seldin, 2008, Definition 8.15).

Nesta subsecção apenas mencionámos equivalências entre a Lógica Combinatória e o Cálculo λ ao nível das conversões. Ao nível da reduções as correspondências complicam-se:

- Tem-se que $M \to_{\beta\eta}^* N$ implica $M_{\mathrm{CL}} \to_s^* N_{\mathrm{CL}}$, mas $F \to_s^* G$ não implica $F_\lambda \to_{\beta\eta}^* G_\lambda$ (Hindley and Seldin, 2008, Discussion 9.18).
- A simulação da redução $\to_\beta^*$ está desde (Curry and Feyes, 1958) em aberto, sendo (Mezghiche, 1984, 1989) a melhor proposta à data (Seldin, 2011).

- A simulação da redução $\rightarrow_w^*$ pelo Cálculo λ também não é uma equivalência "completa", cf. (Çağman and Hindley, 1998, Proposition 3.2, p. 243).

Capítulo 2

Combinadores categoriais

Apresentamos neste capítulo a Lógica Combinatória Categorial e vemos a sua relação, quer com as categorias cartesianas fechadas, quer com o Cálculo λ.

Na secção 2.1 introduzimos a sintaxe e duas conversões da Lógica Combinatória Categorial. Uma das conversões diz-se *forte* e a outra diz-se *fraca*: a fraca está incluída na forte que, por sua vez, está "intimamente" ligada às categorias cartesianas fechadas.

Na secção 2.2 vemos uma correspondência ao nível das conversões forte da Lógica Combinatória Categorial e $\beta\eta SP$ do Cálculo λc, uma extensão do Cálculo λ dotada de pares, projecções e respectivas conversões.

2.1 Lógica Combinatória Categorial

A Lógica Combinatória Categorial é um sistema formal que gira em torno de um conjunto de termos extraídos da teoria das categorias cartesianas fechadas: *identidade*, *composição*, *emparelhamento*, *primeira/segunda projecção*, *abstracção* e *aplicação*. Para além disso, a relação principal entre os termos deste sistema — a chamada *relação forte* — reflecte propriedades acerca destes termos conhecidas no campo da teoria de categorias.

2.1.1 Sintaxe

Convenção 2.1. Seja $\mathcal{V}$ um conjunto numerável (de *variáveis*). Denotamos os elementos deste conjunto pelas meta-variáveis x, y, z, eventualmente etiquetadas.

Definição 2.2. O conjunto dos *termos categoriais*, $\mathcal{CCL}$ (do inglês "Categorical Combinatory Logic"), é definido indutivamente por:

1. Todas as variáveis estão em $\mathcal{CCL}$, isto é, $\mathcal{V} \subseteq \mathcal{CCL}$;
2. As constantes *Id* (*identidade*), *Fst* (*primeira projecção*), *Snd* (*segunda projecção*) e *App* (*aplicação*) estão em $\mathcal{CCL}$;
3. Se A está em $\mathcal{CCL}$, então a *abstracção* $\Lambda(A)$ está em $\mathcal{CCL}$;
4. Se A e B estão em $\mathcal{CCL}$, então o *emparelhamento* $\langle A, B\rangle$ está em $\mathcal{CCL}$;
5. Se A e B estão em $\mathcal{CCL}$, então a *composição* $(A \circ B)$ está em $\mathcal{CCL}$.

Notação 2.3. Adoptamos as seguintes convenções de escrita a respeito de termos categoriais:

1. As meta-variáveis A, B, C, D, eventualmente etiquetadas, denotam termos categoriais.
2. Omitem-se os parêntesis mais externos em composições, isto é, escrevemos $A \circ B$ no lugar de $(A \circ B)$.
3. Escrevemos $\langle A_1, A_2, A_3\rangle$ no lugar de $\langle\langle A_1, A_2\rangle, A_3\rangle$. Mais geralmente,

$$\langle A_1, A_2, \ldots, A_n\rangle = \langle \ldots \langle A_1, A_2\rangle, \ldots, A_n\rangle.$$

Denotamos por $V(A)$ o conjunto das variáves que ocorrem em A. Se $V(A) = \varnothing$, dizemos que A é um *combinador (categorial)*.

Definição 2.4. Define-se $V(\cdot) : \mathcal{CCL} \to \mathcal{P}(\mathcal{V})$ recursivamente por:

1. $V(x) = \{x\}$;

2. $V(\diamond) = \varnothing$, se $\diamond \in \{Id, Fst, Snd, App\}$;

3. $V(\Lambda(B)) = V(B)$;

4. $V(\langle B, C\rangle) = V(B) \cup V(C)$;

5. $V(B \circ C) = V(B) \cup V(C)$.

A conversão forte, e em especial a proposição 2.7, irá reforçar a ligação da Lógica Combinatória Categorial às categorias cartesiansas fechadas.

2.1.2 Conversão forte

Recorde-se (ver 1.9) que uma *redução* (notação: $\rightarrow^*_{\mathcal{R}}$) é dada pelo fecho compatível, reflexivo e transitivo de uma relação binária $\mathcal{R}$ e uma *conversão* (notação: $=_{\mathcal{R}}$) pelo fecho compatível e de equivalência de $\mathcal{R}$. De resto, neste capítulo estaremos mais interessados nas conversões que nas reduções.

Definição 2.5. Dizemos que uma relação binária $\mathcal{R} \subseteq \mathcal{CCL}^2$ é *compatível* se satisfaz as seguintes condições (para todo A, B, C e D):

1. $\Lambda(A)\,\mathcal{R}\,\Lambda(B)$, se $A\,\mathcal{R}\,B$;

2a. $\langle A, B\rangle\,\mathcal{R}\,\langle C, B\rangle$, se $A\,\mathcal{R}\,C$; 2b. $\langle A, B\rangle\,\mathcal{R}\,\langle A, D\rangle$, se $B\,\mathcal{R}\,D$;

3a. $(A \circ B)\,\mathcal{R}\,(C \circ B)$, se $A\,\mathcal{R}\,C$; 3b. $(A \circ B)\,\mathcal{R}\,(A \circ D)$, se $B\,\mathcal{R}\,D$.

Definição 2.6. Define-se a *conversão forte*[1], $=_{CCL\beta\eta SP}$, como sendo a menor relação compatível e de equivalência satisfazendo as seguintes condições (para todo A, B e C):

(Ass) $(A \circ B) \circ C = A \circ (B \circ C)$;

(IdL) $Id \circ A = A$;

(IdR) $A \circ Id = A$;

(Fst) $Fst \circ \langle A, B\rangle = A$;

[1] Ou *Lógica Combinatória Categorial Forte* (Curien, 1993, Definition 1.2.12).

(Snd) $Snd \circ \langle A, B\rangle = B$;

(DPair) $\langle A, B\rangle \circ C = \langle A \circ C, B \circ C\rangle$;

(Beta) $App \circ \langle \Lambda(A), B\rangle = A \circ \langle Id, B\rangle$;

(DΛ) $\Lambda(A) \circ B = \Lambda(A \circ \langle B \circ Fst, Snd\rangle)$;

(AI) $\Lambda(App) = Id$;

(FSI) $\langle Fst, Snd\rangle = Id$.

(onde escrevemos simplesmente = no lugar de $=_{CCL\beta\eta SP}$.)

As condições que definem a conversão forte são reminiscentes de propriedades verificadas pelos constructos das categorias cartesianas fechadas. Com efeito, em (Lambek and Scott, 1986, pp. 52–53) as categorias cartesianas fechadas são estudadas sob o ponto de vista de um sistema dedutivo caracterizado pelo conjunto de axiomas

$$\underbrace{\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR}}_{\text{Categoria}} + \underbrace{\mathrm{Fst} + \mathrm{Snd} + \mathrm{SPair}}_{\text{Cartesiana}} + \underbrace{\mathrm{App} + \mathrm{S}\Lambda}_{\text{Fechada}} \tag{2.1}$$

em que

(SPair) $\langle Fst \circ A, Snd \circ A\rangle = A$;

(App) $App \circ \langle \Lambda(A) \circ Fst, Snd\rangle = A$;

(SΛ) $\Lambda(App \circ \langle A \circ Fst, Snd\rangle) = A$;

excepto apenas que falta aqui equação relativa ao objecto terminal. Curiosamente, a conversão induzida pelo sistema de equações 2.1 é igual à conversão forte.

Proposição 2.7 ((Curien, 1993, Exercise 1.2.12.1))**.** *Seja* $=_{\mathrm{CCF}}$ *(de "Categoria Cartesiana Fechada") a menor relação compatível e de equivalência satisfazendo o sistema de equações 2.1. Então, para todo* $A, B \in \mathcal{CCL}$, $A =_{\mathrm{CCF}} B$ *se e só se* $A =_{CCL\beta\eta SP} B$.

Demonstração. Basta demonstrar, por um lado, que a conversão forte inclui as equações SPair, App e SΛ, e por outro, que a conversão CCF inclui as equações DPair, Beta, DΛ, AI e FSI. Demonstramos:

1. $CCL\beta\eta SP \vdash$ SPair, App, SΛ;
2. CCF $\vdash$ DPair, FSI;
3. CCF, FSI $\vdash$ AI;
4. CCF, DPair $\vdash$ DΛ, Beta.

$CCL\beta\eta SP \vdash$ SPair

$$
\begin{array}{ll}
 & \langle Fst \circ A, Snd \circ A \rangle \\
= & \quad \text{DPair} \\
 & \langle Fst, Snd \rangle \circ A \\
= & \quad \text{FSI, IdL} \\
 & A
\end{array}
$$

$CCL\beta\eta SP \vdash$ App

$$
\begin{array}{ll}
 & App \circ \langle \Lambda(B) \circ Fst, Snd \rangle \\
= & \quad \text{DΛ} \\
 & App \circ \langle \Lambda(B \circ \langle Fst \circ Fst, Snd \rangle), Snd \rangle \\
= & \quad \text{Beta} \\
 & (B \circ \langle Fst \circ Fst, Snd \rangle) \circ \langle Id, Snd \rangle \\
= & \quad \text{Ass, DPair, Fst, Snd, IdR} \\
 & B \circ \langle Fst, Snd \rangle \\
= & \quad \text{FSI, IdR} \\
 & B
\end{array}
$$

$CCL\beta\eta SP \vdash$ SΛ

$$
\begin{array}{ll}
 & \Lambda(App \circ \langle B \circ Fst, Snd \rangle) \\
= & \quad \text{DΛ} \\
 & \Lambda(App) \circ B \\
= & \quad \text{AI, IdL} \\
 & B
\end{array}
$$

CCF $\vdash$ DPair

$$
\begin{array}{ll}
 & \langle A, B \rangle \circ C \\
= & \quad \text{SPair} \\
 & \langle Fst \circ (\langle A, B \rangle \circ C), Snd \circ (\langle A, B \rangle \circ C) \rangle \\
= & \quad \text{Ass, Fst, Snd} \\
 & \langle A \circ C, B \circ C \rangle
\end{array}
$$

CCF $\vdash$ FSI

$$
\begin{array}{ll}
 & Id \\
= & \quad \text{SPair} \\
 & \langle Fst \circ Id, Snd \circ Id \rangle \\
= & \quad \text{IdR}
\end{array}
$$

$\langle Fst, Snd \rangle$

CCF, FSI $\vdash$ AI

$$
\begin{array}{ll}
 & Id \\
= & \quad \text{S}\Lambda \\
 & \Lambda(App \circ \langle Id \circ Fst, Snd \rangle) \\
= & \quad \text{IdL, FSI, IdR} \\
 & \Lambda(App)
\end{array}
$$

CCF, DPair $\vdash$ $D\Lambda$

$$
\begin{array}{ll}
 & \Lambda(A \circ \langle B \circ Fst, Snd \rangle) \\
= & \quad \text{App} \\
 & \Lambda((App \circ \langle \Lambda(A) \circ Fst, Snd \rangle) \circ \langle B \circ Fst, Snd \rangle) \\
= & \quad \text{Ass, DPair, Ass, Fst, Snd} \\
 & \Lambda(App \circ \langle \Lambda(A) \circ (B \circ Fst), Snd \rangle) \\
= & \quad \text{Ass, S}\Lambda \\
 & \Lambda(A) \circ B
\end{array}
$$

CCF, DPair $\vdash$ Beta

$$
\begin{array}{ll}
 & A \circ \langle Id, B \rangle \\
= & \quad \text{App} \\
 & (App \circ \langle \Lambda(A) \circ Fst, Snd \rangle) \circ \langle Id, B \rangle \\
= & \quad \text{Ass, DPair, Ass, Fst, Snd} \\
 & App \circ \langle \Lambda(A) \circ Id, B \rangle \\
= & \quad \text{IdR} \\
 & App \circ \langle \Lambda(A), B \rangle
\end{array}
$$

□

Vale a pena realçar que a proposição 2.7 não apresenta uma equivalência entre as categorias cartesianas fechadas e a Lógica Combinatória Categorial, note-se a omissão do objecto terminal; serve apenas para reforçar, de forma intuitiva, a ligação próxima entre estas duas teorias. Uma consequência útil deste resultado é permitir que invoquemos as relações SPair, App e SΛ na conversão forte. Outra relação que poderemos incluir nesta conversão é a seguinte generalização da relação Beta:

(Beta') $App \circ \langle \Lambda(A) \circ B, C \rangle = A \circ \langle B, C \rangle$.

Lema 2.8 ((Curien, 1993, Exercise 1.2.12.3))**.** *A conversão forte inclui a equação Beta'.*

Demonstração. Temos

$$
\begin{array}{ll}
 & App \circ \langle \Lambda(A) \circ B, C \rangle \\
= & \quad \text{DΛ} \\
 & App \circ \langle \Lambda(A \circ \langle B \circ Fst, Snd \rangle), C \rangle \\
= & \quad \text{Beta} \\
 & (A \circ \langle B \circ Fst, Snd \rangle) \circ \langle Id, C \rangle \\
= & \quad \text{Ass, DPair, Fst, Snd, IdR} \\
 & A \circ \langle B, C \rangle
\end{array}
$$

□

2.1.3 Conversão fraca

Até agora temos visto composições — e.g. $(A \circ B)$ — e emparelhamentos — e.g. $\langle A, B \rangle$. Tratamos em seguida de introduzir *aplicações* — e.g. $A.B$ — e *pares* — e.g. (A, B). Intuitivamente, uma *aplicação* $A.B$ denota a aplicação de uma função A a um argumento B. Isto é diferente da composição $(A \circ B)$ cuja intuição é de que se trata de uma função que, se aplicada a um argumento C, gera a aplicação $A.(B.C)$. Por seu turno, um emparelhamento $\langle A, B \rangle$ denota a função que, quando aplicada a um argumento C, gera o par (no sentido de um objecto de um produto cartesiano) $(A.C, B.C)$. [2]

[2]Em (Curien, 1993), a terminologia é *pairing* para emparelhamento, *couple* para par, *composition* para composição, e *applying* para aplicação.

Definição 2.9. Para todo $A, B \in \mathcal{CCL}$, definem-se os meta-termos $A^{>}$, $A^{<}$, $A.B$ (*aplicação*) e (A, B) (*par*) do seguinte modo:

1. $A^{>} = \Lambda(A \circ \mathit{Snd})$;
2. $A^{<} = \mathit{App} \circ \langle A, \mathit{Id} \rangle$;
3. $A.B = (A \circ B^{>})^{<}$;
4. $(A, B) = \langle A^{>}, B^{>} \rangle^{<}$.

Observação 2.10 (Para o leitor familiarizado com teoria de categorias)**.** Intuitivamente, podemos pensar na meta-operação $^{>}$ como na correspondência entre os morfismos $D \to E$ e $1 \to E^D$ para todo o objecto D, E, e $^{<}$ como na correspondência inversa, onde 1 denota o objecto terminal de uma categoria (Curien, 1993, p. 33).

Notação 2.11. Adoptamos as seguintes convenções de escrita a respeito dos meta-termos par e aplicação:

1. O símbolo "." da aplicação associa para a esquerda, ou seja,

$$A_1.A_2.\ \ldots\ .A_n = (\ldots(A_1.A_2).\ \ldots\ .A_n),$$

2. A aplicação tem prioridade sobre a composição, isto é, escrevemos $A.B \circ C$ no lugar de $(A.B) \circ C$;
3. Escrevemos (A_1, A_2, A_3) no lugar de $((A_1, A_2), A_3)$. Mais geralmente,

$$(A_1, A_2, \ldots, A_n) = (\ldots(A_1, A_2), \ldots, A_n).$$

Definição 2.12. Define-se a *conversão fraca*[3], $=_{CCL}$, como sendo a menor relação compatível e de equivalência satisfazendo (para todo $A, B, C \in \mathcal{CCL}$):

(id) $\mathit{Id}.A = A$;

(ass) $(A \circ B).C = A.(B.C)$;

[3]Ou *Lógica Combinatória Categorial Fraca* (Curien, 1993, p. 34).

(fst) $Fst.(A, B) = A$;

(snd) $Snd.(A, B) = B$;

(dpair) $\langle A, B\rangle.C = (A.C, B.C)$;

(app) $App.(A, B) = A.B$;

(dΛ) $\Lambda(A).B.C = A.(B, C)$.

(onde escrevemos simplesmente $=$ no lugar de $=_{CCL}$.)

Observa-se que as condições da conversão fraca ditam o efeito esperado dos meta-termos aplicação e par acabados de introduzir.

A conversão fraca está incluída na conversão forte. Na verdade, a codificação dos meta-termos aplicação e par escolhida na definição 2.9 é consequência das conversões forte, fraca e *Quote* (Curien, 1993, Exercise 1.2.14.1), em que *Quote* é induzida pelas condições (para todo A, B e C):

(Quote1) $\Lambda(Fst).A \circ B = \Lambda(Fst).A$;

(Quote2) $App \circ \langle A \circ \Lambda(Fst).B, C\rangle = A.B \circ C$;

(Quote3) $\Lambda(A).B = A \circ \langle \Lambda(Fst).B, Id\rangle$.

Terminamos esta subsecção vendo como a conversão forte inclui as conversões fraca e *Quote*.

Lema 2.13 ((Curien, 1993, Lemma 1.2.14))**.** *Denote-se por RA (do inglês "Right Absorving") o conjunto dos termos categoriais B que satisfazem a equação*

$$B =_{CCL\beta\eta SP} B \circ \Lambda(Snd). \tag{2.2}$$

Então, para todo $A \in \mathcal{CCL}$ e $B \in RA$, verificam-se as seguintes condições:

1. $A^{>} \in RA$;
2. $(A^{>})^{<} =_{CCL\beta\eta SP} A$;

3. $B =_{CCL\beta\eta SP} B \circ A$*;*

4. $(B^{<})^{>} =_{CCL\beta\eta SP} B$*.*

Demonstração. Temos:

1.

$$
\begin{array}{ll}
 & A^{>} \circ \Lambda(Snd) \\
= & \quad 2.9 \\
 & \Lambda(A \circ Snd) \circ \Lambda(Snd) \\
= & \quad \mathrm{D}\Lambda \\
 & \Lambda((A \circ Snd) \circ \langle \Lambda(Snd) \circ Fst, Snd \rangle) \\
= & \quad \text{Ass, Snd} \\
 & \Lambda(A \circ Snd) \\
= & \quad 2.9 \\
 & A^{>}
\end{array}
$$

2.

$$
\begin{array}{ll}
 & (A^{>})^{<} \\
= & \quad 2.9 \\
 & App \circ \langle \Lambda(A \circ Snd), Id \rangle \\
= & \quad \text{Beta} \\
 & (A \circ Snd) \circ \langle Id, Id \rangle \\
= & \quad \text{Ass, Snd, IdR} \\
 & A
\end{array}
$$

3.

$$
\begin{array}{ll}
 & B \circ A \\
= & \quad (2.2) \\
 & (B \circ \Lambda(Snd)) \circ A \\
= & \quad \text{Ass, D}\Lambda \\
 & B \circ \Lambda(Snd \circ \langle A \circ Fst, Snd \rangle) \\
= & \quad \text{Snd} \\
 & B \circ \Lambda(Snd) \\
= & \quad (2.2) \\
 & B
\end{array}
$$

4.

$$
\begin{array}{ll}
 & (B^{<})^{>} \\
= & \quad 2.9 \\
 & \Lambda((App \circ \langle B, Id \rangle) \circ Snd) \\
= & \quad \text{Ass, DPair, IdL} \\
 & \Lambda(App \circ \langle B \circ Snd, Snd \rangle) \\
= & \quad 2.13, \text{ alínea 3} \\
 & \Lambda(App \circ \langle B, Snd \rangle) \\
= & \quad 2.13, \text{ alínea 3} \\
 & \Lambda(App \circ \langle B \circ Fst, Snd \rangle) \\
= & \quad \mathrm{S}\Lambda \\
 & B
\end{array}
$$

□

Proposição 2.14. *A conversão forte inclui as conversões fraca e Quote.*

Demonstração. Demonstramos[4]:

[4](Curien, 1993, pp. 35–36)

1. $CCL\beta\eta SP \vdash$ ass, fst, snd, dpair, app, Quote2;
2. $CCL\beta\eta SP$, Quote2 $\vdash$ Quote3;
3. $CCL\beta\eta SP$, Quote3, ass, snd, dpair, app $\vdash$ id;
4. $CCL\beta\eta SP$, Quote3, id, ass, fst, snd, dpair, app $\vdash$ dΛ;
5. $CCL\beta\eta SP \vdash$ Quote1.

Vejamos primeiro alguns resultados auxiliares:

i. $CCL\beta\eta SP \vdash \Lambda(Fst).A = A^{>}$;

$$
\begin{array}{ll}
 & \Lambda(Fst).A \\
= & \quad \text{2.9} \\
 & App \circ \langle \Lambda(Fst) \circ A^{>}, Id \rangle \\
= & \quad \text{Beta', Fst} \\
 & A^{>}
\end{array}
$$

ii. $CCL\beta\eta SP$, Quote3 $\vdash \Lambda(Snd).A = Id$;

$$
\begin{array}{ll}
 & \Lambda(Snd).A \\
= & \quad \text{Quote3} \\
 & Snd \circ \langle \Lambda(Fst).A, Id \rangle \\
= & \quad \text{Snd} \\
 & Id
\end{array}
$$

iii. $CCL\beta\eta SP$, snd, ass $\vdash \Lambda(Fst).A = \Lambda(Snd \circ Fst).(B, A)$;

$$
\begin{array}{ll}
 & \Lambda(Fst).A \\
= & \quad \text{snd, introduzindo } (B, A), \text{ com } B \text{ arbitrário; ass} \\
 & (\Lambda(Fst) \circ Snd).(B, A) \\
= & \quad \text{D}\Lambda \\
 & \Lambda(Fst \circ \langle Snd \circ Fst, Snd \rangle).(B, A) \\
= & \quad \text{Fst} \\
 & \Lambda(Snd \circ Fst).(B, A)
\end{array}
$$

iv. $CCL\beta\eta SP, \text{ass, fst, snd, dpair, app} \vdash \Lambda(Fst).A.B = A.$

$$
\begin{array}{ll}
 & \Lambda(Fst).A.B \\
= & \quad \text{iii., fst} \\
 & (\Lambda(Snd \circ Fst).(B,A)).(Fst.(B,A)) \\
= & \quad \text{app, dpair, ass} \\
 & (App \circ \langle \Lambda(Snd \circ Fst), Fst \rangle).(B,A) \\
= & \quad \text{Beta, Ass, Fst, IdR, snd} \\
 & A
\end{array}
$$

Temos, então:

$\underline{CCL\beta\eta SP \vdash \text{ass}}$

$$
\begin{array}{ll}
 & A.(B.C) \\
= & \quad 2.9 \\
 & (A \circ ((B \circ C^{>})^{<})^{>})^{<} \\
= & \quad 2.13 \\
 & (A \circ (B \circ C^{>}))^{<} \\
= & \quad \text{Ass, } 2.9 \\
 & (A \circ B).C
\end{array}
$$

$\underline{CCL\beta\eta SP \vdash \text{fst}}$

$$
\begin{array}{ll}
 & Fst.(A,B) \\
= & \quad 2.9 \\
 & (Fst \circ (\langle A^{>}, B^{>} \rangle^{<})^{>})^{<} \\
= & \quad 2.13 \\
 & (Fst \circ \langle A^{>}, B^{>} \rangle)^{<} \\
= & \quad \text{Fst, } 2.13 \\
 & A
\end{array}
$$

$\underline{CCL\beta\eta SP \vdash \text{snd}}$ Análogo ao caso fst.

$\underline{CCL\beta\eta SP \vdash \text{dpair}}$

$$
\begin{array}{ll}
 & (A.C, B.C) \\
= & \quad 2.9 \\
 & \langle ((A \circ C^{>})^{<})^{>}, ((B \circ C^{>})^{<})^{>} \rangle^{<} \\
= & \quad 2.13 \\
 & \langle A \circ C^{>}, B \circ C^{>} \rangle^{<} \\
= & \quad 2.9, \text{ DPair} \\
 & \langle A, B \rangle.C
\end{array}
$$

$\underline{CCL\beta\eta SP \vdash \text{app}}$

$$
\begin{array}{ll}
 & App.(A,B) \\
= & \quad 2.9 \\
 & (App \circ (\langle A^{>}, B^{>} \rangle^{<})^{>})^{<} \\
= & \quad 2.13, 2.9 \\
 & (App \circ \langle \Lambda(A \circ Snd), B^{>} \rangle)^{<} \\
= & \quad \text{Beta, Ass, Snd, } 2.9 \\
 & A.B
\end{array}
$$

$\underline{CCL\beta\eta SP \vdash \text{Quote2}}$

$$
\begin{array}{ll}
 & App \circ \langle A \circ \Lambda(Fst).B, C \rangle \\
= & \quad \text{i.} \\
 & App \circ \langle A \circ B^{>}, C \rangle \\
= & \quad \text{2.13, Ass} \\
 & App \circ \langle (A \circ B) \circ C, C \rangle \\
= & \quad \text{IdL, DPair} \\
 & App \circ (\langle A \circ B, Id \rangle \circ C) \\
= & \quad \text{Ass, 2.9} \\
 & A.B \circ C
\end{array}
$$

$\underline{CCL\beta\eta SP, \text{Quote2} \vdash \text{Quote3}}$

$$
\begin{array}{ll}
 & \Lambda(A).B \\
= & \quad \text{IdR} \\
 & \Lambda(A).B \circ Id \\
= & \quad \text{Quote2} \\
 & App \circ \Lambda(A) \circ \Lambda(Fst).B, Id \rangle \\
= & \quad \text{Beta'} \\
 & A \circ \langle \Lambda(Fst).B, Id \rangle
\end{array}
$$

$\underline{CCL\beta\eta SP, \text{Quote3, ass, snd, dpair, app} \vdash \text{id}}$

$$
\begin{array}{ll}
 & Id.A \\
= & \quad \text{ii. introduzindo } (B, A), \text{ com } B \text{ arbitrário} \\
 & (\Lambda(Snd).(B, A)).A \\
= & \quad \text{snd} \\
 & (\Lambda(Snd).(B, A)).(Snd.(B, A)) \\
= & \quad \text{app} \\
 & App.(\Lambda(Snd).(B, A), Snd.(B, A)) \\
= & \quad \text{dpair, ass} \\
 & (App \circ \langle \Lambda(Snd), Snd \rangle).(B, A) \\
= & \quad \text{Beta} \\
 & (Snd \circ \langle Id, Snd \rangle).(B, A) \\
= & \quad \text{Snd, snd} \\
 & A
\end{array}
$$

$\underline{CCL\beta\eta SP, \text{Quote3, id, ass, fst, snd, dpair, app} \vdash \text{d}\Lambda}$

$$
\begin{array}{ll}
 & \Lambda(A).B.C \\
= & \quad \text{Quote3} \\
 & (A \circ \langle \Lambda(Fst).B, Id \rangle).C \\
= & \quad \text{ass, dpair, iv., id} \\
 & A.(B, C)
\end{array}
$$

$\underline{CCL\beta\eta SP \vdash \text{Quote1}}$

$$\begin{array}{ll} & \Lambda(Fst).A \circ B \\ = & \quad \text{i.} \\ & A^{>} \circ B \\ = & \quad 2.13 \\ & A^{>} \\ = & \quad \text{i.} \\ & \Lambda(Fst).A \end{array}$$

□

2.2 Termos categoriais versus termos-λ

Vemos nesta secção uma correspondência entre termos categoriais e termos-λ que se estende às conversões forte e $\beta\eta$. Essencialmente, a correspondência consiste numa codificação do Cálculo λ com índices de de Bruijn na Lógica Combinatória Categorial. A observação-chave é que a operação de substituição para termos-λ de de Bruijn pode ser codificada em termos categoriais e simulada através da conversão forte.

Observação 2.15. A título de curiosidade, é bem conhecida[5] uma correspondência entre o Cálculo λ e as categorias cartesianas fechadas: a teoria do Cálculo λ tipificado dotado de pares, projecções e conversões $\beta\eta$ induz uma categoria cartesiana fechada e vice-versa.

Na subsecção 2.2.1 munimos o Cálculo λ com pares e projecções, introduzindo com isso o Cálculo λc (do inglês "couples"). Apenas consideramos a conversão $\beta\eta SP$ deste Cálculo, que consiste na conversão $\beta\eta$ mais as conversões referentes às projecções e pares. Em 2.20 justificamos a introdução deste Cálculo.

Na subsecção 2.2.2 codificamos os termos-λc de de Bruijn e a respectiva operação de substituição na Lógica Combinatória Categorial. Aqui já vemos que a conversão forte permite simular a conversão $\beta\eta SP$.

[5] Ver (Lambek and Scott, 1986, Chapter 10–11).

Na subsecção 2.2.3 apresentamos finalmente as traduções entre termos categoriais e termos-λc, e vemos como as conversões se simulam uma à outra a partir destas traduções.

2.2.1 Cálculo λc

Definição 2.16. O conjunto dos *termos-λc*, Λc, é definido indutivamente por:

1. Todos os termos-λ são termos-λc, isto é, $\Lambda \subseteq \Lambda c$;
2. Se M e N estão em Λc, então o *par* (M, N) está em Λc;
3. Se M está em Λc, então a *primeira projecção* $fst(M)$ está em Λc;
4. Se M está em Λc, então a *segunda projecção* $snd(M)$ está em Λc.

As definições do Cálculo λ transitam, com as devidas adaptações, para o Cálculo λc. Regra geral, nos novos casos as definições são dadas por distribuição pelos respectivos subtermos, conforme:

- $FV((M, N)) = FV(M) \cup FV(N)$;
- $fst(M)[x \coloneqq N] = fst(M[x \coloneqq N])$;
- ... e assim em diante.

Excepto apenas quando for dito algo em contrário, aplica-se a regra acabada de descrever a todas as definições implícitas encontradas nesta subsecção.

A adição dos novos termos obriga a uma actualização da definição de relação compatível. Definimo-la explicitamente, pese embora esta seguir a regra de construção indicada em 2.16.

Definição 2.17. Uma relação binária $\mathcal{R} \subseteq \Lambda c^2$ diz-se *compatível* se satisfaz as seguintes condições (para todo $M, N, P, Q \in \Lambda c$):

1. $\mathcal{R}$ é compatível sobre Λ, isto é, $\mathcal{R}$ satisfaz todas as condições de 1.6;
2. $fst(M)\,\mathcal{R}\,fst(N)$, se $M\,\mathcal{R}\,N$;

3. $\mathit{snd}(M)\,\mathcal{R}\,\mathit{snd}(N)$, se $M\,\mathcal{R}\,N$;

4a. $(M,N)\,\mathcal{R}\,(P,N)$, se $M\,\mathcal{R}\,P$.

4b. $(M,N)\,\mathcal{R}\,(M,Q)$, se $N\,\mathcal{R}\,Q$.

As relações α, β e η transitam sem qualquer alteração para Λc. Para além destas, consideramos também as relações *fst*, *snd* e *SP*, referentes às projecções e ao par; definimo-las a seguir. A título de excepção notacional, definimos — seguindo (Curien, 1993) — a relação $\beta\eta SP$ como a união, não só das relações $\beta\eta$ e *SP*, como também das relações *fst* e *snd*.

Definição 2.18. Definem-se as seguintes relações binárias sobre Λc:

1. $\mathit{fst} = \{(\mathit{fst}((M,N)), M) \mid M, N \in \Lambda c\}$;

2. $\mathit{snd} = \{(\mathit{snd}((M,N)), N) \mid M, N \in \Lambda c\}$;

3. $SP = \{((\mathit{fst}(M), \mathit{snd}(M)), M) \mid M \in \Lambda c\}$;

4. $\beta\eta SP = \beta \cup \eta \cup \mathit{fst} \cup \mathit{snd} \cup SP$.

Observação 2.19. As relações *fst* e *snd* descrevem o significado subjacente às projecções. A relação *SP* (do inglês "Surjective Pairing") porém, é mais complexa: afirma que todos os termos podem ser vistos como pares. Esta relação é importante para a correspondência entre o Cálculo λc tipificado e as categorias cartesianas fechadas, cf. 2.15. Mais concretamente, *SP* corresponde à equação SPair (ver (2.1)) que, recorde-se, está contida na conversão forte, de acordo com 2.7.

Observação 2.20 ((Curien, 1993, p. 30))**.** A propósito, é bem conhecida uma codificação de termos-λc em termos-λ: veja-se a tradução $\cdot_\lambda : \Lambda c \to \Lambda$ definida por:

1. $x_\lambda = x$;

2. $\mathit{fst}(M)_\lambda = M_\lambda(\lambda xy.x)$;

3. $\mathit{snd}(M)_\lambda = M_\lambda(\lambda xy.y)$;

4. $(M, N)_\lambda = \lambda x.xM_\lambda N_\lambda$, onde $x \notin FV(MN)$;
5. $(MN)_\lambda = M_\lambda N_\lambda$;
6. $(\lambda xM)_\lambda = \lambda x.M_\lambda$.

Porém, sabe-se (Barendregt, 1974) que não é possível simular a relação SP no Cálculo λ, isto é, não existe uma tradução $\cdot_\lambda$ que verifique

$$M =_{\beta\eta SP} N \implies M_\lambda =_{\beta\eta} N_\lambda.$$

Por esta razão, legitima-se a introdução do Cálculo λc.

2.2.2 Versão categorial do Cálculo λc de de Bruijn

Nesta subsecção transitamos do Cálculo λc para a sua variante com índices de de Bruijn. Porém, não definimos um novo Cálculo Λc^{DB}, à semelhança do que é feito em 1.2 com respeito ao Cálculo λ original; em vez disso, refazemos a tradução $\cdot_{DB}$ com uma assinatura $\Lambda c \to \mathcal{CCL}$ e codificamos a operação de substituição à custa da conversão forte. Com efeito, implementamos o Cálculo λc de de Bruijn na Lógica Combinatória Categorial.

Observação 2.21. À partida, não é óbvio como podemos codificar um índice de de Bruijn através de termos categoriais. A ideia-chave por detrás desta subsecção é a de que podemos representar uma substituição simultânea $[i \coloneqq N_i]_0^n$ através da composição e do emparelhamento, como

$$M[i \coloneqq N_i]_0^n \rightsquigarrow M \circ \langle A, N_n, N_{n-1}, \ldots, N_0 \rangle,$$

onde A é um termo a ser definido. Intuitivamente, o emparelhamento representa o *ambiente* de M, donde um índice de de Bruijn é codificado pela composição de projecções que acedem ao seu valor. Na subsecção seguinte veremos mais detalhes a este respeito.

Começamos por introduzir alguma notação auxiliar.

Definição 2.22 ((Curien, 1993, Definition 1.2.11))**.** Para todo $n \in \mathbb{N}_0$ e $A, B \in \mathcal{CCL}$, definem-se os meta-termos $n!$ e $S(A, B)$ do seguinte modo:

- $n! = Snd \circ Fst^n$, onde $Fst^{i+1} = Fst \circ Fst^i$ com $i \in \mathbb{N}_0$.
- $S(A, B) = App \circ \langle A, B \rangle$.

Observação 2.23. O índice j num meta-termo $j!$ indica o total combinadores *Fst*. Por exemplo,

- $0! = Snd$;
- $1! = Snd \circ Fst$;
- $2! = Snd \circ (Fst \circ Fst)$;
- $3! = Snd \circ (Fst \circ (Fst \circ Fst))$.

Definição 2.24 ((Curien, 1993, Definition 1.2.15))**.** Para todo $M \in \Lambda c$ e para todo $x_0, \ldots, x_n \in \mathcal{V}$ tal que $FV(M) \subseteq \{x_0, \ldots, x_n\}$, define-se $M_{DB(x_0,\ldots,x_n)}$ (abreviadamente M_{DB}) recursivamente por:

1. $x_{DB} = i!$, onde $i = \min\{j \mid x = x_j\}$;
2. $fst(M)_{DB} = Fst \circ M_{DB}$;
3. $snd(M)_{DB} = Snd \circ M_{DB}$;
4. $(M, N)_{DB} = \langle M_{DB}, N_{DB} \rangle$;
5. $(MN)_{DB} = S(M_{DB}, N_{DB})$;
6. $(\lambda x M)_{DB(x_0,\ldots,x_n)} = \Lambda(M_{DB(x,x_0,\ldots,x_n)})$.

Exemplo 2.25. Seja $M = \lambda x.xy$. A única variável livre de M é y pelo que basta considerar o conjunto $\{y\}$ de variáveis. Tem-se[6]

$$
\begin{array}{ll}
 & (\lambda x.xy)_{DB(y)} \\
= & \quad 2.24 \\
 & \Lambda(S(x_{DB(x,y)}, y_{DB(x,y)})) \\
= & \quad 2.24 \\
 & \Lambda(S(0!, 1!))
\end{array}
$$

[6]Confrontar com o resultado pela sua tradução original: (λ.01).

Pretendemos agora refazer a substituição de de Bruijn na Lógica Combinatória Categorial. Para começar, a sua codificação deve ser tal que permita obter $S(\Lambda(P), Q) =_{CCL\beta\eta SP} P[0 \coloneqq Q]$, para todo $P, Q \in \mathcal{CCL}$. Ora,

$$S(\Lambda(P), Q) \overset{2.22}{=} App \circ \langle \Lambda(P), Q \rangle \overset{\text{Beta}}{=} P \circ \langle Id, Q \rangle,$$

o que sugere a codificação $P \circ \langle Id, Q \rangle$ para $P[0 \coloneqq Q]$. Por outro lado, também deve permitir obter $\Lambda(P)[0 \coloneqq Q] =_{CCL\beta\eta SP} \Lambda(P[1 \coloneqq Q])$. Ora,

$$\Lambda(P) \circ \langle Id, Q \rangle \overset{\text{D}\Lambda}{=} \Lambda(P \circ \langle \langle Id, Q \rangle \circ Fst, Snd \rangle)$$

o que, por sua vez, sugere a codificação $\langle \langle Id, Q \rangle \circ Fst, Snd \rangle$ para $[1 \coloneqq Q]$. Repetindo-se este exercício, deduzimos uma codificação para $[m \coloneqq Q]$, com $m \in \mathbb{N}_0$. De forma igual, podemos também deduzir uma codificação para as funções de actualização de índices $U_i^m(\cdot)$; formalizamos estas duas codificações nas definições seguintes.

Definição 2.26. Para todo $n \in \mathbb{N}_0$ e para todo $A \in \mathcal{CCL}$, define-se a família dos meta-termos $P^n(A)$ recursivamente por:

1. $P^0(A) = A$;
2. $P^{n+1}(A) = \langle P^n(A) \circ Fst, Snd \rangle$.

Definição 2.27. Sejam $A, B \in \mathcal{CCL}$ e $i, m \in \mathbb{N}_0$. As operações de substituição e de actualização de índices são codificadas como:

- $A[m \coloneqq B] \mapsto A \circ P^m(\langle Id, B \rangle)$;
- $U_i^m(B) \mapsto B \circ P^i(Fst^m)$.

Demonstramos agora que a codificação 2.27 reproduz, de facto, as operações originais. O seguinte lema é chave neste processo.

Lema 2.28. *Para todo $A \in \mathcal{CCL}$ e $m, n \in \mathbb{N}$ tal que $1 \leqslant m \leqslant n$, verifica-se:*

$$Fst^m \circ P^n(A) =_{CCL\beta\eta SP} P^{n-m}(A) \circ Fst^m.$$

Demonstração. Por indução sobre m.

- Caso $m = 1$.

$$
\begin{array}{ll}
 & Fst \circ P^n(A) \\
= & \quad 2.26 \\
 & Fst \circ \langle P^{n-1}(A) \circ Fst, Snd \rangle \\
= & \quad \text{Fst} \\
 & P^{n-1}(A) \circ Fst
\end{array}
$$

- Caso $m + 1$.

$$
\begin{array}{ll}
 & Fst^{m+1} \circ P^n(A) \\
= & \quad 2.22, \text{ Ass} \\
 & Fst \circ (Fst^m \circ P^n(A)) \\
= & \quad \text{H.I.} \\
 & Fst \circ (P^{n-m}(A) \circ Fst^m) \\
= & \quad \text{Ass} \\
 & (Fst \circ P^{n-m}(A)) \circ Fst^m \\
= & \quad 2.26 \\
 & (Fst \circ (\langle P^{n-m-1}(A) \circ Fst, Snd \rangle)) \circ Fst^m \\
= & \quad \text{Fst, Ass, Aritmética} \\
 & P^{n-(m+1)}(A) \circ Fst^{m+1}
\end{array}
$$

□

Confronte-se o seguinte resultado com 1.33 e com a definição de substuição sobre Λc implicitamente definida em 2.2.1.

Proposição 2.29. *Para todo $A, B, C \in \mathcal{CCL}$ e para todo $i, m, n \in \mathbb{N}_0$, verifica-se:*

1. $n![m \coloneqq C] =_{CCL\beta\eta SP} \begin{cases} n! & se\ n < m, \\ U_0^n(C) & se\ n = m, \\ (n-1)! & se\ n > m; \end{cases}$
2. $S(A, B)[m \coloneqq C] =_{CCL\beta\eta SP} S(A[m \coloneqq C], B[m \coloneqq C])$;
3. $\Lambda(A)[m \coloneqq C] =_{CCL\beta\eta SP} \Lambda(A[m + 1 \coloneqq C])$;

4. $(Fst \circ A)[m \coloneqq C] =_{CCL\beta\eta SP} Fst \circ A[m \coloneqq C]$;

5. $(Snd \circ A)[m \coloneqq C] =_{CCL\beta\eta SP} Snd \circ A[m \coloneqq C]$;

6. $\langle A, B\rangle[m \coloneqq C] =_{CCL\beta\eta SP} \langle A[m \coloneqq C], B[m \coloneqq C]\rangle$;

7. $U_i^m(n!) =_{CCL\beta\eta SP} \begin{cases} n! & se\ n < i, \\ (n+m)! & se\ n \geqslant i; \end{cases}$

8. $U_i^m(S(A, B)) =_{CCL\beta\eta SP} S(U_i^m(A), U_i^m(B))$;

9. $U_i^m(\Lambda(A)) =_{CCL\beta\eta SP} \Lambda(U_{i+1}^m(A))$;

10. $U_i^m(Fst \circ A) =_{CCL\beta\eta SP} Fst \circ U_i^m(A)$;

11. $U_i^m(Snd \circ A) =_{CCL\beta\eta SP} Snd \circ U_i^m(A)$;

12. $U_i^m(\langle A, B\rangle) =_{CCL\beta\eta SP} \langle U_i^m(A), U_i^m(B)\rangle$.

Demonstração.

- $U_i^m(n!) = n!$, se $n < i$.

$$
\begin{array}{ll}
 & U_i^m(n!) \\
= & \quad \text{2.22, 2.27, 2.28} \\
 & Snd \circ (P^{i-n}(Fst^m) \circ Fst^n) \\
= & \quad \text{2.26} \\
 & Snd \circ (\langle P^{i-n-1}(Fst^m) \circ Fst, Snd\rangle \circ Fst^m) \\
= & \quad \text{Ass, Snd, 2.22} \\
 & n!
\end{array}
$$

- $U_i^m(n!) = (n+m)!$, se $n \geqslant i$.

$$
\begin{array}{ll}
 & U_i^m(n!) \\
= & \quad \text{2.27, 2.22, Ass} \\
 & (Snd \circ Fst^{n-i}) \circ (Fst^i \circ P^i(Fst^m)) \\
= & \quad \text{2.28, Aritmética} \\
 & (Snd \circ Fst^{n-i}) \circ Fst^{m+i} \\
= & \quad \text{Ass, 2.22, Aritmética} \\
 & (n+m)!
\end{array}
$$

- $U_i^m(S(A,B)) = S(U_i^m(A), U_i^m(B))$.

$$
\begin{array}{ll}
 & U_i^m(S(A,B)) \\
= & \quad \text{2.22, 2.27, Ass} \\
 & App \circ (\langle A, B\rangle \circ P^i(Fst^m)) \\
= & \quad \text{DPair, 2.27} \\
 & App \circ \langle U_i^m(A), U_i^m(B)\rangle \\
= & \quad \text{2.22} \\
 & S(U_i^m(A), U_i^m(B))
\end{array}
$$

- $U_i^m(\Lambda(A)) = \Lambda(U_{i+1}^m(A))$.

$$
\begin{array}{ll}
 & U_i^m(\Lambda(A)) \\
= & \quad \text{2.27} \\
 & \Lambda(A) \circ P^i(Fst^m) \\
= & \quad \text{D}\Lambda \\
 & \Lambda(A \circ \langle P^i(Fst^m) \circ Fst, Snd\rangle) \\
= & \quad \text{2.26, 2.27} \\
 & \Lambda(U_{i+1}^m(A))
\end{array}
$$

- $S(A,B)[m \coloneqq C] = S(A[m \coloneqq C], B[m \coloneqq C])$. Análogo ao caso $U_i^m(S(A,B))$.
- $\Lambda(A)[m \coloneqq C] = \Lambda(A[m+1 \coloneqq C])$. Análogo ao caso $U_i^m(\Lambda(A))$.
- $n![m \coloneqq C] = n!$, se $n < m$. Procedemos tal e qual como no caso $U_i^m(n!)$ quando $n < i$.
- $n![m \coloneqq C] = U_0^m(C)$, se $n = m$.

$$
\begin{array}{ll}
 & n![m \coloneqq C] \\
= & \quad \text{2.22, 2.27, 2.28} \\
 & Snd \circ (\langle Id, C\rangle \circ Fst^m) \\
= & \quad \text{Ass, Snd} \\
 & C \circ Fst^m \\
= & \quad \text{2.27, 2.26} \\
 & U_0^m(C)
\end{array}
$$

- $n![m \coloneqq C] = (n-1)!$, se $n > m$.

$$
\begin{array}{ll}
 & n![m \coloneqq C] \\
= & \quad \text{2.22, 2.27, Ass} \\
 & (Snd \circ Fst^{n-m}) \circ (Fst^m \circ P^m(\langle Id, C\rangle)) \\
= & \quad \text{2.28, 2.26} \\
 & (Snd \circ Fst^{n-m}) \circ (\langle Id, C\rangle \circ Fst^m) \\
= & \quad \text{Ass, Fst, IdR} \\
 & Snd \circ (Fst^{n-m-1} \circ Fst^m) \\
= & \quad \text{Ass, Aritmética, 2.22} \\
 & (n-1)!
\end{array}
$$

- $(Fst \circ A)[m \coloneqq C] =_{CCL\beta\eta SP} Fst \circ A[m \coloneqq C]$. Imediato, por Ass.
- $\langle A, B\rangle[m \coloneqq C] =_{CCL\beta\eta SP} \langle A[m \coloneqq C], B[m \coloneqq C]\rangle$. Imediato, por DPair.

Os restantes casos são análogos. □

A proposição 2.29 mostra que a codificação das operações de substituição e de actualização de índices imita fielmente as operações originais. Sabemos do Cálculo Λ^{DB}, mais particularmente, da proposição 1.34, que estas operações bastam para se reproduzir a relação β. Logo, face ao resultado 2.29, podemos concluir que a relação β também é reproduzível na Lógica Combinatória Categorial (Forte) (Curien, 1993, Implícito na demonstração 1.2.17). Para além da relação β, demonstramos em seguida que a Lógica Combinatória Categorial Forte também implementa (via a tradução $\cdot_{DB}$) as relações η, *fst*, *snd* e *SP*.

Lema 2.30. *Para todo $M \in \Lambda c$, para todo $n \in \mathbb{N}_0$, para todo $x_0, \ldots, x_n \in \mathcal{V}$ variáveis distintas (notação: $\vec{x}$) tais que $FV(M) \subseteq \vec{x}$, para todo $i \in \{0, \ldots, n\}$ tal que $x_i \notin FV(M)$, verifica-se:*

$$M_{DB(\vec{x})} =_{CCL\beta\eta SP} M_{DB(x_0,\ldots,x_{i-1},x_{i+1},\ldots,x_n)} \circ P^i(Fst).$$

Demonstração. Por indução sobre M. Para tornar a apresentação ligeiramente mais leve, vamos escrever $\vec{x'}$ no lugar de $x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$.

- Caso $M = x$. De $FV(x) \subseteq \vec{x}$ e de $x_i \notin FV(x)$ vem que $x_i \neq x$. Logo, uma vez que as variáveis são todas distintas, $x_{DB(\vec{x})} \neq i!$. Surgem então dois casos: o caso em que x ocorre na sublista $x_0, \ldots, x_{i-1}$ e o caso em que x ocorre na sublista $x_{i+1}, \ldots, x_n$. Assim, denote-se o valor de $x_{DB(\vec{x})}$ por

  - $j!$, se $j < i$;
  - $(i + l)!$, se $l > 0$.

  Posto isto, quando $x_{DB(\vec{x})}$ vale

  - $j!$, o valor de $x_{DB(\vec{x'})}$ é também $j!$ porque a sublista $x_0, \ldots, x_{i-1}$ na qual x ocorre mantém-se inalterada com a remoção de x_i;
  - $(i + l)!$, o valor de $x_{DB(\vec{x'})}$ é $(i + l - 1)!$ porque x ocorre na sublista $x_i, x_{i+1}, \ldots, x_n$ que, ao excluirmos x_i, faz recuar uma posição todos termos consecutivos a este.

  Pretendemos assim mostrar:

  1. $j! =_{CCL\beta\eta SP} j! \circ P^i(Fst)$;
  2. $(i + l)! =_{CCL\beta\eta SP} (i + l - 1)! \circ P^i(Fst)$.

  Ora,

  1.
  $$\begin{array}{ll} & j! \circ P^i(Fst) \\ = & \quad \text{2.22} \\ & (Snd \circ Fst^j) \circ P^i(Fst) \\ = & \quad \text{Ass, 2.28} \\ & Snd \circ (P^{i-j}(Fst) \circ Fst^j) \\ = & \quad \text{2.26, Ass, Snd} \\ & Snd \circ Fst^j \\ = & \quad \text{2.22} \\ & j! \end{array}$$

  2.
  $$\begin{array}{ll} & (i + l - 1)! \circ P^i(Fst) \\ = & \quad \text{2.22, Aritmética, Ass} \\ & (Snd \circ Fst^l) \circ (Fst^{i-1} \circ P^i(Fst)) \\ = & \quad \text{2.28} \\ & (Snd \circ Fst^l) \circ (P^{i-(i-1)}(Fst) \circ Fst^{i-1}) \\ = & \quad \text{2.26, Aritmética} \\ & (Snd \circ Fst^l) \circ (\langle Fst \circ Fst, Snd \rangle \circ Fst^{i-1}) \\ = & \quad \text{Ass, Fst} \\ & Snd \circ (Fst^{l+1} \circ Fst^{i-1}) \\ = & \quad \text{2.22, Aritmética} \\ & (i + l)! \end{array}$$

- Caso $M = (\lambda y P)$. Tem-se

$$
\begin{array}{ll}
 & (\lambda y P)_{DB(\vec{x'})} \circ P^i(Fst) \\
= & \quad 2.24 \\
 & \Lambda(P_{DB(y,\vec{x'})}) \circ P^i(Fst) \\
= & \quad \mathrm{D}\Lambda,\ 2.27 \\
 & \Lambda(P_{DB(y,\vec{x'})} \circ P^{i+1}(Fst)) \\
= & \quad \text{H.I.} \\
 & \Lambda(P_{DB(y,\vec{x})}) \\
= & \quad 2.24 \\
 & (\lambda y P)_{DB(\vec{x})}
\end{array}
$$

□

Nota bibliográfica 2.31. O lema 2.30 acabado de demonstrar é apresentado como exercício na sua instância $i = 0$ em (Curien, 1986). Em (Curien, 1993, Exercise 1.2.15.1) é apresentada uma versão diferente, porventura igual ao lema 2.30, mas cuja apresentação não deixa claro o enunciado. Tornando ainda mais confusas as coisas, Curien infere — não explicando como — a instância $i = 0$ a partir da sua demonstração da proposição seguinte, cf. (Curien, 1993, Proof 1.2.20) ou (Curien, 1986, Proof 2.20).

Proposição 2.32. *Sejam $M, N \in \Lambda c$ e $x_0, \ldots, x_n \in \mathcal{V}$ (notação: $\vec{x}$) tais que $FV(MN) \subseteq \vec{x}$. Se $M \to_{\beta\eta SP} N$, então $M_{DB(\vec{x})} =_{CCL\beta\eta SP} N_{DB(\vec{x})}$.*

Demonstração. Por indução sobre a definição de $\to_{\beta\eta SP}$.

- Caso $fst((M, N)) \to_{\beta\eta SP} M$.

$$
\begin{array}{ll}
 & fst((M,N))_{DB} \\
= & \quad 2.24 \\
 & Fst \circ \langle M_{DB}, N_{DB} \rangle \\
= & \quad \text{Fst} \\
 & M_{DB}
\end{array}
$$

- Caso $snd((M, N)) \to_{\beta\eta SP} N$. Análogo ao anterior.

- Caso $(fst(M), snd(M)) \rightarrow_{\beta\eta SP} M$.

$$
\begin{array}{ll}
 & (fst(M), snd(M))_{DB} \\
= & \quad 2.24 \\
 & \langle Fst \circ M_{DB}, Snd \circ M_{DB} \rangle \\
= & \quad \text{SPair} \\
 & M_{DB}
\end{array}
$$

- Caso $(\lambda x M x) \rightarrow_{\beta\eta SP} M$, onde $x \notin FV(M)$.

$$
\begin{array}{ll}
 & (\lambda x M x)_{DB(\vec{x})} \\
= & \quad 2.24,\ 2.22 \\
 & \Lambda(App \circ \langle M_{DB(x,\vec{x})}, Snd \rangle) \\
= & \quad 2.30 \\
 & \Lambda(App \circ \langle M_{DB(\vec{x})} \circ Fst, Snd \rangle) \\
= & \quad \text{S}\Lambda \\
 & M_{DB(\vec{x})}
\end{array}
$$

- Caso $(\lambda x M)N \rightarrow_{\beta\eta SP} M[x \coloneqq N]$.

  Ora $((\lambda x M)N)_{DB} \overset{2.24}{=} S(\Lambda(M_{DB(x,\vec{x})}), N_{DB})$ o que, por 1.34 e por 2.29 conjuntamente, implica (Curien, 1993, Implícito na demonstração 1.2.17)

$$M[x \coloneqq N]_{DB} =_{CCL\beta\eta SP} M_{DB(x,\vec{x})}[0 \coloneqq N_{DB}].$$

  Ora,

$$
\begin{array}{ll}
 & S(\Lambda(M_{DB(x,\vec{x})}), N_{DB(\vec{x})}) \\
= & \quad 2.22,\ \text{Beta} \\
 & M_{DB(x,\vec{x})} \circ \langle Id, N_{DB(\vec{x})} \rangle \\
= & \quad 2.27 \\
 & M_{DB(x,\vec{x})}[0 \coloneqq N_{DB}]
\end{array}
$$

- Os restantes casos são rotineiros.

□

Nota bibliográfica 2.33. A proposição 2.32 é demonstrada em (Curien, 1993, Proposition 1.2.17) ou (Curien, 1986, Proposition 2.17). Contudo, a demonstração aqui apresentada diverge consideravelmente da original para o caso η. A demonstração de Curien não é clara porque utiliza notação não introduzida previamente, a saber, U_0^{+1}; para além disso, depende de uma generalização da operação de substituição que substitui ocorrências de termos em vez de ocorrências de variáveis. A demonstração aqui apresentada é mais simples, recorrendo naturalmente ao lema 2.30.

Encerramos esta secção com o corolário óbvio.

Corolário 2.33.1. *Se* $M =_{\beta\eta SP} N$ *então* $M_{DB(\vec{x})} =_{CCL\beta\eta SP} N_{DB(\vec{x})}$*, para todo* $M, N \in \Lambda c$ *e para todo* $\vec{x} \in \mathcal{V}$ *tal que* $FV(MN) \subseteq \vec{x}$*.*

2.2.3 Correspondência

Estamos finalmente aptos a estabelecer uma ponte entre o Cálculo λc com a conversão $\beta\eta SP$ e a Lógica Combinatória Categorial com a conversão forte. Falta-nos definir traduções entre os conjuntos Λc e $\mathcal{CCL}$ e mostrar que as conversões dos respectivos sistemas simulam-se mutuamente.

Definição 2.34 ((Curien, 1993, Definition 1.2.18))**.** Para todo $M \in \Lambda c$ e para todo $x_0, \ldots, x_n \in \mathcal{V}$ tal que $FV(M) \subseteq \{x_0, \ldots, x_n\}$, define-se $M_{\mathrm{CCL}} \in \mathcal{CCL}$ como

$$M_{\mathrm{CCL}} = M_{DB(x_0,\ldots,x_n)}.(x, x_n, \ldots, x_0),$$

onde x é uma variável distinta de $x_0, \ldots, x_n$.

Esta definição merece vários comentários. Atente-se nas seguintes observações.

Observação 2.35. A componente $(x, x_n, \ldots, x_0)$ representa, intuitivamente, o ambiente de M — uma estrutura na qual se encontram e se podem obter todas as variáveis livres de M. O acesso à variável x_i é conseguido de acordo com a conversão forte pelos termos categoriais da forma $i!$ do termo M_{DB}.

Observação 2.36. O valor concreto de x no ambiente $(x, x_n, \ldots, x_0)$ não é relevante. Este objecto serve para:

1. Quando M é fechado, se tenha $M_{\mathrm{CCL}} = M_{DB}.x$, i.e. permita a construção da aplicação;

2. Quando M tem exactamente uma variável livre, digamos x_0, se tenha $M_{\mathrm{CCL}} = M_{DB}.(x, x_0)$, i.e. permita a construção do par. Nesta circunstância não podemos fazer simplesmente $M_{DB}.x_0$ uma vez que o acesso a x_0 está codificado em M_{DB} pelo combinador categorial $0! = \mathit{Snd}$.

Exemplo 2.37. Considere-se o termo $M = (\lambda x.xy)z$. As variáves livres de M são y, z pelo que basta considerar o conjunto $\{y, z\}$ de variáveis. Tem-se

$$M_{\mathrm{CCL}} = S(\Lambda(S(0!, 1!)), 1!).(x, z, y) \quad \text{ou} \quad M_{\mathrm{CCL}} = S(\Lambda(S(0!, 2!)), 0!).(x, y, z)$$

conforme se considere a sequência yz ou zy de variáveis, respectivamente.

Observação 2.38. A escolha da sequência de variáveis livres é indiferente modulo a conversão forte (Curien, 1993, Exercise 1.2.18.1). Para ajudar a ilustrar, observe-se como as duas traduções de M em 2.37 são iguais pela conversão forte.

$$\begin{array}{ll} & S(\Lambda(S(0!, 1!)), 1!).(x, z, y) \\ = & \quad \text{2.22} \\ & (\mathit{App} \circ \langle \Lambda(\mathit{App} \circ \langle 0!, 1! \rangle), 1! \rangle).(x, z, y) \\ = & \quad \text{Beta} \\ & ((\mathit{App} \circ \langle 0!, 1! \rangle) \circ \langle \mathit{Id}, 1! \rangle).(x, z, y) \\ = & \quad \text{ass, dpair} \\ & (\mathit{App} \circ \langle 0!, 1! \rangle).(\mathit{Id}.(x, z, y), 1!.(x, z, y)) \\ = & \quad \text{id, 2.22, fst, snd} \\ & \mathit{App} \circ \langle 0!, 1! \rangle.(x, z, y, z) \\ = & \quad \text{ass, dpair} \\ & \mathit{App}.(0!(x, z, y, z), 1!(x, z, y, z)) \\ = & \quad \text{app, 2.22, fst, snd} \\ & z.y \end{array}$$

$$\begin{array}{ll} & S(\Lambda(S(0!, 2!)), 0!).(x, y, z) \\ = & \quad \text{2.22, Beta} \\ & ((\mathit{App} \circ \langle 0!, 2! \rangle) \circ \langle \mathit{Id}, 0! \rangle).(x, y, z) \\ = & \quad \text{ass, dpair, id, 2.22, snd} \end{array}$$

$$
\begin{array}{ll}
 & (App \circ \langle 0!, 2! \rangle).(x, y, z, z) \\
= & \qquad \text{ass, dpair, app, 2.22, fst, snd} \\
 & z.y
\end{array}
$$

Este exemplo também ajuda a evidenciar que o valor concreto da componente x não é preciso durante a simplificação de M_{CCL}, sendo por isso irrelevante neste contexto.

A tradução no sentido $\mathcal{CCL} \to \Lambda c$ segue um formato mais familiar, consistindo numa definição recursiva de cada termo categorial em termos-λ.

Definição 2.39. Para todo $A \in \mathcal{CCL}$, define-se $A_{\lambda c} \in \Lambda c$ recursivamente por:

1. $x_{\lambda c} = x$;
2. $Id_{\lambda c} = \lambda x x$;
3. $Fst_{\lambda c} = \lambda x.fst(x)$;
4. $Snd_{\lambda c} = \lambda x.snd(x)$;
5. $App_{\lambda c} = \lambda x.fst(x)snd(x)$;
6. $(A \circ B)_{\lambda c} = \lambda x.A_{\lambda c}(B_{\lambda c}x)$, onde $x \notin V(A) \cup V(B)$;
7. $\langle A, B \rangle_{\lambda c} = \lambda x.(A_{\lambda c}x, B_{\lambda c}x)$, onde $x \notin V(A) \cup V(B)$;
8. $\Lambda(A)_{\lambda c} = \lambda xy.A_{\lambda c}(x, y)$, onde $x, y \notin V(A)$.

Observação 2.40. Para todo $A \in \mathcal{CCL}$, $V(A) = FV(A_{\lambda c})$.

O seguinte lema mostra que a codificação dos meta-termos par e aplicação tem a correspondência esperada do lado do Cálculo λc.

Lema 2.41 ((Curien, 1993, Lemma 1.2.19))**.** *Para todo $A, B \in \mathcal{CCL}$, temos:*

1. $(A.B)_{\lambda c} =_{\beta\eta SP} A_{\lambda c}B_{\lambda c}$;
2. $(A, B)_{\lambda c} =_{\beta\eta SP} (A_{\lambda c}, B_{\lambda c})$.

Demonstração. Vejamos primeiro alguns resultados auxiliares:

i.

$$
\begin{array}{ll}
 & (A^{>})_{\lambda c} \\
= & \quad 2.9,\ 2.39\ \beta \\
 & \lambda xy.A_{\lambda c}(snd(x,y)) \\
= & \quad snd,\ \eta \\
 & \lambda x.A_{\lambda c}
\end{array}
$$

ii.

$$
\begin{array}{ll}
 & (A^{<})_{\lambda c} \\
= & \quad 2.39,\ 2.9,\ \beta \\
 & \lambda x.(\lambda y.fst(y)snd(y))(A_{\lambda c}x,x) \\
= & \quad \beta,\ fst,\ snd \\
 & \lambda x.(A_{\lambda c}x)x
\end{array}
$$

Temos, então:

$$
\begin{array}{ll}
 & (A.B)_{\lambda c} \\
= & \quad 2.9 \\
 & ((A \circ B^{>})^{<})_{\lambda c} \\
= & \quad 2.39,\ \text{i., ii.} \\
 & \lambda x.((\lambda y.A_{\lambda c}((\lambda z.B_{\lambda c})y))x)x \\
= & \quad \beta \\
 & \lambda x.(A_{\lambda c}B_{\lambda c})x \\
= & \quad \eta \\
 & A_{\lambda c}B_{\lambda c}
\end{array}
$$

$$
\begin{array}{ll}
 & (A,B)_{\lambda c} \\
= & \quad 2.9 \\
 & (\langle A^{>}, B^{>}\rangle^{<})_{\lambda c} \\
= & \quad 2.39,\ \text{i., ii.} \\
 & \lambda x.((\lambda y.((\lambda z.A_{\lambda c})y,(\lambda z.B_{\lambda c})y))x)x \\
= & \quad \beta \\
 & \lambda x.(A_{\lambda c},B_{\lambda c})x \\
= & \quad \eta \\
 & (A_{\lambda c},B_{\lambda c})
\end{array}
$$

□

Corolário 2.41.1. *Seja $M \in \Lambda c$ e seja $M_{CCL} = M_{DB}.(x, x_n, \ldots, x_0)$ para algum $x, x_0, \ldots, x_n \in \mathcal{V}$. Então*

$$(M_{CCL})_{\lambda c} =_{\beta\eta SP} (M_{DB})_{\lambda c}(x, x_n, \ldots, x_0).$$

Demonstração. Imediata a partir de 2.34, 2.41 e 2.39. □

Vejamos agora que as traduções $\cdot_{\lambda c}$ e $\cdot_{\mathrm{CCL}}$ são inversas uma da outra.

Lema 2.42 ((Curien, 1993, Exercise 1.2.15.1))**.** *Para todo $M \in \Lambda c$, para todo $n \in \mathbb{N}_0$, para todo $x_0, \ldots, x_n \in \mathcal{V}$ variáveis distintas (notação: $\vec{x}$) tais que $FV(M) \subseteq \vec{x}$, para todo $y \notin \vec{x}$, para todo $i \in \{0, \ldots, n\}$:*

$$M_{DB(\vec{x})} = M[x_i \coloneqq y]_{DB(x_0,\ldots,x_{i-1},y,x_{i+1},\ldots,x_n)}.$$

Demonstração. Por indução sobre M. Para tornar mais leve a apresentação, escrevamos $_{DB(x_0,\ldots,y,\ldots,x_n)}$ subentendendo-se que y ocorre na i-ésima posição.

- Caso $M = x$. Surgem dois casos em virtude do resultado de $x[x_i \coloneqq y]$: o caso em que $x = x_i$, onde o resultado é y; e o caso em que $x \neq x_i$, onde o resultado é x.

  - Caso $x = x_i$. Tem-se $y_{DB(x_0,\ldots,y\ldots,x_n)} = i!$ e $x_{DB(\vec{x})} = i!$, uma vez que as variáveis são todas distintas.
  - Caso $x \neq x_i$. Então $x_{DB(x_0,\ldots,y,\ldots,x_n)} = j!$, com $j \neq i$. Como $x = x_j$, $j \neq i$ e as variáveis são todas distintas, tem-se que $x[x_i \coloneqq y]_{DB(x_0,\ldots,y,\ldots,x_n)} = x_{DB(x_0,\ldots,y,\ldots,x_n)} = j!$.

- Caso $M = (\lambda x P)$. Sem perda de generalidade, podemos assumir que $x \notin \vec{x} \cup \{y\}$. Assim tem-se (por 2.24)

  - $(\lambda x P)_{DB(\vec{x})} = \Lambda(P_{DB(x,\vec{x})})$;
  - $(\lambda x P)[x_i \coloneqq y]_{DB(x_0,\ldots,y,\ldots,x_n)} = \Lambda(P[x_i \coloneqq y]_{DB(x,x_0,\ldots,y,\ldots,x_n)})$.

  Resta por isso ver:

  $$P_{DB(x,\vec{x})} = P[x_i \coloneqq y]_{DB(x,x_0,\ldots,y,\ldots,x_n)}.$$

  Ora, a nossa H.I. é: Para todo $m \in \mathbb{N}_0$, para todo $x'_0, \ldots, x'_m \in \mathcal{V}$ variáveis distintas (notação: $\vec{x'}$), para todo $z \in \mathcal{V}$ tal que $z \notin \vec{x'}$, para todo $j \in \{0, \ldots, m\}$:

  $$P_{DB(\vec{x'})} = P[x'_j \coloneqq z]_{DB(x'_0,\ldots,z,\ldots,x'_m)}.$$

  Basta assim aplicar-se a H.I. considerando-se $m \coloneqq n + 1$, $j \coloneqq i + 1$, $z \coloneqq y$, e $x'_{k+1} \coloneqq x_k$, para todo $k = 0, \ldots, n$.

- Os restantes casos são rotineiros.

□

Teorema 2.43 ((Curien, 1993, Theorem 1.2.20 (3))). *Para todo $M \in \Lambda c$, $(M_{CCL})_{\lambda c} =_{\beta\eta SP} M$.*

Demonstração. Por indução sobre M.

- Caso $M = x$.

$$
\begin{array}{ll}
 & (x_{\mathrm{CCL}})_{\lambda c} \\
= & \quad 2.41.1 \\
 & (x_{DB})_{\lambda c}(y, x_n, \ldots, x) \\
= & \quad 2.24,\ 2.22,\ 2.39 \\
 & (\lambda x.snd(x))(y, x_n, \ldots, x) \\
= & \quad \beta,\ snd \\
 & x
\end{array}
$$

- Caso $M = (PQ)$.

$$
\begin{array}{ll}
 & ((PQ)_{\mathrm{CCL}})_{\lambda c} \\
= & \quad 2.41.1 \\
 & ((PQ)_{DB})_{\lambda c}(x, x_n, \ldots, x_0) \\
= & \quad 2.24,\ 2.39 \\
 & (\lambda x.(\lambda x.fst(x)snd(x))(\lambda x.((P_{DB})_{\lambda c}x, (Q_{DB})_{\lambda c}x)))(x, x_n, \ldots, x_0) \\
= & \quad \beta \\
 & ((P_{DB})_{\lambda c}(x, x_n, \ldots, x_0))((Q_{DB})_{\lambda c}(x, x_n, \ldots, x_0)) \\
= & \quad 2.41.1,\ \text{H.I.} \\
 & PQ
\end{array}
$$

- Caso $M = (\lambda x P)$.

$$
\begin{array}{ll}
 & ((\lambda x P)_{\mathrm{CCL}})_{\lambda c} \\
= & \quad 2.41.1 \\
 & (\lambda x P)_{DB}(y, x_n, \ldots, x_0) \\
= & \quad 2.24 \\
 & \Lambda(P_{DB(x,x_0,\ldots,x_n)})_{\lambda c}(y, x_n, \ldots, x_0) \\
= & \quad 2.39 \\
 & (\lambda uv.(P_{DB(x,x_0,\ldots,x_n)})_{\lambda c}(u, v))(y, x_n, \ldots, x_0) \\
= & \quad \beta
\end{array}
$$

$\lambda v.P(_{DB(x,x_0,\ldots,x_n)})_{\lambda c}(y, x_n, \ldots, x_0, v)$
$=$ 2.42
$\lambda v.(P[x \coloneqq v]_{DB(v,x_0,\ldots,x_n)})_{\lambda c}(y, x_n, \ldots, x_0, v)$
$=$ 2.34, 2.41.1
$\lambda v.((P[x \coloneqq v])_{\mathrm{CCL}})_{\lambda c}$
$=$ H.I.
$\lambda v.P[x \coloneqq v]$
$=$ α
$\lambda x P$

- Caso $M = \mathit{fst}(P)$.

$(\mathit{fst}(P)_{\mathrm{CCL}})_{\lambda c}$
$=$ 2.41.1
$(\mathit{fst}(P)_{DB})_{\lambda c}(y, x_n, \ldots, x_0)$
$=$ 2.39
$(\lambda x.(\lambda x.\mathit{fst}(x))((P_{DB})_{\lambda c}x))(y, x_n, \ldots, x_0)$
$=$ β
$\mathit{fst}((P_{DB})_{\lambda c}(y, x_n, \ldots, x_0))$
$=$ 2.34, 2.41.1
$\mathit{fst}((P_{\mathrm{CCL}})_{\lambda c})$
$=$ H.I.
$\mathit{fst}(P)$

- Caso $M = \mathit{snd}(P)$. Análogo ao caso anterior.
- Caso $M = (P, Q)$.

$((P, Q)_{\mathrm{CCL}})_{\lambda c}$
$=$ 2.41.1, 2.24
$\langle P_{DB}, Q_{DB}\rangle_{\lambda c}(x, x_n, \ldots, x_0)$
$=$ 2.39, β
$((P_{DB})_{\lambda c}(x, x_n, \ldots, x_0), (Q_{DB})_{\lambda c}(x, x_n, \ldots, x_0))$
$=$ 2.41.1, H.I.
(P, Q)

□

Teorema 2.44 ((Curien, 1993, Theorem 1.2.20 (4))). *Para todo* $A \in \mathcal{CCL}$, $(A_{\lambda c})_{CCL} =_{CCL\beta\eta SP} A$.

Demonstração. Por indução sobre A.

- Caso $A = x$.

$$\begin{array}{ll} & (x_{\lambda c})_{\mathrm{CCL}} \\ = & \quad \text{2.39, 2.34, 2.24} \\ & Snd.(y, x_n, \ldots, x) \\ = & \quad \text{snd} \\ & x \end{array}$$

- Caso $A = Fst$.

$$\begin{array}{ll} & (Fst_{\lambda c})_{\mathrm{CCL}} \\ = & \quad \text{2.39, 2.34} \\ & \Lambda(Fst \circ Snd).(x, \ldots, x_0) \\ = & \quad \text{Quote3, Ass, Snd, IdR} \\ & Fst \end{array}$$

- Caso $A = Snd$. Análogo ao caso Fst.

- Caso $A = Id$.

$$\begin{array}{ll} & (Id_{\lambda c})_{\mathrm{CCL}} \\ = & \quad \text{2.39, 2.34} \\ & \Lambda(Snd).(x, \ldots, x_0) \\ = & \quad \text{Quote3, Snd} \\ & Id \end{array}$$

- Caso $A = App$.

$$\begin{array}{ll} & (App_{\lambda c})_{\mathrm{CCL}} \\ = & \quad \text{2.39, 2.34} \\ & \Lambda(App \circ \langle Fst \circ Snd, Snd \circ Snd \rangle).(x, \ldots, x_0) \\ = & \quad \text{Quote3, Ass, DPair, Snd, FSI, IdR} \\ & App \end{array}$$

- Caso $A = (B \circ C)$.

$$\begin{array}{ll} & ((B \circ C)_{\lambda c})_{\mathrm{CCL}} \\ = & \quad \text{2.39} \\ & (\lambda x.B_{\lambda c}(C_{\lambda c}x))_{\mathrm{CCL}} \\ = & \quad \text{2.34 denotando } (y, x_n, \ldots, x_0) \text{ por } u \text{ apenas} \\ & (\lambda x.B_{\lambda c}(C_{\lambda c}x))_{DB}.u \\ = & \quad \text{2.24} \\ & \Lambda(App \circ \langle (B_{\lambda c})_{DB(x,\vec{x})}, App \circ \langle (C_{\lambda c})_{DB(x,\vec{x})}, Snd \rangle \rangle).u \\ = & \quad \text{2.30} \\ & \Lambda(App \circ \langle (B_{\lambda c})_{DB} \circ Fst, App \circ \langle (C_{\lambda c})_{DB} \circ Fst, Snd \rangle \rangle).u \\ = & \quad \text{Quote3} \end{array}$$

$(App \circ \langle (B_{\lambda c})_{DB} \circ Fst, App \langle (C_{\lambda c})_{DB} \circ Fst, Snd \rangle \rangle) \circ \langle \Lambda(Fst).u, Id \rangle$
$=$ Ass, DPair, Fst, Snd
$App \circ \langle (B_{\lambda c})_{DB} \circ \Lambda(Fst).u, App \circ \langle (C_{\lambda c})_{DB} \circ \Lambda(Fst).u, Id \rangle \rangle$
$=$ Quote2
$(B_{\lambda c})_{DB}.u \circ App \langle (C_{\lambda c})_{DB} \circ \Lambda(Fst).u, Id \rangle$
$=$ Quote2, IdR, H.I.
$B \circ C$

- Caso $A = \langle B, C \rangle$.

$(\langle B, C \rangle_{\lambda c})_{\mathrm{CCL}}$
$=$ 2.39; 2.34 denotando $(y, \ldots, x_0)$ por u apenas; 2.24; 2.30
$\Lambda(\langle App \circ \langle (B_{\lambda c})_{DB} \circ Fst, Snd \rangle, App \circ \langle (C_{\lambda c})_{DB} \circ Fst, Snd \rangle \rangle).u$
$=$ Quote3, Ass, DPair, Fst, Snd, Quote2, IdR, H.I. — tal como no caso $A = (B \circ C)$
$\langle A, B \rangle$

- Caso $A = \Lambda(B)$.

$(\Lambda(B)_{\lambda c})_{\mathrm{CCL}}$
$=$ 2.39
$(\lambda zw.B_{\lambda c}(z, w))_{\mathrm{CCL}}$
$=$ 2.34 denotando $(y, \ldots, x_0)$ por u apenas; 2.30
$\Lambda(\Lambda(App \circ \langle (B_{\lambda c})_{DB} \circ (Fst \circ Fst), \langle Snd \circ Fst, Snd \rangle \rangle)).u$
$=$ Quote3
$\Lambda(App \circ \langle (B_{\lambda c})_{DB} \circ (Fst \circ Fst), \langle Snd \circ Fst, Snd \rangle \rangle) \circ \langle \Lambda(Fst).u, Id \rangle$
$=$ DΛ
$\Lambda((App \circ (B_{\lambda c})_{DB} \circ (Fst \circ Fst), \langle Snd \circ Fst, Snd \rangle \rangle) \circ \langle \langle \Lambda(Fst).u, Id \rangle \circ Fst, Snd \rangle)$
$=$ DPair, IdL
$\Lambda((App \circ \langle (B_{\lambda c})_{DB} \circ (Fst \circ Fst), \langle Snd \circ Fst, Snd \rangle \rangle) \circ \langle \langle \Lambda(Fst).u \circ Fst, Fst \rangle, Snd \rangle)$
$=$ Ass, DPair, Fst
$\Lambda(App \circ \langle (B_{\lambda c})_{DB} \circ (\Lambda(Fst).u \circ Fst), \langle Fst, Snd \rangle \rangle)$
$=$ FSI, Quote1
$\Lambda(App \circ \langle (B_{\lambda c})_{DB} \circ \Lambda(Fst).u, Id \rangle)$
$=$ Quote2, IdR
$\Lambda((B_{\lambda c})_{DB}.u)$
$=$ H.I.
$\Lambda(B)$

□

Vejamos agora que as conversões se simulam uma à outra através destas traduções.

Proposição 2.45 ((Curien, 1993, Theorem 1.2.20 (2))). *Se* $A =_{CCL\beta\eta SP} B$, *então* $A_{\lambda c} =_{\beta\eta SP} B_{\lambda c}$, *para todo* $A, B \in \mathcal{CCL}$.

Demonstração. Vamos demonstrar que $A \rightarrow_{CCL\beta\eta SP} B$ implica $A_{\lambda c} =_{\beta\eta SP} B_{\lambda c}$, donde o resultado pretendido sai por corolário. Procedemos por indução sobre $A \rightarrow_{\beta\eta SP} B$.

- Caso $(A \circ B) \circ C \rightarrow_{CCL\beta\eta SP} A \circ (B \circ C)$.

$$\begin{array}{ll} & ((A \circ B) \circ C)_{\lambda c} \\ = & \quad 2.39,\ \beta \\ & \lambda x.A_{\lambda c}(B_{\lambda c}(C_{\lambda c}x)) \\ = & \quad 2.39,\ \beta \\ & (A \circ (B \circ C))_{\lambda c} \end{array}$$

- Caso $Id \circ A \rightarrow_{CCL\beta\eta SP} A$.

$$\begin{array}{ll} & (Id \circ A)_{\lambda c} \\ = & \quad 2.39 \\ & \lambda x.(\lambda yy)(Ax) \\ = & \quad \beta,\ \eta \\ & A \end{array}$$

- Caso $A \circ Id \rightarrow_{CCL\beta\eta SP} A$.

$$\begin{array}{ll} & (A \circ Id)_{\lambda c} \\ = & \quad 2.39 \\ & \lambda x.A((\lambda yy)x) \\ = & \quad \beta,\ \eta \\ & A \end{array}$$

- Caso $Fst \circ \langle A, B \rangle \rightarrow_{CCL\beta\eta SP} A$.

$$\begin{array}{ll} & (Fst \circ \langle A, B \rangle)_{\lambda c} \\ = & \quad 2.39,\ \beta \\ & \lambda x.(\lambda y.fst(y))(Ax, Bx) \\ = & \quad \beta,\ \eta,\ fst \\ & A \end{array}$$

- Caso $Snd \circ \langle A, B\rangle \rightarrow_{CCL\beta\eta SP} B$. Análogo ao caso anterior.
- Caso $\langle A, B\rangle \circ C \rightarrow_{CCL\beta\eta SP} \langle A \circ C, B \circ C\rangle$.

$$
\begin{array}{ll}
 & (\langle A, B\rangle \circ C)_{\lambda c} \\
= & \quad 2.39,\beta \\
 & \lambda x.(\lambda y.(A_{\lambda c}y, B_{\lambda c}y))(C_{\lambda c}x) \\
= & \quad \beta \\
 & \lambda x.((\lambda y.A_{\lambda c}(C_{\lambda c}y))x, (\lambda y.B_{\lambda c}(Cy))x) \\
= & \quad 2.39,\beta \\
 & \langle A \circ C, B \circ C\rangle_{\lambda c}
\end{array}
$$

- Caso $App \circ \langle \Lambda(A), B\rangle \rightarrow_{CCL\beta\eta SP} A \circ \langle Id, B\rangle$.

$$
\begin{array}{ll}
 & (App \circ \langle \Lambda(A), B\rangle)_{\lambda c} \\
= & \quad 2.39,\ \beta \\
 & \lambda x.(\lambda y.fst(y)snd(y))(\lambda z.A_{\lambda c}(x, z), B_{\lambda c}x) \\
= & \quad \beta,\ fst,\ snd \\
 & \lambda x.(\lambda z.A_{\lambda c}(x, z))(B_{\lambda c}x) \\
= & \quad \beta \\
 & \lambda x.A_{\lambda c}(x, B_{\lambda c}x) \\
= & \quad \beta,\ 2.39 \\
 & (A \circ \langle Id, B\rangle)_{\lambda c}
\end{array}
$$

- Caso $\Lambda(A) \circ B \rightarrow_{CCL\beta\eta SP} \Lambda(A \circ \langle B \circ Fst, Snd\rangle)$.

$$
\begin{array}{ll}
 & (\Lambda(A) \circ B)_{\lambda c} \\
= & \quad 2.39,\ \beta \\
 & \lambda xy.A_{\lambda c}(B_{\lambda c}x, y) \\
= & \quad fst,\ snd \\
 & \lambda xy.A_{\lambda c}(B_{\lambda c}(fst(x, y)), snd(x, y)) \\
= & \quad \beta,\ 2.39 \\
 & \Lambda(A \circ \langle B \circ Fst, Snd\rangle)_{\lambda c}
\end{array}
$$

- Caso $\langle Fst, Snd\rangle \rightarrow_{CCL\beta\eta SP} Id$.

$$\begin{array}{ll} & \langle Fst, Snd\rangle_{\lambda c} \\ = & \quad 2.39,\ \beta \\ & \lambda x.(fst(x), snd(x)) \\ = & \quad SP,\ 2.39 \\ & Id_{\lambda c} \end{array}$$

- Caso $\Lambda(App) \rightarrow_{CCL\beta\eta SP} Id$.

$$\begin{array}{ll} & (\Lambda(App))_{\lambda c} \\ = & \quad 2.39,\ \beta \\ & \lambda xy.xy \\ = & \quad \eta \\ & \lambda xx \\ = & \quad 2.39 \\ & Id_{\lambda c} \end{array}$$

Os restantes casos (condições de compatibilidade) são rotineiros. □

Proposição 2.46 ((Curien, 1993, Theorem 1.2.20 (1))). *Se $M =_{\beta\eta SP} N$, então $M_{CCL} =_{CCL\beta\eta SP} N_{CCL}$, para todo $M, N \in \Lambda c$.*

Demonstração. Suponhamos que $M =_{\beta\eta SP} N$. Então $M_{DB} =_{CCL\beta\eta SP} N_{DB}$ por 2.33.1. Daqui segue que

$$M_{DB}.E =_{CCL\beta\eta SP} N_{DB}.E, \text{ para todo } E \in \mathcal{CCL}.$$

Basta (por 2.41.1) então tomarmos $E = (x, x_n, \ldots, x_0)$, onde $x_0, \ldots, x_n$ são as variáveis utilizadas na tradução $\cdot_{DB}$ e x é uma variável diferente de $x_0, \ldots, x_n$. □

Teorema 2.47 (Correspondência). *Para todo $M, N \in \Lambda c$ e $A, B \in \mathcal{CCL}$, verifica-se:*

1. *$M =_{\beta\eta SP} N$ se e só se $M_{CCL} =_{CCL\beta\eta SP} N_{CCL}$;*

2. *$A =_{CCL\beta\eta SP} B$ se e só se $A_{\lambda c} =_{\beta\eta SP} B_{\lambda c}$.*

Demonstração. As implicações "só se" são dadas pelas proposições 2.45 e 2.46. As implicações "se" são corolários:

1.

$$
\begin{array}{ll}
& M_{\mathrm{CCL}} =_{CCL\beta\eta SP} N_{\mathrm{CCL}} \\
\Longrightarrow & \quad 2.45 \\
& (M_{\mathrm{CCL}})_{\lambda c} =_{\beta\eta SP} (N_{\mathrm{CCL}})_{\lambda c} \\
\Longleftrightarrow & \quad 2.43 \\
& M =_{\beta\eta SP} N
\end{array}
$$

2.

$$
\begin{array}{ll}
& A_{\lambda c} =_{\beta\eta SP} B_{\lambda c} \\
\Longrightarrow & \quad 2.46 \\
& (A_{\lambda c})_{\mathrm{CCL}} =_{CCL\beta\eta SP} (B_{\lambda c})_{\mathrm{CCL}} \\
\Longleftrightarrow & \quad 2.44 \\
& A =_{CCL\beta\eta SP} B
\end{array}
$$

□

Confrontem-se os teoremas 2.47, 2.44 e 2.43 que relacionam a Lógica Combinatória Categorial e o Cálculo λc com as alíneas do teorema 1.48 que relacionam a Lógica Combinatória com o Cálculo λ.

2.3 Notas

Outra correspondência

Ao longo da última secção apresentámos uma correspondência envolvendo a conversão $\beta\eta SP$ do Cálculo λ. Substituindo a condição FSI da conversão forte pela condição

(FSA) $App \circ \langle Fst, Snd \rangle = App$,

é possível obter-se uma correspondência semelhante com respeito à conversão $\beta\eta P$ (Curien, 1993, p. 45), onde $P = fst \cup snd$.

Conversão fraca munida de extensionalidade

Um corolário do teorema 2.47 é o de que a conversão fraca pode ser tornada equivalente à forte se munida das condições FSI e ext, em que

(ext) $A = B$, se $A.x = B.x$, onde $x \notin V(A) \cup V(B)$.

Ou seja, tem-se (para todo $A, B \in \mathcal{CCL}$):

$$CCL\beta\eta SP \vdash A = B \text{ se e só se } CCL, \text{FSI}, \text{ext} \vdash A = B.$$

A demonstração pode ser consultada em (Curien, 1993, Corollary 1.2.21).

Constantes

Facilmente podemos incorporar constantes nesta correspondência. Consideremos um conjunto *Const* de *constantes*, as quais denotamos pela meta-variável c, eventualmente etiquetada. Munimos os conjuntos $\mathcal{CCL}$ e Λc com este conjunto da mesma forma que os munimos com o conjunto das variáveis, $\mathcal{V}$. As constantes não introduzem novas noções de redução, pelo que basta revisitarmos as proposições 2.44 e 2.43 para dar por concluída esta extensão à correspondência. Defina-se, para todo $A \in \mathcal{CCL}$, $'(A) = \Lambda(Fst).A$. Defina-se, para todo $c \in Const$:

- $c_{\lambda c} = c$;
- $c_{DB} =' (c)$.

Verifiquemos agora as proposições 2.44 e 2.43. Tem-se:

$$\begin{array}{ll} & (c_{\lambda c})_{\text{CCL}} \\ = & \quad \text{2.39} \\ & c_{\text{CCL}} \\ = & \quad \text{2.34, denotando por } u \text{ o ambiente; def. } c_{DB} \\ & \Lambda(Fst).c.u \\ = & \quad \text{d}\Lambda \\ & Fst.(c, u) \\ = & \quad \text{fst} \\ & c \end{array}$$

$(c_{\mathrm{CCL}})_{\lambda c}$
$=$ 2.34, denotando por u o ambiente
$(\Lambda(Fst).c.u)_{\lambda c}$
$=$ 2.41.1; def. $c_{\lambda} c$
$\Lambda(Fst)_{\lambda c} cu$
$=$ 2.39
$(\lambda xy.(\lambda x.fst(x))(x,y))cu$
$=$ β, *fst*
$(\lambda xy.x)cu$
$=$ β
c

Conversão fraca categorial versus conversão fraca clássica

A conversão fraca da Lógica Combinatória Categorial é *ainda mais fraca* que a conversão fraca da Lógica Combinatória Clássica (Curien, 1993). Recorde-se (cf. 1.3) que a razão para a terminologia fraca tem a ver com o facto destas conversões não reduzirem dentro de abstracções — i.e. não admitem a regra (ξ). Que a conversão fraca categorial não reduz dentro de abstrações é evidente com o exemplo $M = \lambda x.(\lambda xx)y$; tem-se

$$M_{\mathrm{CCL}} = \Lambda(S(\Lambda(0!), 1!)).(x, x_n, \ldots, y),$$

que é desde logo uma forma normal, uma vez que o termo $\Lambda(S\ldots)$ precisa de ser aplicado a dois argumentos para se tornar um redex, o que não acontece em virtude da tradução $\cdot_{\mathrm{CCL}}$. Pela conversão fraca clássica tem-se

$$M_{\mathrm{CL}} = \mathsf{K}(\mathsf{I}y),$$

que é redutível em $(\mathsf{K}y)$. Com efeito, a conversão fraca clássica não reduz, em geral, dentro de abstrações; por exemplo, o termo $N = \lambda x.(\lambda xx)x$ traduz-se em $\mathsf{S}(\mathsf{KI})\mathsf{I}$ que é uma forma normal. Contudo, quando a variável da abstração não ocorre no redex, como é o caso do termo M, já reduz. Em (Çağman

and Hindley, 1998) os autores apresentam um estudo mais detalhado deste fenómeno, incluindo uma redução do Cálculo λ análoga à redução fraca clássica.

Capítulo 3

A Máquina Abstracta Categorial (CAM)

Neste capítulo colocamos em prática a correspondência entre o Cálculo λc e a Lógica Combinatória Categorial vista no capítulo anterior. O resultado é uma implementação de uma linguagem de programação funcional.

3.1 Motivação

Nesta secção vamos reproduzir a dedução[1] de uma máquina abstracta a partir da Lógica Combinatória Categorial.

Convenção 3.1. Graças à introdução de constantes no Cálculo λc e a na Lógica Combinatória Categorial (vide 2.3), iremos fixar a componente arbitrária x de um ambiente na tradução $\cdot_{\mathrm{CCL}}$ pela constante $()$ — leia-se *nil*. Assim sendo, se $FV(M) \subseteq \{x_0, \ldots, x_n\}$, escreveremos $M_{\mathrm{CCL}} = M_{DB}((), x_n, \ldots, x_0)$.

Em virtude do teorema 2.47 sabemos ser possível simular uma conversão $M =_{\beta\eta SP} N$ do Cálculo λc através da Lógica Combinatória Categorial. Vejamos o exemplo muito simples de $M = (\lambda x x)y =_{\beta\eta SP} y = N$. Tem-se:

$$
\begin{array}{ll}
 & ((\lambda x x)y)_{\mathrm{CCL}} \\
= & \qquad 2.34,\ 2.22
\end{array}
$$

[1] Encontrada em (Cousineau et al., 1987) ou (Curien, 1993).

$$
\begin{array}{ll}
 & App \circ \langle \Lambda(Snd), Snd \rangle.((), y) \\
= & \quad \text{ass, dpair} \\
 & App.(\Lambda(Snd).((), y), Snd.((), y)) \\
= & \quad \text{snd, app} \\
 & \Lambda(Snd).((), y).y \\
= & \quad \text{d}\Lambda\text{, snd} \\
 & y
\end{array}
$$

Pode-se observar que esta simulação é feita à custa de uma sequência de reduções pela conversão fraca. Em jeito de relato, a computação deste exemplo decorre da seguinte forma: começa por aplicar o termo M_{DB} ao ambiente $((), y)$; daqui, distribui o ambiente pelos termos constituintes de M_{DB}, $(\lambda xx)_{DB}$ e y_{DB}, onde é finalmente consumido de acordo com termos $\cdot!$ ou suspenso em termos $\Lambda(\cdot)$. Estamos interessados em tornar mecânicas simulações deste género.

As seguintes considerações são chave no processo de dedução de uma implementação para a conversão fraca:

1. O termo M_{DB} é constituído apenas pelas constantes *Fst*, *Snd* e *App*, eventualmente sob a alçada de emparelhamentos ($\langle \cdot, \cdot \rangle$), abstracções ($\Lambda(\cdot)$) ou composições ($\cdot \circ \cdot$). Note-se que a constante *Id* nunca surge, mercê da definição de $\cdot_{DB}$.

2. Atente-se na computação apresentada na secção anterior: observe-se como os redexes são quase todos da forma $A.v$, com A um combinador e v um valor, i.e. uma variável, constante ou par destas. A excepção à regra é o caso dos redexes de dΛ, que têm a forma $A.v_1.v_2$.

3. No seguimento da alínea anterior, observe-se que a seguinte equação:

   (ac) $App.(\Lambda(A).B, C) = A.(B, C)$

   é consequência das condições app e dΛ. Substituindo as condições app e dΛ pela condição ac (do inglês "apply closure") apenas, podemos repetir a computação anterior sem nunca nos cruzarmos com redexes da forma $A.v_1.v_2$.

[1] A terminologia deriva do facto destes termos serem formas normais.

Estas considerações parecem sugerir que basta implementar o seguinte conjunto de equações para obter uma implementação da conversão fraca:

(ass) $(A \circ B).v = A.(B.v)$;

(fst) $\mathit{Fst}.(v_1, v_2) = v_1$;

(snd) $\mathit{Snd}.(v_1, v_2) = v_2$;

(dpair) $\langle A, B\rangle.v = (A.v, B.v)$;

(ac) $\mathit{App}.(\Lambda(A).v_1, v_2) = A.(v_1, v_2)$;

onde A, B e C são combinadores categoriais e v, v_1 e v_2 são valores. Estas equações, por sua vez, designam um efeito único para cada um dos combinadores categoriais quando aplicados a um valor. A partir daqui, podemos deduzir uma máquina de ambiente (representado por um valor) cujas instruções são extraídas dos combinadores categoriais:

- O efeito de $A \circ B$ é o de avaliar primeiro B, produzindo um novo valor v_1, e de seguida avaliar $A.v_1$. Este efeito diz-nos que podemos representar $A \circ B$ através da sequência de instruções $B; A$.

- O efeito de *Fst* (resp. *Snd*) é o de aceder à primeira (resp. segunda) componente do ambiente.

- O efeito de $\langle A, B\rangle$ é o de executar A e B de forma independente e no fim juntar os seus resultados num par. Na prática, precisamos de decidir qual avaliar primeiro: A ou B. Suponhamos que escolhemos avaliar primeiro A e denotemos o ambiente por v. Vamos então avaliar $A.v$, o que produz um novo valor, digamos v_1. Falta-nos avaliar $B.v$, o que pressupõe que tenhamos guardado v de antemão. A avaliação de $B.v$ produz um novo valor, digamos v_2. Resta agora apenas construir o par (v_1, v_2) o que, por sua vez, pressupõe que tenhamos guardado v_1 de antemão.

A explicação do efeito de $\langle, \rangle$ é ligeiramente mais complexa. Porém, o jogo da guarda de valores que esta instrução acarreta é um problema comum em

Ciências da Computação, sendo tipicamente resolvido com o auxílio de uma pilha.

Aproveitando para recapitular, a máquina em dedução tem a seguinte estrutura até agora:

i. Um ambiente, que é um valor;

ii. Um código, que é uma sequência de instruções;

iii. Uma pilha de valores.

Os efeitos dos combinadores podem agora ser explicados com maior especificidade:

- O efeito de $\langle A, B \rangle$ é dado por:
  - "$\langle$" é a instrução que carrega o ambiente para a pilha, ou seja: faz uma cópia do ambiente;
  - "A", i.e. executa o corpo de instruções A;
  - "," é a instrução que troca o ambiente com o topo da pilha, ou seja: se o ambiente for v e o topo da pilha for v_1, o novo estado da máquina é dado por um ambiente v_1 e um topo da pilha v;
  - "B", i.e. executa o corpo de instruções B;
  - "$\rangle$" é a instrução que constrói um par à custa do ambiente e do valor no topo da pilha.
- A instrução *Fst* transforma um ambiente (v_1, v_2) em v_1, e analogamente para a instrução *Snd*.
- O efeito de $\Lambda(A)$ num ambiente v é $\Lambda(A).v$, ou seja, substitui o ambiente v por $\Lambda(A).v$.
- O efeito de *App* é o de transformar um ambiente $(\Lambda(A).v_1, v_2)$ em (v_1, v_2) e executar A.

3.2 Formalização

Formalizamos agora toda esta discussão preliminar através da definição de uma máquina abstracta, denominada a Máquina Abstracta Categorial.

Definição 3.2 (A Máquina Abstracta Categorial (CAM))**.** Uma *configuração* (ou *estado*) desta máquina é um trio ordenado[2] $\{T, C, S\}$, onde:

(Termo)	$T ::= () \mid v \mid (T, T) \mid C : T$
(Código)	$C ::= [] \mid I; C$
(Pilha)	$S ::= [] \mid T::S$
(Instrução)	$I ::= \texttt{fst} \mid \texttt{snd} \mid \texttt{cur}(C) \mid \texttt{push} \mid \texttt{swap} \mid \texttt{cons} \mid \texttt{app}$

A relação de transição entre estados da CAM, $\rightarrow_{\text{CAM}}$, é definida através da tabela 3.1, onde o símbolo @ denota a operação de concatenação.

Observação 3.3. As instruções `fst` e `snd` estão associadas aos combinadores *Fst* e *Snd*, respectivamente; a instrução `cur`$(\cdot)$ (do inglês "curry") está associada ao combinador $\Lambda(\cdot)$; as instruções `push`, `swap` e `cons` estão associadas ao combinador $\langle , \rangle$, respectivamente; a instrução `app` está associada ao combinador *App*.

Observação 3.4. Os termos da forma $C : s$ representam o efeito do combinador $\Lambda(\cdot)$. Por outras palavras, $C : s$ representa o valor $\Lambda(A).s$, se A estiver a ser representado pelo código C.

Exemplo 3.5. Vejamos o exemplo $M = (\lambda xx)y$ através desta máquina. O código de M é obtido através de M_{DB} de acordo com a discussão preliminar:

$App \circ \langle \Lambda(Snd), Snd \rangle$
$\rightsquigarrow$ Tornando composições em sequenciações.
$\langle \Lambda(Snd), Snd \rangle ; App$
$\rightsquigarrow$ Traduzindo os combinadores nas respectivas instruções.
`push`; `cur`(`snd`); `swap`; `snd`; `cons`; `app`

[2]Seguindo a notação de (Cousineau et al., 1987), (Curien, 1993).

Termo	**Código**	**Pilha**	**Termo**	**Código**	**Pilha**
(v_1, v_2)	$\texttt{fst}; C$	S	v_1	C	S
(v_1, v_2)	$\texttt{snd}; C$	S	v_2	C	S
v	$\texttt{cur}(C); C_1$	S	$C : v$	C_1	S
v	$\texttt{push}; C$	S	v	C	$v::S$
v_2	$\texttt{swap}; C$	$v_1::S$	v_1	C	$v_2::S$
v_2	$\texttt{cons}; C$	$v_1::S$	(v_1, v_2)	C	S
$(C : v_1, v_2)$	$\texttt{app}; C_1$	S	(v_1, v_2)	$C@C_1$	S

Tabela 3.1: A Máquina Abstracta Categorial.

O ambiente de M é $((), y)$ e, portanto, o estado inicial da máquina é

$$\{((), y), \texttt{push}; \texttt{cur}(\texttt{snd}); \texttt{swap}; \texttt{snd}; \texttt{cons}; \texttt{app}, []\}.$$

Calculemos:

$\{((), y), \texttt{push}; \texttt{cur}(\texttt{snd}); \texttt{swap}; \texttt{snd}; \texttt{cons}; \texttt{app}, []\}$
$\rightarrow_{\text{CAM}}$
$\{((), y), \texttt{cur}(\texttt{snd}); \texttt{swap}; \texttt{snd}; \texttt{cons}; \texttt{app}, ((), y)\}$
$\rightarrow_{\text{CAM}}$
$\{\texttt{snd} : ((), y), \texttt{swap}; \texttt{snd}; \texttt{cons}; \texttt{app}, ((), y)\}$
$\rightarrow_{\text{CAM}}$
$\{((), y), \texttt{snd}; \texttt{cons}; \texttt{app}, \texttt{snd} : ((), y)\}$
$\rightarrow_{\text{CAM}}$
$\{y, \texttt{cons}; \texttt{app}, \texttt{snd} : ((), y)\}$
$\rightarrow_{\text{CAM}}$
$\{(\texttt{snd} : ((), y), y), \texttt{app}, []\}$
$\rightarrow_{\text{CAM}}$
$\{((), y, y), \texttt{snd}, []\}$
$\rightarrow_{\text{CAM}}$
$\{y, [], []\}$

3.3 Extensões

O Cálculo λc é apenas o esqueleto de uma linguagem de programação funcional. Para exemplos mais realistas iremos considerar constantes, condicionais e

funções recursivas. Em cada caso, a) introduzimos termos categoriais que induzem instruções da CAM e b) equações categoriais que induzem transições da CAM.

Nota bibliográfica 3.6. O modo como iremos deduzir transições da CAM para as extensões a partir de equações categoriais é uma novidade face ao exposto em (Cousineau et al., 1987).

Definições de valores

Começamos por ver como implementar expressões da forma

$$\texttt{let}\ f = E\ \texttt{in}\ B.$$

Ora,

$\texttt{let}\ f = E\ \texttt{in}\ B$
$\rightsquigarrow$ Reescrevendo em termos-λ (Landin, 1964)
$(\lambda f.B)E$
$\rightsquigarrow$ Reescrevendo em termos categoriais
$App \circ \langle \Lambda(B), E \rangle$
$\rightsquigarrow$ Reescrevendo em instruções-máquina
$\texttt{push}; \texttt{cur}(B); \texttt{swap}; E; \texttt{cons}; \texttt{app}$

Percebe-se portanto que as expressões desta forma não requerem adaptações à CAM, donde não se consideram uma extensão.

Constantes

Recorde-se (cf. 2.3) que uma constante $c \in \mathit{Const}$ é codificada na Lógica Combinatória Categorial como $'(c)$, o que satisfaz a equação $'(c).A = c$, para todo $A \in \mathcal{CCL}$. Em termos da CAM, a equação reescreve-se simplesmente como $'(c).v = c$. Introduzimos por isso uma nova instrução, $\texttt{quote}(\cdot)$, com o efeito descrito na tabela 3.2.

Exemplo 3.7. Consideremos o termo $M = (\lambda x.x(4,3))+$. O código de M é obtido a partir de M_{DB} como:

Termo	Código	Pilha	Termo	Código	Pilha
v	$\texttt{quote}(c); C$	S	c	C	S

Tabela 3.2: Instrução $\texttt{quote}(\cdot)$.

$App \circ \langle \Lambda(App \circ \langle Snd, \langle '4, '3 \rangle \rangle), '+ \rangle$

$\rightsquigarrow$ Tornando composições em sequenciações.

$\langle \Lambda(\langle Snd, \langle '4, '3 \rangle \rangle; \texttt{app}), '+ \rangle; App$

$\rightsquigarrow$ Fazendo $C = \texttt{push};\texttt{snd};\texttt{swap};\texttt{push};\texttt{quote}(4);\texttt{swap};\texttt{quote}(3);\texttt{cons};\texttt{cons};\texttt{app}$.

$\texttt{push}; \texttt{cur}(C); \texttt{swap}; '+; \texttt{cons}; \texttt{app}$

Não simplificámos o termo $'+$ como $\texttt{quote}(+)$ para mostrarmos uma codificação sua alternativa: $\Lambda(+ \circ Snd)$. Observe-se que $\Lambda(+ \circ Snd)$ tem o mesmo efeito que $'+$:

$\Lambda(+ \circ Snd).v$

$=$ Quote3

$(+ \circ Snd) \circ \langle \Lambda(Fst).v, Id \rangle$

$=$ Ass, Snd, IdR

$+$

Iremos utilizar esta codificação alternativa para constantes representando operações. A razão para esta escolha será tornada clara no fima da execução deste exemplo. Calculemos (como M é fechado o seu ambiente é ()):

$\{(), \texttt{push}; \texttt{cur}(C); \texttt{swap}; \texttt{cur}(\texttt{snd}; +); \texttt{cons}; \texttt{app}, []\}$

$\rightarrow^2_{\text{CAM}}$

$\{C : (), \texttt{swap}; \texttt{cur}(\texttt{snd}; +); \texttt{cons}; \texttt{app}, ()\}$

$\rightarrow^2_{\text{CAM}}$

$\{(\texttt{snd}; +) : (), \texttt{cons}; \texttt{app}, C : ()\}$

$\rightarrow^2_{\text{CAM}}$

$\{((), (\texttt{snd}; +) : ()), C, []\}$

$\rightarrow_{\text{CAM}}$ Onde $\bar{+}$ denota $(\texttt{snd}; +) : ()$

$\{((), \bar{+}), \texttt{snd}; \texttt{swap}; \texttt{push}; \texttt{quote}(4); \texttt{swap}; \texttt{quote}(3); \texttt{cons}; \texttt{cons}; \texttt{app}, ((), \bar{+})\}$

$\rightarrow^2_{\text{CAM}}$

$\{((), \bar{+}), \texttt{push}; \texttt{quote}(4); \texttt{swap}; \texttt{quote}(3); \texttt{cons}; \texttt{cons}; \texttt{app}, \bar{+}\}$

$\rightarrow^2_{\text{CAM}}$

$\{4, \texttt{swap}; \texttt{quote}(3); \texttt{cons}; \texttt{cons}; \texttt{app}, ((), \bar{+})::\bar{+}\}$

$\rightarrow^3_{\text{CAM}}$

$\{(4,3), \texttt{cons}; \texttt{app}, \bar{+}\}$
$\rightarrow^2_{\text{CAM}}$ Restituindo $\bar{+} = (\texttt{snd}; +) : ()$
$\{((\texttt{snd}; +) : (), (4,3)), \texttt{app}, []\}$
$\rightarrow_{\text{CAM}}$
$\{((), (4,3)), \texttt{snd}; +, []\}$
$\rightarrow^2_{\text{CAM}}$ Assumindo que $\{(m,n), +; C, S\} \rightarrow_{\text{CAM}} \{m+n, C, S\}$
$\{7, [], []\}$

Note-se agora como se tivéssemos optado pela codificação de $'+$ por $\texttt{quote}(+)$, teríamos que ter assumido a regra $\{(+, (m,n)), \texttt{app}; C, S\} \rightarrow_{\text{CAM}} \{m+n, C, S\}$, em vez da mais elegante $\{(m,n), +; C, S\} \rightarrow_{\text{CAM}} \{m+n, C, S\}$.

Condicionais

Estamos agora interessados em simular através da CAM a execução de uma expressão da forma $\texttt{if}\ A\ \texttt{then}\ B\ \texttt{else}\ C$. Intuitivamente, queremos avaliar A e, em função do seu resultado *booleano* (i.e. *verdadeiro* ou *falso*), avaliar B ou C. Introduzimos por isso as constantes true e false. A expressão $\texttt{if}\ A\ \texttt{then}\ B\ \texttt{else}\ C$ aplicada a um ambiente v pode ser computada da seguinte forma:

- Avaliamos $A.v$, produzindo um novo valor v_1;
- Caso $v_1 = \mathsf{true}$, avaliamos $B.v$;
- Caso $v_1 = \mathsf{false}$, avaliamos $C.v$.

Para replicarmos este efeito introduzimos na Lógica Combinatória Categorial uma primitiva que permita escolher entre B e C de acordo com o ambiente actual. Consideremos uma primitiva *Branch* satisfazendo as seguintes condições:

(Branch-true) $\mathit{Branch}(B,C).(v, \mathsf{true}) = B.v$;

(Branch-false) $\mathit{Branch}(B,C).(v, \mathsf{false}) = C.v$.

A expressão $\texttt{if}\ A\ \texttt{then}\ B\ \texttt{else}\ C$ pode então ser traduzida para termos categoriais como $\mathit{Branch}(B,C) \circ \langle \mathit{Id}, A \rangle$. Falta especificar instruções que

Termo	Código	Pilha	Termo	Código	Pilha
v	$\mathtt{skip}; C$	S	v	C	S
(v, true)	$\mathtt{branch}(C_1, C_2); C$	S	v	$C_1 @ C$	S
(v, false)	$\mathtt{branch}(C_1, C_2); C$	S	v	$C_2 @ C$	S

Tabela 3.3: Instruções `skip` e $\mathtt{branch}(\cdot, \cdot)$.

implementem $\mathit{Branch}(\cdot, \cdot)$ e Id. Introduzimos assim as instruções $\mathtt{branch}(\cdot, \cdot)$ e `skip`, cujo efeito é imediatamente dedutível a partir das equações Id, Branch-true e Branch-false; ver tabela 3.3

Nota bibliográfica 3.8. Em (Cousineau et al., 1987) a instrução `branch` é ligeiramente diferente daquela aqui apresentada ao evitar a necessidade de construir o par (v, true) (ou (v, false)) uma vez que utiliza valores no topo da pilha directamente. Este atalho em utilizar a pilha não levanta, por ora, problemas, mas torna impossível a formalização desta transição na redução combinatória $\twoheadrightarrow_{\mathrm{C}}$ a ser vista no próximo capítulo. Fundamentalmente, a razão prende-se com o facto de nas equações categoriais não existir referência à pilha da CAM, donde seria impossível formalizar o efeito descrito em (Cousineau et al., 1987) através de equações categoriais.

Exemplo 3.9. Consideremos o exemplo trivial (mas mais complicado não acrescentaria nada) $M = \mathtt{if}\ 1 > 0\ \mathtt{then}\ 2\ \mathtt{else}\ 3$. Temos

$$
\begin{array}{ll}
 & M_{DB} \\
= & \quad \text{Considerando “}1 > 0\text{” como notação para “}> (1, 0)\text{”.} \\
 & \mathit{Branch}('2, '3) \circ \langle \mathit{Id}, \mathit{App} \circ \langle \Lambda(> \circ \mathit{Snd}), \langle '1, '0 \rangle \rangle \rangle \\
\rightsquigarrow & \quad \text{Reescrevendo composições em sequenciações.} \\
 & \langle \mathit{Id}, \langle \Lambda(\mathit{Snd}; >), \langle '1, '0 \rangle \rangle; \mathit{App} \rangle; \mathit{Branch}('2, '3) \\
\rightsquigarrow & \quad \text{Traduzindo os combinadores em instruções-máquina.} \\
 & \mathtt{push}; \mathtt{skip}; \mathtt{swap}; I; \mathtt{cons}; \mathtt{branch}(\mathtt{quote}(2), \mathtt{quote}(3)) \\
 & \quad \text{onde } I = \mathtt{push};\mathtt{cur}(\mathtt{snd};>);\mathtt{swap};\mathtt{push};\mathtt{quote}(1);\mathtt{swap};\mathtt{quote}(0);\mathtt{cons};\mathtt{cons};\mathtt{app}.
\end{array}
$$

Temos:

$$
\begin{array}{ll}
 & \{(), \mathtt{push}; \mathtt{skip}; \mathtt{swap}; I; \mathtt{cons}; \mathtt{branch}(\mathtt{quote}(2), \mathtt{quote}(3)), []\} \\
\rightarrow^{3}_{\mathrm{CAM}} & \quad \text{Note-se que } \mathtt{push}; \mathtt{skip}; \mathtt{swap} \text{ tem o mesmo efeito que } \mathtt{push} \text{ apenas.} \\
 & \{(), I; \mathtt{cons}; \mathtt{branch}(\mathtt{quote}(2), \mathtt{quote}(3)), ()\}
\end{array}
$$

$\rightarrow^{9}_{\text{CAM}}$ Executando todas menos a última instrução em I
$\{((\texttt{snd}; >) : (), (1, 0)), \texttt{app}; \texttt{cons}; \texttt{branch}(\texttt{quote}(2), \texttt{quote}(3)), ()\}$
$\rightarrow^{3}_{\text{CAM}}$ Assumindo que $\{(1, 0), >; C, S\} \rightarrow_{\text{CAM}} \{\texttt{true}, C, S\}$
$\{\texttt{true}, \texttt{cons}; \texttt{branch}(\texttt{quote}(2), \texttt{quote}(3)), ()\}$
$\rightarrow^{2}_{\text{CAM}}$
$\{(), \texttt{quote}(2), []\}$
$\rightarrow_{\text{CAM}}$
$\{2, [], []\}$

A instrução `skip` parece irrelevante e, com efeito, podemos removê-la do código antes de iniciarmos uma simulação. Esta "optimização" deve-se ao facto da CAM não manipular *referências* (vulgo *apontadores*) para instruções (posições no código). A sua introdução é apenas útil na fase de tradução de expressões funcionais em combinadores categoriais e de seguida em instruções-máquina, auxiliando na uniformização deste processo.

Observação 3.10. A título de curiosidade, a instrução `skip` tem aplicações em processadores reais (e.g. Intel x86, ARM, etc.), servindo tipicamente para forçar um alinhamento constante das instruções como parte de uma estratégia de optimização do desempenho destes dispositivos. Outro uso é o de reservar espaço na componente código para mais tarde ser reescrito com outras instruções.

A instrução `skip` também surge numa optimização relacionada com expressões `let`...`in`.... Considere-se a expressão `let` $f = E$ `in` B. Já vimos que `let` $f = E$ `in` $B \rightsquigarrow \texttt{push}; \texttt{cur}(B); \texttt{swap}; E; \texttt{cons}; \texttt{app}$. Não obstante, segundo a Lógica Combinatória Categorial tem-se:

$App \circ \langle \Lambda(B), E \rangle$
$=$ Beta
$B \circ \langle Id, E \rangle$
$\rightsquigarrow$ Reescrevendo em instruções-máquina
$\texttt{push}; \texttt{skip}; \texttt{swap}; E; \texttt{cons}; B$
$\rightsquigarrow$ `push`; `skip`; `swap` tem o mesmo efeito que `push`
$\texttt{push}; E; \texttt{cons}; B$

Função recursiva

A última extensão que iremos considerar é um mecanismo para a avaliação eficiente de funções recursivas, como por exemplo

$$\texttt{letrec}\ \mathit{fact}\ n = \texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n * \mathit{fact}(n-1)\ \texttt{in}\ \mathit{fact}\ 3.$$

Expressões da forma $\texttt{letrec}\ f\ n = E\ \texttt{in}\ B$ podem ser reescritas (Peyton Jones, 1987) como $\texttt{let}\ f = Y(\lambda fn.E)\ \texttt{in}\ B$, onde Y é um operador de ponto-fixo[3] — ver (1.1). Por seu turno, expressões da forma $\texttt{let}\ f = E\ \texttt{in}\ B$ podem ser reescritas como $(\lambda f.B)E$. Resumindo, queremos avaliar expressões da forma

$$\begin{array}{ll} & \texttt{letrec}\ f\ n = E\ \texttt{in}\ B \\ = & \\ & \texttt{let}\ f = Y(\lambda fn.E)\ \texttt{in}\ B \\ = & \\ & (\lambda f.B)(Y(\lambda fn.E)) \end{array}$$

Assim sendo, falta-nos apenas apresentar uma implementação para Y, onde não queremos utilizar termos-λ no seu lugar porque procuramos uma implementação sua eficiente. Ou seja, queremos olhar para Y como uma constante que satisfaz a condição[3] que o caracteriza.

Em termos categoriais, a equação $YM = M(YM)$ traduz-se (Cousineau et al., 1987) como $YM = \mathit{App} \circ \langle M, YM \rangle$. Introduzimos por isso um combinador *Fix* satisfazendo (para todo $A \in \mathcal{CCL}$):

(Fix) $\mathit{Fix}(A) = \mathit{App} \circ \langle A, \mathit{Fix}(A) \rangle$.

Através da introdução deste combinador obtemos:

$$\begin{array}{ll} & Y(\lambda fn.E) \\ \rightsquigarrow & \quad \text{Traduzindo em combinadores categoriais} \\ & \mathit{Fix}(\Lambda(\Lambda(E))) \\ = & \quad \text{Fix} \\ & \mathit{App} \circ \langle (\Lambda(\Lambda(E)), \mathit{Fix}(\Lambda(\Lambda(E))) \rangle \end{array}$$

[3] Isto é, verifica a condição: $YM =_\beta M(YM)$, para todo $M \in \Lambda$.

$$
\begin{array}{ll}
= & \quad \text{Beta} \\
& \Lambda(E) \circ \langle Id, Fix(\Lambda(\Lambda(E))) \rangle
\end{array}
$$

Queremos replicar este efeito através de instruções da CAM. Apliquemos por isso o termo $Fix(\Lambda(\Lambda(E)))$ a um valor, v, donde obtemos:

$$
\begin{array}{ll}
& Fix(\Lambda(\Lambda(E))).v \\
= & \quad \text{Fix, Beta} \\
& \Lambda(E) \circ \langle Id, Fix(\Lambda(\Lambda(E))) \rangle.v \\
= & \quad \text{ass, dpair, id} \\
& \Lambda(E).(v, Fix(\Lambda(\Lambda(E))).v)
\end{array}
$$

ou seja, $Fix(\Lambda(\Lambda(E))).v = \Lambda(E).(v, Fix(\Lambda(\Lambda(E))).v)$. Esta equação é problemática de traduzir porque do lado direito voltamos a colocar o lado esquerdo, como parte de um valor. A discussão que se segue gira em torno da resolução deste problema.

Escrevamos $FIX(E)$ no lugar de $Fix(\Lambda(\Lambda(E)))$, uma vez que apenas estamos interessados em utilizar o combinador Fix em abstracções "duplas" $Fix(\Lambda(\Lambda(\cdot)))$. Obtem-se assim a equação mais curta

$$FIX(E).v = \Lambda(E).(v, FIX(E).v).$$

Transitando esta equação para a CAM, resolvemos o problema da recorrência introduzindo um novo valor, $E!v$, para todo E termo e v valor:

$$FIX(E).v = \Lambda(E).(v, E!v)),$$

onde $E!v$ deve ser interpretado como o valor $\Lambda(E).(v, E!v))$. O problema da recorrência fica assim resolvido porque do lado esquerdo da equação temos um redex e do lado direito um valor. Introduzimos agora uma instrução `fix`$(\cdot)$, associada ao combinador $FIX(\cdot)$; ver tabela 3.4.

Note-se que $C_1!v$ deve ser interpretado como $C_1 : (v, C_1!v)$, o que explica a entrada referente à instrução `app` na tabela 3.4, que funciona analogamente à instrução `app` previamente vista.

Termo	**Código**	**Pilha**	**Termo**	**Código**	**Pilha**
v	$\texttt{fix}(C_1); C$	S	$C_1 : (v, C_1!v)$	C	S
$(C_1!v_1, v_2)$	$\texttt{app}; C$	S	$(v_1, C_1!v_1, v_2)$	$C_1@C$	S

Tabela 3.4: Instrução $\texttt{fix}(\cdot)$.

Exemplo 3.11. Considere-se a expressão:

$$\texttt{letrec}\ f\ n = \texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n * f(n-1)\ \texttt{in}\ f\ 1.$$

Tem-se:

$\texttt{letrec}\ f\ n = \texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n * f(n-1)\ \texttt{in}\ f\ 1$

$\rightsquigarrow$ Trocando `letrec` por `let`

$\texttt{let}\ f\ n = Y(\lambda f n.\texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n * f(n-1))\ \texttt{in}\ f\ 1$

$\rightsquigarrow$ Representando `let` por termos-λ

$(\lambda f.f1)(Y(\lambda f n.\texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ n * f(n-1)))$

$\rightsquigarrow$ Transitando para combinadores categoriais através da optimização para `let`

$(App \circ \langle 0!, '1\rangle) \circ \langle Id, FIX(F)\rangle$

$\rightsquigarrow$ Transitando para instruções-máquina

$\texttt{push}; \texttt{fix}(F); \texttt{cons}; \texttt{push}; \texttt{snd}; \texttt{swap}; \texttt{quote}(1); \texttt{cons}; \texttt{app}$

onde F é o código de $\texttt{if}\ n = 0\ \texttt{then}\ 1\ \texttt{else}\ G$, G o de $n * f(H)$ e H o de $n - 1$; ou seja:

F

$\rightsquigarrow$

$Branch('1, G) \circ \langle Id, = \circ \langle 0!, '0\rangle\rangle$

$\rightsquigarrow$

$\texttt{push}; \texttt{push}; \texttt{snd}; \texttt{swap}; \texttt{quote}(0); \texttt{cons}; =; \texttt{cons}; \texttt{branch}(\texttt{quote}(1), G)$

G

$\rightsquigarrow$

$\times \circ \langle 0!, App \circ \langle 1!, H\rangle$

$\rightsquigarrow$

$\texttt{push}; \texttt{snd}; \texttt{swap}; \texttt{push}; \texttt{fst}; \texttt{snd}; \texttt{swap}; H; \texttt{cons}; \texttt{app}; \texttt{cons}; \times$

$$
\begin{array}{l}
\quad H \\
\rightsquigarrow \\
\quad - \circ \langle 0!,' 1 \rangle \rangle \\
\rightsquigarrow \\
\quad \mathtt{push}; \mathtt{snd}; \mathtt{swap}; \mathtt{quote}(1); \mathtt{cons}; -
\end{array}
$$

Calculemos:

$$
\begin{array}{l}
\quad \{(), \mathtt{push}; \mathtt{fix}(F); \mathtt{cons}; \mathtt{push}; \mathtt{snd}; \mathtt{swap}; \mathtt{quote}(1); \mathtt{cons}; \mathtt{app}, []\} \\
\rightarrow^{8}_{\mathrm{CAM}} \\
\quad \{(F : ((), F!()), 1), \mathtt{app}, []\} \\
\rightarrow_{\mathrm{CAM}} \\
\quad \{((), F!(), 1), F, []\} \\
\rightarrow^{8}_{\mathrm{CAM}} \quad \text{Assumindo que } \{(1,0), =; C, S\} \rightarrow_{\mathrm{CAM}} \{\mathsf{false}, C, S\} \\
\quad \{((), F!(), 1, \mathsf{false}), \mathtt{branch}(\mathtt{quote}(1), G), []\} \\
\rightarrow_{\mathrm{CAM}} \\
\quad \{((), F!(), 1), G, []\} \\
\rightarrow^{7}_{\mathrm{CAM}} \\
\quad \{((), F!(), 1), H; \mathtt{cons}; \mathtt{app}; \mathtt{cons}; \times, F!()::1\} \\
\rightarrow^{7}_{\mathrm{CAM}} \quad \text{Assumindo que } \{(1,1), -; C, S\} \rightarrow_{\mathrm{CAM}} \{0, C, S\} \\
\quad \{(F!()), 0), \mathtt{app}; \mathtt{cons}; \times, 1\} \\
\rightarrow_{\mathrm{CAM}} \\
\quad \{((), F!(), 0), F; \mathtt{cons}; \times, 1\} \\
\rightarrow^{8}_{\mathrm{CAM}} \quad \text{Assumindo que } \{(0,0), =; C, S\} \rightarrow_{\mathrm{CAM}} \{\mathsf{true}, C, S\} \\
\quad \{((), F!(), 0, \mathsf{true}), \mathtt{branch}(\mathtt{quote}(1), G); \mathtt{cons}; \times, 1\} \\
\rightarrow^{2}_{\mathrm{CAM}} \\
\quad \{1, \mathtt{cons}; \times, 1\} \\
\rightarrow^{2}_{\mathrm{CAM}} \quad \text{Assumindo que } \{(1,1), \times; C, S\} \rightarrow_{\mathrm{CAM}} \{1, C, S\} \\
\quad \{1, [], []\}
\end{array}
$$

Nota bibliográfica 3.12. A instrução-máquina deduzida nesta subsecção é radicalmente diferente da instrução-máquina utilizada em (Cousineau et al., 1987) para a mesma extensão. A razão prende-se com o nosso objectivo de demonstrar a correcção desta extensão no capítulo seguinte. O problema é que Cousineau et al. utilizam técnicas de implementação, a saber, apontadores, difíceis de modelar aqui. A este respeito, observe-se a operação $[\cdot \leftarrow \cdot]$ utilizada (informalmente) em (Cousineau et al., 1987, Table 6), e note-se

como esta não se trata da operação de substituição típica de variáveis, mas antes uma substituição infinita de ocorrências de termos.

3.4 Recapitulação

As secções anteriores permitem-nos agora apresentar formalmente uma implementação de uma linguagem de programação funcional através da Máquina Abstracta Categorial. Consideremos uma linguagem de programação funcional $\mathcal{F}$, cujas expressões são dadas por:

(Cálculo λ com constantes)	$E ::= x \mid c \mid (E_1 E_2) \mid \texttt{fun}\ x.E$
(Par e projecções)	$\mid (E_1, E_2) \mid \texttt{fst}(E) \mid \texttt{snd}(E)$
(Condicional)	$\mid \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3$
(Definição de valores)	$\mid \texttt{let}\ f = E_1\ \texttt{in}\ E_2$
(Definição de funções recursivas)	$\mid \texttt{letrec}\ f\ x = E_1\ \texttt{in}\ E_2$

A compilação de expressões desta linguagem para código CAM retira-se da tradução $\cdot_{DB}$, desta vez não terminando em termos categoriais mas directamente na codificação em instruções destes. Sejam $E \in \mathcal{F}$ e $x_0, \ldots, x_n$ variáveis tais que $FV(E) \subseteq \{x_0, \ldots, x_n\}$ [4]. Seja $\rho = ((), x_n, \ldots, x_0)$ um par. Define-se o código $[\![E]\!]_\rho$ recursivamente por:

1. $[\![x]\!]_{((),x_n,\ldots,x_0)} = \underbrace{\texttt{fst}; \ldots; \texttt{fst}}_{i \text{ vezes}}; \texttt{snd}$ se $x = x_i$, com $0 \leqslant i \leqslant n$

2. $[\![c]\!]_\rho = \begin{cases} \texttt{cur}(\texttt{snd}; c) & \text{se } c \in \{+, \times, -, \div, >, <, =, \neq, \ldots\} \\ \texttt{quote}(c) & \text{caso contrário} \end{cases}$

[4] Tem-se:

- $FV(\texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3) = FV(E_1 E_2 E_3)$;
- $FV(\texttt{let}\ f = E\ \texttt{in}\ B) = FV((\lambda f.B)E) = FV(E) \cup FV(B) \backslash \{f\}$;
- $FV(\texttt{letrec}\ f\ x = E\ \texttt{in}\ B) = FV((\lambda f.B)(Y(\lambda f x.E))) = FV(B) \cup \{f\} \cup FV(\lambda f x.E)$. Recorde-se que Y é como se fosse uma constante.

Os restantes casos seguem directamente da correspondência entre expressões E e termos-λc.

Termo	**Código**	**Pilha**	**Termo**	**Código**	**Pilha**
(v_1, v_2)	$\texttt{fst}; C$	S	v_1	C	S
(v_1, v_2)	$\texttt{snd}; C$	S	v_2	C	S
v	$\texttt{cur}(C); C_1$	S	$C : v$	C_1	S
v	$\texttt{quote}(c); C$	S	c	C	S
v	$\texttt{push}; C$	S	v	C	$v::S$
v_2	$\texttt{swap}; C$	$v_1::S$	v_1	C	$v_2::S$
v_2	$\texttt{cons}; C$	$v_1::S$	(v_1, v_2)	C	S
(v, true)	$\texttt{branch}(C_1, C_2); C$	S	v	$C_1@C$	S
(v, false)	$\texttt{branch}(C_1, C_2); C$	S	v	$C_2@C$	S
v	$\texttt{fix}(C_1); C$	S	$C_1 : (v, C_1!v)$	C	S
$(C_1!v_1, v_2)$	$\texttt{app}; C$	S	$(v_1, C_1!v_1, v_2)$	$C_1@C$	S
$(C : v_1, v_2)$	$\texttt{app}; C_1$	S	(v_1, v_2)	$C@C_1$	S
(m, n)	$+; C$	S	$m + n$	C	S
(m, n)	$>; C$	S	$m > n$	C	S
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabela 3.5: A Máquina Abstracta Categorial estendida.

3. $[\![(E_1 E_2)]\!]_\rho = \texttt{push}; [\![E_1]\!]_\rho; \texttt{swap}; [\![E_2]\!]_\rho; \texttt{cons}; \texttt{app}$

4. $[\![\texttt{fun}\ x.E]\!]_\rho = \texttt{cur}([\![E]\!]_{(\rho,x)})$

5. $[\![(E_1, E_2)]\!]_\rho = \texttt{push}; [\![E_1]\!]_\rho; \texttt{swap}; [\![E_2]\!]_\rho; \texttt{cons}$

6. $[\![\texttt{fst}(E)]\!]_\rho = [\![E]\!]_\rho; \texttt{fst}$

7. $[\![\texttt{snd}(E)]\!]_\rho = [\![E]\!]_\rho; \texttt{snd}$

8. $[\![\texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3]\!]_\rho = \texttt{push}; [\![E_1]\!]_\rho; \texttt{cons}; \texttt{branch}([\![E_2]\!]_\rho, [\![E_3]\!]_\rho)$

9. $[\![\texttt{let}\ f =\ E_1\ \texttt{in}\ E_2]\!]_\rho = \texttt{push}; [\![E_1]\!]_\rho; \texttt{cons}; [\![E_2]\!]_{(\rho,f)}$

10. $[\![\texttt{letrec}\ f\ x = E_1\ \texttt{in}\ E_2]\!]_\rho = \texttt{push}; \texttt{fix}([\![E_1]\!]_{(\rho,f,x)}); \texttt{cons}; [\![E_2]\!]_{(\rho,f)}$

A CAM munida das instruções necessárias para executar estes códigos é dada na tabela 3.5. Os estados iniciais são da forma $\{\rho, [\![E]\!]_\rho, []\}$ e os finais $\{\rho, [], []\}$.

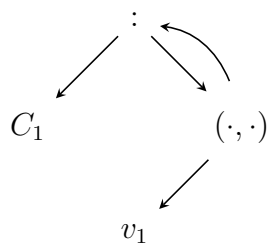


Figura 3.1: Representação em grafo de $C_1!v_1$.

Observação 3.13. Os termos da forma $\cdot!\cdot$ e as transições

$$\{(C_1!v_1, v_2), \texttt{app}; C, S\} \rightarrow_{\text{CAM}} \{(v_1, C_1!v_1, v_2), C_1@C, S\}$$

podem ser omitidas numa implementação real da CAM. A ideia é que $C_1!v_1$ deve ser representado como o grafo descrito pela figura 3.1, tirando proveito do mecanismo de referência/apontador. Esta implementação em grafo remonta à implementação da SECD (Landin, 1964).

Observação 3.14. Nas secções anteriores está presente uma pequena optimização em torno das constantes que não incluimos na definição de $[\![\cdot]\!]$. por uma questão de simplicidade. A optimização é dada pela seguinte equação:

3'. $[\![(cE_2)]\!]_\rho = [\![E_2]\!]_\rho; c$, se $c \in \{+, \times, -, \div, >, \dots\}$.

Esta optimização permite traduzir expressões como $+(1, 2)$ directamente em $\texttt{push}; \texttt{quote}(1); \texttt{swap}; \texttt{quote}(2); \texttt{cons}; +$, em vez da menos eficiente $\texttt{push}; \texttt{cur}(\texttt{snd};+);\texttt{swap};\texttt{push};\texttt{quote}(1);\texttt{swap};\texttt{quote}(2);\texttt{cons};\texttt{cons};\texttt{app}$.

No fim do próximo capítulo iremos argumentar a correcção desta implementação.

Capítulo 4

Correcção da CAM

Neste capítulo demonstramos que a CAM implementa correctamente a estratégia de redução *call-by-value* (CBV) do Cálculo λc, donde se conclui que a CAM pode ser utilizada na implementação de uma linguagem de programação funcional com semântica de avaliação estrita.

4.1 Estratégias de redução

O objectivo principal deste capítulo é demonstrar a comutatividade do diagrama (*da correcção da CAM*)

$$\begin{array}{ccc} M & \xrightarrow{\text{CBV}} & V \\ {\scriptstyle \mathsf{Load}}\downarrow & & \uparrow{\scriptstyle \mathsf{Unload}} \\ s & \xrightarrow{\mathsf{CAM}} & t \end{array} \tag{4.1}$$

para todo M termo-λc fechado, V valor (i.e. forma normal), s estado inicial e t estado final, onde a função Load associa um estado inicial a um termo-λc e Unload associa um valor a um estado final [1]. Deste resultado podemos concluir que a CAM implementa correctamente uma linguagem de programação funcional com semântica de avaliação estrita.

[1] Terminologia extraída de uma prova da correcção da SECD (Plotkin, 1975).

Observação 4.1. O pormenor a respeito de M ser fechado não é uma restrição, uma vez que todos os programas funcionais adequadamente codificados correspondem a termos-λc fechados, ou não seria possível compilá-los: observe-se que um compilador tradicional não é capaz de compilar variáveis não declaradas, i.e. livres.

Já existe na literatura (Cousineau et al., 1987) uma demonstração da correcção da CAM. Não obstante, esta encontra-se abreviada e focada na compilação de termos-λc puros apenas. A demonstração aqui apresentada segue o plano delineado por Cousineau et al., mas formaliza e demonstra todos os resultados necessários para a obtenção da correcção da CAM estendida com constantes, condicionais e funções recursivas.

O nosso plano é mostrar que a CAM implementa, em primeiro lugar, a estratégia de redução *innermost* da conversão fraca categorial, como exemplificada no capítulo anterior, e, em segundo lugar, a estratégia de redução *call-by-value* do Cálculo λc. Para este fim, Cousineau et al. apresentam uma teoria ad-hoc designada como *Cálculo de de Bruijn* $(\mathcal{DBC})$[2] na qual codificam ambas as reduções.

4.1.1 Termos de de Bruijn

Definição 4.2. O conjunto dos *termos de de Bruijn*, $\mathcal{DBC}$, é definido indutivamente por:

1. A *constante* $'(c)$ está em $\mathcal{DBC}$, para todo $c \in \mathit{Const}$;
2. A *variável* $i!$ está em $\mathcal{DBC}$, para todo $i \in \mathbb{N}_0$;
3. A *primeira projecção* $\mathit{fst}(M)$ está em $\mathcal{DBC}$, se M está $\mathcal{DBC}$;
4. A *segunda projecção* $\mathit{snd}(M)$ está em $\mathcal{DBC}$, se M está $\mathcal{DBC}$;
5. O *par* $\langle M, N\rangle$ está em $\mathcal{DBC}$, se M e N estão em $\mathcal{DBC}$;
6. A *aplicação* $S(M, N)$ está em $\mathcal{DBC}$, se M e N estão em $\mathcal{DBC}$;

[2] Do inglês "De Bruijn Calculus".

Λc^{DB}	$\mathcal{DBC}$	$\mathcal{CCL}$
c	$'(c)$	$'(c)$
n	$n!$	$n!$
$fst(M)$	$fst(M)$	$Fst \circ M$
$snd(M)$	$snd(M)$	$Snd \circ M$
(M, N)	$\langle M, N \rangle$	$\langle M, N \rangle$
(MN)	$S(M, N)$	$S(M, N)$
(λM)	$\Lambda(M)$	$\Lambda(M)$

Tabela 4.1: Relação entre termos de de Bruijn e termos-λc/categoriais.

7. A *abstracção* $\Lambda(M)$ está em $\mathcal{DBC}$, se M está em $\mathcal{DBC}$.

Os termos de de Bruijn tanto podem ser vistos como combinadores categoriais (os resultados da tradução $\cdot_{DB}$) como termos-λc em notação de de Bruijn; a tabela 4.1 sumaria esta observação. Posteriormente (secção 4.3) iremos dotar os termos de de Bruijn com condicionais e operadores de ponto-fixo.

Definição 4.3. Para todo $M \in \mathcal{DBC}$, define-se a *ordem de ambiente de* M como:

> a maior diferença, quando não-negativa, entre $n + 1$ e m, com n percorrendo todas as ocorrências de $n!$ de M e m o total de Λ's "acima" dessa ocorrência.

Quando a ordem de ambiente de um determinado M é 0, dizemos que é M é *fechado*.

Exemplo 4.4.

- O termo $0!$ tem ordem de ambiente $0 + 1 - 0 = 1$;
- O termo $\Lambda(0!)$ tem ordem de ambiente $0 + 1 - 1 = 0$;
- O termo $S(\Lambda(0!), 0!)$ tem ordem de ambiente 1, por consequência.

Iremos precisar da substituição simultânea (cf. 1.15) sobre termos de de Bruijn.

Definição 4.5. Sejam $N_0, \ldots, N_n \in \mathcal{DBC}$ termos de de Bruijn. Define-se, para todo $M \in \mathcal{DBC}$, a *substituição simultânea por* $N_0, \ldots, N_n$ *em* M, $M[N_0, N_1, \ldots, N_n]$ (abreviadamente $M[\vec{N}]$), recursivamente por:

1. $'(c)[\vec{N}] = '(c)$;
2. $i![N_0, \ldots, N_n] = \begin{cases} N_i & \text{se } 0 \leqslant i \leqslant n \\ i! & \text{caso contrário} \end{cases}$
3. $\mathit{fst}(M)[\vec{N}] = \mathit{fst}(M[\vec{N}])$;
4. $\mathit{snd}(M)[\vec{N}] = \mathit{snd}(M[\vec{N}])$;
5. $\langle P, Q\rangle[\vec{N}] = \langle P[\vec{N}], Q[\vec{N}]\rangle$;
6. $S(P, Q)[\vec{N}] = S(P[\vec{N}], Q[\vec{N}])$;
7. $\Lambda(M)[\vec{N}] = \Lambda(M[0!, \vec{N'}])$, onde $N'_i = N_i[1!, 2!, \ldots]$, para todo $0 \leqslant i \leqslant n$.

Observação 4.6 ((Cousineau et al., 1987, p. 189))**.** A definição 4.5 tem a complicação de, na alínea 7, depender de uma invocação sua infinita. Porém, isto não levantará dificuldades no decorrer da demonstração uma vez que apenas iremos considerar termos N_i fechados. Quando $N_0, \ldots, N_n$ são termos fechados, a equação 7 pode reescrever-se simplesmente como:

7. $\Lambda(M)[\vec{N}] = \Lambda(M[0!, \vec{N}])$,

o que resulta do facto $M[\vec{N}] = M$, para todo M fechado.

Lema 4.7. *Para todo* $n \in \mathbb{N}_0$*, para todo* $m \geqslant n + 1$*, para todo* $N_0, \ldots, N_m \in \mathcal{DBC}$*, verifica-se:*

$$(n + 1)![N_0, \ldots, N_m] = n![N_1, \ldots, N_m].$$

Demonstração. Segue da definição, cf.

$$\begin{array}{ll} & (n+1)![N_0, \ldots, N_m] \\ = & \quad 4.5 \\ & N_{n+1} \\ = & \quad 4.5 \\ & n![N_1, \ldots, N_m] \end{array}$$

□

Lema 4.8. *Fixem-se $M, Q \in \mathcal{DBC}$, com Q fechado. Para todo $N_1, N_2 \ldots, N_n \in \mathcal{DBC}$ fechados (notação: $\vec{N}$), para todo $k \in \mathbb{N}_0$, verifica-se:*

$$M[\mathsf{Id}_k, k!, \vec{N}][\mathsf{Id}_k, Q] = M[\mathsf{Id}_k, Q, \vec{N}],$$

onde $\mathsf{Id}_k = 0!, 1!, \ldots, (k-1)!$.

Demonstração. Por indução sobre M. Os únicos casos interessantes são os das variáveis e das abstracções – os restantes são aplicações rotineiras da H.I.

- Caso $M = i!$. Procedemos nos casos de i.

  - Caso $i \in \{0, \ldots, k\}$. Temos, por definição,

$$\begin{aligned} i![\mathsf{Id}_k, k!, \vec{N}][\mathsf{Id}_k, Q] &= i![\mathsf{Id}_k, Q] \\ &= i![\mathsf{Id}_k, Q, \vec{N}]. \end{aligned}$$

  - Caso $i \in \{k+1, \ldots, k+n\}$. Temos, por definição e por N_{n-i} ser fechado,

$$\begin{aligned} i![\mathsf{Id}_k, k!, \vec{N}][\mathsf{Id}_k, Q] &= N_{n-i}[\mathsf{Id}_k, Q] \\ &= N_{n-i} \\ &= i![\mathsf{Id}_k, Q, \vec{N}]. \end{aligned}$$

– Caso $i > k + n$. Temos, por definição,

$$\begin{aligned} i![\mathsf{Id}_k, k!, \vec{N}][\mathsf{Id}_k, Q] &= i![\mathsf{Id}_k, Q] \\ &= i! \\ &= i![\mathsf{Id}_k, Q, \vec{N}]. \end{aligned}$$

- Caso $M = \Lambda(M_1)$. Temos

$$\begin{array}{ll} & \Lambda(M_1)[\mathsf{Id}_k, k!, \vec{N}][\mathsf{Id}_k, Q] \\ = & \quad 4.5 \text{ e por } N_i \text{ ser fechado} \\ & \Lambda(M_1[\mathsf{Id}_{k+1}, (k+1)!, \vec{N}])[\mathsf{Id}_k, Q] \\ = & \quad 4.5 \text{ e por } Q \text{ ser fechado} \\ & \Lambda(M_1[\mathsf{Id}_{k+1}, (k+1)!, \vec{N}][\mathsf{Id}_{k+1}, Q]) \\ = & \quad \text{H.I.} \\ & \Lambda(M_1[\mathsf{Id}_{k+1}, Q, \vec{N}]) \\ = & \quad 4.5 \text{ e por } N_i \text{ e } Q \text{ serem fechados} \\ & \Lambda(M_1)[\mathsf{Id}_k, Q, \vec{N}] \end{array}$$

□

Corolário 4.8.1. *Fixem-se $M, Q \in \mathcal{DBC}$, com Q fechado. Para todo $N_1, N_2, \ldots, N_n \in \mathcal{DBC}$ fechados (notação: $\vec{N}$), verifica-se:*

$$M[0!, \vec{N}][Q] = M[Q, \vec{N}].$$

Demonstração. Ora,

$$\begin{array}{ll} & M[0!, \vec{N}][Q] \\ = & \quad \text{definição } \mathsf{Id}_k \\ & M[\mathsf{Id}_0, 0!, \vec{N}][\mathsf{Id}_0, Q] \\ = & \quad 4.8 \\ & M[\mathsf{Id}_0, Q, \vec{N}] \\ = & \quad \text{definição } \mathsf{Id}_k \\ & M[Q, \vec{N}] \end{array}$$

□

4.1.2 Redução *call-by-value* (CBV)

Codificamos nesta subsecção a redução *call-by-value* do Cálculo λ, $\twoheadrightarrow_{\mathrm{B}}$, sobre o conjunto $\mathcal{DBC}$. Apenas nos importamos com termos fechados, cf. 4.1.

Começamos por definir os valores, i.e. as formas normais desta redução.

Definição 4.9. O conjunto dos *valores* é o menor subconjunto de $\mathcal{DBC}$ satisfazendo:

1. $'(c)$ é um valor, para todo $c \in \mathit{Const}$;
2. $\langle V_1, V_2 \rangle$ é um valor, se V_1 e V_2 são valores;
3. $\mathit{fst}(V)$ é um valor, se V é um valor e não é um par;
4. $\mathit{snd}(V)$ é um valor, se V é um valor e não é um par;
5. $\Lambda(M)$ é um valor, para todo $M \in \mathcal{DBC}$;
6. $S(V_1, V_2)$ é um valor, se V_1 e V_2 são valores e V_1 não é uma abstracção.

Denotamos valores pela meta-variável V, eventualmente etiquetada.

Definição 4.10. Define-se a relação binária $\twoheadrightarrow_{\mathrm{B}}$ (de "de Bruijn") indutivamente por:

1. $V \twoheadrightarrow_{\mathrm{B}} V$;
2. $\dfrac{M \twoheadrightarrow_{\mathrm{B}} V_1 \qquad N \twoheadrightarrow_{\mathrm{B}} V_2}{\langle M, N \rangle \twoheadrightarrow_{\mathrm{B}} \langle V_1, V_2 \rangle}$;
3. $\dfrac{M \twoheadrightarrow_{\mathrm{B}} \Lambda(M_1) \qquad N \twoheadrightarrow_{\mathrm{B}} V_2}{S(M, N) \twoheadrightarrow_{\mathrm{B}} M_1[V_2]}$;
4. $\dfrac{M \twoheadrightarrow_{\mathrm{B}} V_1 \neq \Lambda(\cdot) \qquad N \twoheadrightarrow_{\mathrm{B}} V_2}{S(M, N) \twoheadrightarrow_{\mathrm{B}} S(V_1, V_2)}$;
5. $\dfrac{M \twoheadrightarrow_{\mathrm{B}} \langle V_1, V_2 \rangle}{\mathit{fst}(M) \twoheadrightarrow_{\mathrm{B}} V_1}$;
6. $\dfrac{M \twoheadrightarrow_{\mathrm{B}} V \neq \langle \cdot, \cdot \rangle}{\mathit{fst}(M) \twoheadrightarrow_{\mathrm{B}} \mathit{fst}(V)}$;

7. $\dfrac{M \twoheadrightarrow_{\mathrm{B}} \langle V_1, V_2 \rangle}{snd(M) \twoheadrightarrow_{\mathrm{B}} V_2}$;

8. $\dfrac{M \twoheadrightarrow_{\mathrm{B}} V \neq \langle \cdot, \cdot \rangle}{snd(M) \twoheadrightarrow_{\mathrm{B}} snd(V)}$;

9. $\dfrac{M_1 \twoheadrightarrow_{\mathrm{B}} M_2 \qquad M_2 \twoheadrightarrow_{\mathrm{B}} M_3}{M_1 \twoheadrightarrow_{\mathrm{B}} M_3}$;

(onde quantificamos universalmente todas as variáveis.)

Observação 4.11. Note-se como a alínea 3 de 4.10 codifica a regra β. Confronte-se com 1.32.

4.1.3 Redução combinatória

Definimos nesta subsecção a estratégia de redução *innermost* da conversão fraca da Lógica Combinatória Categorial (notação: $\twoheadrightarrow_{\mathrm{C}}$) sobre um par da forma $M?v$, em que M é um termo de de Bruijn (representando o termo categorial) e v um valor (representando o ambiente). Modelamos através desta relação o efeito individual de cada combinador categorial, bem com as reduções fst, snd, dpair, ac e quote.

Começamos por definir os valores, que são certos termos categoriais.

Definição 4.12. O conjunto dos *valores* é definido indutivamente por:

1. A *constante* c é um valor, para todo $c \in Const$. Supomos que $Const$ contém um objecto distinto, ();

2. A *closure* $\Lambda(M).((), v_{n-1}, \ldots, v_0)$ é um valor, para todo $M \in \mathcal{DBC}$ com ordem de ambiente, no máximo, n, e $v_0, \ldots, v_{n-1}$ valores;

3. O *par* (v_1, v_2) é um valor, se v_1 e v_2 são valores;

4. A *primeira projecção* $Fst.v_1$ é um valor, se v_1 é um valor e não é um par;

5. A *segunda projecção* $Snd.v_1$ é um valor, se v_1 é um valor e não é um par;

6. A *aplicação* $App.(v_1, v_2)$ é um valor, se v_1 e v_2 são valores e v_1 não é uma closure.

Denotamos pela meta-variável v, eventualmente etiquetada, valores.

Observação 4.13. Note-se que acabámos de introduzir dois conceitos (4.12 e 4.9) com o mesmo nome: "valor". Isto não será um problema porque será sempre possível deduzir pelo contexto a qual destes conceitos nos referimos.

Nota bibliográfica 4.14. Em (Cousineau et al., 1987) o objecto () utilizado em *closures* não é declarado como uma constante; contudo é utilizado mais tarde em instruções `quote`$(\cdot)$, o que pressupõe que () esteja em *Const*. Também não é declarado como um valor, o que não é coerente no uso de termos da forma $M?()$. Preferimos por isso declará-lo como uma constante por uma questão de uniformização.

Definição 4.15. Dados $M \in \mathcal{DBC}$ e v valor, dizemos que v é *compatível com* M se $v = (v_n, \ldots, v_0)$ e M tem ordem de ambiente, no máximo, n.

Observação 4.16. Intuitivamente, v ser compatível com M significa que v é um ambiente grande o suficiente que permite atribuir significado a todas as variáveis livres de M.

Definição 4.17. O conjunto dos *estados* é definido por extensão como:

1. Todos os valores são estados;

2. Um objecto da forma $M?v$ é um estado, se $M \in \mathcal{DBC}$ e v é um valor compatível com M;

Denotamos estados pelas meta-variáveis s e t, eventualmente etiquetadas.

Observação 4.18. Um objecto da forma $M?v$ pretende representar o termo categorial $M.v$.

Definição 4.19. Define-se a relação binária $\twoheadrightarrow_{\mathrm{C}}$ (de "Combinatória") indutivamente por:

1. $v \twoheadrightarrow_{\mathrm{C}} v$;

2. $'(c)?v \twoheadrightarrow_{\mathrm{C}} c$;
3. $\Lambda(M)?v \twoheadrightarrow_{\mathrm{C}} \Lambda(M).v$;
4. $(n+1)!?(v_1, v_2) \twoheadrightarrow_{\mathrm{C}} n!?v_1$;
5. $0!?(v_1, v_2) \twoheadrightarrow_{\mathrm{C}} v_2$;
6. $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} \Lambda(M_1).v_1 \qquad N?v \twoheadrightarrow_{\mathrm{C}} v_2}{S(M,N)?v \twoheadrightarrow_{\mathrm{C}} M_1?(v_1, v_2)}$;
7. $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} v_1 \neq \Lambda(\cdot).\cdot \qquad N?v \twoheadrightarrow_{\mathrm{C}} v_2}{S(M,N)?v \twoheadrightarrow_{\mathrm{C}} App.(v_1, v_2)}$;
8. $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} v_1 \qquad N?v \twoheadrightarrow_{\mathrm{C}} v_2}{\langle M, N\rangle?v \twoheadrightarrow_{\mathrm{C}} (v_1, v_2)}$;
9. $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} (v_1, v_2)}{fst(M)?v \twoheadrightarrow_{\mathrm{C}} v_1}$;
10. $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} v_1 \neq (\cdot, \cdot)}{fst(M)?v \twoheadrightarrow_{\mathrm{C}} Fst.v_1}$;
11. $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} (v_1, v_2)}{snd(M)?v \twoheadrightarrow_{\mathrm{C}} v_2}$;
12. $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} v_1 \neq (\cdot, \cdot)}{snd(M)?v \twoheadrightarrow_{\mathrm{C}} Snd.v_1}$;
13. $\dfrac{s_1 \twoheadrightarrow_{\mathrm{C}} s_2 \qquad s_2 \twoheadrightarrow_{\mathrm{C}} s_3}{s_1 \twoheadrightarrow_{\mathrm{C}} s_3}$.

(onde quantificamos universalmente todas as variáveis.)

Observação 4.20. Apenas os estados da forma $M?v$ (i.e. questões) reduzem "produtivamente"; os valores reduzem neles mesmos.

Exemplo 4.21. Seja $M = (\lambda xx)y$. Em termos de de Bruijn, M escreve-se como $S(\Lambda(0!), 0!)$. Tem-se:

$$\dfrac{\dfrac{\overset{3.}{\overline{\Lambda(0!)?((), y) \twoheadrightarrow_{\mathrm{C}} \Lambda(0!).((), y)}} \quad \overset{5.}{\overline{0!?((), y) \twoheadrightarrow_{\mathrm{C}} y}}}{\overset{6.}{S(\Lambda(0!), 0!)?((), y) \twoheadrightarrow_{\mathrm{C}} 0!?((), y, y)}} \quad \overset{5.}{\overline{0!?((), y, y) \twoheadrightarrow_{\mathrm{C}} y}}}{\overset{13.}{S(\Lambda(0!), 0!)?((), y) \twoheadrightarrow_{\mathrm{C}} y}}$$

Confronte-se esta derivação com a sequência de passos obtida pela redução fraca da Lógica Combinatória Categorial para o mesmo termo M:

$$\begin{array}{ll} & S(\Lambda(0!),0!).((),y) \\ = & \qquad \text{ass, dpair, snd} \\ & App.(\Lambda(0!).((),y),y) \\ = & \qquad \text{ac} \\ & 0!.((),y,y) \\ = & \qquad \text{snd} \\ & y \end{array}$$

Observe-se como a alínea 5. corresponde à redução snd e a alínea 6. à redução ac. A alínea 3., por outro lado, não tem uma redução correspondente uma vez que $\Lambda(M)?v$ corresponde a $\Lambda(M).v$, que já é uma forma normal; contudo, esta alínea permite modelar o efeito da instrução $\texttt{cur}(\cdot)$ da CAM, o que será útil quando quisermos relacionar as relações $\twoheadrightarrow_{\mathrm{C}}$ e $\rightarrow_{\mathrm{CAM}}$ mais à frente.

Observação 4.22. As alíneas 7., 10. e 12. da definição 4.19 reflectem reduções "estagnadas" que podem surgir porque não impomos qualquer restrição à forma como são construidos os termos categoriais. Por exemplo, o termo-λc $M = \mathit{fst}(\lambda xx)$ traduzido em termos categoriais é $\mathit{Fst} \circ \Lambda(\mathit{Snd})$. Temos

$$\begin{array}{ll} & \mathit{Fst} \circ \Lambda(\mathit{Snd}).() \\ = & \qquad \text{ass} \\ & \mathit{Fst}.(\Lambda(\mathit{Snd}).()) \end{array}$$

Em termos de de Bruijn, M reescreve-se como $\mathit{fst}(\Lambda(0!))$; temos:

$$\begin{array}{ll} & \mathit{fst}(\Lambda(0!))?() \\ \twoheadrightarrow_{\mathrm{C}} & \qquad 3., 10. \\ & \mathit{fst}(\Lambda(0!).()) \end{array}$$

Estas considerações levam-nos a actualizar a redução $\rightarrow_{\mathrm{CAM}}$, como veremos na subsecção 4.2.

A demonstração da correcção da CAM passa agora por relacionar $\rightarrow_{\mathrm{CAM}}$ com $\twoheadrightarrow_{\mathrm{C}}$ (redução combinatória) e $\twoheadrightarrow_{\mathrm{C}}$ com $\twoheadrightarrow_{\mathrm{B}}$ (redução *call-by-value*). Vamos começar por esta última correspondência.

4.1.4 Redução combinatória versus redução *call-by-value*

Um estado $M?v$, como já vimos, representa um termo de de Bruijn e o respectivo ambiente. Podemos extrair o termo que este estado representa substituindo todas as variáveis livres em M pelos valores a elas atribuído pelo ambiente. Formalizamos de seguida esta transformação.

Definição 4.23. Seja s um estado. Define-se $\mathrm{REAL}(s) \in \mathcal{DBC}$ por exaustão dos casos possíveis do seguinte modo:

1. $\mathrm{REAL}(M?((), v_{n-1}, \ldots, v_0)) = M[\mathrm{REAL}(v_0), \mathrm{REAL}(v_1), \ldots, \mathrm{REAL}(v_{n-1})]$;
2. $\mathrm{REAL}(\Lambda(M).((), v_{n-1}, \ldots, v_0)) = \mathrm{REAL}(\Lambda(M)?((), v_{n-1}, \ldots, v_0))$;
3. $\mathrm{REAL}(c) = '(c)$;
4. $\mathrm{REAL}((v_1, v_2)) = (\mathrm{REAL}(v_1), \mathrm{REAL}(v_2))$;
5. $\mathrm{REAL}(\mathit{Fst}.v) = \mathit{fst}(\mathrm{REAL}(v))$, onde v não é um par;
6. $\mathrm{REAL}(\mathit{Snd}.v) = \mathit{snd}(\mathrm{REAL}(v))$, onde v não é um par;
7. $\mathrm{REAL}(\mathit{App}.(v_1, v_2)) = S(\mathrm{REAL}(v_1), \mathrm{REAL}(v_2))$, onde v_1 não é uma closure.

Observação 4.24. Pela forma como construímos os valores e os estados, $\mathrm{REAL}(s)$ é sempre um termo fechado (Cousineau et al., 1987, p. 189). Isto deve-se, em grande parte, à restrição por ambientes compatíveis.

Lema 4.25. *Para todo v valor, $\mathit{REAL}(v) = V$, para algum V valor.*

Demonstração. Por indução sobre v.

- Caso $v = c$. Ora $\mathrm{REAL}(c) \overset{4.23}{=} '(c)$ e $'(c)$ é um valor, cf. 4.9.
- Caso $v = \Lambda(M).((), v_{n-1}, \ldots, v_0)$. Temos

$$\begin{array}{ll} & \mathrm{REAL}(\Lambda(M).((), v_{n-1}, \ldots, v_0)) \\ = & \quad 4.23 \\ & \Lambda(M)[\mathrm{REAL}(v_0), \ldots, \mathrm{REAL}(v_{n-1})] \\ = & \quad 4.5 \\ & \Lambda(M[0!, \mathrm{REAL}(v_0), \ldots, \mathrm{REAL}(v_{n-1})]) \end{array}$$

e $\Lambda(M[0!, \mathrm{REAL}(v_0), \ldots, \mathrm{REAL}(v_{n-1})])$ é um valor, cf. 4.9.

- Caso $v = (v_1, v_2)$. Ora, $\mathrm{REAL}((v_1, v_2)) \overset{4.23}{=} \langle \mathrm{REAL}(v_1), \mathrm{REAL}(v_2) \rangle$. Por H.I., $\mathrm{REAL}(v_1), \mathrm{REAL}(v_2)$ são valores. Logo, $\langle \mathrm{REAL}(v_1), \mathrm{REAL}(v_2) \rangle$ é um valor, cf. 4.9.

- Caso $v = \mathit{Fst}.v_1$, onde v_1 não é um par. Ora $\mathrm{REAL}(\mathit{Fst}.v_1) \overset{4.23}{=} \mathit{fst}(\mathrm{REAL}(v_1))$. Por H.I., $\mathrm{REAL}(v_1)$ é um valor. Como v_1 não é um par, podemos concluir, mercê da definição de REAL, que $\mathrm{REAL}(v_1)$ também não é um par. Logo, $\mathit{fst}(\mathrm{REAL}(v_1))$ é um valor, cf. 4.9.

- Caso $v = \mathit{Snd}.v_1$, onde v_1 não é um par. Análogo ao caso anterior.

- Caso $v = \mathit{App}.(v_1, v_2)$, onde v_1 não é uma closure. Ora $\mathrm{REAL}(\mathit{App}.(v_1, v_2)) \overset{4.23}{=} S(\mathrm{REAL}(v_1), \mathrm{REAL}(v_2))$. Por H.I., $\mathrm{REAL}(v_1), \mathrm{REAL}(v_2)$ são valores. Como v_1 não é uma closure, podemos concluir, mercê da definição de REAL, que $\mathrm{REAL}(v_1)$ não é uma abstracção, Logo, $S(\mathrm{REAL}(v_1), \mathrm{REAL}(v_2))$ é um valor, cf. 4.9.

□

Proposição 4.26. *Para todo $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$: $\mathit{REAL}(s_1) \twoheadrightarrow_{\mathrm{B}} \mathit{REAL}(s_2)$.*

Demonstração. Por indução sobre $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$. Omitimos os casos 11. e 12. (referentes às reduções $\mathit{snd}(\cdot)?v \twoheadrightarrow_{\mathrm{C}} \cdot$) por serem análogos aos casos 9. e 10. (referentes às reduções $\mathit{fst}(\cdot)?v \twoheadrightarrow_{\mathrm{C}} \cdot$).

- Caso $v \twoheadrightarrow_{\mathrm{C}} v$. Por 4.25, $\mathrm{REAL}(v) = V$, para algum valor V, e $V \twoheadrightarrow_{\mathrm{B}} V$, por definição.

- Caso $'(c)?v \twoheadrightarrow_{\mathrm{C}} c$. Ora

$$\begin{array}{ll} & \mathrm{REAL}('(c)?v) \\ = & \quad \text{4.23, 4.5} \\ & '(c) \\ = & \quad \text{4.23} \\ & \mathrm{REAL}(c) \end{array}$$

e $'(c) \twoheadrightarrow_{\mathrm{B}} '(c)$ porque $'(c)$ é um valor e a relação $\twoheadrightarrow_{\mathrm{B}}$ é reflexiva para estes termos, por definição.

- Caso $\Lambda(M)?v \twoheadrightarrow_{\mathrm{C}} \Lambda(M).v$. Como $\mathrm{REAL}(\Lambda(M)?v) = \mathrm{REAL}(\Lambda(M).v)$ por definição, $\mathrm{REAL}(\Lambda(M).v)$ é um valor (4.25) e a relação $\twoheadrightarrow_{\mathrm{B}}$ é reflexiva para valores (4.10), temos que $\mathrm{REAL}(\Lambda(M)?v) \twoheadrightarrow_{\mathrm{B}} \mathrm{REAL}(\Lambda(M).v)$, como pretendíamos.

- Caso $(n+1)!?(v_1, v_2) \twoheadrightarrow_{\mathrm{C}} n!?v_1$. Podemos supor que $(v_1, v_2) = ((), v'_m, \ldots, v'_{n+1}, \ldots, v'_0)$ onde $v_1 = ((), v'_m, \ldots, v'_{n+1}, \ldots, v'_1)$ e $v_2 = v'_0$. Por um lado, temos

$$\begin{array}{ll} & \mathrm{REAL}((n+1)!?(v_1, v_2)) \\ = & \quad 4.23 \\ & (n+1)![\mathrm{REAL}(v'_0), \ldots, \mathrm{REAL}(v'_m)] \\ = & \quad 4.7 \\ & n![\mathrm{REAL}(v'_1), \ldots, \mathrm{REAL}(v'_m)] \\ = & \quad 4.23 \\ & \mathrm{REAL}(n!?v_1) \end{array}$$

  Por outro, $n![\mathrm{REAL}(v'_1), \ldots, \mathrm{REAL}(v'_m)] \overset{4.5}{=} \mathrm{REAL}(v'_{n+1})$ e $\mathrm{REAL}(v'_{n+1})$ é um valor (cf. 4.25). Como $\twoheadrightarrow_{\mathrm{B}}$ é reflexiva para valores, terminamos.

- Caso $0!?(v_1, v_2) \twoheadrightarrow_{\mathrm{C}} v_2$. Análoga ao caso anterior.

- Caso $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} \Lambda(M_1).v_1 \qquad N?v \twoheadrightarrow_{\mathrm{C}} v_2}{S(M,N)?v \twoheadrightarrow_{\mathrm{C}} M_1?(v_1, v_2)}$. Seja $v = ((), v'_{n-1}, \ldots, v'_0)$ e façamos $[\vec{V}] = [\mathrm{REAL}(v'_0), \ldots, \mathrm{REAL}(v'_{n-1})]$. Observe-se que

$$\begin{array}{ll} & \mathrm{REAL}(S(M,N)?v) \\ = & \quad 4.23 \\ & S(M,N)[\vec{V}] \\ = & \quad 4.5 \\ & S(M[\vec{V}], N[\vec{V}]) \\ = & \quad 4.23 \\ & S(\mathrm{REAL}(M?v), \mathrm{REAL}(N?v)) \end{array}$$

  Por H.I. temos

i.

$$\begin{array}{ll} & \mathrm{REAL}(N?v) \twoheadrightarrow_{\mathrm{B}} \mathrm{REAL}(v_2) \\ \iff & \text{Tomando } V_2 = \mathrm{REAL}(v_2) \\ & \mathrm{REAL}(N?v) \twoheadrightarrow_{\mathrm{B}} V_2 \end{array}$$

ii.

$$\begin{array}{ll} & \mathrm{REAL}(M?v) \twoheadrightarrow_{\mathrm{B}} \mathrm{REAL}(\Lambda(M_1).v_1) \\ \iff & \text{4.23, tomando } [\vec{V_1}] = [\mathrm{REAL}(v_{1_0}), \ldots, \mathrm{REAL}(v_{1_{m-1}})] \\ & \mathrm{REAL}(M?v) \twoheadrightarrow_{\mathrm{B}} \Lambda(M_1)[\vec{V_1}] \\ \iff & \text{4.5} \\ & \mathrm{REAL}(M?v) \twoheadrightarrow_{\mathrm{B}} \Lambda(M_1[0!, \vec{V_1}]) \end{array}$$

Logo,

$$\begin{array}{ll} & S(\mathrm{REAL}(M?v), \mathrm{REAL}(N?v)) \\ \twoheadrightarrow_{\mathrm{B}} & \text{i., ii., 4.10} \\ & M_1[0!, \vec{V_1}][V_2] \\ = & \text{4.8.1} \\ & M_1[V_2, \vec{V_1}] \\ = & \text{4.23} \\ & \mathrm{REAL}(M_1?(v_1, v_2)) \end{array}$$

- Caso $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} v_1 \neq \Lambda(\cdot).\cdot \qquad N?v \twoheadrightarrow_{\mathrm{C}} v_2}{S(M,N)?v \twoheadrightarrow_{\mathrm{C}} App.(v_1, v_2)}$. Note-se que $\mathrm{REAL}(v_1) \neq \Lambda(\cdot)$, em virtude da definição de REAL. Ora,

$$\begin{array}{ll} & \mathrm{REAL}(S(M,N)?v) \\ = & \text{4.23,4.5} \\ & S(\mathrm{REAL}(M?v), \mathrm{REAL}(N?v)) \\ \twoheadrightarrow_{\mathrm{B}} & \text{H.I., 4.10} \\ & S(\mathrm{REAL}(v_1), \mathrm{REAL}(v_2)) \\ = & \text{4.23} \\ & \mathrm{REAL}(App.(v_1, v_2)) \end{array}$$

- Caso $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} v_1 \qquad N? \twoheadrightarrow_{\mathrm{C}} v_2}{\langle M, N\rangle?v \twoheadrightarrow_{\mathrm{C}} (v_1, v_2)}$. Temos

$$\begin{array}{ll} & \mathrm{REAL}(\langle M, N\rangle?v) \\ = & \qquad 4.23{,}4.5 \\ & \langle \mathrm{REAL}(M?v), \mathrm{REAL}(N?v)\rangle \\ \twoheadrightarrow_{\mathrm{B}} & \qquad \text{H.I., } 4.10 \\ & \langle \mathrm{REAL}(v_1), \mathrm{REAL}(v_2)\rangle \\ = & \qquad 4.23 \\ & \mathrm{REAL}((v_1, v_2)) \end{array}$$

- Caso $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} (v_1, v_2)}{\mathit{fst}(M)?v \twoheadrightarrow_{\mathrm{C}} v_1}$. Temos

$$\begin{array}{ll} & \mathrm{REAL}(\mathit{fst}(M)?v) \\ = & \qquad 4.23{,}4.5 \\ & \mathit{fst}(\mathrm{REAL}(M?v)) \\ \twoheadrightarrow_{\mathrm{B}} & \qquad \text{H.I., } \mathrm{REAL}((v_1, v_2)) \overset{4.23}{=} \langle \mathrm{REAL}(v_1), \mathrm{REAL}(v_2)\rangle, \ 4.10 \\ & \mathrm{REAL}(v_1) \end{array}$$

- Caso $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} v_1 \neq (\cdot, \cdot)}{\mathit{fst}(M)?v \twoheadrightarrow_{\mathrm{C}} \mathit{Fst}.v_1}$. Note-se que $\mathrm{REAL}(v_1) \neq \langle \cdot, \cdot \rangle$, em virtude da definição de REAL. Ora

$$\begin{array}{ll} & \mathrm{REAL}(\mathit{fst}(M)?v) \\ = & \qquad 4.23{,}4.5 \\ & \mathit{fst}(\mathrm{REAL}(M?v)) \\ \twoheadrightarrow_{\mathrm{B}} & \qquad \text{H.I., } 4.10 \\ & \mathit{fst}(\mathrm{REAL}(v_1)) \\ = & \qquad 4.23 \\ & \mathrm{REAL}(\mathit{Fst}.v_1) \end{array}$$

- Caso $\dfrac{s_1 \twoheadrightarrow_{\mathrm{C}} s_2 \qquad s_2 \twoheadrightarrow_{\mathrm{C}} s_3}{s_1 \twoheadrightarrow_{\mathrm{C}} s_3}$. Imediato, em consequência da transitividade de $\twoheadrightarrow_{\mathrm{B}}$ e H.I.

□

Lema 4.27. *Para todo s estado, se $REAL(s) = V$, para algum valor V, então $s \twoheadrightarrow_{\mathrm{C}} v'$, para algum valor v' tal que $REAL(v') = V$.*

Demonstração. No caso em que s é um valor temos $\mathrm{REAL}(s) = V$ (por 4.25) e $s \twoheadrightarrow_{\mathrm{C}} s$, por definição de $\twoheadrightarrow_{\mathrm{C}}$. No caso em que $s = M?v$, procedemos por indução sobre M.

- Caso $M = \prime(c)$. Temos $\mathrm{REAL}(\prime(c)?v) = \prime(c) = \mathrm{REAL}(c)$, por 4.23. Tome-se $v' = c$. Então $\prime(c)?v \twoheadrightarrow_{\mathrm{C}} v'$ e $\mathrm{REAL}(v') = \prime(c)$.

- Caso $M = \langle M_1, M_2 \rangle$. Observe-se que

$$
\begin{array}{ll}
 & \mathrm{REAL}(\langle M_1, M_2 \rangle ?v) = V \\
\Longrightarrow & \text{4.23,4.5} \\
 & \langle \mathrm{REAL}(M_1?v), \mathrm{REAL}(M_2?v) \rangle = V \\
\Longrightarrow & \text{4.9, introduzindo } V_1, V_2 \\
 & \begin{cases} V = \langle V_1, V_2 \rangle \\ V_1 = \mathrm{REAL}(M_1?v) \\ V_2 = \mathrm{REAL}(M_2?v) \end{cases} \\
\Longrightarrow & \text{H.I., onde } \mathrm{REAL}(v_1) = V_1 \text{ e } \mathrm{REAL}(v_2) = V_2 \\
 & \begin{cases} M_1?v \twoheadrightarrow_{\mathrm{C}} v_1 \\ M_2?v \twoheadrightarrow_{\mathrm{C}} v_2 \end{cases}
\end{array}
$$

  Posto isto, suponhamos que $\mathrm{REAL}(\langle M_1, M_2 \rangle ?v) = V$. Tome-se $v' = (v_1, v_2)$. Então $\mathrm{REAL}(v') = V$ e $\langle M_1, M_2 \rangle ?v \twoheadrightarrow_{\mathrm{C}} v'$.

- Caso $M = S(M_1, M_2)$. Observe-se que

$$
\begin{array}{ll}
 & \mathrm{REAL}(S(M_1, M_2)?v) = V \\
\Longrightarrow & \text{4.23,4.5} \\
 & S(\mathrm{REAL}(M_1?v), \mathrm{REAL}(M_2?v)) = V \\
\Longrightarrow & \text{4.9, introduzindo } V_1, V_2 \\
 & \begin{cases} V = S(V_1, V_2) \\ V_1 = \mathrm{REAL}(M_1?v) \neq \Lambda(\cdot) \\ V_2 = \mathrm{REAL}(M_2?v) \end{cases} \\
\Longrightarrow & \text{H.I., onde } \mathrm{REAL}(v_1) = V_1 \text{ e } \mathrm{REAL}(v_2) = V_2
\end{array}
$$

$$\begin{cases} M_1?v \twoheadrightarrow_{\mathrm{C}} v_1 \\ M_2?v \twoheadrightarrow_{\mathrm{C}} v_2 \end{cases}$$

Como v_1 é um valor e $\mathrm{REAL}(v_1) \neq \Lambda(\cdot)$, temos que $v_1 \neq \Lambda(\cdot).\cdot$. Suponhamos portanto que $\mathrm{REAL}(S(M_1, M_2)?v) = V$. Tome-se $v' = App.(v_1, v_2)$. Então $\mathrm{REAL}(v') = V$ e $S(M_1, M_2)?v \twoheadrightarrow_{\mathrm{C}} v'$.

- Caso $M = fst(M_1)$. Observe-se que

$$\begin{array}{ll} & \mathrm{REAL}(fst(M_1)?v) = V \\ \Longrightarrow & \quad 4.23{,}4.5 \\ & fst(\mathrm{REAL}(M_1?v)) = V \\ \Longrightarrow & \quad 4.9, \text{ introduzindo } V_1 \\ & \begin{cases} V = fst(V_1) \\ V_1 = \mathrm{REAL}(M_1?v) \neq \langle \cdot, \cdot \rangle \end{cases} \\ \Longrightarrow & \quad \text{H.I., onde } \mathrm{REAL}(v_1) = V_1 \\ & M_1?v \twoheadrightarrow_{\mathrm{C}} v_1 \end{array}$$

Como v_1 é um valor e $\mathrm{REAL}(v_1) \neq \langle \cdot, \cdot \rangle$, temos que $v_1 \neq (\cdot, \cdot)$. Suponhamos portanto que $\mathrm{REAL}(fst(M_1)?v) = V$. Tome-se $v' = Fst.v_1$. Então $\mathrm{REAL}(v') = V$ e $fst(M_1)?v \twoheadrightarrow_{\mathrm{C}} v'$.

- Caso $M = snd(M_1)$. Análogo ao caso anterior.

- Caso $M = \Lambda(M_1)$. Tome-se $v' = \Lambda(M_1).v$. Então $\Lambda(M_1)?v \twoheadrightarrow_{\mathrm{C}} v'$ e $\mathrm{REAL}(v') = \mathrm{REAL}(\Lambda(M_1)?v)$.

- Caso $M = n!$. Podemos supor que $v = ((), v_m, \ldots, v_n, \ldots, v_0)$. Tem-se

$$\begin{array}{ll} & \mathrm{REAL}(n!?v) \\ = & \quad 4.23 \\ & n![\mathrm{REAL}(v_0), \ldots, \mathrm{REAL}(v_m)] \\ = & \quad 4.5 \\ & \mathrm{REAL}(v_n) \end{array}$$

e $\mathrm{REAL}(v_n)$ é um valor (4.25). Tome-se $v' = v_n$. Então $\mathrm{REAL}(n!?v) = \mathrm{REAL}(v')$ e $n!?v \twoheadrightarrow_{\mathrm{C}} v'$ (4.34).

□

Proposição 4.28. *Para todo $M \twoheadrightarrow_{\mathrm{B}} N$, se existe s_1 tal que $REAL(s_1) = M$, então $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$, para algum s_2 tal que $REAL(s_2) = N$.*

Demonstração. Por indução sobre $M \twoheadrightarrow_{\mathrm{B}} N$.

- Caso $V \twoheadrightarrow_{\mathrm{B}} V$. Seja s_1 tal que $\mathrm{REAL}(s_1) = V$. Ora, por 4.27, $s_1 \twoheadrightarrow_{\mathrm{C}} v$, para algum v tal que $\mathrm{REAL}(v) = V$, como pretendíamos.
- Caso $\dfrac{M_1 \twoheadrightarrow_{\mathrm{B}} V_1 \qquad M_2 \twoheadrightarrow_{\mathrm{B}} V_2}{\langle M_1, M_2\rangle \twoheadrightarrow_{\mathrm{B}} \langle V_1, V_2\rangle}$. Seja s_1 tal que $\mathrm{REAL}(s_1) = \langle M_1, M_2\rangle$. Procedemos por casos de s_1.
  - Caso $s_1 = v$, para algum valor v. Temos

    $$\begin{array}{ll} & \mathrm{REAL}(s_1) = \mathrm{REAL}(v) = \langle M_1, M_2\rangle \\ \Longrightarrow & \text{4.23, introduzindo } v_1, v_2 \text{ valores} \\ & \begin{cases} v = (v_1, v_2) \\ M_1 = \mathrm{REAL}(v_1) \\ M_2 = \mathrm{REAL}(v_2) \end{cases} \\ \Longrightarrow & \text{H.I., introduzindo } s_1', s_2' \\ & \begin{cases} v_1 \twoheadrightarrow_{\mathrm{C}} s_1', \text{ onde } \mathrm{REAL}(s_1') = V_1 \\ v_2 \twoheadrightarrow_{\mathrm{C}} s_2', \text{ onde } \mathrm{REAL}(s_2') = V_2 \end{cases} \end{array}$$

    Como v_1, v_2 são valores e $v_1 \twoheadrightarrow_{\mathrm{C}} s_1', v_2 \twoheadrightarrow_{\mathrm{C}} s_2'$, temos $v_1 = s_1'$ e $v_2 = s_2'$, pela definição de $\twoheadrightarrow_{\mathrm{C}}$. Tome-se portanto $s_2 = (s_1', s_2')$. Então $s_1 = s_2$, $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$ e $\mathrm{REAL}(s_2) = \langle V_1, V_2\rangle$.
  - Caso $s_1 = P?v$, para algum $P \in \mathcal{DBC}$ e v valor. Temos

    $$\begin{array}{ll} & \mathrm{REAL}(s_1) = \mathrm{REAL}(P?v) = \langle M_1, M_2\rangle \\ \Longrightarrow & \text{4.23, 4.5, introduzindo } P_1, P_2 \\ & \begin{cases} P = \langle P_1, P_2\rangle \\ M_1 = \mathrm{REAL}(P_1?v) \\ M_2 = \mathrm{REAL}(P_2?v) \end{cases} \\ \Longrightarrow & \text{H.I., introduzindo } t_1, t_2 \text{ estados} \\ & \begin{cases} P_1?v \twoheadrightarrow_{\mathrm{C}} t_1, \text{ onde } \mathrm{REAL}(t_1) = V_1 \\ P_2?v \twoheadrightarrow_{\mathrm{C}} t_2, \text{ onde } \mathrm{REAL}(t_2) = V_2 \end{cases} \end{array}$$

Por 4.27, $t_1 \twoheadrightarrow_{\mathrm{C}} v_1$ e $t_2 \twoheadrightarrow_{\mathrm{C}} v_2$ para alguns v_1, v_2 valores tais que $\mathrm{REAL}(v_1) = \mathrm{REAL}(t_1)$ e $\mathrm{REAL}(v_2) = \mathrm{REAL}(t_2)$. Tome-se portanto $s_2 = (v_1, v_2)$. Então $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$ cf.

$$\frac{13.\dfrac{P_1?v \twoheadrightarrow_{\mathrm{C}} t_1 \qquad t_1 \twoheadrightarrow_{\mathrm{C}} v_1}{P_1?v \twoheadrightarrow_{\mathrm{C}} v_1} \qquad 13.\dfrac{P_2?v \twoheadrightarrow_{\mathrm{C}} t_2 \qquad t_2 \twoheadrightarrow_{\mathrm{C}} v_2}{P_2?v \twoheadrightarrow_{\mathrm{C}} v_2}}{\langle P_1, P_2\rangle?v \twoheadrightarrow_{\mathrm{C}} (v_1, v_2)}$$

e $\mathrm{REAL}(s_2) = \langle V_1, V_2\rangle$.

- Caso $\dfrac{M_1 \twoheadrightarrow_{\mathrm{B}} \Lambda(M_3) \qquad M_2 \twoheadrightarrow_{\mathrm{B}} V_2}{S(M_1, M_2) \twoheadrightarrow_{\mathrm{B}} M_3[V_2]}$. Seja s_1 tal que $\mathrm{REAL}(s_1) = S(M_1, M_2)$. Procedemos por casos de s_1.

  - Caso $s_1 = v$, para algum valor v. Mostramos que este caso conduz a uma contradição, donde se conclui que é impossível ter-se, simultaneamente, $s_1 = v$, $\mathrm{REAL}(s_1) = S(M_1, M_2)$, $M_1 \twoheadrightarrow_{\mathrm{B}} \Lambda(M_3)$ e $M_2 \twoheadrightarrow_{\mathrm{B}} V_2$. Por um lado,

    $$\begin{array}{ll} & \mathrm{REAL}(s_1) = \mathrm{REAL}(v) = S(M_1, M_2) \\ \Longrightarrow & \quad 4.23, \text{ introduzindo } v_1, v_2 \text{ valores} \\ & s_1 = \mathit{App.}(v_1, v_2), \text{ onde } v_1 \text{ não é uma closure} \end{array}$$

    Por outro,

    $$\begin{array}{ll} & \mathrm{REAL}(\mathit{App.}(v_1, v_2)) = S(M_1, M_2) \\ \Longrightarrow & \quad 4.23 \\ & \begin{cases} \mathrm{REAL}(v_1) = M_1 \\ \mathrm{REAL}(v_2) = M_2 \end{cases} \\ \Longrightarrow & \quad \text{H.I.} \\ & \begin{cases} v_1 \twoheadrightarrow_{\mathrm{C}} t_1, \text{ onde } \mathrm{REAL}(t_1) = \Lambda(M_3) \\ v_2 \twoheadrightarrow_{\mathrm{C}} t_2, \text{ onde } \mathrm{REAL}(t_2) = V_2 \end{cases} \end{array}$$

    Como v_1 e v_2 são valores, temos $v_1 = t_1$ e $v_2 = t_2$, pela definição de $\twoheadrightarrow_{\mathrm{C}}$. Por $\mathrm{REAL}(t_1) = \Lambda(M_3)$, temos que t_1 é da forma $\Lambda(\cdot).\cdot$, por definição de REAL. Portanto, v_1 não é uma closure, $v_1 = t_1$ e t_1 é uma closure: contradição!

– Caso $s_1 = P?v$, para algum $P \in \mathcal{DBC}$ e v valor. Temos

$$\begin{array}{ll} & \mathrm{REAL}(s_1) = \mathrm{REAL}(P?v) = S(M_1, M_2) \\ \Longrightarrow & \text{4.23, introduzindo } P_1, P_2 \\ & \begin{cases} P = S(P_1, P_2) \\ M_1 = \mathrm{REAL}(P_1?v) \\ M_2 = \mathrm{REAL}(P_2?v) \end{cases} \\ \Longrightarrow & \text{H.I., introduzindo } t_1, t_2 \text{ estados} \\ & \begin{cases} P_1?v \twoheadrightarrow_{\mathrm{C}} t_1, \text{ onde } \mathrm{REAL}(t_1) = \Lambda(M_3) \\ P_2?v \twoheadrightarrow_{\mathrm{C}} t_2, \text{ onde } \mathrm{REAL}(t_2) = V_2 \end{cases} \end{array}$$

Por 4.27, podemos assumir que $t_2 \twoheadrightarrow_{\mathrm{C}} v_2$ para algum valor v_2, e que

$$(*) \qquad \mathrm{REAL}(v_2) = \mathrm{REAL}(t_2) = V_2.$$

Por $\mathrm{REAL}(t_1) = \Lambda(M_3)$ vem (por 4.23,4.5):

$$(**) \qquad \begin{cases} t_1 = \Lambda(M_3').v' \\ M_3 = M_3'[0!, \mathrm{REAL}(v_0'), \ldots, \mathrm{REAL}(v_n')] \end{cases}$$

para algum $M_3' \in \mathcal{DBC}$ e $v' = ((), v_n', \ldots, v_0')$ compatível com M_3'. Tome-se $s_2 = M_3'?(v', v_2)$. Então $\mathrm{REAL}(s_2) = M_3[V_2]$, cf.

$$\begin{array}{ll} & \mathrm{REAL}(s_2) \\ = & \quad 4.23 \\ & M_3'[\mathrm{REAL}(v_2), \mathrm{REAL}(v_0'), \ldots, \mathrm{REAL}(v_n')] \\ = & \quad 4.8.1 \\ & M_3'[0!, \mathrm{REAL}(v_0'), \ldots, \mathrm{REAL}(v_n')][\mathrm{REAL}(v_2)] \\ = & \quad (**) \\ & M_3[\mathrm{REAL}(v_2)] \\ = & \quad (*) \\ & M_3[V_2] \end{array}$$

e $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$, cf.

$$\frac{P_1?v \twoheadrightarrow_{\mathrm{C}} \Lambda(M_3').v' \qquad 13.\ \dfrac{P_2?v \twoheadrightarrow_{\mathrm{C}} t_2 \qquad t_2 \twoheadrightarrow_{\mathrm{C}} v_2}{P_2?v \twoheadrightarrow_{\mathrm{C}} v_2}}{S(P_1,P_2)?v \twoheadrightarrow_{\mathrm{C}} M_3'?(v', v_2)}$$

- Caso $\dfrac{M_1 \twoheadrightarrow_{\mathrm{B}} V_1 \neq \Lambda(\cdot) \qquad M_2 \twoheadrightarrow_{\mathrm{B}} V_2}{S(M_1,M_2) \twoheadrightarrow_{\mathrm{B}} S(V_1,V_2)}$. Seja s_1 tal que $\mathrm{REAL}(s_1) = S(M_1,M_2)$. Procedemos nos casos de s_1.

  - Caso $s_1 = v$, para algum valor v. Temos

    $$\begin{array}{ll} & \mathrm{REAL}(s_1) = \mathrm{REAL}(v) = S(M_1,M_2) \\ \Longrightarrow & \text{4.23, introduzindo } v_1, v_2 \text{ valores} \\ & \begin{cases} v = App.(v_1,v_2), \text{ onde } v_1 \text{ não é uma closure} \\ M_1 = \mathrm{REAL}(v_1) \\ M_2 = \mathrm{REAL}(v_2) \end{cases} \\ \Longrightarrow & \text{H.I., introduzindo } t_1, t_2 \text{ estados} \\ & \begin{cases} v_1 \twoheadrightarrow_{\mathrm{C}} t_1, \text{ onde } \mathrm{REAL}(t_1) = V_1 \\ v_2 \twoheadrightarrow_{\mathrm{C}} t_2, \text{ onde } \mathrm{REAL}(t_2) = V_2 \end{cases} \end{array}$$

    Como v_1, v_2 são valores e $v_1 \twoheadrightarrow_{\mathrm{C}} t_1, v_2 \twoheadrightarrow_{\mathrm{C}} t_2$, temos que $v_1 = t_1$ e $v_2 = t_2$, por definição de $\twoheadrightarrow_{\mathrm{C}}$. Tome-se $s_2 = s_1 = App.(v_1,v_2)$. Então $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$ e $\mathrm{REAL}(s_2) = S(M_1,M_2)$.

  - Caso $s_1 = P?v$, para algum $P \in \mathcal{DBC}$ e v valor. Temos

    $$\begin{array}{ll} & \mathrm{REAL}(s_1) = \mathrm{REAL}(P?v) = S(M_1,M_2) \\ \Longrightarrow & \text{4.23, 4.5, introduzindo } P_1, P_2 \\ & \begin{cases} P = S(P_1,P_2) \\ M_1 = \mathrm{REAL}(P_1?v) \\ M_2 = \mathrm{REAL}(P_2?v) \end{cases} \\ \Longrightarrow & \text{H.I., introduzindo } t_1, t_2 \text{ estados} \\ & \begin{cases} P_1?v \twoheadrightarrow_{\mathrm{C}} t_1, \text{ onde } \mathrm{REAL}(t_1) = V_1 \\ P_2?v \twoheadrightarrow_{\mathrm{C}} t_2, \text{ onde } \mathrm{REAL}(t_2) = V_2 \end{cases} \end{array}$$

    Por 4.27, $t_1 \twoheadrightarrow_{\mathrm{C}} v_1$ e $t_2 \twoheadrightarrow_{\mathrm{C}} v_2$ para alguns v_1, v_2 valores tais que $\mathrm{REAL}(v_1) = \mathrm{REAL}(t_1)$ e $\mathrm{REAL}(v_2) = \mathrm{REAL}(t_2)$. Como

$\mathrm{REAL}(v_1) = V_1 \neq \Lambda(\cdot)$, tem-se que v_1 não é uma closure, por definição de REAL. Tome-se $s_2 = \mathit{App.}(v_1, v_2)$. Então $\mathrm{REAL}(s_2) = S(V_1, V_2)$ e $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$, cf.

$$\dfrac{13.\dfrac{P_1?v \twoheadrightarrow_{\mathrm{C}} t_1 \quad t_1 \twoheadrightarrow_{\mathrm{C}} v_1 \neq \Lambda(\cdot).\cdot}{P_1 \twoheadrightarrow_{\mathrm{C}} v_1 \neq \Lambda(\cdot).\cdot} \qquad 13.\dfrac{P_2?v \twoheadrightarrow_{\mathrm{C}} t_2 \quad t_2 \twoheadrightarrow_{\mathrm{C}} v_2}{P_2?v \twoheadrightarrow_{\mathrm{C}} v_2}}{S(P_1, P_2)?v \twoheadrightarrow_{\mathrm{C}} \mathit{App.}(v_1, v_2)}$$

- Caso $\dfrac{M_1 \twoheadrightarrow_{\mathrm{B}} \langle V_1, V_2 \rangle}{\mathit{fst}(M_1) \twoheadrightarrow_{\mathrm{B}} V_1}$. Seja s_1 tal que $\mathrm{REAL}(s_1) = \mathit{fst}(M_1)$. Procedemos por casos de s_1.

  - Caso $s_1 = v$, para algum valor v. Mostramos que este caso conduz a uma contradição, donde se conclui que é impossível ter-se, simultaneamente, $s_1 = v$, $\mathrm{REAL}(s_1) = \mathit{fst}(M_1)$ e $M_1 \twoheadrightarrow_{\mathrm{B}} \langle V_1, V_2 \rangle$. Temos

    $$\begin{array}{ll} & \mathrm{REAL}(s_1) = \mathrm{REAL}(v) = \mathit{fst}(M_1) \\ \Longrightarrow & \text{4.23} \\ & \begin{cases} v = \mathit{Fst}.v_1, \text{ onde } v_1 \text{ não é um par} \\ M_1 = \mathrm{REAL}(v_1) \end{cases} \\ \Longrightarrow & \text{H.I., introduzindo } t_1 \\ & v_1 \twoheadrightarrow_{\mathrm{C}} t_1, \text{ onde } \mathrm{REAL}(t_1) = \langle V_1, V_2 \rangle \end{array}$$

    Como v_1 é um valor e $v_1 \twoheadrightarrow_{\mathrm{C}} t_1$, temos $v_1 = t_1$, pela definição de $\twoheadrightarrow_{\mathrm{C}}$. Por $\mathrm{REAL}(t_1) = \langle V_1, V_2 \rangle$, temos que t_1 é da forma $(\cdot, \cdot)$, pela definição de REAL. Portanto, v_1 não é um par, $v_1 = t_1$ e t_1 é um par: contradição!

  - Caso $s_1 = P?v$, para algum $P \in \mathcal{DBC}$ e v valor. Temos

    $$\begin{array}{ll} & \mathrm{REAL}(s_1) = \mathrm{REAL}(P?v) = \mathit{fst}(M_1) \\ \Longrightarrow & \text{4.23} \\ & \begin{cases} P = \mathit{fst}(\mathrm{REAL}(P_1?v)) \\ M_1 = \mathrm{REAL}(P_1?v) \end{cases} \\ \Longrightarrow & \text{H.I., introduzindo } t_1 \text{ estado} \\ & P_1?v \twoheadrightarrow_{\mathrm{C}} t_1, \text{ onde } \mathrm{REAL}(t_1) = \langle V_1, V_2 \rangle \end{array}$$

Por 4.27, $t_1 \twoheadrightarrow_{\mathrm{C}} v_1$ para algum v_1 valor tal que $\mathrm{REAL}(v_1) = \mathrm{REAL}(t_1)$. De $\mathrm{REAL}(v_1) = \langle V_1, V_2 \rangle$ vem por definição:

$$\begin{cases} v_1 = (v_1', v_2') \\ V_1 = \mathrm{REAL}(v_1') \\ V_2 = \mathrm{REAL}(v_2') \end{cases}$$

para alguns v_1', v_2' valores. Tome-se $s_2 = v_1'$. Então $\mathrm{REAL}(s_2) = V_1$ e $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$, cf.

$$13.\ \frac{\dfrac{P_1?v \twoheadrightarrow_{\mathrm{C}} t_1 \qquad t_1 \twoheadrightarrow_{\mathrm{C}} (v_1', v_2')}{P_1?v \twoheadrightarrow_{\mathrm{C}} (v_1', v_2')}}{\mathit{fst}(P_1)?v \twoheadrightarrow_{\mathrm{C}} v_1'}$$

- Caso $\dfrac{M_1 \twoheadrightarrow_{\mathrm{B}} V_1 \neq \langle \cdot, \cdot \rangle}{\mathit{fst}(M_1) \twoheadrightarrow_{\mathrm{B}} \mathit{fst}(V_1)}$. Seja s_1 tal que $\mathrm{REAL}(s_1) = \mathit{fst}(M_1)$. Procedemos por casos de s_1.

  - Caso $s_1 = v$, para algum valor v. Temos

    $\mathrm{REAL}(s_1) = \mathrm{REAL}(v) = \mathit{fst}(M_1)$
    $\Longrightarrow$ 4.23
    $\begin{cases} v = \mathit{Fst}.v_1, \text{ onde } v_1 \text{ não é um par} \\ M_1 = \mathrm{REAL}(v_1) \end{cases}$
    $\Longrightarrow$ H.I., introduzindo t_1 estado
    $v_1 \twoheadrightarrow_{\mathrm{C}} t_1$, onde $\mathrm{REAL}(t_1) = V_1 \neq \langle \cdot, \cdot \rangle$

    Como v_1 é um valor e $v_1 \twoheadrightarrow_{\mathrm{C}} t_1$, temos $v_1 = t_1$, por definição de $\twoheadrightarrow_{\mathrm{C}}$. Tome-se $s_2 = s_1 = \mathit{Fst}.v_1$. Então $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$ e $\mathrm{REAL}(s_2) = \mathit{fst}(V_1)$.

  - Caso $s_1 = P?v$, para algum $P \in \mathcal{DBC}$ e v valor. Temos

    $\mathrm{REAL}(s_1) = \mathrm{REAL}(P?v) = \mathit{fst}(M_1)$
    $\Longrightarrow$ 4.23,4.5, introduzindo P_1
    $\begin{cases} P = \mathit{fst}(P_1) \\ M_1 = \mathrm{REAL}(P_1?v) \end{cases}$
    $\Longrightarrow$ H.I., introduzindo v_1
    $P_1?v \twoheadrightarrow_{\mathrm{C}} t_1$, onde $\mathrm{REAL}(t_1) = V_1 \neq \langle \cdot, \cdot \rangle$

Por 4.27, $t_1 \twoheadrightarrow_{\mathrm{C}} v_1$ para algum v_1 valor tal que $\mathrm{REAL}(v_1) = \mathrm{REAL}(t_1)$. Como $\mathrm{REAL}(v_1) = V_1 \neq \langle \cdot, \cdot \rangle$, tem-se que v_1 não é um par, por definição de REAL. Tome-se $s_2 = \mathit{Fst}.v_1$. Então $\mathrm{REAL}(s_2) = \mathit{fst}(V_1)$ e $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$, cf.

$$13.\ \frac{\dfrac{P_1?v \twoheadrightarrow_{\mathrm{C}} t_1 \qquad t_1 \twoheadrightarrow_{\mathrm{C}} v_1 \neq (\cdot,\cdot)}{P_1?v \twoheadrightarrow_{\mathrm{C}} v_1 \neq (\cdot,\cdot)}}{\mathit{fst}(P_1)?v \twoheadrightarrow_{\mathrm{C}} \mathit{Fst}.v_1}$$

- Caso $\dfrac{M_1 \twoheadrightarrow_{\mathrm{B}} M_2 \qquad M_2 \twoheadrightarrow_{\mathrm{B}} M_3}{M_1 \twoheadrightarrow_{\mathrm{B}} M_3}$. Imediato, em consequência da transitividade de $\twoheadrightarrow_{\mathrm{C}}$.

□

Teorema 4.29. *Para todo $M \in \mathcal{DBC}$ fechado, $M \twoheadrightarrow_{\mathrm{B}} V$ se e só se $M?() \twoheadrightarrow_{\mathrm{C}} v$, onde $\mathit{REAL}(v) = V$.*

Demonstração. Note-se que $M = \mathrm{REAL}(M?())$, por definição. A implicação "se" é consequência imediata de 4.26 e da observação acabada de fazer. Quanto à implicação "só se", tem-se:

$M \twoheadrightarrow_{\mathrm{B}} V$
$\Longrightarrow$ $M = \mathrm{REAL}(M?())$ e 4.28, introduzindo s_2 estado
$M?() \twoheadrightarrow_{\mathrm{C}} s_2$, onde $\mathrm{REAL}(s_2) = V$
$\Longrightarrow$ 4.27, introduzindo v_2 valor
$M?() \twoheadrightarrow_{\mathrm{C}} s_2 \twoheadrightarrow_{\mathrm{C}} v_2$, onde $\mathrm{REAL}(v_2) = V$
$\Longrightarrow$ transitividade de $\twoheadrightarrow_{\mathrm{C}}$
$M?() \twoheadrightarrow_{\mathrm{C}} v_2$, onde $\mathrm{REAL}(v_2) = V$

□

4.2 CAM versus redução combinatória

Fixamos o processo de compilação de termos de de Bruijn em código CAM através da seguinte função.

Definição 4.30. Define-se a função $\mathit{CAM} : \mathcal{DBC} \to C$ (recorde-se: C representa código da CAM) recursivamente por:

Termo	**Código**	**Pilha**	**Termo**	**Código**	**Pilha**
(v_1, v_2)	$\mathtt{fst}; C$	S	v_1	C	S
v	$\mathtt{fst}; C$	S	$Fst.v$	C	S
(v_1, v_2)	$\mathtt{snd}; C$	S	v_2	C	S
v	$\mathtt{snd}; C$	S	$Snd.v$	C	S
v	$\mathtt{cur}(C); C_1$	S	$\Lambda(CAM^{-1}(C)).v$	C_1	S
v	$\mathtt{quote}(c); C$	S	c	C	S
v	$\mathtt{push}; C$	S	v	C	$v::S$
v_2	$\mathtt{swap}; C$	$v_1::S$	v_1	C	$v_2::S$
v_2	$\mathtt{cons}; C$	$v_1::S$	(v_1, v_2)	C	S
$(\Lambda(M).v_1, v_2)$	$\mathtt{app}; C_1$	S	(v_1, v_2)	$CAM(M)@C_1$	S
v	$\mathtt{app}; C_1$	S	$App.v$	C_1	S

Tabela 4.2: Definição da relação $\rightarrow_{\mathrm{CAM}}$, onde as entradas encontram-se dispostas por ordem decrescente de prioridade de utilização.

1. $CAM('(c)) = \mathtt{quote}(c)$;
2. $CAM(0!) = \mathtt{snd}$;
3. $CAM((n+1)!) = \mathtt{fst}; CAM(n!)$;
4. $CAM(\mathit{fst}(M)) = CAM(M); \mathtt{fst}$;
5. $CAM(\mathit{snd}(M)) = CAM(M); \mathtt{snd}$;
6. $CAM(\langle M, N\rangle) = \mathtt{push}; CAM(M); \mathtt{swap}; CAM(N); \mathtt{cons}$;
7. $CAM(S(M, N)) = \mathtt{push}; CAM(M); \mathtt{swap}; CAM(N); \mathtt{cons}; \mathtt{app}$;
8. $CAM(\Lambda(M)) = \mathtt{cur}(CAM(M))$.

A função CAM é injectiva, pelo que podemos escrever $CAM^{-1}(C) = M$, sempre que $CAM(M) = C$. Esta observação permite poupar a notação $C : v$, reescrevendo valores desta forma como $\Lambda(CAM^{-1}(C)).v$. Veja-se a tabela 4.2.

Definição 4.31. Redefine-se a relação entre estados da CAM, $\rightarrow_{\mathrm{CAM}}$, através da tabela 4.2. Nessa tabela as linhas 2, 4 e 11 devem ser utilizadas quando as respectivas entradas precedentes não forem aplicáveis. Estas entradas servem para lidar com o código "estagnado" (aludido em 4.22), permitindo que a

máquina continue a executar o código e portanto a não bloquear num estado final indesejado.

Lema 4.32. *Para todo v, v_1 valores, C, C_1 códigos e S, S_1 pilhas: se*

$$\{v, C, S\} \rightarrow^*_{\text{CAM}} \{v_1, C_1, S_1\},$$

então, para todo C_2 código e S_2 pilha:

$$\{v, C@C_2, S@S_2\} \rightarrow^*_{\text{CAM}} \{v_1, C_1@C_2, S_1@S_2\}.$$

Demonstração. Por indução sobre $\{v, C, S\} \rightarrow^*_{\text{CAM}} \{v_1, C_1, S_1\}$. O único caso interessante é do passo singular $\rightarrow_{\text{CAM}}$; os restantes casos (i.e. reflexivo e transitivo) são imediatos por reflexividade e transitividade de $\rightarrow^*_{\text{CAM}}$ e H.I.

- Caso $\{(v_1, v_2), \texttt{fst}; C', S\} \rightarrow_{\text{CAM}} \{v_1, C', S\}$. Tem-se, por definição,

$$\{(v_1, v_2), \texttt{fst}; C'@C_2, S@S_2\} \rightarrow_{\text{CAM}} \{v_1, C'@C_2, S@S_2\}.$$

- Caso $\{v, \texttt{fst}; C', S\} \rightarrow_{\text{CAM}} \{Fst.v, C', S\}$. Tem-se, por definição,

$$\{v, \texttt{fst}; C'@C_2, S@S_2\} \rightarrow_{\text{CAM}} \{Fst.v, C'@C_2, S@S_2\}.$$

Os restantes casos são demonstrados precisamente da mesma forma, pelo que os omitimos.

□

Lema 4.33. *Para todo $M \in \mathcal{DBC}$, v valor e S pilha: se*

$$\{v, CAM(M), S\} \rightarrow^*_{\text{CAM}} \{v', [], S'\},$$

então $S = S'$.

Demonstração. Observe-se que cada transição $\rightarrow_{\text{CAM}}$ (linhas da tabela 4.2) possível designa, implicitamente, uma sequência de operações sobre a pilha:

i. As linhas 1–6 não realizam qualquer acção;

ii. A linha 7 coloca um elemento;

iii. A linha 8 retira um elemento e coloca outro;

iv. A linha 9 retira um elemento;

v. As linhas 10 e 11 não realizam qualquer accção.

Tratando-se de uma pilha, sabemos que uma sequência de inserções e remoções balanceada deixa a pilha no seu estado original. Desta feita, basta demonstrar que a função *CAM* produz código cujas acções sobre a pilha resumem-se a uma sequência balanceada de inserções e de remoções. Portanto, defina-se:

1. O conjunto das acções possíveis sobre a pilha por extensão como

$$\{\mathsf{Push}, \mathsf{Pop}, \mathsf{Nop}\},$$

   onde Push denota uma inserção, Pop uma remoção e Nop uma não acção.

2. O conjunto das expressões balanceadas indutivamente por:

   (a) Nop é balanceada;

   (b) $\mathsf{Push}@\alpha@\mathsf{Pop}$ é balanceada, se α é balanceada;

   (c) $\alpha@\beta$ é balanceada, se α e β são balanceadas.

3. Uma correspondência ϕ entre instruções da CAM em sequências de acções sobre a pilha de acordo com a discussão anterior:

   (a) $\phi(\texttt{fst}) = \mathsf{Nop}$ (de acordo com as linhas 1–2);

   (b) $\phi(\texttt{snd}) = \mathsf{Nop}$ (de acordo com as linhas 2–4);

   (c) $\phi(\texttt{cur}(\cdot)) = \mathsf{Nop}$ (de acordo com a linha 5);

   (d) $\phi(\texttt{quote}(\cdot)) = \mathsf{Nop}$ (de acordo com a linha 6);

   (e) $\phi(\texttt{push}) = \mathsf{Push}$ (de acordo com a linha 7);

   (f) $\phi(\texttt{swap}) = \mathsf{Pop}; \mathsf{Push}$ (de acordo com a linha 8);

(g) $\phi(\texttt{cons}) = \mathsf{Pop}$ (de acordo com a linha 9);

(h) $\phi(\texttt{app}) = \mathsf{Nop}$ (de acordo com as linhas 10 e 11).

4. A função ϕ estende-se naturalmente a sequências de instruções da CAM, fazendo-se

$$\phi(i_1; i_2; \ldots; i_n) = \phi(i_1)@\phi(i_2)@\ldots@\phi(i_n),$$

para todo $i_1, i_2, \ldots, i_n$ instrução da CAM.

Resta agora demonstrar que $\phi(CAM(M))$ é uma expressão balanceada, para todo $M \in \mathcal{DBC}$. Procedemos por indução sobre M.

- Caso $M = n!$. Procedemos por indução sobre n.
  - Caso $M = 0!$. Tem-se $\phi(\texttt{snd}) = \mathsf{Nop}$, que é balanceada.
  - Caso $M = (n+1)!$. Tem-se $\phi(\texttt{fst}; CAM(n!)) = \mathsf{Nop}@\phi(CAM(n!))$, que é balanceda, uma vez que $\phi(CAM(n!))$ é balanceada por H.I.
- Caso $M = \,'(c)$. Tem-se $\phi(\texttt{quote}(c)) = \mathsf{Nop}$, que é balanceada.
- Caso $M = \langle M_1, M_2 \rangle$. Tem-se

$$\begin{array}{ll} & \phi(\texttt{push}; CAM(M_1); \texttt{swap}; CAM(M_2); \texttt{cons}) \\ = & \qquad \text{definição, introduzindo } \alpha = \phi(CAM(M_1)) \text{ e } \beta = CAM(M_2) \\ & \mathsf{Push}@\alpha@\mathsf{Pop}; \mathsf{Push}@\beta@\mathsf{Pop} \end{array}$$

que é balanceada, porque:

1. α e β são balanceadas, por H.I.;
2. $\delta_1 = \mathsf{Push}@\alpha@\mathsf{Pop}$ é balanceada, por definição;
3. $\delta_2 = \mathsf{Push}@\beta@\mathsf{Pop}$ é balanceada, por definição.
4. $\delta_1@\delta_2$ é balanceada, por definição.

- Caso $M = S(M_1, M_2)$. Observe-se que

$$CAM(S(M_1, M_2)) = CAM(\langle M_1, M_2\rangle; \mathtt{app},$$

donde podemos repetir o raciocínio anterior e obter a expressão

$$\delta_1 @ \delta_2 @ \mathsf{Nop},$$

uma vez que $\phi(\mathtt{app}) = \mathsf{Nop}$, donde se conclui que também é balanceada.

- Caso $M = \mathit{fst}(M_1)$. Tem-se

$$\begin{array}{ll} & \phi(CAM(M_1); \mathtt{fst}) \\ = & \qquad \text{definição, introduzindo } \alpha = \phi(CAM(M_1)) \\ & \alpha @ \mathsf{Nop} \end{array}$$

que é balanceda, uma vez que α é balanceada por H.I., e Nop e $\alpha @ \mathsf{Nop}$ são balanceadas por definição.

- Caso $M = \mathit{snd}(M_1)$. Análogo ao anterior.

- Caso $M = \Lambda(M_1)$. Tem-se $\phi(\mathtt{cur}(CAM(M_1))) = \mathsf{Nop}$, que é balanceada.

□

Lema 4.34. *Para todo $n \in \mathbb{N}_0$, para todo $m \geqslant n$ e para todo*

$$v = ((), v_m, \ldots, v_n, \ldots, v_0)$$

valor compatível com $n!$, verifica-se:

$$n!?v \twoheadrightarrow_{\mathrm{C}} v_n.$$

Demonstração. Por indução natural sobre n.

- Caso $n = 0$. Temos imediatemente $0!?v \twoheadrightarrow_{\mathrm{C}} v_0$ por definição de $\twoheadrightarrow_{\mathrm{C}}$.

- Caso $n+1$. Seja $v' = ((), v_m, \ldots, v_{n+1}, v_n, \ldots, v_1)$, e logo $v = (v', v_0)$. Temos $(n+1)!?v \twoheadrightarrow_{\mathrm{C}} n!?v'$, por definição de $\twoheadrightarrow_{\mathrm{C}}$. Por H.I. segue que $n!?v' \twoheadrightarrow_{\mathrm{C}} v_{n+1}$, uma vez que v_{n+1} ocorre na n-ésima posição de v'. Logo, por transitividade de $\twoheadrightarrow_{\mathrm{C}}$, $(n+1)!?v \twoheadrightarrow_{\mathrm{C}} v_{n+1}$.

□

Proposição 4.35. *Para todo $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$ e para todo S pilha:*

1. $\{s_1, [], S\} \rightarrow^*_{\mathrm{CAM}} \{s_2, [], S\}$, *se* $s_1 = v$;
2. $\{v, \mathit{CAM}(M), S\} \rightarrow^*_{\mathrm{CAM}} \{v_1, [], S\}$, *se* $s_1 = M?v$ *e* $s_2 = v_1$;
3. $\{v, \mathit{CAM}(M), S\} \rightarrow^*_{\mathrm{CAM}} \{v_1, \mathit{CAM}(M_1), S\}$, *se* $s_1 = M?v$ *e* $s_2 = M_1?v_1$.

Demonstração. Por indução sobre $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$. Omitimos os casos 11. e 12. (referentes às reduções $\mathit{snd}(\cdot)?v \twoheadrightarrow_{\mathrm{C}} \cdot$) por serem análogos aos casos 9. e 10. (referentes às reduções $\mathit{fst}(\cdot)?v \twoheadrightarrow_{\mathrm{C}} \cdot$).

- Caso $v \twoheadrightarrow_{\mathrm{C}} v$. Portanto $s_1 = s_2 = v$, donde $\{s_1, [], S\} \rightarrow^*_{\mathrm{CAM}} \{s_2, [], S\}$ por reflexividade de $\rightarrow^*_{\mathrm{CAM}}$.
- Caso $'(c)?v \twoheadrightarrow_{\mathrm{C}} c$. Portanto $s_1 = '(c)?v$ e $s_2 = c$. Ora, $\{v, \mathtt{quote}(c), S\} \rightarrow_{\mathrm{CAM}} \{c, [], S\}$ por definição.
- Caso $\Lambda(M)?v \twoheadrightarrow_{\mathrm{C}} \Lambda(M).v$. Portanto $s_1 = \Lambda(M)?v$ e $s_2 = \Lambda(M).v$. Ora, $\{v, \mathtt{cur}(\mathit{CAM}(M)), S\} \rightarrow_{\mathrm{CAM}} \{\Lambda(M).v, [], S\}$ por definição.
- Caso $(n+1)!?(v_1, v_2) \twoheadrightarrow_{\mathrm{C}} n!?v_1$. Portanto $s_1 = (n+1)!?(v_1, v_2)$ e $s_2 = n!?v_1$. Ora, $\{(v_1, v_2), \mathtt{fst}; \mathit{CAM}(n!), S\} \rightarrow_{\mathrm{CAM}} \{v_1, \mathit{CAM}(n!), S\}$ por definição.
- Caso $0!?(v_1, v_2) \twoheadrightarrow_{\mathrm{C}} v_2$. Portanto $s_1 = 0!?(v_1, v_2)$ e $s_2 = v_2$. Ora, $\{(v_1, v_2), \mathtt{snd}, S\} \rightarrow_{\mathrm{CAM}} \{v_2, [], S\}$ por definição.
- Caso $\dfrac{M?v \twoheadrightarrow_{\mathrm{C}} \Lambda(M_1).v_1 \qquad N?v \twoheadrightarrow_{\mathrm{C}} v_2}{S(M,N)?v \twoheadrightarrow_{\mathrm{C}} M_1?(v_1, v_2)}$. Portanto $s_1 = S(M,N)?v$ e $s_2 = M_1?(v_1, v_2)$. Ora,

$\{v, \texttt{push}; CAM(M); \texttt{swap}; CAM(N); \texttt{cons}; \texttt{app}, S\}$
$\rightarrow_{\text{CAM}}$
$\{v, CAM(M); \texttt{swap}; CAM(N); \texttt{cons}; \texttt{app}, v::S\}$
$\rightarrow^*_{\text{CAM}}$ H.I., 4.32
$\{\Lambda(M_1).v_1, \texttt{swap}; CAM(N); \texttt{cons}; \texttt{app}, v::S\}$
$\rightarrow_{\text{CAM}}$
$\{v, CAM(N); \texttt{cons}; \texttt{app}, \Lambda(M_1).v_1::S\}$
$\rightarrow^*_{\text{CAM}}$ H.I., 4.32
$\{v_2, \texttt{cons}; \texttt{app}, \Lambda(M_1).v_1::S\}$
$\rightarrow_{\text{CAM}}$
$\{(\Lambda(M_1).v_1, v_2), \texttt{app}, S\}$
$\rightarrow_{\text{CAM}}$
$\{(v_1, v_2), CAM(M_1), S\}$

- Caso $\dfrac{M?v \twoheadrightarrow_{\text{C}} v_1 \neq \Lambda(\cdot).\cdot \qquad N?v \twoheadrightarrow_{\text{C}} v_2}{S(M,N)?v \twoheadrightarrow_{\text{C}} App.(v_1, v_2)}$. Portanto $s_1 = S(M,N)?v$ e $s_2 = App.(v_1, v_2)$. Replicamos o argumento do caso anterior até à penúltima configuração, obtendo-se agora $\{(v_1, v_2), \texttt{app}, S\} \rightarrow_{\text{CAM}} \{App.(v_1, v_2), [], S\}$, por definição.

- Caso $\dfrac{M?v \twoheadrightarrow_{\text{C}} v_1 \qquad N? \twoheadrightarrow_{\text{C}} v_2}{\langle M, N\rangle?v \twoheadrightarrow_{\text{C}} (v_1, v_2)}$. Portanto $s_1 = \langle M, N\rangle?v$ e $s_2 = (v_1, v_2)$. Replicamos o argumento do penúltimo caso até à ante-penúltima configuração, obtendo-se desta vez $\{v_1, \texttt{cons}, v_2::S\} \rightarrow_{\text{CAM}} \{(v_1, v_2), [], S\}$, por definição.

- Caso $\dfrac{M?v \twoheadrightarrow_{\text{C}} (v_1, v_2)}{fst(M)?v \twoheadrightarrow_{\text{C}} v_1}$. Portanto $s_1 = fst(M)?v$ e $s_2 = v_1$. Ora,

$\{v, CAM(M); \texttt{fst}, S\}$
$\rightarrow^*_{\text{CAM}}$ H.I.,4.32
$\{(v_1, v_2), \texttt{fst}, S\}$
$\rightarrow_{\text{CAM}}$
$\{v_1, [], S\}$

- Caso $\dfrac{M?v \twoheadrightarrow_{\text{C}} v_1 \neq (\cdot, \cdot)}{fst(M)?v \twoheadrightarrow_{\text{C}} Fst.v_1}$. Portanto $s_1 = fst(M)?v$ e $s_2 = Fst.v_1$. Ora,

$$
\begin{array}{l}
\quad \{v, CAM(M); \mathtt{fst}, S\} \\
\rightarrow^{*}_{\mathrm{CAM}} \quad \text{H.I.,4.32} \\
\quad \{v_1, \mathtt{fst}, S\} \\
\rightarrow_{\mathrm{CAM}} \\
\quad \{Fst.v_1, [], S\}
\end{array}
$$

- Caso $\dfrac{s_1 \twoheadrightarrow_{\mathrm{C}} s_2 \qquad s_2 \twoheadrightarrow_{\mathrm{C}} s_3}{s_1 \twoheadrightarrow_{\mathrm{C}} s_3}$. Defina-se, para todo s estado e S pilha, a configuração $\Sigma(s, S)$ de acordo com:

  1. $\Sigma(M?v, S) = \{v, CAM(M), S\}$;
  2. $\Sigma(v, S) = \{v, [], S\}$.

  Desta feita, temos por H.I. $\Sigma(s_1, S) \rightarrow^{*}_{\mathrm{CAM}} \Sigma(s_2, S)$ e $\Sigma(s_2, S) \rightarrow^{*}_{\mathrm{CAM}} \Sigma(s_3, S)$. Ora, $\Sigma(s_1, S) \rightarrow^{*}_{\mathrm{CAM}} \Sigma(s_3, S)$ segue por transitividade de $\rightarrow^{*}_{\mathrm{CAM}}$.

□

Proposição 4.36. *Para todo $M \in \mathcal{DBC}$, v valor compatível com M e S pilha: se*

$$\{v, CAM(M), S\} \rightarrow^{*}_{\mathrm{CAM}} \{v', [], S\},$$

então $M?v \twoheadrightarrow_{\mathrm{C}} v'$.

Demonstração. Por indução forte sobre o comprimento da sequência de reduções $\rightarrow^{*}_{\mathrm{CAM}}$.

- Caso $M = \langle M_1, M_2 \rangle$. Por hipótese e 4.33, conclui-se que

  i.

  $$
  \begin{array}{l}
  \quad \{v, CAM(M_1); \mathtt{swap}; CAM(M_2); \mathtt{cons}, v{::}S\} \\
  \rightarrow^{*}_{\mathrm{CAM}} \\
  \quad \{v_1, \mathtt{swap}; CAM(M_2); \mathtt{cons}, v{::}S\}
  \end{array}
  $$

  ii.

$$\begin{array}{l} \quad \{v, CAM(M_2); \mathtt{cons}, v_1{::}S\} \\ \rightarrow^*_{\mathrm{CAM}} \\ \quad \{v_2, \mathtt{cons}, v_1{::}S\} \end{array}$$

para alguns v_1, v_2 valores. Assim sendo, temos $v' = (v_1, v_2)$ e, por H.I.,

I. $M_1?v \twoheadrightarrow_{\mathrm{C}} v_1$; II. $M_2?v \twoheadrightarrow_{\mathrm{C}} v_2$.

Falta ver que $\langle M_1, M_2\rangle?v \twoheadrightarrow_{\mathrm{C}} (v_1, v_2)$. Ora,

$$8.\dfrac{\text{I. } \dfrac{}{M_1? \twoheadrightarrow_{\mathrm{C}} v_1} \qquad \text{II. } \dfrac{}{M_2?v \twoheadrightarrow_{\mathrm{C}} v_2}}{\langle M_1, M_2\rangle?v \twoheadrightarrow_{\mathrm{C}} (v_1, v_2)}$$

- Caso $M = S(M_1, M_2)$. Observe-se que
  - $CAM(S(M_1, M_2)) = CAM(\langle M_1, M_2\rangle); \mathtt{app}$,

  donde podemos repetir o raciocínio empregue no caso $M = \langle M_1, M_2\rangle$ e concluir, a partir da hipótese, que

$$\begin{array}{l} \quad \{v, CAM(\langle M_1, M_2\rangle); \mathtt{app}, S\} \\ \rightarrow^*_{\mathrm{CAM}} \\ \quad \{(v_1, v_2), \mathtt{app}, S\} \end{array}$$

  donde, de forma análoga, se extrai, por H.I.,

  - $M_1?v \twoheadrightarrow_{\mathrm{C}} v_1$; - $M_2?v \twoheadrightarrow_{\mathrm{C}} v_2$.

  A computação pode prosseguir de duas formas distintas, conforme a forma do valor v_1:

  - Caso $v_1 = \Lambda(M_3).v_3$, para algum $M_3 \in \mathcal{DBC}$ e v_3 valor. Então

$$\begin{array}{l} \quad \{(v_1, v_2), \mathtt{app}, S\} \\ \rightarrow_{\mathrm{CAM}} \\ \quad \{(v_3, v_2), CAM(M_3), S\} \\ \rightarrow^*_{\mathrm{CAM}} \quad \text{hipótese e 4.33} \\ \quad \{v', [], S\} \end{array}$$

donde podemos concluir, por H.I., que $M_3?(v_3, v_2) \twoheadrightarrow_{\mathrm{C}} v'$. Assim sendo, tem-se:

$$13.\dfrac{6.\dfrac{\text{H.I.}\dfrac{}{M_1?v \twoheadrightarrow_{\mathrm{C}} \Lambda(M_3).v_3} \quad \text{H.I.}\dfrac{}{M_2?v \twoheadrightarrow_{\mathrm{C}} v_2}}{S(M_1, M_2)?v \twoheadrightarrow_{\mathrm{C}} M_3?(v_3, v_2)} \quad \text{H.I.}\dfrac{}{M_3?(v_3, v_2) \twoheadrightarrow_{\mathrm{C}} v'}}{S(M_1, M_2)?v \twoheadrightarrow_{\mathrm{C}} v'}$$

– Caso contrário. Então

$$\begin{array}{l} \{(v_1, v_2), \mathtt{app}, S\} \\ \rightarrow_{\mathrm{CAM}} \\ \{App.(v_1, v_2), [], S\} \end{array}$$

donde se retira que $v' = App.(v_1, v_2)$ e, assim sendo, tem-se:

$$7.\dfrac{\text{H.I.}\dfrac{}{M_1?v \twoheadrightarrow_{\mathrm{C}} v_1} \quad \text{H.I.}\dfrac{}{M_2?v \twoheadrightarrow_{\mathrm{C}} v_2}}{S(M_1, M_2)?v \twoheadrightarrow_{\mathrm{C}} App.(v_1, v_2)}$$

- Caso $M = \Lambda(M_1)$. Temos $\{v, \mathtt{cur}(M_1), S\} \rightarrow_{\mathrm{CAM}} \{\Lambda(M_1).v, [], S\}$ donde $v' = \Lambda(M_1).v$. Assim sendo, temos trivialmente

$$3.\dfrac{}{\Lambda(M_1)?v \twoheadrightarrow_{\mathrm{C}} \Lambda(M_1).v}.$$

- Caso $M = \mathit{fst}(M_1)$. Temos, por hipótese e 4.33

$$\begin{array}{l} \{v, CAM(M_1); \mathtt{fst}, S\} \\ \rightarrow^{*}_{\mathrm{CAM}} \\ \{v_1, \mathtt{fst}, S\} \end{array}$$

donde, por H.I. segue que $M_1?v \twoheadrightarrow_{\mathrm{C}} v_1$. A computação pode prosseguir de duas formas distintas, conforme a forma de v_1:

– Caso $v_1 = (v_2, v_3)$, para alguns v_2, v_3 valores. Então

$$\begin{array}{l} \{v_1, \mathtt{fst}, S\} \\ \rightarrow_{\mathrm{CAM}} \\ \{v_2, [], S\} \end{array}$$

donde $v' = v_2$. Assim sendo, temos

$$9.\dfrac{\overset{\text{H.I.}}{\dfrac{}{M_1?v \twoheadrightarrow_{\mathrm{C}} (v_2, v_3)}}}{\mathit{fst}(M_1)?v \twoheadrightarrow_{\mathrm{C}} v_2}$$

– Caso contrário. Então

$$\begin{array}{l} \{v_1, \mathtt{fst}, S\} \\ \rightarrow_{\mathrm{CAM}} \\ \quad \{\mathit{Fst}.v_1, [], S\} \end{array}$$

donde $v' = \mathit{Fst}.v_1$. Assim sendo, temos

$$10.\dfrac{\overset{\text{H.I.}}{\dfrac{}{M_1?v \twoheadrightarrow_{\mathrm{C}} v_1}}}{\mathit{fst}(M_1)?v \twoheadrightarrow_{\mathrm{C}} \mathit{Fst}.v_1}$$

- Caso $M = \mathit{snd}(M_1)$. Análogo ao caso anterior.

- Caso $M = \mathit{'}(c)$. Temos $\{v, \mathtt{quote}(c), S\} \rightarrow_{\mathrm{CAM}} \{c, [], S\}$, donde $v' = c$. Assim sendo, temos trivialmente

$$2.\dfrac{}{\mathit{'}(c)?v \twoheadrightarrow_{\mathrm{C}} c}.$$

- Caso $M = n!$. Pelo facto de v ser compatível com M, tem-se que $v = ((), v_k, \ldots, v_n, \ldots, v_0)$, para algum $k \geqslant n$. Ora

$$\begin{array}{l} \{v, \underbrace{\mathtt{fst}; \ldots, \mathtt{fst}}_{n}; \mathtt{snd}, S\} \\ \rightarrow^{*}_{\mathrm{CAM}} \\ \quad \{(v_{n+1}, v_n), \mathtt{snd}, S\} \\ \rightarrow_{\mathrm{CAM}} \\ \quad \{v_n, [], S\} \end{array}$$

donde se conclui que $v' = v_n$. Falta ver que $n!?v \twoheadrightarrow_{\mathrm{C}} v_n$, o que é dado por 4.34.

□

Teorema 4.37. *Para todo $M \in \mathcal{DBC}$, v valor compatível com M e S pilha, $M?v \twoheadrightarrow_{\mathrm{C}} v_1$ se e só se $\{v, CAM(M), S\} \rightarrow^*_{\mathrm{CAM}} \{v_1, [], S\}$.*

Demonstração. Por dupla implicação.

$$
\begin{array}{ll}
 & M?v \twoheadrightarrow_{\mathrm{C}} v_1 \\
\Longrightarrow & \quad 4.35 \\
 & \{v, CAM(M), S\} \rightarrow^*_{\mathrm{CAM}} \{v_1, [], S\} \\
\Longrightarrow & \quad 4.36 \\
 & M?v \twoheadrightarrow_{\mathrm{C}} v_1
\end{array}
$$

□

Os resultados obtidos até agora já nos permitem concluir que a CAM implementa correctamente a estratégia de redução *call-by-value* do Cálculo λc dotado de constantes. Vejamos como.

Teorema 4.38. *Para todo $M \in \mathcal{DBC}$ fechado, são equivalentes:*

1. $M \twoheadrightarrow_{\mathrm{B}} V$;
2. $\{(), CAM(M), []\} \rightarrow^*_{\mathrm{CAM}} \{v, [], []\}$, *onde* $REAL(v) = V$.

Demonstração. Por transitividade de $\Longleftrightarrow$ no seguinte argumento.

$$
\begin{array}{ll}
 & M \twoheadrightarrow_{\mathrm{B}} V \\
\Longleftrightarrow & \quad 4.29 \\
 & M?() \twoheadrightarrow_{\mathrm{C}} v, \text{ onde } \mathrm{REAL}(v) = V \\
\Longleftrightarrow & \quad \text{Instância } (S = []) \text{ de } 4.37 \\
 & \{(), CAM(M), []\} \rightarrow^*_{\mathrm{CAM}} \{v, [], []\}, \text{ onde } \mathrm{REAL}(v) = V
\end{array}
$$

□

Recordemos o diagrama 4.1 da correcção visto no início deste capítulo:

$$
\begin{array}{ccc}
M & \xrightarrow{\mathrm{CBV}} & V \\
{\scriptstyle \mathsf{Load}}\downarrow & & \uparrow{\scriptstyle \mathsf{Unload}} \\
s & \xrightarrow{\mathsf{CAM}} & t
\end{array}
\tag{4.2}
$$

O enunciando do teorema 4.38 descreve precisamente este diagrama uma vez definidas as funções Load e Unload da seguinte forma:

$$\mathsf{Load}(M) = \{(), \mathit{CAM}(M), []\}$$
$$\mathsf{Unload}(\{v, [], []\}) = \mathrm{REAL}(v).$$

4.3 Extensões

Nesta secção estendemos o resultado 4.38 ao Cálculo λ dotado de condicionais e de um operador de ponto-fixo, como aludido na secção 3.3. Basta para isso introduzirmos os devidos termos de de Bruijn e de seguida revisitarmos os resultados que permitiram chegar ao teorema da última secção.

Condicional

Definição 4.39 (Revisita a 4.2)**.** Introduzimos, para todo $M_1, M_2, M_3 \in \mathcal{DBC}$, o termo-$\mathcal{DBC}$ $\mathit{Cond}(M_1, M_2, M_3)$. Introduzimos também as constantes true e false.

Definição 4.40 (Revisita a 4.12, 4.19)**.** Ao nível da redução combinatória, adicionamos:

1. O valor $\mathit{Branch}(M_1, M_2).(v_1, v_2)$, para todo $M_1, M_2 \in \mathcal{DBC}$ e v_1, v_2 valores tais que $v_1 \notin \{\mathsf{true}, \mathsf{false}\}$;

2. As reduções

   (a) $$\frac{M_1?v \twoheadrightarrow_{\mathrm{C}} \mathsf{true}}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} M_2?v};$$

   (b) $$\frac{M_1?v \twoheadrightarrow_{\mathrm{C}} \mathsf{false}}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} M_3?v};$$

   (c) $$\frac{M_1?v \twoheadrightarrow_{\mathrm{C}} v_1 \notin \{\mathsf{true}, \mathsf{false}\}}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} \mathit{Branch}(M_2, M_3).(v, v_1)}.$$

Definição 4.41 (Revisita a 4.9, 4.10)**.** Ao nível da redução call-by-value, adicionamos:

Termo	Código	Pilha	Termo	Código	Pilha
(v, true)	$\mathtt{branch}(C_1, C_2); C$	S	v	$C_1@C$	S
(v, false)	$\mathtt{branch}(C_1, C_2); C$	S	v	$C_2@C$	S
(v, v_1)	$\mathtt{branch}(C_1, C_2); C$	S	$\mathit{Branch}(M_1, M_2).(v, v_1)$	C	S

Tabela 4.3: Actualização da relação $\rightarrow_{\mathrm{CAM}}$, onde $M_1 = \mathit{CAM}^{-1}(C_1)$ e $M_2 = \mathit{CAM}^{-1}(C_2)$.

1. O valor $\mathit{Cond}(V, M_2, M_3)$, para todo $M_2, M_3 \in \mathcal{DBC}$ e V valor tal que $V \notin \{'(\mathsf{true}), '(\mathsf{false})\}$.

2. As reduções:

   (a) $$\frac{M_1 \twoheadrightarrow_{\mathrm{B}} '(\mathsf{true})}{\mathit{Cond}(M_1, M_2, M_3) \twoheadrightarrow_{\mathrm{B}} M_2};$$

   (b) $$\frac{M_1 \twoheadrightarrow_{\mathrm{B}} '(\mathsf{false})}{\mathit{Cond}(M_1, M_2, M_3) \twoheadrightarrow_{\mathrm{B}} M_3};$$

   (c) $$\frac{M_1 \twoheadrightarrow_{\mathrm{B}} V \notin \{'(\mathsf{true}), '(\mathsf{false})\}}{\mathit{Cond}(M_1, M_2, M_3) \twoheadrightarrow_{\mathrm{B}} \mathit{Cond}(V, M_2, M_3)}.$$

Definição 4.42 (Revisita a 4.23)**.** Actualizamos a função REAL com o caso:

1. $\mathrm{REAL}(\mathit{Branch}(M_2, M_3).(v, v_1)) = \mathit{Cond}(\mathrm{REAL}(v_1), \mathrm{REAL}(M_2?v), \mathrm{REAL}(M_3?v))$.

Definição 4.43 (Revisita a 4.5)**.** Actualizamos a função de substituição com o seguinte caso:

1. $\mathit{Cond}(M_1, M_2, M_3)[\vec{N}] = \mathit{Cond}(M_1[\vec{N}], M_2[\vec{N}], M_3[\vec{N}])$.

Definição 4.44 (Revisita a 4.30, 4.31)**.** Actualizamos a função CAM com o caso

1. $\mathit{CAM}(\mathit{Cond}(M_1, M_2, M_3)) = \mathtt{push}; \mathit{CAM}(M_1); \mathtt{cons}; \mathtt{branch}(\mathit{CAM}(M_2), \mathit{CAM}(M_3))$

e a redução $\rightarrow_{\mathrm{CAM}}$ com os casos da tabela 4.3.

Demonstração (Revisita a 4.25). Caso $v = \mathit{Branch}(M_2, M_3).(v, v_1)$, onde $v_1 \notin \{\mathsf{true}, \mathsf{false}\}$. Ora,

$$\begin{array}{ll} & \mathrm{REAL}(\mathit{Branch}(M_2, M_3).(v, v_1)) \\ = & \quad 4.23 \\ & \mathit{Cond}(\mathrm{REAL}(v_1), \mathrm{REAL}(M_2?v), \mathrm{REAL}(M_3?v)) \end{array}$$

Por H.I. $\mathrm{REAL}(v_1)$ é um valor. Como $v_1 \notin \{\mathsf{true}, \mathsf{false}\}$, vem que $\mathrm{REAL}(v_1) \notin \{'(\mathsf{true}), '(\mathsf{false})\}$, por definição de REAL. Logo,

$$\mathit{Cond}(\mathrm{REAL}(v_1), \mathrm{REAL}(M_2?v), \mathrm{REAL}(M_3?v))$$

é um valor, por definição. □

Demonstração (Revisita a 4.26).

- Caso $\dfrac{M_1?v \twoheadrightarrow_{\mathrm{C}} \mathsf{true}}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} M_2?v}$. Por H.I. tem-se que

  $$\mathrm{REAL}(M_1?v) \twoheadrightarrow_{\mathrm{B}} '(\mathsf{true}).$$

  Logo,

  $$\begin{array}{ll} & \mathrm{REAL}(\mathit{Cond}(M_1, M_2, M_3)?v) \\ = & \quad 4.23,\ 4.5 \\ & \mathit{Cond}(\mathrm{REAL}(M_1?v), \mathrm{REAL}(M_2?v), \mathrm{REAL}(M_3?v)) \\ \twoheadrightarrow_{\mathrm{B}} & \quad \text{H.I.} \\ & \mathrm{REAL}(M_2?v) \end{array}$$

- Caso $\dfrac{M_1?v \twoheadrightarrow_{\mathrm{C}} v_1 \notin \{\mathsf{true}, \mathsf{false}\}}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} \mathit{Branch}(M_2, M_3).(v, v_1)}$. Por H.I. tem-se que (onde $V_1 = \mathrm{REAL}(v_1)$)

  $$\mathrm{REAL}(M_1?v) \twoheadrightarrow_{\mathrm{B}} V_1 \notin \{'(\mathsf{true}), '(\mathsf{false})\}.$$

  Logo,

  $$\begin{array}{ll} & \mathrm{REAL}(\mathit{Cond}(M_1, M_2, M_3)?v) \\ = & \quad 4.23 \\ & \mathit{Cond}(\mathrm{REAL}(M_1?v), \mathrm{REAL}(M_2?v), \mathrm{REAL}(M_3?v)) \\ \twoheadrightarrow_{\mathrm{B}} & \quad \text{H.I.} \end{array}$$

$$
\begin{array}{ll}
 & \mathit{Cond}(V_1, \mathrm{REAL}(M_2?v), \mathrm{REAL}(M_3?v)) \\
= & \quad 4.23 \\
 & \mathrm{REAL}(\mathit{Branch}(M_2, M_3).(v, v_1))
\end{array}
$$

□

Demonstração (Revisita a 4.27). Caso $M = \mathit{Cond}(M_1, M_2, M_3)$. Observe-se que

$$
\begin{array}{l}
\quad \mathrm{REAL}(\mathit{Cond}(M_1, M_2, M_3)?v) = V \\
\Longrightarrow \quad 4.23,\ 4.5 \\
\quad \mathit{Cond}(\mathrm{REAL}(M_1?v), \mathrm{REAL}(M_2?v), \mathrm{REAL}(M_3?v)) = V \\
\Longrightarrow \quad 4.9, \text{ introduzindo } V_1, N_2, N_3 \\
\quad \begin{cases} V = \mathit{Cond}(V_1, N_2, N_3) \\ V_1 = \mathrm{REAL}(M_1?v) \notin \{'(\mathsf{true}), '(\mathsf{false})\} \\ N_2 = \mathrm{REAL}(M_2?v) \\ N_3 = \mathrm{REAL}(M_3?v) \end{cases} \\
\Longrightarrow \quad \text{H.I., introduzindo } v_1 \\
\quad M_1?v \twoheadrightarrow_{\mathrm{C}} v_1, \text{ onde } \mathrm{REAL}(v_1) = V_1
\end{array}
$$

Como v_1 é um valor e $\mathrm{REAL}(V_1) \notin \{'(\mathsf{true}), '(\mathsf{false})\}$, vem que $v_1 \notin \{\mathsf{true}, \mathsf{false}\}$. Posto isto, suponhamos que $\mathrm{REAL}(\mathit{Cond}(M_1, M_2, M_3)?v) = V$. Tome-se $v' = \mathit{Branch}(M_2, M_3).(v, v_1)$. Então, por definição de REAL e de $\twoheadrightarrow_{\mathrm{C}}$, $\mathrm{REAL}(v') = V$ e $\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} v'$.

□

Demonstração (Revisita a 4.28).

- Caso $\dfrac{M_1 \twoheadrightarrow_{\mathrm{B}} '(\mathsf{true})}{\mathit{Cond}(M_1, M_2, M_3) \twoheadrightarrow_{\mathrm{B}} M_2}$. Seja s_1 tal que $\mathrm{REAL}(s_1) = \mathit{Cond}(M_1, M_2, M_3)$. Procedemos por casos de s_1.

  - Caso $s_1 = v$, para algum valor v. Mostramos que este caso conduz a uma contradição, donde se conclui que é impossível ter-se, simultaneamente, $s_1 = v$, $\mathrm{REAL}(s_1) = \mathit{Cond}(M_1, M_2, M_3)$ e $M_1 \twoheadrightarrow_{\mathrm{B}} '(\mathsf{true})$. Tem-se

$$
\begin{array}{l}
\quad \mathrm{REAL}(s_1) = \mathrm{REAL}(v) = \mathit{Cond}(M_1, M_2, M_3) \\
\Longrightarrow \quad \text{4.23, introduzindo } v_1, v_2, N_2, N_3 \\
\quad \begin{cases} v = \mathit{Branch}(N_1, N_2).(v_1, v_2), \text{ onde } v_2 \notin \{\mathsf{true}, \mathsf{false}\} \\ M_1 = \mathrm{REAL}(v_2) \\ M_2 = \mathrm{REAL}(N_2?v_1) \\ M_3 = \mathrm{REAL}(N_3?v_1) \end{cases} \\
\Longrightarrow \quad \text{H.I., introduzindo } t_2 \\
\quad v_2 \twoheadrightarrow_{\mathrm{C}} t_2, \text{ onde } \mathrm{REAL}(t_2) = \mathit{'}(\mathsf{true})
\end{array}
$$

Como v_2 é um valor e $v_2 \twoheadrightarrow_{\mathrm{C}} t_2$, temos que $v_2 = t_2$. Por $\mathrm{REAL}(t_2) = \mathit{'}(\mathsf{true})$, temos que $t_2 = \mathsf{true}$, pela definição de REAL. Portanto, $v_2 \notin \{\mathsf{true}, \mathsf{false}\}$, $v_2 = t_2$ e $t_2 = \mathsf{true}$: contradição!

– Caso $s_1 = P?v$, para algum $P \in \mathcal{DBC}$ e v valor. Temos

$$
\begin{array}{l}
\quad \mathrm{REAL}(s_1) = \mathrm{REAL}(P?v) = \mathit{Cond}(M_1, M_2, M_3) \\
\Longrightarrow \quad \text{4.23, 4.5, introduzindo } P_1, P_2, P_3 \\
\quad \begin{cases} P = \mathit{Cond}(P_1, P_2, P_3) \\ M_1 = \mathrm{REAL}(P_1?v) \\ M_2 = \mathrm{REAL}(P_2?v) \\ M_3 = \mathrm{REAL}(P_3?v) \end{cases} \\
\Longrightarrow \quad \text{H.I., introduzindo } t_1 \\
\quad P_1?v \twoheadrightarrow_{\mathrm{C}} t_1, \text{ onde } \mathrm{REAL}(t_1) = \mathit{'}(\mathsf{true})
\end{array}
$$

Por 4.27, $t_1 \twoheadrightarrow_{\mathrm{C}} v_1$ para algum valor v_1 tal que $\mathrm{REAL}(v_1) = \mathrm{REAL}(t_1) = \mathit{'}(\mathsf{true})$. Logo, $v_1 = \mathsf{true}$ pela definição de REAL. Tome-se $s_2 = P_2?v$. Então $\mathrm{REAL}(s_2) = M_2$ e $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$.

- Caso $\dfrac{M_1 \twoheadrightarrow_{\mathrm{B}} V_1 \notin \{\mathit{'}(\mathsf{true}), \mathit{'}(\mathsf{false})\}}{\mathit{Cond}(M_1, M_2, M_3) \twoheadrightarrow_{\mathrm{B}} \mathit{Cond}(V_1, M_2, M_3)}$. Seja s_1 tal que $\mathrm{REAL}(s_1) = \mathit{Cond}(M_1, M_2, M_3)$. Procedemos por casos de s_1.

– Caso $s_1 = v$, para algum valor v. Temos

$$
\begin{array}{l}
\quad \mathrm{REAL}(s_1) = \mathrm{REAL}(v) = \mathit{Cond}(M_1, M_2, M_3) \\
\Longrightarrow \quad \text{4.23, introduzindo } v_1, v_2, N_2, N_3
\end{array}
$$

$$\begin{cases} v = \mathit{Branch}(N_1, N_2).(v_1, v_2), \text{ onde } v_2 \notin \{\mathsf{true}, \mathsf{false}\} \\ M_1 = \mathrm{REAL}(v_2) \\ M_2 = \mathrm{REAL}(N_2?v_1) \\ M_3 = \mathrm{REAL}(N_3?v_1) \end{cases}$$
$$\Longrightarrow \quad \text{H.I., introduzindo } t_2$$
$$v_2 \twoheadrightarrow_{\mathrm{C}} t_2, \text{ onde } \mathrm{REAL}(t_2) = V_1$$

Como v_2 é um valor e $v_2 \twoheadrightarrow_{\mathrm{C}} t_2$ temos que $v_2 = t_2$. Logo, $\mathrm{REAL}(v_2) = V_1$. Tome-se $s_2 = s_1 = \mathit{Branch}(N_2, N_3).(v_1, v_2)$. Então $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$ e $\mathrm{REAL}(s_2) = \mathit{Cond}(V_1, M_2, M_3)$.

– Caso $s_1 = P?v$, para algum $P \in \mathcal{DBC}$ e v valor. Temos

$$\mathrm{REAL}(s_1) = \mathrm{REAL}(P?v) = \mathit{Cond}(M_1, M_2, M_3)$$
$$\Longrightarrow \quad \text{4.23, 4.5, introduzindo } P_1, P_2, P_3$$
$$\begin{cases} P = \mathit{Cond}(P_1, P_2, P_3) \\ M_1 = \mathrm{REAL}(P_1?v) \\ M_2 = \mathrm{REAL}(P_2?v) \\ M_3 = \mathrm{REAL}(P_3?v) \end{cases}$$
$$\Longrightarrow \quad \text{H.I., introduzindo } t_1$$
$$P_1?v \twoheadrightarrow_{\mathrm{C}} t_1, \text{ onde } \mathrm{REAL}(t_1) = V_1 \notin \{'(\mathsf{true}), '(\mathsf{false})\}$$

Por 4.27, $t_1 \twoheadrightarrow_{\mathrm{C}} v_1$ para algum valor v_1 tal que $\mathrm{REAL}(v_1) = \mathrm{REAL}(t_1) = V_1$. Logo, $v_1 \notin \{\mathsf{true}, \mathsf{false}\}$ pela definição de REAL. Tome-se $s_2 = \mathit{Branch}(P_2, P_3).(v, v_1)$. Então $\mathrm{REAL}(s_2) = \mathit{Cond}(V_1, M_2, M_3)$ e $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$.

□

Demonstração (Revisita a 4.32). Encaixa-se naturalmente nos casos da demonstração original. □

Demonstração (Revisita a 4.33). Caso $M = \mathit{Cond}(M_1, M_2, M_3)$. Note-se que $\phi(\mathtt{branch}(\cdot, \cdot)) = \mathsf{Nop}$. Tem-se:

$$\begin{aligned} & \phi(\mathtt{push}; \mathit{CAM}(M_1); \mathtt{cons}; \mathtt{branch}(\mathit{CAM}(M_2), \mathit{CAM}(M_3))) \\ = & \quad \text{definição, introduzindo } \alpha = \phi(\mathit{CAM}(M_1)) \\ & \mathsf{Push}@\alpha@\mathsf{Pop}; \mathsf{Nop} \end{aligned}$$

que é balanceada, porque:

1. α é balanceada, por H.I.;
2. $\mathsf{Push}@\alpha@\mathsf{Pop}$ é balanceada, por definição;
3. Nop é balanceada, por definição;
4. $(\mathsf{Push}@\alpha@\mathsf{Pop})@\mathsf{Nop}$ é balanceada, por definição.

□

Demonstração (Revisita a 4.35).

- Caso $\dfrac{M_1?v \twoheadrightarrow_{\mathrm{C}} \mathsf{true}}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} M_2?v}$. Portanto, $s_1 = \mathit{Cond}(M_1, M_2, M_3)?v$ e $s_2 = M_2?v$. Sejam $C_i = \mathit{CAM}(M_i)$, $i \in \{1, 2, 3\}$. Tem-se:

$$
\begin{array}{ll}
 & \{v, \mathtt{push}; C_1; \mathtt{cons}; \mathtt{branch}(C_2, C_3), S\} \\
\rightarrow_{\mathrm{CAM}} & \\
 & \{v, C_1; \mathtt{cons}; \mathtt{branch}(C_2, C_3), v::S\} \\
\rightarrow^{*}_{\mathrm{CAM}} & \text{H.I., 4.32} \\
 & \{\mathsf{true}, \mathtt{cons}; \mathtt{branch}(C_2, C_3), v::S\} \\
\rightarrow_{\mathrm{CAM}} & \\
 & \{(v, \mathsf{true}), \mathtt{branch}(C_2, C_3), S\} \\
\rightarrow_{\mathrm{CAM}} & \\
 & \{v, C_2, S\}
\end{array}
$$

- Caso $\dfrac{M_1?v \twoheadrightarrow_{\mathrm{C}} v_1 \neq \{\mathsf{true}, \mathsf{false}\}}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} \mathit{Branch}(M_2, M_3).(v, v_1)}$. Portanto $s_1 = \mathit{Cond}(M_1, M_2, M_3)?v$ e $s_2 = \mathit{Branch}(M_2, M_3).(v, v_1)$. Temos (onde novamente $C_i = \mathit{CAM}(M_i)$):

$$
\begin{array}{ll}
 & \{v, \mathtt{push}; C_1; \mathtt{cons}; \mathtt{branch}(C_2, C_3), S\} \\
\rightarrow^{*}_{\mathrm{CAM}} & \text{H.I., 4.32} \\
 & \{(v, v_1), \mathtt{branch}(C_2, C_3), S\} \\
\rightarrow_{\mathrm{CAM}} & \mathit{CAM}^{-1}(C_i) = M_i \\
 & \{\mathit{Branch}(M_2, M_3).(v, v_1), [], S\}
\end{array}
$$

□

Demonstração (Revisita a 4.36). Caso $M = \mathit{Cond}(M_1, M_2, M_3)$. Por hipótese e por 4.33, conclui-se que

i.

$$\begin{array}{l} \quad \{v, \mathit{CAM}(M_1); \mathtt{cons}; \mathtt{branch}(\mathit{CAM}(M_2), \mathit{CAM}(M_3)), v{::}S\} \\ \rightarrow^*_{\mathrm{CAM}} \\ \quad \{v_1, \mathtt{cons}; \mathtt{branch}(\mathit{CAM}(M_2), \mathit{CAM}(M_3)), v{::}S\} \end{array}$$

ii.

$$\begin{array}{l} \quad \{(v, v_1), \mathtt{branch}(\mathit{CAM}(M_2), \mathit{CAM}(M_3)), S\} \\ \rightarrow_{\mathrm{CAM}} \\ \quad \{v', C', S\} \end{array}$$

para alguns v_1, v' valores e C' código. Por i. e por H.I. conclui-se que $M_1?v \twoheadrightarrow_{\mathrm{C}} v_1$. Temos três possibilidades:

- Caso $v_1 = \mathsf{true}$. Então $v' = v$ e $C' = \mathit{CAM}(M_2)$ e

  $$\begin{array}{ll} \quad \{v', \mathit{CAM}(M_2), S\} & \\ \rightarrow^*_{\mathrm{CAM}} & \text{hipótese e 4.33} \\ \quad \{v'', [], S\} & \end{array}$$

  para algum v'' valor, donde podemos concluir (por H.I.) que $M_2?v' \twoheadrightarrow_{\mathrm{C}} v''$. Assim sendo, $\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} v''$, cf.

  $$13.\frac{\dfrac{M_1?v \twoheadrightarrow_{\mathrm{C}} \mathsf{true}}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} M_2?v} \qquad M_2?v \twoheadrightarrow_{\mathrm{C}} v''}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} v''}$$

- Caso $v_1 = \mathsf{false}$. Análogo ao caso anterior.

- Caso $v_1 \notin \{\mathsf{true}, \mathsf{false}\}$. Então $v' = \mathit{Branch}(M_2, M_3).(v, v_1)$ e $C' = []$. Assim sendo, $\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} \mathit{Branch}(M_2, M_3).(v, v_1)$, cf.

$$\frac{M_1?v \twoheadrightarrow_{\mathrm{C}} v_1 \notin \{\mathsf{true}, \mathsf{false}\}}{\mathit{Cond}(M_1, M_2, M_3)?v \twoheadrightarrow_{\mathrm{C}} \mathit{Branch}(M_2, M_3).(v, v_1)}.$$

□

Ponto-fixo

Definição 4.45 (Revisita a 4.2)**.** Introduzimos, para todo $M \in \mathcal{DBC}$, o termo-$\mathcal{DBC}$ $\mathit{FIX}(M)$ com o intuito de representar o termo-λc $Y(\Lambda(\Lambda(M)))$.

Definição 4.46 (Revisita a 4.10)**.** Actualizamos a redução $\twoheadrightarrow_{\mathrm{B}}$ com o caso

1. $\mathit{FIX}(M) \twoheadrightarrow_{\mathrm{B}} \Lambda(M)[\mathit{FIX}(M)]$.

Observação 4.47 (Racional para 4.46)**.** Imediato, uma vez que no Cálculo λc temos a redução $Y(\Lambda(\Lambda(M))) \to \Lambda(M)[Y(\Lambda(\Lambda(M)))]$ e $\mathit{FIX}(M)$ pretende representar $Y(\Lambda(\Lambda(M))$.

Definição 4.48 (Revisita a 4.12, 4.19)**.** Do lado da redução combinatória incluimos o valor

1. $M!v$, para todo $M \in \mathcal{DBC}$ e v valor compatível com M.

e as reduções

1. $\mathit{FIX}(M)?v \twoheadrightarrow_{\mathrm{C}} M!v$;
2. $\dfrac{M_1?v \twoheadrightarrow_{\mathrm{C}} \mathsf{fix} M_3 v_1 \qquad M_2?v \twoheadrightarrow_{\mathrm{C}} v_2}{S(M_1, M_2)?v \twoheadrightarrow_{\mathrm{C}} M_3?(v_1, M_3!v_1, v_2)}$.

de acordo com a discussão em 3.3.

Definição 4.49 (Revisita a 4.5)**.** Actualizamos a função de substituição com o caso

1. $\mathit{FIX}(M)[\vec{N}] = \mathit{FIX}(M[0!, 1!, \vec{N}])$.

Observação 4.50 (Racional para 4.49)**.** Como $FIX(M)$ representa $Y(\Lambda(\Lambda(M)))$, queremos que $FIX(M)[\vec{N}]$ represente $Y(\Lambda(\Lambda(M)))[\vec{N}]$. Ora,

$$
\begin{array}{ll}
 & Y(\Lambda(\Lambda(M)))[\vec{N}] \\
= & \quad \text{definição de substituição (semelhante a 4.5)} \\
 & Y[\vec{N}]\Lambda(\Lambda(M))[\vec{N}] \\
= & \quad Y \text{ é uma constante; definição de substituição} \\
 & Y(\Lambda(\Lambda(M[0,1,\vec{N}])))
\end{array}
$$

Logo, $FIX(M)[\vec{N}]$ — $Y(\Lambda(\Lambda(M)))[\vec{N}]$ — deve ser igual a $FIX(M[0!,1!,\vec{N}])$ — $Y(\Lambda(\Lambda(M[0!,1!,\vec{N}])))$.

Definição 4.51 (Revisita a 4.23)**.** Actualizamos a função REAL com o caso

1. $\mathrm{REAL}(M!(),v_n,\ldots,v_0)) = \Lambda(M)[FIX(M)[\vec{V}],\vec{V}]$,

onde $\vec{V} = \mathrm{REAL}(v_0),\mathrm{REAL}(v_1),\ldots,\mathrm{REAL}(v_n)$.

Observação 4.52 (Racional para 4.51)**.** Como $M!v$ representa $\Lambda(M).(v,M!v)$, queremos satisfazer $\mathrm{REAL}(M!v) = \mathrm{REAL}(\Lambda(M).(v,M!v))$. Ora,

$$
\begin{array}{ll}
 & \mathrm{REAL}(\Lambda(M).(v,M!v)) \\
= & \quad 4.23 \\
 & \Lambda(M)[\mathrm{REAL}(M!v),\mathrm{REAL}(v_0),\ldots,\mathrm{REAL}(v_n)]
\end{array}
$$

Ou seja, $\mathrm{REAL}(M!v)$ deve satisfazer a equação sobre X:

$$
\begin{array}{ll}
 & X = \Lambda(M)[X,\mathrm{REAL}(v)] \\
\Longleftrightarrow & \quad 4.8.1 \\
 & X = \Lambda(M)[0!,\mathrm{REAL}(v)][X] \\
\Longleftrightarrow & \quad 4.5 \\
 & X = \Lambda(M[0!,1!,\mathrm{REAL}(v)])[X]
\end{array}
$$

Observe-se que $FIX(M[0!,1!,\mathrm{REAL}(v)])$ satisfaz esta equação, cf.

$$
\begin{array}{ll}
 & FIX(M[0!,1!,\mathrm{REAL}(v)]) \\
=_{\mathrm{B}} & \quad 4.46 \\
 & \Lambda(M[0!,1!,\mathrm{REAL}(v)])[FIX(M[0!,1!,\mathrm{REAL}(v)])]
\end{array}
$$

Termo	**Código**	**Pilha**	**Termo**	**Código**	**Pilha**
v	$\texttt{fix}(C_1); C$	S	$CAM^{-1}(C_1)!v$	C	S
$(M!v_1, v_2)$	$\texttt{app}; C$	S	$(v_1, M!v_1, v_2)$	$CAM(M)@C$	S

Tabela 4.4: Actualização da relação $\rightarrow_{\text{CAM}}$.

Portanto,

$$\begin{array}{ll} & \text{REAL}(M!v) = \Lambda(M[0!, 1!, \text{REAL}(v)])[FIX(M[0!, 1!, \text{REAL}(v)])] \\ \Longleftrightarrow & \quad \text{Simplificando (4.49, 4.8.1, 4.5)} \\ & \text{REAL}(M!v) = \Lambda(M)[FIX(M)[\text{REAL}(v)], \text{REAL}(v)] \end{array}$$

Definição 4.53 (Revisita a 4.30, 4.31)**.** Actualizamos a função CAM com o caso

1. $CAM(FIX(M)) = \texttt{fix}(CAM(M))$

e a redução $\rightarrow_{\text{CAM}}$ com os casos da tabela 4.4.

Demonstração (Revisita a 4.25). Caso $v = M!v_1$. Tem-se

$$\begin{array}{ll} & \text{REAL}(M!v_1) \\ = & \quad \text{4.51, introduzindo } \vec{V_1} = \text{REAL}(v_{1_0}), \ldots, \text{REAL}(v_{1_n}) \\ & \Lambda(M)[FIX(M)[\vec{V_1}], \vec{V_1}] \end{array}$$

que é um valor, cf. 4.9. □

Demonstração (Revisita a 4.26).

- Caso $FIX(M)?v \twoheadrightarrow_{\text{C}} Fix(M, v)$. Tem-se

$$\begin{array}{ll} & \text{REAL}(FIX(M)?v) \\ = & \quad \text{4.23, introduzindo } \vec{V} = \text{REAL}(v_0), \ldots, \text{REAL}(v_n) \\ & FIX(M)[\vec{V}] \\ = & \quad \text{4.49} \\ & FIX(M[0!, 1!, \vec{V}]) \\ \twoheadrightarrow_{\text{B}} & \quad \text{4.46} \\ & \Lambda(M[0!, 1!, \vec{V}])[FIX(M[0!, 1!, \vec{V}])] \\ = & \quad \text{4.5} \end{array}$$

$$
\begin{array}{ll}
& \Lambda(M)[0!, \vec{V}][FIX(M[0!, 1!, \vec{V}])] \\
= & \quad 4.8.1 \\
& \Lambda(M)[FIX(M[0!, 1!, \vec{V}]), \vec{V}] \\
= & \quad 4.49 \\
& \Lambda(M)[FIX(M)[\vec{V}], \vec{V}] \\
= & \quad 4.51 \\
& \mathrm{REAL}(M!v)
\end{array}
$$

- Caso $\dfrac{M_1?v \twoheadrightarrow_{\mathrm{C}} M_3!v_1 \qquad N?v \twoheadrightarrow_{\mathrm{C}} v_2}{S(M_1, M_2)?v \twoheadrightarrow_{\mathrm{C}} M_3?(v_1, M_3!v_1, v_2)}$. Por H.I. temos

  i.

$$
\begin{array}{ll}
& \mathrm{REAL}(M_1?v) \twoheadrightarrow_{\mathrm{B}} REAL(M_3!v_1) \\
\Longleftrightarrow & \quad 4.51, \text{ introduzindo } \vec{V_1} = \mathrm{REAL}(v_{1_0}), \ldots, \mathrm{REAL}(v_{1_n}) \\
& \mathrm{REAL}(M_1?v) \twoheadrightarrow_{\mathrm{B}} \Lambda(M_3)[FIX(M_3)[\vec{V_1}], \vec{V_1}] \\
\Longleftrightarrow & \quad 4.5{,}4.49 \\
& \mathrm{REAL}(M_1?v) \twoheadrightarrow_{\mathrm{B}} \Lambda(M_3[0!, FIX(M_3[0!, 1!, \vec{V_1}]), \vec{V_1}])
\end{array}
$$

  ii.

$$
\begin{array}{ll}
& \mathrm{REAL}(M_2?v) \twoheadrightarrow_{\mathrm{B}} \mathrm{REAL}(v_2) \\
\Longleftrightarrow & \quad \text{introduzindo } V_2 = \mathrm{REAL}(v_2) \\
& \mathrm{REAL}(M_2?v) \twoheadrightarrow_{\mathrm{B}} V_2
\end{array}
$$

  Logo,

$$
\begin{array}{ll}
& \mathrm{REAL}(S(M_1, M_2)?v) \\
= & \quad 4.23{,}4.5 \\
& S(\mathrm{REAL}(M_1?v), \mathrm{REAL}(M_2?v)) \\
\twoheadrightarrow_{\mathrm{B}} & \quad \mathrm{i.,ii.,}4.10 \\
& M_3[0!, FIX(M_3[0!, 1!, \vec{V_1}]), \vec{V_1}])[V_2] \\
= & \quad 4.8.1 \\
& M_3[V_2, FIX(M_3[0, !, 1!, \vec{V_1}]), \vec{V_1}] \\
= & \quad 4.49 \\
& M_3[V_2, FIX(M_3)[\vec{V_1}], \vec{V_1}] \\
= & \quad 4.51 \\
& \mathrm{REAL}(M_3?(v_1, M_3!v_1, v_2))
\end{array}
$$

□

Demonstração (Revisita a 4.27). Caso $M = FIX(M_1)$. Tem-se

$$\begin{array}{ll} & \mathrm{REAL}(FIX(M_1)?v) \\ = & \quad 4.23 \\ & FIX(M_1)[\mathrm{REAL}(v_0), \ldots, \mathrm{REAL}(v_n)] \\ = & \quad 4.49 \\ & FIX(M_1[0!, 1!, \mathrm{REAL}(v_0), \ldots, \mathrm{REAL}(v_n)]) \end{array}$$

que não é um valor, donde terminamos. □

Demonstração (Revisita a 4.28). Caso $FIX(M) \twoheadrightarrow_{\mathrm{B}} \Lambda(M)[FIX(M)]$. Seja s_1 tal que $\mathrm{REAL}(s_1) = FIX(M)$. Procedemos por casos de s_1.

- Caso $s_1 = v$, para algum valor v. Tem-se

$$\begin{array}{ll} & \mathrm{REAL}(v) = FIX(M) \\ \Longrightarrow & \quad 4.51, \text{ introduzindo } M_1,\ v_1 \text{ e } \vec{V_1} = \mathrm{REAL}(v_{1_0}), \ldots, \mathrm{REAL}(v_{1_n}) \\ & \begin{cases} v = M_1!v_1 \\ M = M_1[0!, 1!, \vec{V_1}] \end{cases} \end{array}$$

  Tome-se $s_2 = s_1 = M_1!v_1$. Então $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$ e $\mathrm{REAL}(s_2) = \Lambda(M)[FIX(M)]$, cf.

$$\begin{array}{ll} & \mathrm{REAL}(M_1!v_1) \\ = & \quad 4.51 \\ & \Lambda(M_1)[FIX(M_1)[\vec{V_1}], \vec{V_1}] \\ = & \quad 4.8.1 \\ & \Lambda(M_1)[0!, \vec{V_1}][FIX(M_1)[\vec{V_1}]] \\ = & \quad 4.5 \\ & \Lambda(M_1[0!, 1!, \vec{V_1}])[FIX(M_1)[\vec{V_1}]] \\ = & \quad 4.49 \\ & \Lambda(M_1[0!, 1!, \vec{V_1}])[FIX(M_1[0!, 1!, \vec{V_1}])] \\ = & \quad M = M_1[0!, 1!, \vec{V_1}] \\ & \Lambda(M)[FIX(M)] \end{array}$$

- Caso $s_1 = P?v$, para algum $P \in \mathcal{DBC}$ e v valor. Tem-se

$$\begin{array}{ll} & \mathrm{REAL}(P?v) = \mathit{FIX}(M) \\ \Longrightarrow & \text{4.23,4.49, introduzindo } \vec{V} = \mathrm{REAL}(v_0), \ldots, \mathrm{REAL}(v_n) \\ & \begin{cases} P = \mathit{FIX}(M_1) \\ M = M_1[0!, 1!, \vec{V}] \end{cases} \end{array}$$

  Tome-se $s_2 = M_1!v$. Então $s_1 \twoheadrightarrow_{\mathrm{C}} s_2$ e $\mathrm{REAL}(s_2) = \Lambda(M)[\mathit{FIX}(M)]$ (procedendo-se tal como no caso anterior).

□

Demonstração (Revisita a 4.32 e 4.33). A demonstração de 4.32 transita naturalmente; quanto a 4.33, observa-se que a acção de $\mathit{FIX}(\cdot)$ sobre a pilha é Nop, que é balanceada. □

Demonstração (Revisita a 4.35).

- Caso $\mathit{FIX}(M)?v \twoheadrightarrow_{\mathrm{C}} M!v$. Portanto $s_1 = \mathit{FIX}(M)?v$ e $s_2 = M!v$. Temos, trivialmente:

$$\begin{array}{ll} & \{v, \mathtt{fix}(\mathit{CAM}(M)), S\} \\ \rightarrow_{\mathrm{CAM}} & \mathit{CAM}^{-1}(\mathit{CAM}(M)) = M \\ & \{M!v, [], S\} \end{array}$$

- Caso $\dfrac{M_1?v \twoheadrightarrow_{\mathrm{C}} \mathit{Fix}(M_3, v_1) \qquad M_2?v \twoheadrightarrow_{\mathrm{C}} v_2}{S(M_1, M_2)?v \twoheadrightarrow_{\mathrm{C}} M_3?(v_1, M_3!v_1, v_2)}$. Portanto, $s_1 = S(M_1, M_2)?v$ e $s_2 = M_3?(v_1, M_3!v_1, v_2)$. Ora,

$$\begin{array}{ll} & \{v, \mathtt{push}; \mathit{CAM}(M_1); \mathtt{swap}; \mathit{CAM}(M_2); \mathtt{cons}; \mathtt{app}, S\} \\ \rightarrow_{\mathrm{CAM}} & \\ & \{v, \mathit{CAM}(M_1); \mathtt{swap}; \mathit{CAM}(M_2); \mathtt{cons}; \mathtt{app}, v::S\} \\ \rightarrow^*_{\mathrm{CAM}} & \text{H.I., 4.32} \\ & \{M_3!v_1, \mathtt{swap}; \mathit{CAM}(M_2); \mathtt{cons}; \mathtt{app}, v::S\} \\ \rightarrow_{\mathrm{CAM}} & \\ & \{v, \mathit{CAM}(M_2); \mathtt{cons}; \mathtt{app}, M_3!v_1::S\} \\ \rightarrow^*_{\mathrm{CAM}} & \text{H.I., 4.32} \end{array}$$

$$
\begin{array}{l}
\quad \{v_2, \mathtt{cons}; \mathtt{app}, M_3!v_1::S\} \\
\rightarrow_{\mathrm{CAM}} \\
\quad \{(M_3!v_1, v_2), \mathtt{app}, S\} \\
\rightarrow_{\mathrm{CAM}} \\
\quad \{(v_1, M_3!v_1, v_2), \mathit{CAM}(M_3), S\}
\end{array}
$$

□

Demonstração (Revisita a 4.36). Caso $M = \mathit{FIX}(M_1)$. Tem-se

$$
\begin{array}{l}
\quad \{v, \mathtt{fix}(\mathit{CAM}(M_1)), S\} \\
\rightarrow_{\mathrm{CAM}} \\
\quad \{M_1!v, [], S\}
\end{array}
$$

e $\mathit{FIX}(M_1)?v \twoheadrightarrow_{\mathrm{C}} M_1!v$, por definição de $\twoheadrightarrow_{\mathrm{C}}$. □

4.4 Epílogo

As revisitas efectuadas ao longo da secção anterior permitem concluir novamente o teorema 4.38 desta vez incluindo os termos de de Bruijn $\mathit{Cond}(\cdot,\cdot,\cdot)$ e $\mathit{FIX}(\cdot)$, obtendo-se com isso que a CAM implementa correctamente a estratégia de redução *call-by-value* para o Cálculo λc de de Bruijn dotado, para além de constantes, de condicional e de ponto-fixo.

Discutimos agora a correcção da implementação pela CAM de uma linguagem de programação funcional. Podemos resumir o teorema 4.38 no seguinte diagrama:

(4.3)

$$
\begin{array}{ccc}
M_{DB} & \xrightarrow{\twoheadrightarrow_B} & V_{DB} \\
\downarrow & \mathcal{DBC} \text{ vs. CCL} & \uparrow \\
M_{DB}?((),\ldots) & \xrightarrow{\twoheadrightarrow_C} & v \\
\downarrow & \text{CCL vs. CAM} & \uparrow \\
\{((),\ldots), \mathit{CAM}(M_{DB}), []\} & \xrightarrow{\twoheadrightarrow_{\mathrm{CAM}}} & \{v, [], []\}
\end{array}
$$

Apesar de não o provarmos neste trabalho, assumimos, por se tratarem de meras variantes, que são equivalentes as estratégias de redução *call-by-value* no Cálculo λc e no Cálculo λc com índices de de Bruijn. Ou seja, assumimos que o seguinte diagrama comuta.

$$\begin{array}{ccc} M & \xrightarrow{\twoheadrightarrow_{\mathrm{CBV}}} & V \\ {\scriptstyle \cdot_{DB}}\downarrow & \Lambda \text{ vs. } \mathcal{DBC} & \downarrow{\scriptstyle \cdot_{DB}} \\ M_{DB} & \xrightarrow{\twoheadrightarrow_{B}} & V_{DB} \end{array} \tag{4.4}$$

Considere-se a linguagem $\mathcal{F}$ que serviu de exemplo no capítulo anterior. A linguagem $\mathcal{F}$ é apenas uma versão "açucarada" do Cálculo λc com as extensões que temos vindo a considerar. Ou seja, assumimos também a comutatividade do seguinte diagrama:

$$\begin{array}{ccc} M & \xrightarrow{\twoheadrightarrow_{\mathcal{F}}} & V \\ \downarrow & \mathcal{F} \text{ vs. } \Lambda & \downarrow \\ M & \xrightarrow{\twoheadrightarrow_{\mathrm{CBV}}} & V \end{array} \tag{4.5}$$

Somando as partes, obtém-se o diagrama

$$\begin{array}{ccc} M & \xrightarrow{\twoheadrightarrow_{\mathcal{F}}} & V \\ \vdots & \mathcal{F} \text{ vs. } \Lambda & \vdots \\ M & \xrightarrow{\twoheadrightarrow_{\mathrm{CBV}}} & V \\ {\scriptstyle \cdot_{DB}}\vdots & \Lambda \text{ vs. } \mathcal{DBC} & \vdots{\scriptstyle \cdot_{DB}} \\ M_{DB} & \xrightarrow{\twoheadrightarrow_{B}} & V_{DB} \\ \downarrow & \mathcal{DBC} \text{ vs. CCL} & \uparrow \\ M_{DB}.((),\dots) & \xrightarrow{\twoheadrightarrow_{C}} & v \\ \downarrow & \text{CCL vs. CAM} & \uparrow \\ \{((),\dots), M_{DB}, []\} & \xrightarrow{\twoheadrightarrow_{\mathrm{CAM}}} & \{v, [], []\} \end{array} \tag{4.6}$$

no qual o tracejado realça os resultados não demonstrados neste trabalho. Na

implementação da linguagem $\mathcal{F}$ do capítulo anterior, observa-se que a função $[\![\cdot]\!]. : \mathcal{F} \to C$ utilizada consiste na composição das setas do lado esquerdo deste último diagrama módulo optimizações ao nível do código-máquina gerado, donde a sua correcção segue imediatamente.

Capítulo 5

Notas históricas

A primeira máquina abstracta para a implementação de linguages de programação funcionais a ser publicada foi a SECD, em 1964, por P. J. Landin (1930–2009). Em (Landin, 1964) o autor idealizou uma linguagem de programação funcional[1], mostrou como esta podia ser traduzida em termos-λ, e por fim concebeu um processo mecânico — a máquina abstracta SECD — com o qual podia animar a execução destes termos-λ. Deste empreendimento resultaram várias ideias em voga ainda à data, a saber:

1. Definição de tipos de dados estruturados;
2. Expressões `let ... in ...`, `... where ...` e `letrec ...`;
3. Terminologia *closure*;
4. Animação de linguagens de programação através de máquinas abstractas.

Dez anos mais tarde, em 1974, G. D. Plotkin (1946–) demonstrou a correcção da SECD ao relacionar a linguagem de programação ISWIM com o Cálculo λ (Plotkin, 1975). Na verdade, o seu trabalho a este respeito vai mais longe: para além de demonstrar que a SECD implementa a estratégia de redução *call-by-value*, também mostra como simular a estratégia de redução *call-by-name* através da estratégia *call-by-value* e vice-versa.

[1] Mais tarde devidamente formalizada sob o nome ISWIM, do inglês "If You See What I Mean" (Landin, 1966).

Em 1977, D. A. Turner (1946–) tirou proveito da relação entre o Cálculo λ e a Lógica Combinatória (Clássica) ao implementar uma linguagem de programação funcional[2] *lazy* através da compilação para termos combinatoriais e posterior avaliação por passos de reescrita de acordo com as regras da Lógica Combinatória (Turner, 1979). A execução *lazy* significa que os argumentos das funções são substituídos verbatim nos corpos destas. Este aspecto, até então, levantava problemas de eficiência por tipicamente levar à avaliação em repetido dos argumentos das funções nas diversas partes dos seus corpos. Turner colmatou este problema aproveitando também a *redução normal de grafos* apresentada em (Wadsworth, 1971), permitindo-lhe assim partilhar redexes em memória e, consequentemente, avaliar os argumentos das funções, no máximo, uma única vez.

Turner apelidou o seu processo de reescrita por SK-Machine, em honra aos combinadores S e K. A SK-Machine é bastante diferente da SECD: a SECD incorpora um compartimento distinto (o chamado "ambiente") no qual se encontram os valores de todas as variáveis do programa, sendo em função deste ambiente que se determina o resultado final do programa; a SK-Machine, por seu lado, apenas conhece o programa, o qual manipula continuamente via uma filosofia de simplificações até este atingir uma forma irredutível, a partir do qual se extrai o resultado final do programa. Em (Turner, 1979, Secção Results and Conclusions), Turner constatou que a SK-Machine é mais adequada, em termos de eficiência, para a implementação de linguagens *lazy*, enquanto que a SECD é mais adequada para a implementação de linguagens *strict*.

Em meados da década de 1980, após Pierre-Louis Curien (1963–) ter introduzido a Lógica Combinatória Categorial (Curien, 1986), Guy Cousineau (1949–) constatou que a tradução de termos-λc em termos categoriais poderia ser vista como uma técnica de compilação (Cousineau, 1996). Desta observação nasceu a Máquina Abstracta Categorial (CAM), que foi utilizada de imediato para implementar uma variante da linguagem de programação ML: a linguagem Caml[3], em 1987. Esta implementação da Caml foi desenvolvida até 1992 por Ascánder Suárez (principalmente), Pierre Weis e Michel Mauny,

[2]SASL, do inglês "St. Andrews Static Language".

estudantes de doutoramento de Cousineau. A tese de doutoramento de Suárez (Suárez, 1993), em particular, debruça-se sobre a implementação da linguagem ML através da CAM onde, entre outras coisas, reúne várias optimizações possíveis.

A CAM é, tal como a SECD, uma máquina de ambiente. De facto, a única diferença essencial entre estas duas máquinas reside na forma como salvaguardam os valores (ou ambientes): na SECD esta tarefa é realizada numa só instrução, ao passo que na CAM esta tarefa é difundida em várias instruções (`push`, `cons`, `swap`); de acordo com (Cousineau et al., 1987, p. 184), esta diferença torna a prova da correcção da CAM mais simples.

A implementação inicial da linguagem Caml levada a cabo por Cousineau e seus estudantes era suportada na verdade por uma outra máquina abstracta para a linguagem Lisp. Esta abordagem simplificou o trabalho dos implementadores, mas penalizou os seus utilizadores: por um lado exigia uma máquina com mais recursos de memória e processamento; por outro, tornava impraticável a sua utilização para tarefas "mais pesadas". Para além disso, esta implementação apenas funcionava em certo hardware, o que limitava consideravelmente a sua audiência (Cousineau, 1996; Leroy, 1990).

Em 1990–1991, Xavier Leroy (1968–), que conhecia a implementação da linguagem Caml quer do ponto de vista do implementador quer do ponto de vista do utilizador, decidiu reescrevê-la tendo em mente os seguintes pontos (Leroy, 1990, Chapter 1):

1. A implementação devia ser autónoma e portável, colmatando assim os problemas da implementação original acabados de relatar.

2. A implementação devia ser simples e permitir estender as funcionalidades da linguagem facilmente.

3. A implementação devia ser documentada. Este ponto deve-se ao facto de, à data, a tese de doutoramento de Suárez ainda não se encontrar concluída e o código-fonte disponível ainda estar por documentar. Por

[3]Caml resulta da junção das palavras CAM e ML.

> outro lado, também não existia bibliografia dedicada à implementação de linguagens de programação funcionais *strict*.

Estas eram as suas preocupações mais pragmáticas. Uma outra preocupação era tornar a aplicação n-ária eficiente. Na CAM, a aplicação de uma função f a n argumentos $a_1, a_2, \ldots, a_n$,

$$(\ldots((fa_1)a_2)\ldots)a_n,$$

leva à construção de $n-1$ "closures" intermédias, o que tem um impacto severo na eficiência. Este problema, porém, não é particular à CAM.

Leroy conseguiu satisfazer todos estes pontos e o resultado foi o sistema ZINC[4] que tinha por base uma máquina abstracta diferente da CAM: a ZAM[5]. Curiosamente, e sem que Leroy se apercebesse[6], a ZAM consiste numa adaptação de uma outra máquina abstracta conhecida na altura, mas que só recentemente foi publicada pelo seu autor, Jean-Louis Krivine (1939–): a máquina de Krivine, ou K-Machine, ou KAM (Krivine, 2007). Originalmente, a KAM serve para a avaliação call-by-name, mas a adaptação de Leroy permite que esta realize a avaliação call-by-value (Leroy, 1990, Chapter 3).

O sistema ZINC de Leroy substituiu a implementação original da linguagem Caml, dando origem à linguagem "Caml Light". Em 1996, esta linguagem cresceu na linguagem "Objective Caml" utilizada na actualidade (2014), com a ZAM ainda por base.

[4] Acrónimo para "ZINC is not Caml".

[5] De "ZINC Abstract Machine".

[6] Foi Curien quem observou a Leroy que a ZAM se tratava, essencialmente, da KAM (Leroy, 1990, Secção 3.3).

Capítulo 6

Conclusão

Contributos Grosso modo este trabalho segue a apresentação original dos conteúdos realizada em (Curien, 1986), (Curien, 1993) e (Cousineau et al., 1987). Contudo, observamos as seguintes diferenças:

- No capítulo 2, porque incluímos a demonstração de certos exercícios, demonstrámos efectivamente todos os resultados necessários para observarmos a correspondência entre a Lógica Combinatória Categorial e o Cálculo λc. Nalgumas demonstrações divergimos do autor original fazendo uso das reduções em vez das conversões, obtendo-se pontualmente um resultado mais preciso. A demonstração dos resultados 2.30 e 2.32 em particular, cremos serem uma melhoria em relação aos originais, vide os comentários 2.31, 2.33.

- No capítulo 3 realçamos a forma diferente de (Cousineau et al., 1987) como apresentámos as extensões condicional e ponto-fixo uma vez que procurávamos incluí-las na demonstração da correcção da CAM. A implementação da linguagem de programação funcional $\mathcal{F}$ descrita neste capítulo pode ser utilizada interactivamente em `http://github.com/xavierpinho/CAM`.

- No capítulo 4 detalhámos a demonstração da correcção delineada em (Cousineau et al., 1987), incluindo com isso alguns lemas omitidos originalmente. Alargámos posteriormente a demonstração por forma a

incluir as extensões constante, condicional e ponto-fixo.

Para além disto, no capítulo 5 fizemos um breve enquadramento histórico em volta da CAM, onde relatámos como esta surgiu e como deixou de ser usada.

Sugestão de continuação Neste trabalho debruçámo-nos sobre a estratégia de redução *call-by-value*, apenas. Uma continuação possível seria explorar o caso da redução *call-by-name*. Observações a este respeito podem ser encontradas desde já na literatura principal sobre a CAM, e.g. (Cousineau et al., 1987) ou (Curien, 1993).

Bibliografia

Barendregt, H. (1974). Pairing Without Conventional Restraints. Mathematical Logic Quarterly *20*, 289–306.

Barendregt, H. (1997). The impact of the lambda calculus in logic and computer science. Bulletin of Symbolic Logic *3*.

Cardone, F. and Hindley, J. R. (2006). History of Lambda-calculus and Combinatory Logic. Swansea University Mathematics Research Report No. MRRS-05-06.

Church, A. (1941). The Calculi of Lambda-Conversion. Princeton University Press.

Cousineau, G. (1996). A brief history of Caml. `http://www.pps.univ-paris-diderot.fr/~cousinea/Caml/caml_history.html`. Consultado a 16 de Outubro de 2014.

Cousineau, G., Curien, P.-L. and Mauny, M. (1987). The categorical abstract machine. Science of Computer Programming *8*, 173 – 202.

Curien, P.-L. (1986). Categorical combinators. Information and Control *69*, 188 – 254.

Curien, P.-L. (1993). Categorical Combinators, Sequential Algorithms, and Functional Programming (2nd ed.), Chapter 1. Birkhauser Boston Inc., Cambridge, MA, USA.

Curry, H. B. and Feyes, R. (1958). Combinatory Logic, vol. 1,. North-Holland Publishing Company, Amsterdam.

de Bruijn, N. (1972). Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. Indagationes Mathematicae (Proceedings) *75*, 381 – 392.

Hankin, C. (1994). Lambda Calculi: A Guide for Computer Scientists. Clarendon Press.

Hindley, J. R. and Seldin, J. P. (2008). Lambda Calculus and Combinators: An Introduction. Cambridge University Press.

Krivine, J. (2007). A call-by-name lambda-calculus machine. Higher-Order and Symbolic Computation *20*, 199–207.

Lambek, J. and Scott, P. J. (1986). Introduction to higher order categorical logic. Cambridge University Press, New York, NY, USA.

Landin, P. J. (1964). The Mechanical Evaluation of Expressions. The Computer Journal *6*, 308–320.

Landin, P. J. (1966). The next 700 programming languages. Commun. ACM *9*, 157–166.

Leroy, X. (1990). The ZINC experiment: an economical implementation of the ML language. Technical report 117 INRIA.

Mezghiche, M. (1984). Une nouvelle cβ-réduction dans la logique combinatoire. Theoretical Computer Science *31*, 151 – 163.

Mezghiche, M. (1989). On pseudo-cβ normal form in combinatory logic. Theoretical Computer Science *66*, 323 – 331.

Peyton Jones, S. L. (1987). The Implementation of Functional Programming Languages. Prentice-Hall, Inc.

Plotkin, G. (1975). Call-by-name, call-by-value and the lambda-calculus. Theoretical Computer Science *1*, 125 – 159.

Seldin, J. P. (2011). The search for a reduction in combinatory logic equivalent to -reduction. Theoretical Computer Science *412*, 4905 – 4918.

Sestoft, P. (2002). Demonstrating Lambda Calculus Reduction. In The Essence of Computation, (Mogensen, T., Schmidt, D. and Sudborough, I., eds), vol. 2566, of Lecture Notes in Computer Science pp. 420–435. Springer Berlin Heidelberg.

Sørensen, M. H. and Urzyczyn, P. (2006). Lectures on the Curry-Howard Isomorphism, Volume 149 (Studies in Logic and the Foundations of Mathematics). Elsevier Science Inc., New York, NY, USA.

Suárez, A. (1993). Une implementation de ML en ML. Université Paris VII.

Turner, D. A. (1979). A New Implementation Technique for Applicative Languages. Softw., Pract. Exper. *9*, 31–49.

Wadsworth, C. P. (1971). Semantics and Pragmatics of the lambda-calculus. Oxford University; DPhil. Thesis.

Çağman, N. and Hindley, J. (1998). Combinatory weak reduction in lambda calculus. Theoretical Computer Science *198*, 239 – 247.