Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Matemática

Comparando Cálculos de Substituições Explícitas com *Eta*-conversão

por

Flávio Leonardo Cavalcanti de Moura

Brasília, 05 de abril de 2002

Aos meus pais,

José Leonardo de Moura e *Petrúcia de Holanda Cavalcanti*.

Resumo

O λ-cálculo é uma importante ferramenta teórica no campo da computação, essencial para o desenvolvimento de noções como computabilidade assim como para a implementação de linguagens de programação funcional tipadas e assistentes de prova. A noção de substituição utilizada para a definição da β-conversão é uma meta-operação que não está formalmente definida no λ-cálculo. Por esta razão, apesar de termos uma notação concisa para algoritmos, não estamos aptos para analisar complexidade de tempo ou espaço dos mesmos. Na intenção de resolver esse problema foram criadas algumas variações do λ-cálculo, que manipulam a operação de substituição explicitamente, como, por exemplo, $\lambda\sigma$, λs_e, λu, etc. Neste trabalho estamos interessados em comparar três desses cálculos: o $\lambda\sigma$, o λs_e e o Cálculo de Suspensão, que aqui denotaremos por λ_{Susp}. Inicialmente introduzimos uma regra *Eta* para o λ_{Susp} e mostramos que o cálculo de substituições explícitas obtido ao se incluir esta nova regra é convergente. Depois mostramos que o λ_{Susp} e o $\lambda\sigma$ são incomparáveis entre si e, que o λs_e é mais adequado que o λ_{Susp}. Utilizamos a linguagem de programação funcional Ocaml, para implementarmos um ambiente que simula os sistemas de reescrita dos cálculos $\lambda\sigma$, λs_e e λ_{Susp}.

Introdução

O λ-cálculo surgiu no início do século XX com os trabalhos de Alonzo Church [Chu32] e [Chu33]. Esses estudos faziam parte de teorias mais gerais de funções e lógica de ordem superior, cujo intuito era servir como fundamento para a Matemática. Quando Kleene & Rosser [KR35] mostraram a inconsistência dessas teorias, Church abandonou o programa de fundamentos e extraiu a subteoria referente à parte funcional que hoje corresponde ao que chamamos de λ-cálculo, cuja consistência foi mostrada por [CR36].

Conhecido como o primeiro sistema de reescrita no contexto computacional, o λ-cálculo é uma teoria que modela funções computáveis, com uma notação compacta para algoritmos, mas que no entanto, não nos fornece meios para estimar importantes noções da computação como, por exemplo, complexidade de tempo ou espaço para estes algoritmos. As operações básicas do λ-cálculo são a β-conversão e a η-conversão, sendo a primeira um mecanismo básico para aplicar funções abstratas a argumentos e, a segunda uma simples abstração para a equivalência funcional. Além disso, uma terceira operação, denominada a α-conversão, permite mudar nomes de variáveis.

A operação de substituição utilizada para definir a β-conversão do λ-cálculo é uma operação implícita que não está definida formalmente. Muito trabalho tem sido desenvolvido desde então no intuito de tentar controlar melhor essa operação como, por exemplo, [CF58] e [Cur86]. O trabalho de Curien [Cur86] constituiu a base para o trabalho de Abadi *et al.* [ACCL91] onde foi apresentado o primeiro cálculo de substituições explícitas, chamado de $\lambda\sigma$. Os cálculos de substituições explícitas constituem uma variação do λ-cálculo onde se tenta controlar explicitamente a operação de substituição. Aqui estudaremos três desses cálculos: o $\lambda\sigma$, o λs_e [KR97] e o Cálculo de Suspensão [NW98], que aqui denotaremos por λ_{Susp}. O λ_{Susp} foi desenvolvido por Nadathur e Wilson quando eles estavam interessados em utilizar λ-termos como dispositivos computacionais em questões pertinentes à im-

plementação do λProlog [NM88], uma linguagem de programação lógica que utiliza λ-termos como estrutura de dados.

Nosso objetivo é comparar o λ_{Susp} com os cálculos $\lambda\sigma$ e λs_e. Compararemos no estilo de [KR00], onde Kamareddine e Ríos mostraram, entre outras coisas, que os cálculos $\lambda\sigma$ e λs (uma versão restrita do λs_e para termos sem meta-variáveis) são incomparáveis.

O λ_{Susp} foi apresentado sem a regra Eta[†]. Aqui introduziremos uma regra Eta que preserva a terminalidade e a confluência da parte de substituição deste cálculo. Posteriormente mostraremos a correspondência entre as regras Eta desses cálculos.

Outra parte interessante deste trabalho consiste de uma implementação desenvolvida com o intuito de simular as derivações de um termo no λ_{Susp}, no λs_e ou no $\lambda\sigma$. A linguagem utilizada para esta implementação foi o Ocaml, da família ML, e que utiliza tipagem implícita. Aqui apresentaremos os primeiros resultados obtidos com essa implementação, em particular, mostraremos que a implemetação da regra Eta para cada um dos cálculos aqui tratados é correta.

A presente dissertação está dividida em quatro capítulos de acordo com a seguinte ordem:

O primeiro capítulo contém os resultados e noções importantes sobre teoria de reescrita e λ-cálculo. Um cuidado especial é dado para a seção que trata da notação de De Bruijn haja vista que esta notação é utilizada pelos cálculos tratados. Na notação de De Bruijn, variáveis são substituídas por índices representados por números naturais. Esses índices representam o número de abstratores que envolvem aquela variável. Uma consequência imediata é que termos em notação de De Bruijn são insensíveis a α-conversão. Por esse motivo podemos citar aqui, fora as vantagens computacionais óbvias dessa indexação, pelo menos duas grandes vantagens em se utilizar essa notação. A primeira diz respeito ao fato de que a equivalência entre λ-termos, módulo renomeamento de variáveis, é imediata enquanto que na notação convencional estabelecer a dita equivalência, em geral, tem um alto custo computacional. A segunda vantagem está no fato de que não precisamos nos preocupar com captura de variáveis livres após β-conversões.

O segundo capítulo versa sobre substituições explícitas. Apresentaremos os cálculos de substituições explícitas com os quais trabalharemos assim como suas propriedades. Ainda no capítulo 2 introduziremos a regra Eta para o λ_{Susp} e mostraremos a terminalidade e confluência da parte de substituição deste cálculo

[†]Restringiremos o uso do nome η ao λ-cálculo puro

com a *Eta*-conversão. Em seguida estabeleceremos a equivalência entre as regras *Eta* dos cálculos aqui tratados.

No terceiro capítulo utilizaremos a noção de adequabilidade de [KR00] para compararmos o λ_{Susp}, $\lambda\sigma$ e λs_e. Esta comparação é importante para que possamos ter uma boa idéia do comportamento de cada um desses cálculos verificando suas vantagens e desvantagens.

Por fim, o quarto capítulo se refere a uma implementação para as regras dos sistemas de reescrita dos cálculos aqui estudados. Vale salientar que implementamos a regra *Eta* para cada um dos cálculos aqui tratados. Apresentamos um pouco da estrutura da implementação de forma que se possa observar as vantagens de se trabalhar com λ-termos no ambiente da linguagem Ocaml. Além disso, veremos a corretude das regras *Eta* aqui implementadas.

Índice

Capítulo 1

Sistemas de Reescrita e o λ-Cálculo

1.1 Sistemas de Reescrita

Iniciaremos este trabalho fazendo uma breve introdução aos sitemas de reescrita com ênfase especial para o λ-cálculo a título de motivação. Iniciaremos com as definições básicas de sistemas de reescrita que podem ser encontradas com mais detalhes em [AR00] e [BN98].

Definição 1.1 *O número de argumentos de um símbolo de função é denominado a "aridade" da função. Um símbolo de função de aridade n é denominado um símbolo n-***ário**.

Definição 1.2 *Um* **termo** *é definido indutivamente por:*

- *Uma variável é um termo;*
- *Uma constante é um termo;*
- *Se f é um símbolo de função de aridade n e, $t_1, \ldots, t_n$ são termos então $f(t_1, \ldots, t_n)$ é um termo.*

Definição 1.3 *Uma* **substituição** *θ é um conjunto finito da forma $\{t_1/v_1, \ldots, t_n/v_n\}$, onde cada v_i é uma variável distinta e cada t_i é um termo distinto de v_i.*

Observação. Utilizaremos $=$ para denotar a igualdade semântica e $\equiv$ para denotar a igualdade sintática.

Definição 1.4 *Sejam $\theta = \{t_1/v_1, \ldots, t_n/v_n\}$ uma substituição e E um expressão. Então $E\theta$, a* **instância de** E **por** θ*, é a expressão obtida de E substituindo simultaneamente cada ocorrência das variáveis v_i respectivamente por t_i, para $i = 1, 2, \ldots, n$.*

Definição 1.5 *Seja S um conjunto finito de expressões. Uma substituição θ é denominada um* **unificador** *de S se $S\theta$ é um conjunto unitário.*

Definição 1.6 *Um unificador θ para um conjunto finito S de expressões é denominado um* **unificador mais geral** *de S, denotado por* **mgu(S)**, *se para todo unificador σ de S, existe uma substituição γ tal que $\sigma = \theta\gamma$.*

Definição 1.7 (Posições Válidas) *O conjunto $O(t)$ das* **posições válidas** *de um termo t é definido indutivamente por:*

(1) Se t é uma constante ou variável então $O(t) = \lambda$, onde λ denota a sequência vazia.

(2) Se t é um termo da forma $f(t_1, \ldots, t_n)$ então $O(t) = \{i.\pi \mid 1 \leq i \leq n \text{ e } \pi \in O(t_i)\} \cup \{\lambda\}$ e $t\ |_{i.\pi} \equiv t_i\ |_\pi$, onde $t\ |_i$ denota o subtermo de t que se encontra na posição i.

Definição 1.8 (Sistema de Reescrita) *Seja M um conjunto e $\rightarrow$ uma relação binária sobre M. Um* **sistema de reescrita** *é um par $(M, \rightarrow)$ onde $\rightarrow$ é denominada a* **relação de redução** *ou* **relação de reescrita** *correspondente ao sistema.*

Notação. Seja $R = (M, \rightarrow)$ um sistema de reescrita. Denotamos por:

- $\leftarrow$ a relação inversa de $\rightarrow$, de forma que $u \leftarrow v$ se e somente se $v \rightarrow u$;
- $\leftrightarrow = \rightarrow \cup \leftarrow$, de forma que $u \leftrightarrow v$ se e somente se $u \rightarrow v$ ou $u \leftarrow v$;
- $\rightarrow^n$ a relação que é definida indutivamente por

$$u \rightarrow^0 v \text{ se e somente se } u \equiv v \text{ e,}$$
$$u \rightarrow^{n+1} v \text{ se e somente se } \exists w, u \rightarrow^n w \rightarrow v;$$

Intuitivamente $\rightarrow^n$ pode ser vista como a aplicação de n passos da relação $\rightarrow$. Neste caso dizemos que temos uma **derivação** de comprimento n. Uma derivação de um único passo é denominada uma **redução**.

A mesma relação indutiva também pode ser extendida, de maneira óbvia, para $\leftarrow^n$ e $\leftrightarrow^n$. Por elegância na notação, $\leftarrow^n$ é substituído por ${}^n\leftarrow$.

- $\rightarrow^*$ o fecho reflexivo transitivo da relação $\rightarrow$;
- $\leftrightarrow^*$ o fecho reflexivo simétrico transitivo da relação $\rightarrow$. Ou seja, $\leftrightarrow^*$ é a menor relação de equivalência que contém a relação $\rightarrow$.

  Notação. Para o sistema de reescrita $R = (M, \rightarrow)$, essa relação de equivalência será denotada por $=_R$.

Definição 1.9 *Uma relação binária é uma* **ordem parcial** *se for anti-reflexiva e transitiva.*

Definição 1.10 *Uma ordem parcial é dita* **bem-fundada** *(do inglês, well-founded) quando não gera cadeias de redução infinitas.*

Definição 1.11 *Seja $R = (M, \rightarrow)$ um sistema de reescrita. Um elemento $u \in M$ é denominado* **irredutível** *se não existe nenhum $v \in M$ tal que $u \rightarrow v$; caso contrário u é dito* **redutível**. *Se $u \rightarrow^* v$ e v é irredutível, então v é denominado uma* **forma normal** *de u, que denotaremos por $R\text{-}norm(u)$. Dois termos u e v são ditos* **juntáveis** *se existe um $w \in M$ tal que $u \rightarrow^* w \ {}^*\leftarrow v$ e, denotamos por $u \downarrow v$.*

Definição 1.12 *Uma relação de reescrita $\rightarrow$ sobre um conjunto M é dita* **terminante** *se não existem cadeias infinitas de derivação $u_0 \rightarrow u_1 \rightarrow \ldots$. Um sistema de reescrita associado com uma relação de reescrita terminante será também denominado terminante.*

Definição 1.13 *Um sistema de reescrita é dito* **confluente** *se*

$$({}^*\leftarrow \circ \rightarrow^*) \subseteq (\rightarrow^* \circ {}^*\leftarrow).$$

Isto significa que para todo $u, v, w \in M$ com $v \ {}^*\leftarrow u \rightarrow^* w$, existe algum $r \in M$ com $v \rightarrow^* \ r \ {}^*\leftarrow w$.

Definição 1.14 *Um sistema de reescrita $R = (M, \rightarrow)$ é* **localmente confluente** *ou local confluente se e somente se $(\leftarrow \circ \rightarrow) \subseteq (\rightarrow^* \circ {}^*\leftarrow)$.*

Definição 1.15 *Um sistema de reescrita $R = (M, \rightarrow)$ satisfaz a* **propriedade de Church-Rosser (CR)** *se e somente se $\leftrightarrow^* \subseteq (\rightarrow^* \circ {}^*\leftarrow)$.*

Observação. É bem conhecido que Confluência e a Propriedade de Church-Rosser são equivalentes.

Notação. Por analogia, as situações $v \ {}^*\leftarrow u \rightarrow^* \ w$ e $v \leftarrow u \rightarrow w$ são denominadas **divergência** e **divergência local**, respectivamente.

Definição 1.16 *Um sistema de reescrita $R = (M, \rightarrow)$ é denominado* **convergente** *se é terminante e confluente.*

Lema 1.17 *Seja $R = (M, \rightarrow)$ um sistema de reescrita convergente. Então $u \leftrightarrow^* v$ se e somente se $R\text{-}norm(u) =_R R\text{-}norm(v)$.*

Lema 1.18 (do Diamante de Newman, 1942). *Seja $R = (M, \rightarrow)$ um sistema de reescrita terminante. Então a relação $\rightarrow$ é confluente se e somente se é localmente confluente.*

Definição 1.19 (Pares Críticos) *Sejam $l_1 \rightarrow r_1$ e $l_2 \rightarrow r_2$ regras de um sistema de reescrita R. Suponha que $Var(l_1) \cap Var(l_2) = \emptyset$, onde $Var(t)$ denota o conjunto das variáveis que ocorrem no termo t. Seja $\pi \in O(l_1)$, tal que o termo na posição π de l_1, isto é, $l_1 \mid_\pi$, não seja uma variável, e $\sigma = mgu(l_1 \mid_\pi, l_2)$. Então o par ordenado de termos*

$$\langle r_1\sigma, (l_1[\pi \leftarrow r_2])\sigma \rangle$$

é um **par crítico** *de R das regras $l_1 \rightarrow r_1$ e $l_2 \rightarrow r_2$. Denotamos por $CP(R)$ o* **conjunto dos pares críticos** *de R.*

Pelo Lema do Diamante de Newman [New42], sob a hipótese de terminação, confluência e confluência local são equivalentes. Esse resultado é de grande interesse teórico, mas na prática não pode ser aplicado diretamente já que testar confluência local é, em geral, um processo infinito. Para solucionar esse problema, enunciaremos

o conhecido Lema dos Pares Críticos de Knuth-Bendix. Esse lema nos permitirá concluir a confluência de um sistema de reescrita pela simples análise dos seus pares críticos.

Lema 1.20 (Pares Críticos de Knuth-Bendix, 1970) *Seja R um sistema de reescrita de termos. Se $t_1 \leftarrow t \rightarrow t_2$ então $t_1 \downarrow t_2$ ou $t_1 \leftrightarrow_{CP(R)} t_2$, onde $\leftrightarrow_{CP(R)}$ é o fecho simétrico da relação dada pelo conjunto de pares críticos de R.*

1.2 O λ-Cálculo

O λ-cálculo surgiu no início do século XX com os trabalhos de Alonzo Church [Chu32] e [Chu33]. Esses estudos faziam parte de teorias mais gerais de funções e lógica de ordem superior, cujo intuito era o de servir como fundamento para a Matemática. Quando Kleene e Rosser [KR35] mostraram a inconsistência dessas teorias, Church abandonou o programa de fundamentos e extraiu a subteoria referente à parte funcional que hoje corresponde ao que chamamos de λ-cálculo, cuja consistência foi mostrada por [CR36].

O λ-cálculo é o primeiro sistema de reescrita conhecido no contexto computacional. É também um modelo conveniente para representar funções computáveis e, além disso possui uma notação compacta para algoritmos, sendo por esse motivo a base do paradigma de programação funcional.

O λ-cálculo é uma teoria que modela funções e seu comportamento aplicativo, tendo, por isso, a *aplicação* como operação básica. Dada uma função f e um argumento a denotamos o resultado da aplicação da função f ao argumento a por $f\ a$.

O λ-cálculo possui uma outra operação básica que, num certo sentido, corresponde à operação inversa da aplicação. Essa operação é chamada de *abstração*. O significado da abstração pode ser compreendido da seguinte forma, se $g(x)$ representa uma expressão, possivelmente contendo a variável x, então representamos por $\lambda x.g(x)$ a função que associa a cada argumento a o valor $g(a)$. Dado um abstrator qualquer $\lambda x.M$, dizemos que M é o *corpo* do abstrator. Em geral, o corpo de um abstrator inicia após o ponto ligado a esse abstrator e, quando necessário, é delimitado por parênteses. Por exemplo, em $(\lambda x.(\lambda y.M)\ N)$ o corpo do abstrator λy é M enquanto que o corpo do abstrator λx é $\lambda y.M$. Os parênteses podem ser eliminados quando a abrangência dos abstratores é evidente como em $\lambda x.M$.

Nesse ponto podemos definir *variável livre* e *variável ligada*:

Definição 1.21 *O conjunto das variáveis livres do termo M, denotado por $VL(M)$, é definido indutivamente por:*

(i) $VL(x) = \{x\}$, para qualquer variável x;
(ii) $VL((M\ N)) = VL(M) \cup VL(N)$;
(iii) $VL(\lambda x.M) = VL(M) \backslash \{x\}$.

As ocorrências da variável x na expressão $\lambda x.M$ são ditas ligadas.

Por exemplo, na expressão $\lambda x.2x + 1$ dizemos que x é uma variável ligada pois ela ocorre no corpo do abstrator λx. Já na expressão $\lambda x.2x + y$, dizemos que x é variável ligada enquanto que y é uma variável livre porque a mesma não está no corpo de nenhum abstrator da forma λy. Note que uma variável pode ocorrer tanto livre quanto ligada em um mesmo termo. De fato, no termo $a(\lambda a.a)$ a primeira ocorrência da variável a é livre enquanto que a segunda ocorrência é ligada.

Em [Bar97], Barendregt explica o aparecimento do símbolo λ para denotar a abstração de função: "Em *Principia Mathematica* [RW13], a notação para função f com $f(x) = 2x + 1$ é $2\widehat{x} + 1$. Church originalmente pretendia utilizar a notação $\widehat{x}.2x + 1$. No entanto, o tipógrafo não conseguiu posicionar o símbolo $\widehat{\ }$ em cima da letra x, e o colocou em frente da mesma resultando em$\widehat{x}.2x + 1$. Depois outro tipógrafo mudou$\widehat{x}.2x + 1$ para $\lambda x.2x + 1$".

A seguir definimos formalmente os λ-termos:

Definição 1.22 *O λ-cálculo opera sobre termos definidos pela seguinte sintaxe:*

TERMOS $M, N ::= a \mid \lambda.M \mid (M\ N)$ *onde a é uma variável.*

Notação. (i) $M\ N_1 \ldots N_k$ é o mesmo que $(\ldots((M\ N_1)N_2)\ldots N_k)$ e,
(ii) $\lambda x_1 \ldots x_k.M$ significa $(\lambda x_1.(\lambda x_2.(\ldots(\lambda x_k.M)\ldots)))$

As substituições desempenham um papel fundamental no λ-cálculo. De fato, a principal regra computacional desse formalismo, a β-conversão, pode ser representada da seguinte maneira:

$$(\beta)\ \ ((\lambda x.M)\ N) \to M\{N/x\} \tag{1.1}$$

De acordo com a definição 1.4, $M\{N/x\}$ representa a substituição por N de todas as ocorrências da variável x no termo M, no entanto, adaptada ao contexto

do λ-cálculo, devemos substituir N apenas pelas ocorrências livres da variável x. Note que não queremos com essa substituição que alguma variável livre venha a se tornar ligada após a substituição. Por exemplo, $(\lambda y.x)\{yy/x\} \neq (\lambda y.yy)$. Assim, sempre que necessário podemos fazer um *renomeamento de variáveis* de forma que $(\lambda y.x)\{yy/x\} \rightarrow_\beta (\lambda z.yy)$. Esse renomeamento é denominado α-*conversão* na linguagem do λ-cálculo. Fazer um renomeamento como descrito acima não é uma regra formal e, assim como a substituição $\{N/x\}$, ambas operações estão fora da linguagem do λ-cálculo, ou seja, estas operações são na verdade meta-operações.

Adicionalmente, o λ-cálculo possui uma outra regra chamada η-conversão. Intuitivamente essa regra nos diz que abstrações que computam o mesmo valor para o mesmo argumento são convertíveis. Formalmente a regra η é dada por:

$$(\eta) \quad \lambda x.(M\ x) \rightarrow M, \text{ sempre que } x \notin VL(M) \tag{1.2}$$

Informalmente podemos dizer que a regra η está intimamente ligada à noção de igualdade extendida de funções que pode ser melhor compreendida através da seguinte afirmação: "Se $f(x) = g(x)$, para todo x, então $f = g$". Agora se tentarmos traduzir esta afirmação para a linguagem do λ-cálculo, obteremos exatamente a regra η. De fato, de acordo com nossa hipótese precisamos tomar dois λ-termos f e g que sejam "iguais" sempre que aplicados ao mesmo argumento, isto é, $f\ x =_\lambda g\ x$. Até o presente momento a única "igualdade" que temos é a obtida pela regra β (equação 1.1). Sendo assim, podemos escrever

$$(\lambda a.M\ a)\ x \rightarrow_\beta (M\ a)\{a/x\} = M\ x$$

onde o segundo passo acima se justifica pelo fato de que $x \notin VL(M)$.

Então temos que $(\lambda a.M\ a)\ x \rightarrow M\ x$, isto é, $(\lambda a.M\ a) \rightarrow M$.

A estrutura do λ-cálculo pode ser associada a uma teoria de tipos e, nesse caso temos o chamado λ-cálculo tipado. Nesse trabalho exploraremos apenas aspectos do λ-cálculo não tipado.

Essa diferenciação entre λ-cálculo simplesmente tipado e, λ-cálculo não tipado é importante porque suas propriedades são diferentes. De fato, apesar de ambos serem confluentes, o λ-cálculo simplesmente tipado é terminante enquanto que o λ-cálculo não tipado é não terminante. O contra-exemplo clássico para mostrar a não terminalidade do λ-cálculo não tipado consiste em considerar a β-redução do λ-termo $((\lambda x.(x\ x))\ (\lambda x(x\ x)))$ já que este se β-reduz a si mesmo.

1.3 O λ-Cálculo em Notação de De Bruijn

No ínicio dos anos 70, De Bruijn desenvolveu uma notação que elimina a necessidade de se realizar qualquer tipo de α-conversão, pois nessa notação variáveis são substituidas por *índices* [dB72]. Esses índices são números naturais que denotaremos por $\underline{\mathbf{n}}$. Os índices de De Bruijn, a grosso modo, indicam o abstrator a que aquela variável está ligada, ou melhor, o índice de De Bruijn $\underline{\mathbf{n}}$ indica que n representa uma variável que está dentro do corpo de n abstratores. Por exemplo, o termo $\lambda x.x$ é representado na notação de De Bruijn por $\lambda\underline{\mathbf{1}}$. Já $\lambda x.\lambda y.(x\ y)$ é representado por $\lambda\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})$. As variáveis livres também podem ser representadas na notação de De Bruijn. Para isto precisamos estabelecer um *referencial* e, a contagem sobre esse referencial se faz de maneira similar ao caso de variáveis ligadas. Como exemplo considere o termo $(\lambda x.(\lambda y.(x\ z)\ x)\ (z\ \lambda z.(x\ z)))$. Sua representação no referencial x, y, z é $(\lambda(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{5}})\ \underline{\mathbf{1}})\ (\underline{\mathbf{3}}\ \lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})))$.

Daqui em diante utilizaremos termos na notação de De Bruijn. É importante salientar que as propriedades do λ-cálculo em notação de De Bruijn são idênticas às do λ-cálculo puro. Este é um dos motivos para o fato de que, neste trabalho, os termos do λ-cálculo sejam representados nesta notação.

A notação de De Bruijn tem vantagens importantes, além, é claro, das vantagens computacionais óbvias. De fato, existem pelo menos duas razões importantes para se utilizar a notação de De Bruijn. A primeira diz respeito quanto a dificuldade computacional de se evitar a captura de variáveis livres ao se realizar β-reduções. Essa dificuldade pode ser controlada, de forma muito custosa em termos computacionais, via α-conversões. A segunda razão diz respeito à dificuldade de se estabelecer a equivalência entre λ-termos. Na prática essa dificuldade também é resolvida via α-conversão. Como termos na notação de De Bruijn são insensíveis a α-conversão, essas dificuldades desaparecem ao se utilizar essa notação.

Podemos definir formalmente os termos na notação de De Bruijn da seguinte maneira:

Definição 1.23 *O conjunto Λ_{dB} dos λ-termos na notação de De Bruijn é definido indutivamente como se segue:*

$$\textsc{Termos}\quad M,\ N\ ::=\ \underline{\mathbf{n}} \mid \lambda M \mid (M\ N)$$

A pergunta que surge naturalmente agora é: Como ficam as regras β e η para termos na notação de De Bruijn? Para responder a essa pergunta precisaremos de algumas definições para substituição e atualização dos índices de De Bruijn.

Definição 1.24 ([ARM00]) *Seja $M \in \Lambda_{dB}$. Então a i-elevação do termo M, denotada por M^{+i}, é definida indutivamente como segue:*

1. $(M\ N)^{+i} = (M^{+i}\ N^{+i})$
2. $(\lambda M)^{+i} = \lambda M^{+(i+1)}$
3. $\underline{\mathbf{n}}^{+i} = \begin{cases} \underline{\mathbf{n+1}} & \text{se } n > i \\ \underline{\mathbf{n}} & \text{se } n \leq i \end{cases}$

A elevação de um termo M é a sua 0-elevação denotada por M^{+}

Definição 1.25 ([ARM00]) *A substituição pelo termo N no nível $n-1$ do termo M, denotada por $M\{N/\underline{\mathbf{n}}\}$, é definida indutivamente com segue:*

1. $(M_1\ M_2)\{N/\underline{\mathbf{n}}\} = (M_1\{N/\underline{\mathbf{n}}\}\ M_2\{N/\underline{\mathbf{n}}\})$
2. $(\lambda M)\{N/\underline{\mathbf{n}}\} = \lambda M\{N^{+}/\underline{\mathbf{n+1}}\}$
3. $\underline{\mathbf{m}}\{N/\underline{\mathbf{n}}\} = \begin{cases} \underline{\mathbf{m-1}} & \text{se } m > n \\ N & \text{se } m = n \\ \underline{\mathbf{m}} & \text{se } m \leq n \end{cases}$

Agora podemos definir as regras β e η na notação de De Bruijn.

Definição 1.26 ([ARM00]) *As regras β e η são definidas para o conjunto dos Λ_{dB}-termos como segue:*

$$(\beta) \qquad (\lambda M\ N) \rightarrow_{\beta} M\{N/\underline{\mathbf{1}}\}$$
$$(\eta) \qquad \lambda(M\ \underline{\mathbf{1}}) \rightarrow_{\eta} N,\ \textit{se}\ N^{+} = M$$

Para exemplificar a aplicação destas regras considere a seguinte β-conversão do λ-cálculo puro: $((\lambda x.(\lambda y.(x\ y)))\ z) \rightarrow_{\beta} \lambda y.(z\ y)$. A equivalente β-conversão na notação de De Bruijn é dada como mostraremos a seguir. Considerando o referencial x, y, z o termo $((\lambda x.(\lambda y.(x\ y)))\ z)$ pode ser escrito como $((\lambda(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})))\ \underline{\mathbf{3}})$. Aplicando as regras dadas pelas definições anteriores temos:

$((\lambda(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})))\ \underline{\mathbf{3}}) \rightarrow$
$(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}}))\{\underline{\mathbf{3}}/\underline{\mathbf{1}}\} \rightarrow$
$\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})\{\underline{\mathbf{3}^{+}}/\underline{\mathbf{2}}\} \rightarrow$
$\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})\{\underline{\mathbf{4}}/\underline{\mathbf{2}}\} \rightarrow$
$\lambda(\underline{\mathbf{2}}\{\underline{\mathbf{4}}/\underline{\mathbf{2}}\}\ \underline{\mathbf{1}}\{\underline{\mathbf{4}}/\underline{\mathbf{2}}\}) \rightarrow$
$\lambda(\underline{\mathbf{4}}\ \underline{\mathbf{1}})$

No capítulo seguinte estudaremos alguns cálculos que tentam simular, computacionalmente falando, o λ-cálculo. Como veremos isto não é uma tarefa fácil mas que, por outro lado é muito interessante. As idéias que surgiram nessa área criaram um novo campo de trabalho em Teoria de Reescrita que chamamos de Substituições Explícitas.

Capítulo 2

Substituições Explícitas

O λ-cálculo é um importante modelo teórico equivalente às Máquinas de Turing. No entanto, a operação de substituição utilizada na definição da β-conversão do λ-cálculo, por se tratar de uma meta-operação, não nos permite analisar noções computacionais importantes como, por exemplo, complexidade de tempo ou espaço. Como vimos anteriormente, isto acontece porque a operação de substituição não está definida formalmente no λ-cálculo.

Para solucionar esse problema Abadi *et al.* desenvolveram em [ACCL91] uma variação do λ-cálculo, chamado de $\lambda\sigma$. No $\lambda\sigma$ substituições são manipuladas explicitamente, mas como veremos adiante esse cálculo ainda não resolve completamente esse problema pois não satisfaz todas as propriedades necessárias.

De fato, um cálculo de substituições explícitas λ_ξ adequado para os propósitos de ser uma variação do λ-cálculo (que preserve suas principais características) deve possuir as seguintes propriedades:

1. **Simulação da β-conversão:** Essa condição é essencial haja vista que o objetivo maior de um cálculo de substituições explícitas é simular o λ-cálculo. Formalmente, considere $A, B \in \Lambda_{dB}$ tal que $A \to_\beta B$. Uma **simulação** dessa β-conversão no cálculo λ_ξ, é uma λ_ξ-derivação da forma $A \to_r C \to_\xi \xi\text{-}norm(C) = B$, onde r é a regra que inicia a simulação da β-conversão, aplicando-a ao mesmo redex que em $A \to_\beta B$. Dizemos que o cálculo λ_ξ **simula um passo de β-conversão** se qualquer β-conversão $A \to_\beta B$ tem uma simulação no cálculo λ_ξ.

2. **Terminação:** Para um cálculo de substituições explícitas λ_ξ, considere o cálculo de substituição subjacente ξ, isto é, considere o conjunto de regras

do sistema de reescrita de λ_ξ retirando-se a regra *Eta* e a regra que inicia a simulação da β-conversão. A questão é saber se o cálculo ξ é terminante.

3. **Confluência (CR):** A análise da confluência de um cálculo de substituições explícitas é subdividida em:

   (a) Confluência em **termos fechados**, isto é, termos sem a ocorrência de meta-variáveis. As meta-variáveis são variáveis que pertencem a um contexto mais amplo do λ-cálculo; elas podem, por exemplo, representar λ-termos em um problema de unificação de ordem superior. Dessa forma, termos fechados são termos cujas variáveis pertencem exclusivamente ao conjunto Λ_{dB}.

   (b) Confluência em **termos abertos**, isto é, termos com a ocorrência de meta-variáveis.

4. **Preservação da Terminação Forte (PSN):** Seja A um termo tal que qualquer derivação a partir de A no λ-cálculo seja sempre finita. A questão é saber se qualquer derivação a partir do termo A, via o cálculo de substituições explícitas λ_ξ, é também finita.

Posteriormente surgiram outros cálculos de substituições explícitas, como por exemplo, o $\lambda\sigma_{\Uparrow}$, λv, λs, λs_e, λt, λu e o λ_{Susp}. Neste trabalho nos restringiremos a analisar três desses cálculos: o $\lambda\sigma$, λs_e [KR95b] e o λ_{Susp} [NW98]. Na próxima seção apresentaremos esses cálculos e suas propriedades.

2.1 Alguns Cálculos de Substituições Explícitas

2.1.1 O Cálculo $\lambda\sigma$

Originalmente proposto por [ACCL91], o cálculo $\lambda\sigma$ surge como uma variação do λ-cálculo onde as substituições são manipuladas explicitamente. O $\lambda\sigma$ é um sistema de reescrita cujas expressões podem ser de um dos seguintes tipos: **termos** ou **substituições**. Esses tipos podem ser assim definidos:

Termos	M, N	::=	$\underline{\mathbf{1}} \mid X \mid \lambda M \mid (M\ N) \mid M[S]$
Substituições	S, T	::=	$id \mid \uparrow \mid M.S \mid S \circ T$

O sistema de reescrita do cálculo $\lambda\sigma$ é mostrado na figura 2.1. Nessa figura, a regra $(Beta)$ inicia a simulação de um passo de β-conversão, a regra Eta_σ corresponde à η-conversão do λ-cálculo, enquanto que as outras regras são necessárias para a

(Beta)	$(\lambda M\ N) \to M[N.id]$
(App)	$(M\ N)[S] \to (M[S]\ N[S])$
(Abs)	$(\lambda M)[S] \to \lambda M[\underline{\mathbf{1}}.(S \circ \uparrow)]$
(Clos)	$M[S][T] \to M[S \circ T]$
(VarCons)	$\underline{\mathbf{1}}[M.S] \to M$
(Id)	$M[id] \to M$
(Assoc)	$(S_1 \circ S_2) \circ T \to S_1 \circ (S_2 \circ T)$
(Map)	$(M.S) \circ T \to M[T].(S \circ T)$
(IdL)	$id \circ S \to S$
(IdR)	$S \circ id \to S$
(ShiftCons)	$\uparrow \circ (M.S) \to S$
(VarShift)	$\underline{\mathbf{1}}. \uparrow \to id$
(SCons)	$\underline{\mathbf{1}}[S].(\uparrow \circ S) \to S$
(Eta_σ)	$\lambda(M\ \underline{\mathbf{1}}) \to N$ se $M =_\sigma N[\uparrow]$

Figura 2.1: As regras do cálculo $\lambda\sigma$

manipulação dos termos do $\lambda\sigma$. Sendo assim, como foi citado anteriormente denotaremos por $\lambda\sigma$ ao sistema de reescrita representado pelas regras dadas na figura 2.1, enquanto que σ denotará o mesmo sistema $\lambda\sigma$ retirando-se as regras $(Beta)$ e (Eta_σ). Como podemos observar, esse cálculo utiliza apenas o índice de De Bruijn $\underline{\mathbf{1}}$. Todos os outros índices são codificados da seguinte forma: $\underline{\mathbf{2}} = \underline{\mathbf{1}}[\uparrow]$, $\underline{\mathbf{3}} = \underline{\mathbf{1}}[\uparrow][\uparrow]$ e assim sucessivamente. Ou seja, denotando $\underbrace{\uparrow \circ \uparrow \circ \ldots \circ \uparrow}_{n \text{ vezes}}$ por $\uparrow^n$, podemos escrever $\underline{\mathbf{n}} = \underline{\mathbf{1}}[\uparrow^{n-1}]$. Intuitivamente a substituição $\uparrow$ é utilizada para atualizar índices de De Bruijn. O ponto ".", por sua vez, separa os termos de uma lista que contém termos que devem substituir índices de De Bruijn. Isto é, o primeiro termo da lista substitui as ocorrências do índice $\underline{\mathbf{1}}$, o segundo, as ocorrências de $\underline{\mathbf{2}}$, e assim sucessivamente. Os índices $\underline{\mathbf{n}}$, onde n é maior do que o comprimento desta lista, são incrementados de acordo com o último termo dessa lista, já que os mesmos correspondem a variáveis livres.

Quanto às propriedades desejáveis para um cálculo de substituições explícitas apresentadas na página 11, [ACCL91] mostra que o cálculo $\lambda\sigma$ satisfaz as propriedades 1, 2 e 3a. Em [DHK00] estende-se o $\lambda\sigma$ para lidar com meta-variáveis, obtendo confluência local. Em [Mel95], Melliès mostra que o cálculo $\lambda\sigma$ não preserva terminação forte, isto é, não é PSN e portanto não satisfaz a propriedade 4.

Para exemplificar uma simulação de um passo de β-conversão no $\lambda\sigma$ considere o seguinte exemplo: $((\lambda x.(\lambda y.(x\ y)))\ z) \rightarrow_\beta \lambda y.(z\ y)$. Como já vimos anteriormente, esta β-conversão em notação de De Bruijn, considerando o referencial x, y, z, equivale a $((\lambda(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})))\ \underline{\mathbf{3}}) \rightarrow_\beta \lambda(\underline{\mathbf{4}}\ \underline{\mathbf{1}})$. No cálculo $\lambda\sigma$, precisamos inicialmente codificar o termo $((\lambda(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})))\ \underline{\mathbf{3}})$ obtendo $((\lambda(\lambda(\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}})))\ \underline{\mathbf{1}}[\uparrow^2])$. Aplicando agora as regras dadas pela figura 2.1, temos, por exemplo, a seguinte derivação:

$$
\begin{aligned}
&((\lambda(\lambda(\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}})))\ \underline{\mathbf{1}}[\uparrow^2]) \rightarrow_{Beta}\\
&(\lambda(\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}}))[\underline{\mathbf{1}}[\uparrow^2].id] \rightarrow_{Abs}\\
&\lambda(\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}})[\underline{\mathbf{1}}.((\underline{\mathbf{1}}[\uparrow^2].id)\circ\uparrow)] \rightarrow_{Map}\\
&\lambda(\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}})[\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow^2][\uparrow].(id\circ\uparrow))] \rightarrow_{Clos}\\
&\lambda(\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}})[\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow^3].(id\circ\uparrow))] \rightarrow_{App}\\
&\lambda(\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow^3].(id\circ\uparrow))]\ \underline{\mathbf{1}}[\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow^3].(id\circ\uparrow))]) \rightarrow_{VarCons}\\
&\lambda(\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow^3].(id\circ\uparrow))]\ \underline{\mathbf{1}}) \rightarrow_{Clos}\\
&\lambda(\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow^3].(id\circ\uparrow)))]\ \underline{\mathbf{1}}) \rightarrow_{ShiftCons}\\
&\lambda(\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow^3].(id\circ\uparrow)]\ \underline{\mathbf{1}}) \rightarrow_{VarCons}\\
&\lambda(\underline{\mathbf{1}}[\uparrow^3]\ \underline{\mathbf{1}})
\end{aligned}
$$

E este último termo, como podemos ver corresponde a $\lambda(\underline{\mathbf{4}}\ \underline{\mathbf{1}})$ como queríamos.

2.1.2 O Cálculo λs_e

Em [KR95a], Kamareddine e Ríos apresentam um novo cálculo de substituições explícitas: o λs. Diferentemente do $\lambda\sigma$, as expressões do λs são de apenas um tipo, denominado **termos**. Isto faz com que o λs mantenha uma estrutura sintática mais próxima do λ-cálculo. Os termos do cálculo λs são definidos por:

$$\text{Termos}\quad \text{M, N}\quad ::=\quad \underline{\mathbf{n}} \mid \lambda M \mid (M\ N) \mid M\ \sigma^i\ N \mid \varphi^i_k\ M, \text{ para } k \geq 0 \text{ e } i \geq 1$$

Esse cálculo satisfaz as propriedades 1, 2, 3a e 4. As regras do λs são dadas pela figura 2.2. Pouco depois, em [KR95b], Kamareddine e Ríos estendem o cálculo λs para termos abertos. Essa extensão é denominada λs_e. Até esse ponto sabia-se que

o cálculo λs_e satisfazia as propriedades 1, 2 e 3. Kamareddine e Ríos conjecturaram, então, que o cálculo λs_e preservava a terminação forte até que em [Gui00], Guillaume mostrou falsa essa conjectura. As regras do λs_e são formadas pela união das regras dadas pelas figuras 2.2 e 2.3. Aqui, denotaremos por s_e ao sistema λs_e (figuras 2.2 e 2.3) retirando-se a regra (Eta_{s_e}) e a regra que inicia a propagação da β-conversão, ou seja, a (σ-*generation*).

A noção intuitiva dos operadores σ^i e φ_k^i pode ser vista da seguinte forma: o operador σ^i é um operador que faz substituições, como podemos observar pela regra σ-*destruction*. O super-índice i deste operador vai sendo incrementado à medida em que o mesmo entra dentro de novos abstratores (veja σ-λ-*transition*). O operador φ_k^i é responsável pela atualização de índices de De Bruijn. O super-índice de φ_k^i é herdado do super-índice de σ^i por meio de uma σ-*destruction*. O sub-índice, por sua vez, é incrementado ao entrar dentro de novos abstratores como podemos observar em φ-λ-*transition*. A atualização de índices é feita pela regra φ-*destruction*, considerando-se que todos os índices $\underline{\mathbf{n}}$ maiores do que o sub-índice k são variáveis livres e, portanto precisam ser atualizados em $i-1$. Os outros índices permanecem inalterados.

(σ-*generation*)	$(\lambda M\ N) \to M\ \sigma^1\ N$
(σ-λ-*transition*)	$(\lambda M)\ \sigma^i\ N \to \lambda(M\ \sigma^{i+1}\ N)$
(σ-*app-transition*)	$(M_1\ M_2)\ \sigma^i\ N \to ((M_1\ \sigma^i\ N)(M_2\ \sigma^i\ N))$
(σ-*destruction*)	$\underline{\mathbf{n}}\sigma^i N \to \begin{cases} \underline{\mathbf{n-1}} & \text{se } n > i \\ \varphi_0^i N & \text{se } n = i \\ \underline{\mathbf{n}} & \text{se } n < i \end{cases}$
(φ-λ-*transition*)	$\varphi_k^i(\lambda M) \to \lambda(\varphi_{k+1}^i M)$
(φ-*app-transition*)	$\varphi_k^i(M_1\ M_2) \to ((\varphi_k^i M_1)\ (\varphi_k^i M_2))$
(φ-*destruction*)	$\varphi_k^i \underline{\mathbf{n}} \to \begin{cases} \underline{\mathbf{n+i-1}} & \text{se } n > k \\ \underline{\mathbf{n}} & \text{se } n \leq k \end{cases}$

Figura 2.2: Regras do cálculo λs

(σ-σ-*transition*)	$(M_1\ \sigma^i\ M_2)\sigma^j N \to (M_1\ \sigma^{j+1}\ N)\sigma^i(M_2\ \sigma^{j-i+1}\ N)$ se $i \leq j$
(σ-φ-*transition*1)	$(\varphi^i_k M)\sigma^j N \to \varphi^{i-1}_k M$ se $k < j < k+i$
(σ-φ-*transition*2)	$(\varphi^i_k M)\sigma^j N \to \varphi^i_k(M\sigma^{j-i+1}N)$ se $k+i \leq j$
(φ-σ-*transition*)	$\varphi^i_k(M\sigma^j N) \to (\varphi^i_{k+1}M)\sigma^j(\varphi^i_{k+1-j}N)$ se $j \leq k+1$
(φ-φ-*transition*1)	$\varphi^i_k(\varphi^j_l M) \to \varphi^j_l(\varphi^i_{k+1-j}M)$ se $l+j \leq k$
(φ-φ-*transition*2)	$\varphi^i_k(\varphi^j_l M) \to \varphi^{j+i-1}_l M$ se $l \leq k < l+j$
(Eta_{s_e})	$\lambda(M\ \underline{\mathbf{1}}) \to N$ se $M =_{S_e} \varphi^2_0 N$

Figura 2.3: Regras para manipulação de termos abertos no λs_e

Como exemplo de simulação de um passo de β-conversão via λs_e considere: $((\lambda(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})))\ \underline{\mathbf{3}}) \to_\beta \lambda(\underline{\mathbf{4}}\ \underline{\mathbf{1}})$. Note que a seguinte simulação no λs_e é única:

$$
\begin{aligned}
&((\lambda(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})))\ \underline{\mathbf{3}}) \to_{\sigma\text{-}gen}\\
&(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}}))\ \sigma^1\ \underline{\mathbf{3}} \to_{\sigma\text{-}\lambda}\\
&\lambda((\underline{\mathbf{2}}\ \underline{\mathbf{1}})\ \sigma^2\ \underline{\mathbf{3}}) \to_{\sigma\text{-}app}\\
&\lambda((\underline{\mathbf{2}}\ \sigma^2\ \underline{\mathbf{3}})\ (\underline{\mathbf{1}}\ \sigma^2\ \underline{\mathbf{3}})) \to^2_{\sigma\text{-}destr}\\
&\lambda(\varphi^2_0\underline{\mathbf{3}}\ \underline{\mathbf{1}}) \to_{\varphi\text{-}destr} \lambda(\underline{\mathbf{4}}\ \underline{\mathbf{1}})
\end{aligned}
$$

Como podemos observar aqui, o λs_e opera diretamente com todos os índices de De Bruijn sem a necessidade da codificação utilizada pelo cálculo $\lambda\sigma$.

2.1.3 O Cálculo λ_{Susp}

O Cálculo de Suspensão, que aqui denominamos λ_{Susp}, foi desenvolvido por Nadathur e Wilson [NW98] com a intensão de tratar λ-termos como dispositivos computacionais. Esse interesse foi motivado inicialmente por questões de implementação relativas ao λProlog, uma linguagem de programação lógica que utiliza termos do λ-cálculo tipado como estrutura de dados [NM88].

As expressões do λ_{Susp} podem ser dos seguintes tipos: **termos suspensos**, **contextos** ou **termos de contexto**. Esses tipos são definimos como a seguir:

Termos Suspensos	$M,\ N$	::=	$Cons \mid \underline{\mathbf{n}} \mid \lambda M \mid (M\ N) \mid [\![M, i, j, e_1]\!]$
Contextos	$e_1,\ e_2$	::=	$nil \mid et :: e_1 \mid \{\!\{e_1, i, j, e_2\}\!\}$
Termos de Contexto	et	::=	$@i \mid (M, i) \mid \langle\!\langle et, i, j, e_1\rangle\!\rangle$

onde $Cons$ denota uma constante qualquer e i, j são números naturais.

Veja que os termos suspensos podem conter constantes e índices de De Bruijn. Essas constantes podem ser vistas como meta-variáveis e, portanto, os termos abertos fazem parte da linguagem do λ_{Susp}.

Os contextos aqui têm um papel semelhante às listas que são separadas pelos pontos no $\lambda\sigma$. Os termos de contexto, por sua vez constituem os termos dessa lista, ou seja, do contexto.

Observação. Denotaremos por $\dot{-}$ a subtração truncada, que pode ser considerada da seguinte forma: $n \dot{-} m = \text{máx}\{n - m, 0\}$.

Como acontece em geral nos cálculos de substituições explícitas, o λ_{Susp} possui uma regra que inicia a simulação de um passo de β-conversão, chamada de β_s e um conjunto de regras utilizadas para manipular os termos suspensos. Estas regras são dadas pelas figuras 2.4, 2.5 e 2.6. As regras (m_9) e (m_{10}) da figura 2.6 dependem de uma função denominada *índice* que pode ser encontrada na definição 2.3. Para manter a uniformidade de notação com o $\lambda\sigma$ e λs_e, denotaremos por $Susp$ ao cálculo de substituição subjacente à λ_{Susp} retirando-se a regra β_s.

(β_s)	$((\lambda\, t_1)\ t_2) \to [\![t_1, 1, 0, (t_2, 0) :: nil]\!]$

Figura 2.4: Regra que inicia a simulação de um passo de β-conversão

(r_1)	$[\![c, ol, nl, e]\!] \to c$, onde c é uma constante.
(r_2)	$[\![\underline{\mathbf{i}}, 0, nl, nil]\!] \to \underline{\mathbf{i + nl}}$.
(r_3)	$[\![\underline{\mathbf{1}}, ol, nl, @l :: e]\!] \to \underline{\mathbf{nl - l}}$.
(r_4)	$[\![\underline{\mathbf{1}}, ol, nl, (t, l) :: e]\!] \to [\![t, 0, (nl - l), nil]\!]$.
(r_5)	$[\![\underline{\mathbf{i}}, ol, nl, et :: e]\!] \to [\![\underline{\mathbf{i - 1}}, (ol - 1), nl, e]\!]$, para $i > 1$.
(r_6)	$[\![(t_1\ t_2), ol, nl, e]\!] \to ([\![t_1, ol, nl, e]\!]\ [\![t_2, ol, nl, e]\!])$.
(r_7)	$[\![(\lambda\, t), ol, nl, e]\!] \to (\lambda\, [\![t, (ol + 1), (nl + 1), @nl :: e]\!])$.

Figura 2.5: Regras para manipulação dos termos suspensos

(m_1) $[\![[\![t, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!] \to [\![t, ol', nl', \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!]$,
onde $ol' = ol_1 + (ol_2 \dot{-} nl_1)$ e $nl' = nl_2 + (nl_1 \dot{-} ol_2)$.

(m_2) $\{\!\{nil, nl, 0, nil\}\!\} \to nil$.

(m_3) $\{\!\{nil, nl, ol, et :: e\}\!\} \to \{\!\{nil, (nl - 1), (ol - 1), e\}\!\}$,
para $nl, ol \geq 1$.

(m_4) $\{\!\{nil, 0, ol, e\}\!\} \to e$.

(m_5) $\{\!\{et :: e_1, nl, ol, e_2\}\!\} \to \langle\!\langle et, nl, ol, e_2\rangle\!\rangle :: \{\!\{e_1, nl, ol, e_2\}\!\}$.

(m_6) $\langle\!\langle et, nl, 0, nil\rangle\!\rangle \to et$.

(m_7) $\langle\!\langle @m, nl, ol, @l :: e\rangle\!\rangle \to @(l + (nl \dot{-} ol))$,
para $nl = m + 1$.

(m_8) $\langle\!\langle @m, nl, ol, (t, l) :: e\rangle\!\rangle \to (t, (l + (nl \dot{-} ol)))$,
para $nl = m + 1$.

(m_9) $\langle\!\langle (t, nl), nl, ol, et :: e\rangle\!\rangle \to ([\![t, ol, l', et :: e]\!], m)$,
onde $l' = ind(et)$ e $m = l' + (nl \dot{-} ol)$.

(m_{10}) $\langle\!\langle et, nl, ol, et' :: e\rangle\!\rangle \to \langle\!\langle et, (nl - 1), (ol - 1), e\rangle\!\rangle$,
para $nl \neq ind(et)$.

Figura 2.6: Regras para imersão em termos suspensos

Observações.

(1) Denotaremos por $\to_r$ e $\to_m$ a relação de redução definida pelas regras das figuras 2.5 e 2.6, respectivamente. A união de ambos os sistemas será, naturalmente, denotada por $\to_{rm}$.

(2) O fecho reflexivo transitivo simétrico de $\to_{rm}$ será denotado por $=_{rm}$.

Como vimos anteriormente, existem algumas propriedades desejáveis para um cálculo de substituições explícitas. O λ_{Susp}, por sua vez, satisfaz as propriedades 1, 2 e 3 [NW98]. Em [Nad99] Nadathur conjectura que o λ_{Susp} preserva terminação forte, ou seja, é PSN.

Vejamos agora, como exemplo, uma simulação de um passo de β-conversão no λ_{Susp}. Considere a seguinte β-conversão: $((\lambda(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})))\ \underline{\mathbf{3}}) \to_\beta \lambda(\underline{\mathbf{4}}\ \underline{\mathbf{1}})$. Uma possível simulação é dada por:

$((\lambda(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})))\ \underline{\mathbf{3}}) \rightarrow_{\beta_s}$
$[\![(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{1}})), 1, 0, (\underline{\mathbf{3}}, 0) :: nil]\!] \rightarrow_{r_7}$
$\lambda[\![(\underline{\mathbf{2}}\ \underline{\mathbf{1}}), 2, 1, @0 :: (\underline{\mathbf{3}}, 0) :: nil]\!] \rightarrow_{r_6}$
$\lambda([\![\underline{\mathbf{2}}, 2, 1, @0 :: (\underline{\mathbf{3}}, 0) :: nil]\!]\ [\![\underline{\mathbf{1}}, 2, 1, @0 :: (\underline{\mathbf{3}}, 0) :: nil]\!]) \rightarrow_{r_3}$
$\lambda([\![\underline{\mathbf{2}}, 2, 1, @0 :: (\underline{\mathbf{3}}, 0) :: nil]\!]\ \underline{\mathbf{1}}) \rightarrow_{r_5}$
$\lambda([\![\underline{\mathbf{1}}, 1, 1, (\underline{\mathbf{3}}, 0) :: nil]\!]\ \underline{\mathbf{1}}) \rightarrow_{r_4}$
$\lambda([\![\underline{\mathbf{3}}, 0, 1, nil]\!]\ \underline{\mathbf{1}}) \rightarrow_{r_2}$
$\lambda(\underline{\mathbf{4}}\ \underline{\mathbf{1}})$

As definições a seguir determinam as condições que devem existir sobre as expressões que trabalharemos no λ_{Susp}.

Definição 2.1 ([NW98]) *Uma expressão é dita* **simples** *quando não possui subexpressão da forma* $\langle\langle et, j, k, e\rangle\rangle$ *ou* $\{\!\{e_1, j, k, e_2\}\!\}$. *Se a expressão em questão é um termo, um contexto ou um termo de contexto, então dizemos* **termo simples**, **contexto simples** *ou* **termo de contexto simples**, *respectivamente.*

Definição 2.2 ([NW98]) *O* **comprimento** *de um contexto* e, *denotado por* $len(e)$, *é dado por:*

(a) *se* $e = nil$ *então* $len(e) = 0$;

(b) *se* $e = et :: e'$ *então* $len(e) = len(e') + 1$;

(c) *se* $e = \{\!\{e_1, i, j, e_2\}\!\}$ *então* $len(e) = len(e_1) + (len(e_2) \dot{-} i)$.

Definição 2.3 ([NW98]) *O* **índice** *de um termo de contexto* et, *denotado por* $ind(et)$, *e para cada número natural* l, *o* **índice do** l**-ésimo elemento** *do contexto* e, *denotado por* $ind_l(e)$, *são definidos simultaneamente por indução sobre a estrutura das expressões da seguinte maneira:*

(i) *Se* $et = @m$ *então* $ind(et) = m + 1$;

(ii) *Se* $et = (t', m)$ *então* $ind(et) = m$;

(iii) *Se* $et = \langle\langle et', j, k, e\rangle\rangle$, *seja* $m = (j \dot{-} ind(et'))$.

$$\textit{Então } ind(et) = \begin{cases} ind_m(e) + (j \dot{-} k) & \text{se } len(e) > m \\ ind(et') & \textit{caso contrário.} \end{cases}$$

(iv) *Se* $e = nil$ *então* $ind_l(e) = 0$

(v) *Se* $e = et :: e'$ *então* $ind_0(e) = ind(et)$ *e* $ind_{l+1}(e) = ind_l(e')$

(vi) *Se* $e = \{\!\{e_1, j, k, e_2\}\!\}$, *seja* $m = (j \dot{-} ind_l(e_1))$ *e* $l_1 = len(e_1)$.

$$Então\ ind_l(e) = \begin{cases} ind_m(e_2) + (j \dot{-} k) & \text{se } l < l_1 \text{ e } len(e_2) > m \\ ind_l(e_1) & \text{se } l < l_1 \text{ e } len(e_2) \leq m \\ ind_{l-l_1+j}(e_2) & \text{se } l \geq l_1 \end{cases}$$

O índice de um contexto, denotado por $ind(e)$, *é o* $ind_0(e)$.

Definição 2.4 ([NW98]) *Uma expressão do* λ_{Susp} *é dita* **bem-formada** *se as seguintes condições são verdadeiras para toda subexpressão* s *da expressão dada:*

(i) *Se* s *é da forma* $[\![t, ol, nl, e]\!]$ *então* $len(e) = ol$ *e* $ind(e) \leq nl$;

(ii) *Se* s *é da forma* $et :: e$ *então* $ind(e) \leq ind(et)$;

(iii) *Se* s *é da forma* $\langle\!\langle et, j, k, e \rangle\!\rangle$ *então* $len(e) = k$ *e* $ind(et) \leq j$;

(iv) *Se* s *é da forma* $\{\!\{e_1, j, k, e_2\}\!\}$ *então* $len(e_2) = k$ *e* $ind(e_1) \leq j$

Definição 2.5 ([NW98]) *A(s)* **subexpressão(ões) imediata(s)** *de uma expressão* x *são dadas por:*

(1) *Se* x *é um termo então,*

(a) se x *é da forma* $(t_1\ t_2)$ *então* t_1 *e* t_2 *são suas subexpressões imediatas;*

(b) se x *é da forma* (λt) *então* t *é sua subexpressão imediata;*

(c) se x *é da forma* $[\![t, ol, nl, e]\!]$ *então* t *e* e *são suas subexpressões imediatas;*

(2) *se* x *é um contexto então,*

(a) se x *é da forma* $et :: e$ *então* et *e* e *são suas subexpressões imediatas;*

(b) se x *é da forma* $\{\!\{e_1, i, j, e_2\}\!\}$ *então* e_1 *e* e_2 *são suas subexpressões imediatas;*

(3) *se* x *é um termo de contexto então,*

(a) se x *é da forma* (t, l) *então* t *é sua subexpressão imediata;*

(b) se x *é da forma* $\langle\!\langle et, i, j, e \rangle\!\rangle$ *então* et *e* e *são suas subexpressões imediatas.*

Definição 2.6 ([NW98]) *Duas expressões têm a mesma* **estrutura externa** *se ambas são constantes, índices de De Bruijn, abstrações, aplicações ou termos suspensos ou se ambas são da forma nil, et* :: *e,* $\{\!\{e_1, i, j, e_2\}\!\}$*,* $@l$*,* (t, l) *ou* $\langle\!\langle et, i, j, e\rangle\!\rangle$*. Se duas expressões na linguagem do* λ_{Susp} *têm a mesma estrutura externa, então existe uma correspondência óbvia entre suas subexpressões.*

O lema a seguir caracteriza a importante noção de *forma rm-normal*.

Lema 2.7 ([NW98]) *Uma expressão bem-formada x está na sua forma rm-normal se e somente se uma das seguintes afirmações é verdadeira:*

(a) x é um termo puro na notação de De Bruijn;

(b) x é um termo de contexto da forma $@l$ *ou* (t, l) *onde t é um termo em sua forma rm-normal;*

(c) x é um contexto da forma nil ou et :: *e onde et e e são, respectivamente, um termo de contexto e um contexto na forma rm-normal.*

Demonstração.(Esquema)

A demonstração é feita por inspeção direta das regras contidas nas figuras 2.5 e 2.6. Note que termos puros na notação de De Bruijn não podem ser reduzidos por nenhuma das regras citadas acima, e portanto, os termos puros já estão na sua forma *rm*-normal. Para termos de contexto da forma $@l$ ou (t, l) também não temos nenhuma redução possível desde que t esteja em sua forma *rm*-normal. Analogamente, para contextos simples da forma nil ou $et :: e$ como não temos nenhuma regra de reescrita que se aplique a eles concluímos que os mesmos estão na sua forma *rm*-normal.

Logo expressões bem-formadas que tenham subexpressões da forma $[\![t, i, j, e]\!]$, $\{\!\{e_1, i, j, e_2\}\!\}$ ou $\langle\!\langle et, i, j, e\rangle\!\rangle$ podem ser reescritas utilizando as regras (r_1)-(r_7) e (m_1)-(m_{10}) e, portanto essas expressões não podem estar na sua forma *rm*-normal. □

Observação. Para um termo A qualquer do λ_{Susp}, denotaremos por $rm\text{-}norm(A)$, a sua forma *rm*-normal.

2.2 Adição da regra Eta para o Cálculo λ_{Susp}

O cálculo λ_{Susp} foi originalmente proposto sem a regra *Eta*. Aqui vamos introduzí-la e compará-la com as regras *Eta* correspondentes dos cálculos $\lambda\sigma$ e λs_e a fim de entendermos melhor as relações entre os mesmos. Nos cálculos $\lambda\sigma$ e λs_e as regras *Eta* foram introduzidas, respectivamente, em [DHK00] e [ARK01b], para aplicação no tratamento de problemas de unificação de ordem superior.

A relação $\rightarrow_{rm}$ é terminante e confluente [NW98]. É, portanto, de fundamental importância que a inclusão da regra *Eta* que estamos propondo preserve estas propriedades.

A regra *Eta* do λ_{Susp} é dada pela figura 2.7. Denotaremos esta regra por Eta_{Susp}.

$$\boxed{(Eta_{Susp})\ (\lambda\ (t_1\ \underline{\mathbf{1}})) \rightarrow t_2, \quad \text{se} \quad t_1 =_{rm} [\![t_2, 0, 1, nil]\!]}$$

Figura 2.7: A Regra *Eta* do λ_{Susp}

Intuitivamente a regra Eta_{Susp} pode ser interpretada da seguinte maneira: sempre que for possível aplicar a regra Eta_{Susp} à raiz do redex $\lambda(t_1\ \underline{\mathbf{1}})$ obteremos um termo t_2 que consiste exatamente do termo t_1 com todas as suas variáveis livres decrementadas em 1. Será possível aplicar a regra Eta_{Susp} ao redex $\lambda(t_1\ \underline{\mathbf{1}})$ sempre que não tivermos a ocorrência de $\underline{\mathbf{1}}$ como variável livre em t_1. A proposição 2.9 mostra a corretude dessa regra. Antes disso vejamos o seguinte:

Lema 2.8 *Seja A um termo bem-formado. Então a Susp-normalização do termo $[\![A, k, k+1, @k :: @k-1 :: \ldots :: @1 :: nil]\!]$ nos fornece um novo termo, obtido a partir de A incrementando todas as suas variáveis livres maiores do que k em 1 e deixando todas as outras variáveis inalteradas.*

Demonstração.

Por indução sobre a estrutura do termo A, temos que:

1. Se $A = \underline{\mathbf{n}}$ então,

   (a) se $n > k$ então $[\![\underline{\mathbf{n}}, k, k+1, @k :: \ldots :: @1 :: nil]\!] \rightarrow_{r_5}^{k}$
   $[\![\underline{\mathbf{n-k}}, 0, k+1, nil]\!] \rightarrow_{r_2} \underline{\mathbf{n+1}}$;

   (b) se $n \leq k$ então $[\![\underline{\mathbf{n}}, k, k+1, @k :: \ldots :: @1 :: nil]\!] \rightarrow_{r_5}^{n-1}$
   $[\![\underline{\mathbf{1}}, k-n+1, k+1, @k-n+1 :: \ldots :: @1 :: nil]\!] \rightarrow_{r_3} \underline{\mathbf{n}}$;

2. Se $A = (B\ C)$ então aplicamos (r_6) e utilizamos a hipótese de indução para os termos B e C;

3. Se $A = (\lambda B)$, para B um termo bem-formado, então, como o termo B está limitado por um abstrator adicional, apenas as variáveis livres de B maiores do que $k+1$ é que devem ser incrementadas em 1, enquanto que as outras variáveis devem permanecer inalteradas. Como $[\![(\lambda B), k, k+1, @k :: \ldots :: @1 :: nil]\!] \rightarrow_{r_7} \lambda[\![B, k+1, k+2, @k+1 :: \ldots :: @1 :: nil]\!]$, aplicando a hipótese de indução sobre este último termo e obtemos o resultado desejado.

4. Se $A = [\![t, ol, nl, e]\!]$ então primeiro rm-normalizamos o termo A. Pelo lema 2.7 temos que a análise do termo assim obtido recai em um dos casos anteriores.

□

Proposição 2.9 *Uma aplicação da regra Eta_{Susp}, sempre que possível, ao redex $\lambda(t_1\ \underline{\mathbf{1}})$, produz efetivamente o termo t_2, obtido a partir do termo t_1 da seguinte forma: t_2 consiste do termo t_1 com todas as suas variáveis livres decrementadas em 1.*

Demonstração.

A demonstração é feita por indução sobre a estrutura do termo t_2 considerando-se a premissa $t_1 =_{rm} [\![t_2, 0, 1, nil]\!]$. Veremos que o efeito da rm-normalização de $[\![t_2, 0, 1, nil]\!]$ é incrementar as variáveis livres de t_2 em 1. Sendo assim, quando:

(a) $t_2 = \underline{\mathbf{n}}$ temos $[\![\underline{\mathbf{n}}, 0, 1, nil]\!] \rightarrow_{r_2} \underline{\mathbf{n+1}} =_{rm} t_1$, ou seja, t_2 é obtido de t_1 decrementando em 1 sua única variável livre, a saber, $\underline{\mathbf{n+1}}$.

(b) $t_2 = (A\ B)$, podemos assumir sem perda de generalidade que os termos A e B estão rm-normalizados. Agora, assumindo a hipótese de indução válida para os termos A e B, temos que $[\![(A\ B), 0, 1, nil]\!] \rightarrow_{r_6} [\![A, 0, 1, nil]\!]\ [\![B, 0, 1, nil]\!]$, onde o termo suspenso $[\![A, 0, 1, nil]\!]$ incrementa em 1 todas as variáveis livres de A. Da mesma forma, $[\![B, 0, 1, nil]\!]$ incrementa em 1 todas as variáveis livres do termo B. Como as ocorrências de variáveis livres no termo A não têm interferência sobre as ocorrências de variáveis livres no termo B, podemos concluir que o termo suspenso $[\![(A\ B), 0, 1, nil]\!]$ incrementa em 1 todas as variáveis livres do termo $(A\ B) = t_2$, ou seja, t_1 é obtido de t_2 decrementando em 1 todas as suas variáveis livres.

(c) $t_2 = (\lambda A)$, como no caso anterior, assumiremos que o termo A está na sua forma rm-normal. Assim, $[\![(\lambda A), 0, 1, nil]\!] \rightarrow_{r_7} (\lambda [\![A, 1, 2, @1 :: nil]\!])$. Agora aplicamos o lema 2.8 ao termo $[\![A, 1, 2, @1 :: nil]\!]$ para concluir que todas as variáveis livres de A maiores que 1 são incrementadas em 1 enquanto que as outras permanecem inalteradas. Ou seja, t_2 é obtido de t_1 decrementando em 1 todas as suas variáveis livres maiores que 1.

(d) $t_2 = [\![t, i, j, e]\!]$, para t rm-normalizado temos que $[\![t, i, j, e]\!] \rightarrow^*_{rm} t'$ onde t' é um termo puro na notação de De Bruijn de acordo com o lema 2.7. A análise do termo assim obtido recai nos casos anteriores.

□

Os resultados a seguir serão importantes no sentido de se estabelecer a terminalidade do λ_{Susp} com a regra Eta_{Susp}.

Lema 2.10 ([NW98]) *Seja et um termo de contexto tal que* $ind(et) \leq nl$. *Então para* $j \geq 1$, $\langle\langle et, nl + j, ol + j, et_1 :: \ldots :: et_j :: e\rangle\rangle \rightarrow^*_{rm} \langle\langle et, nl, ol, e\rangle\rangle$.

Demonstração. (Esquema)

Indução em j e repetidas aplicações da regra (m_{10}). □

Lema 2.11 ([NW98]) *Seja* e_1 *um contexto simples. Além disso, suponha que nl e ol sejam números naturais tais que* $(nl - ind(e_1)) \geq ol$. *Então* $\{\!\{e_1, nl, ol, e_2\}\!\} \rightarrow^*_{rm} e_1$.

Demonstração. (Esquema)

Sem perda de generalidade assumiremos que e_2 é um contexto simples, pois caso contrário, podemos rm-reduzí-lo a um tal contexto. A demonstração se dá por indução sobre o comprimento de e_1, $len(e_1)$.

- Se $len(e_1) = 0$ então $e_1 = nil$. Como $ind(nil) = 0$ e $nl \geq ol$ temos por indução sobre ol que $\{\!\{e_1, nl, ol, e_2\}\!\} \rightarrow^*_{rm} nil$

- Se $len(e_1) > 0$ então e_1 é da forma $et_1 :: e'_1$. Portanto

$$\{\!\{e_1, nl, ol, e_2\}\!\} \rightarrow_{m_5} \langle\langle et_1, nl, ol, e_2\rangle\rangle :: \{\!\{e'_1, nl, ol, e_2\}\!\}$$

Temos que $ind(et_1) = ind(e_1)$ pela definição de índice, e adicionalmente como estamos trabalhando com termos bem-formados temos que $ind(e'_1) \leq ind(e_1)$. O resultado desejado é consequência do lema 2.10 e da regra (m_6) juntamente com a hipótese de indução.

□

Construiremos agora uma ordem parcial bem-fundada a fim de estabelecermos a terminalidade de λ_{Susp} com a regra Eta_{Susp}.

A idéia fundamental por trás dessa ordem parcial está na definição de duas medidas δ e μ que calculam o trabalho que ainda precisa ser feito ao se propagar as substituições. Se considerarmos, por exemplo, um termo suspenso da forma $[\![t, ol, nl, e]\!]$ é de se esperar que essas medidas computem a complexidade da estrutura do termo t, já que as substituições devem ser propagadas dentro do mesmo, e também devem computar a complexidade do contexto e, pois estes constituem substituições que, em geral, precisam ser propagadas como evidencia a regra (r_4).

Definição 2.12 ([NW98]) *As medidas δ, sobre expressões, e μ, sobre termos, são dadas pela tabela abaixo:*

Categoria	*Expressão*	$\delta(exp)$	$\mu(exp)$
Termo	*constante*	*0*	*1*
	$\underline{\mathbf{i}}$	*0*	*1*
	$(t_1\ t_2)$	$max(\delta(t_1), \delta(t_2))$	$max(\mu(t_1), \mu(t_2)) + 1$
	(λt)	$\delta(t)$	$\mu(t) + 1$
	$[\![t, ol, nl, e]\!]$	$\mu(t) + \delta(e)$	$\mu(t) + \delta(e) + 1$
Contexto	*nil*	*0*	-
	$et :: e$	$max(\delta(et), \delta(e))$	-
	$\{\!\{e_1, nl, ol, e_2\}\!\}$	$\delta(e_1) + \delta(e_2) + 1$	-
Termo de Contexto	$@l$	*0*	-
	(t, l)	$\mu(t)$	-
	$\langle\!\langle et, nl, ol, e\rangle\!\rangle$	$\delta(et) + \delta(e) + 1$	-

Infelizmente a medida δ ainda não constitui a relação de ordem que desejamos por dois motivos. Primeiro, para algumas regras de reescrita, em particular para (r_5), (m_1), (m_3), (m_5) e (m_{10}), não temos uma simplificação das expressões obtidas pois os termo do lado esquerdo e direito de cada regra têm o mesmo valor de δ. Segundo, ao substituirmos uma subexpressão t_1 de t por outra subexpressão t_2 com $\delta(t_2) < \delta(t_1)$ não necessariamente obtemos uma expressão t' com $\delta(t') < \delta(t)$. Para resolver esses problemas extendemos a medida δ sobre a estrutura das expressões. Os resultados seguintes caminham nessa direção.

Definição 2.13 ([NW98]) *Dadas duas expressões x_1 e x_2, dizemos que $x_1 \sqsupset x_2$ se $\delta(x_1) > \delta(x_2)$ ou se $\delta(x_1) = \delta(x_2)$ e uma das seguintes afirmações é verdadeira:*

(1) $x_1 = \underline{\mathbf{i}}$ *e* $x_2 = \underline{\mathbf{j}}$ *onde* $i > j$*;*

(2) $x_1 = [\![t_1, ol_1, nl_1, e_1]\!]$, $x_2 = [\![t_2, ol_2, nl_2, e_2]\!]$ *e* $\delta(t_1) > \delta(t_2)$*;*

(3) $x_1 = \{\!\{e_1, nl, ol, e_2\}\!\}$, $x_2 = et :: e$ *e* $x_1 \sqsupset e$*;*

(4) x_1 *e* x_2 *têm a mesma estrutura externa e também têm subexpressões imediatas tais que cada subexpressão imediata de* x_1 *é idêntica a correspondente subexpressão imediata de* x_2 *exceto por um par de subexpressões imediatas* x_1' *de* x_1 *e* x_2' *de* x_2 *tais que* $x_1' \sqsupset x_2'$*.*

(5) x_2 *é uma subexpressão imediata de* x_1*.*

A relação $\sqsupset$ não é transitiva, logo, não é uma ordem parcial. Considere então a seguinte:

Definição 2.14 ([NW98]) *A relação $\succ$ define-se sobre expressões da linguagem do λ_{Susp} como o fecho transitivo da relação $\sqsupset$.*

Lema 2.15 ([NW98]) *Não existem cadeias infinitas de redução de expressões $x_1, x_2, \ldots, x_n, \ldots$ tais que $x_1 \sqsupset x_2 \sqsupset \ldots \sqsupset x_n \sqsupset \ldots$*

Demonstração.(Esquema) A prova é feita por indução sobre $\delta(x_1)$. Note que se $x \sqsupset y$ então $\delta(x) \geq \delta(y)$. Então precisamos mostrar que não existem cadeias infinitas de redução de expressões $x_1, x_2, \ldots, x_n, \ldots$, com $\delta(x_i) \geq \delta(x_j)$, para $i, j \geq 1$. Note que o caso estrito é imediato. Quando $\delta(x) = \delta(y)$ a demonstração é feita por indução sobre a estrutura de x considerando sua categoria sintática. □

Proposição 2.16 ([NW98]) *A relação $\succ$ é uma ordem parcial bem-fundada definida sobre expressões do λ_{Susp}.*

Demonstração.(Esquema)

Precisamos mostrar que $\succ$ é anti-reflexiva para que seja uma ordem parcial. Além disso, precisamos mostrar que $\succ$ é bem fundada. Ambos os fatos são consequência imediata do fato de não existirem cadeias infinitas de redução para a relação $\succ$ (lema 2.15). □

Lema 2.17 ([NW98]) *Sejam x_1 e x_2 expressões da mesma categoria sintática tais que $\delta(x_1) \geq \delta(x_2)$ e, se x_1 e x_2 são termos então $\mu(x_1) \geq \mu(x_2)$. Se x resulta de y substituindo a subexpressão x_1 por x_2 então $\delta(y) \geq \delta(x)$ e, se x e y são termos, $\mu(y) \geq \mu(x)$.*

Demonstração.(Esquema) Indução sobre a estrutura das expressões e inspeção da definição 2.12. □

Lema 2.18 ([NW98]) *Se $l \to r$ é uma instância de alguma das regras das figuras 2.5 ou 2.6 então $\delta(l) \geq \delta(r)$ e, se l e r são termos, então $\mu(l) \geq \mu(r)$.*

Demonstração.(Esquema) Inspeção direta das regras em questão. □

Lema 2.19 ([NW98]) *Sejam x_1 e x_2 expressões bem-formadas. Se $x_1 \to^*_{rm} x_2$ então $\delta(x_1) \geq \delta(x_2)$.*

Demonstração.(Esquema) Consequência imediata dos lemas 2.17 e 2.18. □

Lema 2.20 ([NW98]) *Se $l \to r$ é uma instância de alguma das regras das figuras 2.5 ou 2.6 então $l \succ r$.*

Demonstração.(Esquema) A idéia que está por trás da demonstração segue do fato que para qualquer instância $l \to r$ das regras citadas, o termo r é mais simples do que o termo l, no seguinte sentido: O termo r é mais simples do que o termo l quando $\mu(l) > \mu(r)$. Analogamente, a expressão r é mais simples do que a expressão l quando $\delta(l) > \delta(r)$. □

Lema 2.21 *Seja M uma expressão bem-formada do λ_{Susp}. Se $\lambda(M\ \underline{\mathbf{1}}) \to_{Eta_{Susp}} N$ então $\mu(\lambda(M\ \underline{\mathbf{1}})) \geq \mu(N)$.*

Demonstração.

A idéia que aqui utilizaremos foi desenvolvida por [Bor95] para o $\lambda\sigma$ e por [ARK01a] para o λs_e. Essa idéia é um algoritmo para aplicação da regra *Eta* que pode ser assim resumida: Para $M \in \Lambda_{dB}$ sabemos que a regra η se aplica ao redex $\lambda(M\ \underline{\mathbf{1}})$ se M não possui ocorrências livres da variável $\underline{\mathbf{1}}$. Por outro lado isto equivale a normalizar o termo $((\lambda M)\ \diamond)$ onde $\diamond$ é um símbolo "dummy" que não

pertence a linguagem do λ_{Susp}. Esse símbolo "dummy" pode ser considerado como uma constante que estende a linguagem do λ_{Susp}. Essa idéia, que será desenvolvida com detalhes no capítulo 4, pode ser resumida através da seguinte regra:

$$\lambda(M\ \underline{1}) \rightarrow_{Eta_{Susp}} N \ \text{ onde } N = \textit{rm-norm}([\![M, 1, 0, (\diamond, 0) :: \textit{nil}]\!])$$

se $\diamond$ não ocorre em $\textit{rm-norm}([\![M, 1, 0, (\diamond, 0) :: \textit{nil}]\!])$.

Por um lado, como M é um termo, $\mu(\lambda(M\ \underline{1})) = \mu((M\ \underline{1}))+1 = \text{máx}(\mu(M), 1)+2 = \mu(M) + 2$.

Por outro lado, $\mu([\![M, 1, 0, (\diamond, 0) :: \textit{nil}]\!]) = \mu(M) + \delta((\diamond, 0) :: \textit{nil}) + 1 = \mu(M) + \text{máx}(\delta((\diamond, 0), \delta(\textit{nil})) + 1 = \mu(M) + \delta((\diamond, 0)) + 1 = \mu(M) + \mu(\diamond) + 1 = \mu(M) + 2$. Portanto, $\mu(\lambda(M\ \underline{1})) = \mu([\![M, 1, 0, (\diamond, 0) :: \textit{nil}]\!])$ e, pelo lema 2.17 concluimos que $\mu([\![M, 1, 0, (\diamond, 0) :: \textit{nil}]\!]) \geq \mu(\textit{rm-norm}([\![M, 1, 0, (\diamond, 0) :: \textit{nil}]\!])) = N$. □

Notação. Para simplificar a notação vamos nos referir ao sistema $rm + Eta_{Susp}$ como sendo Sp.

Lema 2.22 *Sejam x_1 e x_2 expressões bem-formadas. Se $x_1 \rightarrow_{Sp} x_2$ então $x_1 \succ x_2$.*

Demonstração.

Por indução sobre a estrutura de x_1, temos que se x_1 é uma instância de uma das regras de Sp então o resultado é consequência imediata dos lemas 2.20 e 2.21 e da definição 2.13. Caso contrário, se x_1 e x_2 têm a mesma estrutura externa então, pelo lema 2.17, temos $\delta(x_1) \geq \delta(x_2)$. No caso em que $\delta(x_1) > \delta(x_2)$ o resultado é imediato a partir da definição de $\succ$. Quando $\delta(x_1) = \delta(x_2)$, temos por hipótese de indução e aplicação da relação Sp que existe uma subexpressão imediata x_1' de x_1 e uma correspondente subexpressão imediata x_2' de x_2 tais que $x_1' \succ x_2'$ e qualquer outra subexpressão imediata de x_1 é idêntica à correspondente subexpressão imediata de x_2. Usando a definição de $\succ$ concluimos então que $x_1 \succ x_2$ □

Proposição 2.23 *A relação Sp é terminante.*

Demonstração.

O resultado é consequência imediata dos lemas 2.16 e 2.22. □

A terminalidade da relação Sp nos garante a existência de formas normais, cuja notação obedece à forma determinada pela definição 2.7.

Proposição 2.24 *A relação Sp é local-confluente.*

Demonstração.

Sabemos que a relação rm é local-confluente [NW98]. Para mostrar que a nova relação Sp é também local-confluente basta verificar que todos os pares críticos gerados pela regra Eta_{Susp} com alguma das regras de rm são juntáveis. Para isso utilizamos o critério dos pares críticos de Knuth-Bendix [KB70], dado pelo lema 1.20.

Inicialmente note que não há sobreposição da regra Eta_{Susp} com ela mesma, assim como entre as regras dadas nas figuras 2.6 e 2.7. Temos uma única sobreposição possível da regra Eta_{Susp} com a regra (r_7) da figura 2.5.

Na figura 2.8 podemos observar o termo obtido com esta sobreposição, assim como a divergência ao se aplicar (r_7) por um lado e Eta_{Susp} por outro.

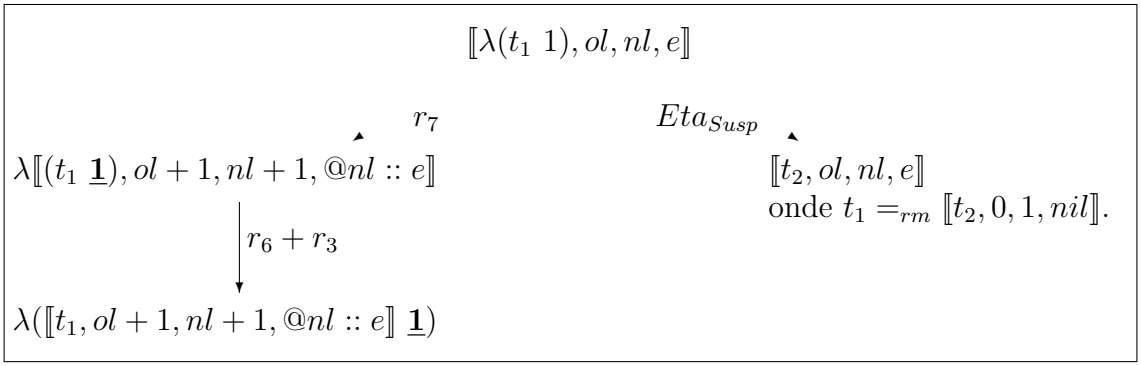

Figura 2.8: Divergência gerada pelas regras (r_7) e Eta_{Susp}

Considerando os termos t_1 e t_2 rm-normalizados, vejamos que esta divergência é juntável. A demonstração se dá pela análise da estrutura do termo t_1:

1. $t_1 = \underline{\mathbf{n}}$. Aqui precisamos que $n > 1$ para que a regra Eta_{Susp} possa ser aplicada. Neste ponto estamos com $\lambda([\![\underline{\mathbf{n}}, ol+1, nl+1, @nl :: e]\!]\ \underline{\mathbf{1}})$. De acordo com o comprimento do contexto $@nl :: e$ temos os seguintes casos:

   (a) Se $ol + 1 < n$ temos a seguinte cadeia de redução:

$\lambda([\![\underline{\mathbf{n}}, ol+1, nl+1, @nl :: e]\!]\ \underline{\mathbf{1}}) \to_{r_5}^{ol+1}$
$\lambda([\![\underline{\mathbf{n-ol-1}}, 0, nl+1, nil]\!]\ \underline{\mathbf{1}}) \to_{r_2}$
$\lambda(\underline{\mathbf{n-ol+nl}}\ \underline{\mathbf{1}}) \to_{Eta_{Susp}}$
$\underline{\mathbf{n-ol+nl-1}}$.

Por outro lado, temos que $t_1 =_{rm} [\![t_2, 0, 1, nil]\!]$ e portanto, $t_2 = \underline{\mathbf{n-1}}$. Logo,

$[\![t_2, ol, nl, e]\!] =$
$[\![\underline{\mathbf{n-1}}, ol, nl, e]\!] \to_{r_5}^{ol}$
$[\![\underline{\mathbf{n-1-ol}}, 0, nl, nil]\!] \to_{r_2}$
$\underline{\mathbf{n-ol+nl-1}}$

(b) Quando $ol + 1 \geq n$ temos

$\lambda([\![\underline{\mathbf{n}}, ol+1, nl+1, @nl :: e]\!]\ \underline{\mathbf{1}}) \to_{r_5}^{n-1}$
$\lambda([\![\underline{\mathbf{1}}, ol-n+2, nl+1, e_1 :: e']\!]\ \underline{\mathbf{1}})$.

A redução a partir desse ponto depende da estrutura de e_1: quando $e_1 = @l$ aplicamos (r_3) obtendo $\lambda(\underline{\mathbf{nl+1-l}}\ \underline{\mathbf{1}}) \to_{Eta_{Susp}} \underline{\mathbf{nl-l}}$. Por outro lado, temos

$[\![t_2, ol, nl, e]\!] =$
$[\![\underline{\mathbf{n-1}}, ol, nl, e]\!] \to_{r_5}^{n-2}$
$[\![\underline{\mathbf{1}}, ol-n+2, nl, e_1 :: e']\!] =$
$[\![\underline{\mathbf{1}}, ol-n+2, nl, @l :: e']\!] \to_{r_3} \underline{\mathbf{nl-l}}$

A outra possibilidade é $e_1 = (t, l)$, onde, sem perda de generalidade, podemos considerar t rm-normalizado. Neste caso,

$\lambda([\![\underline{\mathbf{1}}, ol-n+2, nl+1, (t,l) :: e']\!]\ \underline{\mathbf{1}}) \to_{r_4}$
$\lambda([\![t, 0, nl-l+1, nil]\!]\ \underline{\mathbf{1}}) \to_{Eta_{Susp}}$
$rm\text{-}norm([\![[\![t, 0, nl+1-l, nil]\!], 1, 0, (\diamond, 0) :: nil]\!]) \to_{m_1}$
$rm\text{-}norm([\![t, 0, nl-l, \{\!\{nil, nl+1-l, 1, (\diamond, 0) :: nil\}\!\}]\!]) \to_{m_3}$
$rm\text{-}norm([\![t, 0, nl-l, \{\!\{nil, nl-l, 0, nil\}\!\}]\!]) \to_{m_2}$
$rm\text{-}norm([\![t, 0, nl-l, nil]\!])$.

Por outro lado,

$$[\![t_2, ol, nl, e]\!] = [\![\underline{\mathbf{n-1}}, ol, nl, e]\!] \to_{r_5}^{n-2}$$
$$[\![\underline{\mathbf{1}}, ol - n + 2, nl, (t, l) :: e']\!] \to_{r_4} [\![t, 0, nl - l, nil]\!].$$

como *rm-norm*$([\![t, 0, nl - l, nil]\!])$ e $[\![t, 0, nl - l, nil]\!]$ são obviamente juntáveis, temos a convergência.

2. $t_1 = (A\ B)$.
Temos $\lambda([\![t_1, ol + 1, nl + 1, @nl :: e]\!]\ \underline{\mathbf{1}}) =_{rm} \lambda([\![(A\ B), ol + 1, nl + 1, @nl :: e]\!]\ \underline{\mathbf{1}})$.
Agora note que $(A\ B) =_{rm} [\![t_2, 0, 1, nil]\!]$ e, portanto podemos fazer a seguinte redução:

$$\lambda([\![(A\ B), ol + 1, nl + 1, @nl :: e]\!]\ \underline{\mathbf{1}}) =$$
$$\lambda([\![[\![t_2, 0, 1, nil]\!], ol + 1, nl + 1, @nl :: e]\!]\ \underline{\mathbf{1}}) \to_{m_1}$$
$$\lambda([\![t_2, ol, nl + 1, \{\!\{nil, 1, ol + 1, @nl :: e\}\!\}]\!]\ \underline{\mathbf{1}}) \to_{m_3}$$
$$\lambda([\![t_2, ol, nl + 1, \{\!\{nil, 0, ol, e, \}\!\}]\!]\ \underline{\mathbf{1}}) \to_{m_4}$$
$$\lambda([\![t_2, ol, nl + 1, e]\!]\ \underline{\mathbf{1}}) \to_{Eta_{Susp}}$$
$$rm\text{-}norm([\![[\![t_2, ol, nl + 1, e]\!], 1, 0, (\diamond, 0) :: nil]\!]) \to_{m_1}$$
$$rm\text{-}norm([\![t_2, ol, nl, \{\!\{e, nl + 1, 1, (\diamond, 0) :: nil\}\!\}]\!]) \to_{lema2.11}$$
$$rm\text{-}norm([\![t_2, ol, nl, e]\!])$$

Por outro lado tínhamos o termo $[\![t_2, ol, nl, e]\!]$, mas esses dois últimos termos se juntam trivialmente.

3. $t_1 = (\lambda A)$. Neste caso, temos $\lambda([\![(\lambda A), ol + 1, nl + 1, @nl :: e]\!]\ \underline{\mathbf{1}})$ onde $\lambda A =_{rm} [\![t_2, 0, 1, nil]\!]$. Fazendo essa substituição, temos:

$$\lambda([\![[\![t_2, 0, 1, nil]\!], ol + 1, nl + 1, @nl :: e]\!]\ \underline{\mathbf{1}}) \to_{m_1}$$
$$\lambda([\![t_2, ol, nl + 1, \{\!\{nil, 1, ol + 1, @nl :: e\}\!\}]\!]\ \underline{\mathbf{1}}) \to_{m_3}$$
$$\lambda([\![t_2, ol, nl + 1, \{\!\{nil, 0, ol, e\}\!\}]\!]\ \underline{\mathbf{1}}) \to_{m_4}$$
$$\lambda([\![t_2, ol, nl + 1, e]\!]\ \underline{\mathbf{1}}) \to_{Eta_{Susp}}$$
$$rm\text{-}norm([\![[\![t_2, ol, nl + 1, e]\!], 1, 0, (\diamond, 0) :: nil]\!]) \to_{m_1}$$
$$rm\text{-}norm([\![t_2, ol, nl, \{\!\{e, nl + 1, 1, (\diamond, 0) :: nil\}\!\}]\!]) \to_{lema2.11}$$
$$rm\text{-}norm([\![t_2, ol, nl, e]\!])$$

Por outro lado tínhamos o termo $[\![t_2, ol, nl, e]\!]$, mas esses dois últimos termos se juntam trivialmente.

□

Proposição 2.25 *A relação Sp é confluente.*

Demonstração.

Aplicando o Lema do Diamante de Newman (lema 1.18) tendo como hipótese a proposição 2.23 e o lema anterior, obtemos o resultado desejado. □

2.3 A Correspondência entre as Regras Eta

Nesta seção veremos a correspondência entre as regras Eta_σ, Eta_{s_e} e Eta_{Susp}.

Em [ARK01b], Ayala-Rincón e Kamareddine mostram a correspondência entre as regras *Eta* do $\lambda\sigma$ e do λs_e, isto é, a correspondência entre as condições $t_1[\uparrow] =_\sigma M$ e $\varphi_0^2 t_1 =_{s_e} M$, onde $t_1 \in \Lambda_{dB}$, como veremos a seguir.

Lema 2.26 ([ARK01b]) *Seja* $\underline{\mathbf{n}}$ *um índice de De Bruijn. Então para* $k \geq 0$ *a forma* s_e*-normal de* $\varphi_k^2\underline{\mathbf{n}}$ *e a forma* σ*-normal de* $\underline{\mathbf{n}}[\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\underline{\mathbf{1}}[\uparrow^2].\ldots.\underline{\mathbf{1}}[\uparrow^{k-1}].\uparrow^{k+1}]$ *são índices de De Bruijn correspondentes entre si.*

Lema 2.27 ([ARK01b]) *Seja* λt_1 *uma abstração sobre a linguagem* Λ_{dB}*. Então temos, para* $k \geq 0$*, que o termo* $(\lambda t_1)[\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\underline{\mathbf{1}}[\uparrow^2].\ldots.\underline{\mathbf{1}}[\uparrow^{k-1}].\uparrow^{k+1}]$ σ*-reduz-se para* $\lambda(t_1[\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\underline{\mathbf{1}}[\uparrow^2]\ldots\underline{\mathbf{1}}[\uparrow^k].\uparrow^{k+2}])$.

A correspondência entre $t_1[\uparrow]$ e $\varphi_0^2 t_1$ é o caso $k = 0$ do seguinte lema:

Lema 2.28 ([ARK01b]) *Seja* $t_1 \in \Lambda_{dB}$ *e* t_1' *sua codificação na linguagem do cálculo* $\lambda\sigma$*, onde todos os índices* $\underline{\mathbf{n}} \in \mathbb{N}$ *que ocorrem em* t_1 *são substituidos por* $\underline{\mathbf{1}}[\uparrow^{n-1}]$*. Então, para* $k \geq 0$*, a forma* σ*-normal de* $t_1'[\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\underline{\mathbf{1}}[\uparrow^2].\ldots.\underline{\mathbf{1}}[\uparrow^{k-1}].\uparrow^{k+1}]$ *corresponde à forma* s_e*-normal de* $\varphi_k^2 t_1$*.*

De forma análoga estabeleceremos a correspondência entre as regras *Eta* do λ_{Susp} e do λs_e. Isto corresponde à equivalência entre as condições $[\![t_1, 0, 1, nil]\!] =_{Susp} M$ e $\varphi_0^2 t_1 =_{s_e} M$ para $t_1 \in \Lambda_{dB}$. A correspondência desejada corresponde ao caso $k = 0$ da seguinte proposição:

Proposição 2.29 *Seja $t_1 \in \Lambda_{dB}$. Então, para todo $k \geq 0$, a forma Susp-normal de $[\![t_1, k, k+1, @k :: @k-1 :: \ldots :: @1 :: nil]\!]$ corresponde à forma s_e-normal de $\varphi_k^2 t_1$.*

Demonstração.

A demonstração é feita por indução sobre a estrutura do termo t_1. No caso em que:

- $t_1 = \underline{\mathbf{n}}$. De fato, pelo lema 2.8 temos que para todo $k \geq 0$,
  $[\![\underline{\mathbf{n}}, k, k+1, @k :: @k-1 :: \ldots :: @1 :: nil]\!] \rightarrow \begin{cases} \underline{\mathbf{n+1}} & \text{se } n > k \\ \underline{\mathbf{n}} & \text{se } n \leq k \end{cases}$
  Por outro lado, temos $\varphi_k^2 \underline{\mathbf{n}} \rightarrow_{\varphi\text{-}destr} \begin{cases} \underline{\mathbf{n+1}} & \text{se } n > k \\ \underline{\mathbf{n}} & \text{se } n \leq k \end{cases}$

- $t_1 = (A\ B)$ aplicamos a hipótese de indução a A e a B. De fato,
  $Susp\text{-}norm([\![(A\ B), k, k+1, @k :: @k-1 :: \ldots :: @1 :: nil]\!]) \rightarrow_{r_6} Susp\text{-}norm($
  $[\![A, k, k+1, @k :: \ldots :: @1 :: nil]\!]\ \ [\![B, k, k+1, @k :: \ldots :: @1 :: nil]\!]) \overset{HI}{\equiv}$
  $s_e\text{-}norm(\varphi_k^2 A\ \varphi_k^2 B) \equiv s_e\text{-}norm(\varphi_k^2 a)$

- $t_1 = (\lambda A)$ temos
  $Susp\text{-}norm([\![(\lambda A), k, k+1, @k :: @k-1 :: \ldots :: @1 :: nil]\!]) \rightarrow_{r_7}$
  $Susp\text{-}norm((\lambda [\![A, k+1, k+2, @k+1 :: \ldots :: @1 :: nil]\!])) \overset{HI}{\equiv}$
  $s_e\text{-}norm((\lambda \varphi_{k+1}^2 A))$.
  Por outro lado, $\varphi_k^2(\lambda A) \rightarrow_{\sigma\lambda\text{-}trans} (\lambda \varphi_{k+1}^2 A)$. □

Aqui vale salientar que essa correspondência não ocorre de maneira óbvia para termos abertos. De fato, seja c uma constante qualquer. Por um lado, no λ_{Susp}, temos que $[\![c, k, k+1, @k :: \ldots :: @1 :: nil]\!] \rightarrow_{r_1} c$. Mas por outro lado, no cálculo λs_e temos que $\varphi_k^2 c$ é irredutível. Esses dois termos podem, num certo sentido, serem considerados iguais pois a função φ_k^2 não vai modificar a constante c e esta equivalência pode ser estabelecida num contexto mais amplo como, por exemplo, em unificação de ordem superior.

Capítulo 3

Adequabilidade

Neste capítulo compararemos os cálculos λ_{Susp}, λs_e e $\lambda\sigma$. Essa comparação é feita no estilo de [KR00]. Isto nos permitirá concluir que, num certo sentido, o λs_e simula um passo de β-conversão de maneira mais adequada que o λ_{Susp}. Intuitivamente, esta noção de adequabilidade está relacionada com o comprimento das simulações de um passo de β-conversão. De forma que se um cálculo de substituições explícitas simula um passo de β-conversão com derivações mais curtas ele será considerado mais adequado. A seguir definiremos formalmente adequabilidade.

Veremos que os cálculos λ_{Susp} e $\lambda\sigma$ são incomparáveis assim como também o são o λs_e e o $\lambda\sigma$. Informalmente isto significa que, no caso do λ_{Susp} e $\lambda\sigma$, para algumas simulações o λ_{Susp} se comporta de maneira mais adequada enquanto que para outras o $\lambda\sigma$ é mais adequado.

3.1 Noções de Adequabilidade para Comparação de Cálculos de Substituições Explícitas

Definição 3.1 ([KR00]) *Dizemos que λ_{ξ_1} é* **mais adequado** *que λ_{ξ_2} (na simulação de um passo de β-conversão), denotado por $\lambda_{\xi_1} \prec \lambda_{\xi_2}$ se:*

1. *Para toda β-redução $A \to_\beta B$ e para toda λ_{ξ_2}-simulação $A \to^n_{\lambda_{\xi_2}} B$ existe uma λ_{ξ_1}-simulação $A \to^m_{\lambda_{\xi_1}} B$ tal que $m \leq n$.*
2. *Existe uma β-redução $A \to_\beta B$ e uma λ_{ξ_1}-simulação $A \to^m_{\lambda_{\xi_1}} B$ tal que para toda λ_{ξ_2}-simulação $A \to^n_{\lambda_{\xi_2}} B$ temos $m < n$.*

É fácil verificar que $\prec$ é transitiva e anti-simétrica. De fato, suponha que $\lambda_{\xi_1} \prec \lambda_{\xi_2}$ e $\lambda_{\xi_2} \prec \lambda_{\xi_3}$. De $\lambda_{\xi_2} \prec \lambda_{\xi_3}$, temos que para toda β-conversão $A \to_\beta B$ e para toda

λ_{ξ_3}-simulação $A \to^{p}_{\lambda_{\xi_3}} B$ existe uma λ_{ξ_2}-simulação $A \to^{n}_{\lambda_{\xi_2}} B$ com $n \leq p$. Por outro lado, como $\lambda_{\xi_1} \prec \lambda_{\xi_2}$, para a mesma λ_{ξ_2}-simulação $A \to^{n}_{\lambda_{\xi_2}} B$ citada acima existe uma λ_{ξ_1}-simulação $A \to^{m}_{\lambda_{\xi_1}} B$ com $m \leq n$. Sendo assim, para toda λ_{ξ_3}-simulação $A \to^{p}_{\lambda_{\xi_3}} B$ existe uma λ_{ξ_1}-simulação $A \to^{m}_{\lambda_{\xi_1}} B$ com $m \leq p$ pois $m \leq n$ e $n \leq p$. O caso estrito é tratado de forma semelhante.

Para verificar que $\prec$ é anti-simétrica, precisamos verificar que não é o caso de termos simultaneamente $\lambda_{\xi_1} \prec \lambda_{\xi_2}$ e $\lambda_{\xi_2} \prec \lambda_{\xi_1}$. De fato, suponha verdadeiro $\lambda_{\xi_1} \prec \lambda_{\xi_2}$, então, pela condição 2 da definição temos que existe uma β-conversão $A \to_{\beta} B$ e uma λ_{ξ_1}-simulação $A \to^{m}_{\lambda_{\xi_1}} B$ tal que para toda λ_{ξ_2}-simulação $A \to^{n}_{\lambda_{\xi_2}} B$ temos $m < n$. Suponha, por absurdo, que $\lambda_{\xi_2} \prec \lambda_{\xi_1}$. Então, pela condição 1 da definição podemos tomar a β-conversão do caso anterior, obtendo que $n \leq m$. Absurdo! Pois $m < n$.

Segundo a definição de adequabilidade dada acima, para mostrar que dois cálculos, por exemplo λ_{ξ_1} e λ_{ξ_2}, não podem ser comparados basta encontrarmos duas β-conversões clássicas $A \to_{\beta} B$ e $C \to_{\beta} D$ tais que:

1. Existe uma simulação $A \to_{\lambda_{\xi_1}} B$ mais curta do que a mais curta das simulações $A \to_{\lambda_{\xi_2}} B$.

2. Existe uma simulação $C \to_{\lambda_{\xi_2}} D$ mais curta do que a mais curta das simulações $C \to_{\lambda_{\xi_1}} D$.

Neste caso dizemos que os cálculos λ_{ξ_1} e λ_{ξ_2} são *incomparáveis*.

3.2 Incomparabilidade entre Cálculos de Substituições Explícitas

3.2.1 Os Cálculos $\lambda\sigma$ e λs_e são Incomparáveis

Iniciaremos definindo duas funções importantes para estimar comprimentos de derivações nos cálculos que estamos trabalhando.

Definição 3.2 *Sejam $A, B \in \Lambda_{dB}$. Definimos a função $M : \Lambda_{dB} \to \mathbb{N}$ indutivamente como segue:*

- $M(\underline{\mathbf{n}}) = 1$

- $M(A\ B) = M(A) + M(B) + 1$
- $M(\lambda A) = M(A) + 1$

Definição 3.3 *Sejam $A, B, C \in \Lambda_{dB}$ e $k \geq 0$. Definimos a função $Q_k : \Lambda_{dB} \times \Lambda_{dB} \to \mathbb{N}$ indutivamente como segue:*

- $Q_k(\underline{\mathbf{n}}, B) = \begin{cases} n & \text{se } n < k \\ n + M(B) & \text{se } n = k \\ k + 1 & \text{se } n > k \end{cases}$
- $Q_k(A\ B, C) = Q_k(A, C) + Q_k(B, C) + 1$
- $Q_k(\lambda A, B) = Q_{k+1}(A, B) + 1$

Em [KR00], Kamareddine e Ríos mostraram que os cálculos λs e $\lambda\sigma$ são incomparáveis. Aqui extenderemos esse resultado para o λs_e:

Proposição 3.4 *Os cálculos λs_e e $\lambda\sigma$ são incomparáveis.*

Antes da demonstração dessa proposição consideremos alguns lemas que serão importantes para estabelecer o resultado desejado. Os lemas a seguir foram tratados por [KR00] apenas para o cálculo λs. Os enunciados aqui propostos extendem o resultado para o λs_e. Note, no entanto, que essa extensão é imediata já que estamos considerando termos puros do λ-cálculo.

Lema 3.5 ([KR00]) *Para $A \in \Lambda_{dB}$, toda s_e-derivação de $\varphi_k^i A$ para sua forma s_e-normal tem comprimento $M(A)$.*

Demonstração.

Indução sobre a estrutura do termo A. □

Lema 3.6 ([KR00]) *Toda λs_e-derivação de $(\lambda\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}}))\ \underline{\mathbf{1}}^n$ para sua forma λs_e-normal tem comprimento $4n + 3$*

Demonstração.

Considere a seguinte derivação $(\lambda\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}}))\ \underline{\mathbf{1}}^n \to (\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}}))\sigma^1\underline{\mathbf{1}}^n \to \lambda((\underline{\mathbf{2}}\ \underline{\mathbf{2}})\sigma^2\underline{\mathbf{1}}^n) \to \lambda((\underline{\mathbf{2}}\sigma^2\underline{\mathbf{1}}^n)\ (\underline{\mathbf{2}}\sigma^2\underline{\mathbf{1}}^n))$. Como as duas ocorrências de $(\underline{\mathbf{2}}\sigma^2\underline{\mathbf{1}}^n)$ não interagem entre si,

é suficiente mostrar que todas as derivações de $(\underline{\mathbf{2}}\sigma^2\underline{\mathbf{1}}^n)$ têm comprimento $2n$. De fato, $(\underline{\mathbf{2}}\sigma^2\underline{\mathbf{1}}^n)$ se reduz diretamente para $\varphi_0^2(\underline{\mathbf{1}}^n)$ cujo comprimento, de acordo com o lema 3.5 é $M(\underline{\mathbf{1}}^n)$ que por sua vez é igual a $2n-1$. A derivação total, portanto, tem comprimento $4n+3$. □

Lema 3.7 ([KR00]) *Existe uma derivação de* $(\lambda\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}}))\ \underline{\mathbf{1}}^n$ *para sua forma* $\lambda\sigma$*-normal cujo comprimento é* $n+9$.

Demonstração.

Basta considerar a seguinte derivação:

$$
\begin{array}{l}
(\lambda\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}}))\ \underline{\mathbf{1}}^n = \\
(\lambda\lambda(\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}}[\uparrow]))\ \underline{\mathbf{1}}^n \to_{Beta} \\
(\lambda(\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}}[\uparrow]))[\underline{\mathbf{1}}^n.id] \to_{Abs} \\
\lambda((\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.((\underline{\mathbf{1}}^n.id)\circ\uparrow)]) \to_{Map} \\
\lambda((\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.(\underline{\mathbf{1}}^n[\uparrow].(id\circ\uparrow))]) \to_{App}^{n-1} \\
\lambda((\underline{\mathbf{1}}[\uparrow]\ \underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.((\underline{\mathbf{1}}[\uparrow])^n.(id\circ\uparrow))]) \to_{App} \\
\lambda((\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}[\uparrow])^n.(id\circ\uparrow))])\ (\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}[\uparrow])^n.(id\circ\uparrow))])) \to_{Clos} \\
\lambda((\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow])^n.(id\circ\uparrow))])\ (\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}[\uparrow])^n.(id\circ\uparrow))])) \to_{ShiftCons} \\
\lambda((\underline{\mathbf{1}}[(\underline{\mathbf{1}}[\uparrow])^n.(id\circ\uparrow)])\ (\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}[\uparrow])^n.(id\circ\uparrow))])) \to_{VarCons} \\
\lambda((\underline{\mathbf{1}}[\uparrow])^n\ (\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}[\uparrow])^n.(id\circ\uparrow))])) \to^3 \\
\lambda((\underline{\mathbf{1}}[\uparrow])^n\ (\underline{\mathbf{1}}[\uparrow])^n) = \lambda(\underline{\mathbf{2}}^n\ \underline{\mathbf{2}}^n)
\end{array}
$$

□

Demonstração (da proposição 3.4)

Pelos lemas 3.6 e 3.7 temos que, para $n \geq 3$, $\lambda s_e \not\prec \lambda\sigma$.

Por outro lado, é fácil ver que o termo $(\lambda\underline{\mathbf{2}})\ \underline{\mathbf{1}} \to_\beta \underline{\mathbf{1}}$ tem simulação única tanto no λs_e quanto no $\lambda\sigma$, cujos comprimentos são respectivamente iguais a 2 e 4. Portanto, $\lambda\sigma \not\prec \lambda s_e$. □

3.2.2 Os Cálculos λ_{Susp} e $\lambda\sigma$ são Incomparáveis.

Vejamos agora que os cálculos λ_{Susp} e $\lambda\sigma$ também são incomparáveis entre si.

Lema 3.8 *Toda* λ_{Susp}*-derivação de* $(\lambda\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}}))\underline{\mathbf{1}}^n$ *para sua forma* λ_{Susp}*-normal tem comprimento* $4n+5$.

Demonstração.

De fato, note que a derivação a seguir é única:

$$
\begin{array}{l}
(\lambda\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}}))\underline{\mathbf{1}}^n \to_{\beta_s} \\
[\![(\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}})), 1, 0, (\underline{\mathbf{1}}^n, 0) :: nil]\!] \to_{r_7} \\
\lambda[\![(\underline{\mathbf{2}}\ \underline{\mathbf{2}}), 2, 1, @0 :: (\underline{\mathbf{1}}^n, 0) :: nil]\!] \to_{r_6} \\
\lambda([\![\underline{\mathbf{2}}, 2, 1, @0 :: (\underline{\mathbf{1}}^n, 0) :: nil]\!]\ [\![\underline{\mathbf{2}}, 2, 1, @0 :: (\underline{\mathbf{1}}^n, 0) :: nil]\!]) \to^2_{r_5} \\
\lambda([\![\underline{\mathbf{1}}, 1, 1, (\underline{\mathbf{1}}^n, 0) :: nil]\!]\ [\![\underline{\mathbf{1}}, 1, 1, (\underline{\mathbf{1}}^n, 0) :: nil]\!]) \to^2_{r_4} \\
\lambda([\![\underline{\mathbf{1}}^n, 0, 1, nil]\!]\ [\![\underline{\mathbf{1}}^n, 0, 1, nil]\!]) \to^{2(n-1)}_{r_6} \\
\lambda(([\![\underline{\mathbf{1}}, 0, 1, nil]\!])^n\ ([\![\underline{\mathbf{1}}, 0, 1, nil]\!])^n) \to^{2n}_{r_2} \\
\lambda(\underline{\mathbf{2}}^n\ \underline{\mathbf{2}}^n).
\end{array}
$$

□

Lema 3.9 *Todas as derivações de* $(\lambda\lambda\underline{\mathbf{2}})\ \underline{\mathbf{1}}$ *para sua forma* $\lambda\sigma$*-normal têm comprimento maior ou igual a 6.*

Demonstração.

De fato, as possíveis derivações de $(\lambda\lambda\underline{\mathbf{2}})\ \underline{\mathbf{1}}$ são:

- $(\lambda\lambda\underline{\mathbf{1}}[\uparrow])\ \underline{\mathbf{1}} \to_{Beta} (\lambda\underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.id] \to_{Abs} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow)] \to_{Clos}$
  $\lambda\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow))] \to_{ShiftCons} \lambda\underline{\mathbf{1}}[(\underline{\mathbf{1}}.id)\circ\uparrow] \to_{Map}$
  $\lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)] \to_{VarCons} \lambda\underline{\mathbf{1}}[\uparrow] = \lambda\underline{\mathbf{2}}$;
- $(\lambda\lambda\underline{\mathbf{1}}[\uparrow])\ \underline{\mathbf{1}} \to_{Beta} (\lambda\underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.id] \to_{Abs} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow)] \to_{Clos}$
  $\lambda\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow))] \to_{ShiftCons} \lambda\underline{\mathbf{1}}[(\underline{\mathbf{1}}.id)\circ\uparrow] \to_{Map}$
  $\lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)] \to_{IdL} \lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].\uparrow] \to_{VarCons} \lambda\underline{\mathbf{1}}[\uparrow] = \lambda\underline{\mathbf{2}}$;
- $(\lambda\lambda\underline{\mathbf{1}}[\uparrow])\ \underline{\mathbf{1}} \to_{Beta} (\lambda\underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.id] \to_{Abs} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow)] \to_{Clos}$
  $\lambda\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow))] \to_{Map} \lambda\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)))] \to_{ShiftCons}$
  $\lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)] \to_{VarCons} \lambda\underline{\mathbf{1}}[\uparrow] = \lambda\underline{\mathbf{2}}$;
- $(\lambda\lambda\underline{\mathbf{1}}[\uparrow])\ \underline{\mathbf{1}} \to_{Beta} (\lambda\underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.id] \to_{Abs} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow)] \to_{Clos}$
  $\lambda\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow))] \to_{Map} \lambda\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)))] \to_{ShiftCons}$
  $\lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)] \to_{IdL} \lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].\uparrow] \to_{VarCons} \lambda\underline{\mathbf{1}}[\uparrow] = \lambda\underline{\mathbf{2}}$;
- $(\lambda\lambda\underline{\mathbf{1}}[\uparrow])\ \underline{\mathbf{1}} \to_{Beta} (\lambda\underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.id] \to_{Abs} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow)] \to_{Map}$
  $\lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow))] \to_{Clos} \lambda\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)))] \to_{ShiftCons}$
  $\lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)] \to_{VarCons} \lambda\underline{\mathbf{1}}[\uparrow] = \lambda\underline{\mathbf{2}}$;

- $(\lambda\lambda\underline{\mathbf{1}}[\uparrow])\ \underline{\mathbf{1}} \to_{Beta} (\lambda\underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.id] \to_{Abs} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow)] \to_{Map} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow))] \to_{Clos} \lambda\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)))] \to_{ShiftCons} \lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow)] \to_{IdL} \lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].\uparrow] \to_{VarCons} \lambda\underline{\mathbf{1}}[\uparrow] = \lambda\underline{\mathbf{2}}$;
- $(\lambda\lambda\underline{\mathbf{1}}[\uparrow])\ \underline{\mathbf{1}} \to_{Beta} (\lambda\underline{\mathbf{1}}[\uparrow])[\underline{\mathbf{1}}.id] \to_{Abs} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.((\underline{\mathbf{1}}.id)\circ\uparrow)] \to_{Map} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].(id\circ\uparrow))] \to_{IdL} \lambda\underline{\mathbf{1}}[\uparrow][\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].\uparrow)] \to_{Clos} \lambda\underline{\mathbf{1}}[\uparrow\circ(\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].\uparrow))] \to_{ShiftCons} \lambda\underline{\mathbf{1}}[\underline{\mathbf{1}}[\uparrow].\uparrow] \to_{VarCons} \lambda\underline{\mathbf{1}}[\uparrow] = \lambda\underline{\mathbf{2}}$.

□

Proposição 3.10 *Os cálculos λ_{Susp} e $\lambda\sigma$ são incomparáveis.*

Demonstração.

Pelos lemas 3.8 e 3.7 temos que existe uma simulação $(\lambda\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}}))\ \underline{\mathbf{1}}^n \to^*_{\lambda\sigma} \lambda(\underline{\mathbf{2}}^n\ \underline{\mathbf{2}}^n)$ mais curta que a mais curta das simulações $(\lambda\lambda(\underline{\mathbf{2}}\ \underline{\mathbf{2}}))\ \underline{\mathbf{1}}^n \to^*_{\lambda_{Susp}} \lambda(\underline{\mathbf{2}}^n\ \underline{\mathbf{2}}^n)$ sempre que $n > 1$. Isto nos permite concluir que $\lambda_{Susp} \not\leftarrow \lambda\sigma$.

Por outro lado, considere a seguinte simulação no λ_{Susp}:

$$
\begin{aligned}
&(\lambda\lambda\underline{\mathbf{2}})\underline{\mathbf{1}} \to_{\beta_s}\\
&[\![(\lambda\underline{\mathbf{2}}), 1, 0, (\underline{\mathbf{1}}, 0) :: nil]\!] \to_{r_7}\\
&\lambda[\![\underline{\mathbf{2}}, 2, 1, @0 :: (\underline{\mathbf{1}}, 0) :: nil]\!] \to_{r_5}\\
&\lambda[\![\underline{\mathbf{1}}, 1, 1, (\underline{\mathbf{1}}, 0) :: nil]\!] \to_{r_4}\\
&\lambda[\![\underline{\mathbf{1}}, 0, 1, nil]\!] \to_{r_2} \lambda\underline{\mathbf{2}}
\end{aligned}
$$

Podemos, agora, utilizar o fato da derivação acima ter 5 passos juntamente com o lema 3.9 para concluirmos que existe uma simulação $(\lambda\lambda\underline{\mathbf{2}})\ \underline{\mathbf{1}} \to^*_{\lambda_{Susp}} (\lambda\underline{\mathbf{2}})$ mais curta que a mais curta das simulações $(\lambda\lambda\underline{\mathbf{2}})\ \underline{\mathbf{1}} \to^*_{\lambda\sigma} (\lambda\underline{\mathbf{2}})$. Ou seja, $\lambda\sigma \not\leftarrow \lambda_{Susp}$. Isto conclui a prova. □

3.3 O Cálculo λs_e é mais adequado do que o λ_{Susp}.

Os lemas e proposições a seguir caminham no sentido de estimar os comprimentos de derivações de termos no λ_{Susp} e λs_e.

Lema 3.11 *Sejam $A \in \Lambda_{dB}$, $i \geq 0$ e $[\![A, i, i, @i-1 :: \ldots :: @0 :: nil]\!]$ um termo bem-formado. Então toda Susp-derivação de $[\![A, i, i, @i-1 :: \ldots :: @0 :: nil]\!]$ para sua forma Susp-normal tem comprimento maior ou igual a $M(A)$.*

Demonstração.

Por indução sobre a estrutura do termo A temos que:

(1) $A = \underline{\mathbf{n}}$. Neste caso, temos que se $n > i$ então $[\![\underline{\mathbf{n}}, i, i, @i - 1 :: \ldots :: @0 :: nil]\!] \to_{r_5}^{i}$ $[\![\underline{\mathbf{n - i}}, 0, i, nil]\!] \to_{r_2} \underline{\mathbf{n}}$. Aqui a derivação tem comprimento $i + 1 \geq M(A)$.

Se $n \leq i$ então $[\![\underline{\mathbf{n}}, i, i, @i - 1 :: \ldots :: @0 :: nil]\!] \to_{r_5}^{n-1}$
$[\![\underline{\mathbf{1}}, i - n + 1, i, @i - n :: \ldots :: @0 :: nil]\!] \to_{r_3} \underline{\mathbf{n}}$. O comprimento da derivação acima é $n \geq 1 = M(A)$, como queríamos.

(2) $A = (B\ C)$. Neste caso, temos $[\![(B\ C), i, i, @i - 1 :: \ldots :: @0 :: nil]\!] \to_{r_6}$ $[\![B, i, i, @i - 1 :: \ldots :: @0 :: nil]\!]\ [\![C, i, i, @i - 1 :: \ldots :: @0 :: nil]\!]$. Agora utilizamos a hipótese de indução e concluimos que o comprimento da derivação acima é maior ou igual a $1 + M(B) + M(C) = M(B\ C) = M(A)$.

(3) $A = (\lambda B)$. Agora temos $[\![(\lambda B), i, i, @i - 1 :: \ldots :: @0 :: nil]\!] \to_{r_7}$ $\lambda[\![B, i + 1, i + 1, @i :: \ldots :: @0 :: nil]\!]$. Agora aplicamos a hipótese de indução e concluímos que a derivação acima tem comprimento maior ou igual a $1 + M(B) = M(\lambda B) = M(A)$.

□

Lema 3.12 *Para $B \in \Lambda_{dB}$ e $i, j \geq 0$, a derivação do termo bem-formado $[\![B, i, j, @j - 1 :: e]\!]$ para sua forma Susp-normal tem comprimento maior ou igual a $M(B)$.*

Demonstração.

De fato, se $B = \underline{\mathbf{n}}$ então $[\![\underline{\mathbf{n}}, i, j, @j - 1 :: e]\!]$ se reduz a sua forma *Susp*-normal em 1 ou mais passos conforme $\underline{\mathbf{n}}$ seja 1 ou maior do que 1, respectivamente. Como $M(B) = 1$ temos o resultado desejado.

Quando $B = (C\ D)$ temos

$$[\![(C\ D), i, j, @j - 1 :: e]\!] \to_{r_6} [\![C, i, j, @j - 1 :: e]\!]\ [\![D, i, j, @j - 1 :: e]\!].$$

Agora aplicando a hipótese de indução temos o resultado desejado.

Por fim, quando $B = (\lambda C)$ temos

$$[\![(\lambda C), i, j, @j - 1 :: e]\!] \to_{r_7} \lambda[\![C, i + 1, j + 1, @j :: e']\!].$$

Aplicando a hipótese de indução temos o resultado desejado. □

Proposição 3.13 *Sejam $A, B \in \Lambda_{dB}$ e $k \geq 0$. Então toda Susp-derivação do termo bem-formado*

$$[\![A, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!]$$

para sua forma Susp-normal tem comprimento maior ou igual a $Q_k(A, B)$.

Demonstração.

Por indução sobre a estrutura de A temos que:

(1) $A = \underline{\mathbf{n}}$. Temos o termo $[\![\underline{\mathbf{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!]$.

Se $n < k$ então $[\![\underline{\mathbf{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_5}^{n-1}$
$[\![\underline{\mathbf{1}}, k-n+1, k-1, @k-n-1 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_3} \underline{\mathbf{n}}$. O comprimento dessa derivação foi $n \geq Q_k(\underline{\mathbf{n}}, B)$ como queríamos.

Se $n = k$ então $[\![\underline{\mathbf{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_5}^{n-1}$
$[\![\underline{\mathbf{1}}, 1, k-1, (B, l) :: nil]\!] \rightarrow_{r_4} [\![B, 0, k-1-l, nil]\!]$. Pelo lema 3.12 este último termo se reduz a sua forma *Susp*-normal em $M(B)$ ou mais passos. Logo a derivação completa tem comprimento maior ou igual a $n+M(B) = Q_k(\underline{\mathbf{n}}, B) = Q_k(A, B)$.

Se $n > k$ então $[\![\underline{\mathbf{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_5}^{k}$
$[\![\underline{\mathbf{n-k}}, 0, k-1, nil]\!] \rightarrow_{r_2} \underline{\mathbf{n-1}}$. O comprimento dessa derivação foi $k+1 \geq Q_k(\underline{\mathbf{n}}, B) = Q_k(A, B)$ como queríamos.

(2) $A = (C\ D)$. Temos $[\![(C\ D), k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_6}$

$$[\![C, k, k-1, @k-2 :: \ldots :: @0 :: (B, 0) :: nil]\!]$$
$$[\![D, k, k-1, @k-2 :: \ldots :: @0 :: (B, 0) :: nil]\!].$$

Aplicando a hipótese de indução temos que a derivação completa tem comprimento maior ou igual a $1 + Q_k(C, B) + Q_k(D, B) = Q_k(C\ D, B) = Q_k(A, B)$.

(3) $A = (\lambda C)$. Temos $[\![(\lambda C), k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_7}$
$\lambda[\![C, k+1, k, @k-1 :: \ldots :: @0 :: (B, l) :: nil]\!]$. Agora aplicamos a hipótese de indução e concluimos que esta derivação tem comprimento maior ou igual a $1 + Q_{k+1}(C, B) = Q_k(\lambda C, B) = Q_k(A, B)$.

□

Proposição 3.14 *Quaisquer que sejam $A, B \in \Lambda_{dB}$ e $k \geq 0$, toda s_e-derivação de $A\sigma^{k+1}B$ para sua forma s_e-normal tem comprimento menor ou igual a $Q_{k+1}(A, B)$.*

Demonstração.

A demonstração é feita por indução sobre a estrutura do termo A:

- $A = \underline{\mathbf{n}}$. Temos $\underline{\mathbf{n}}\sigma^{k+1}B \rightarrow \begin{cases} \underline{\mathbf{n-1}} & \text{se } n > k+1 \\ \varphi_0^{k+1}B & \text{se } n = k+1 \\ \underline{\mathbf{n}} & \text{se } n < k+1 \end{cases}$

  Nos casos $n > k+1$ e $n < k+1$ a derivação tem comprimento 1 e, claramente, $1 \leq Q_{k+1}(\underline{\mathbf{n}}, B), \forall n$, pois $Q_{k+1}(\underline{\mathbf{n}}, B) \geq 1, \forall n$. Portanto a derivação completa tem comprimento menor ou igual a $Q_{k+1}(\underline{\mathbf{n}}, B) = Q_{k+1}(A, B)$. Quando $n = k+1$ utilizamos o lema 3.5 para concluirmos que $\varphi_0^{k+1}B$ se reduz a sua forma s_e-normal em $M(B)$ passos. Assim, a derivação completa tem comprimento $1 + M(B)$. Observando a definição da função Q_{k+1} concluimos que esse comprimento é sempre menor ou igual a $Q_{k+1}(\underline{\mathbf{n}}, B) = Q_{k+1}(A, B)$.

- $A = (C\ D)$. Temos $(C\ D)\sigma^{k+1}B \rightarrow\ C\sigma^{k+1}B\ D\sigma^{k+1}B$. Aplicando a hipótese de indução concluimos que esta derivação tem comprimento menor ou igual a $1 + Q_{k+1}(C, B) + Q_{k+1}(D, B) = Q_{k+1}(C\ D, B) = Q_{k+1}(A, B)$.

- $A = (\lambda C)$. Temos $(\lambda C)\sigma^{k+1}B \rightarrow\ \lambda C\sigma^{k+2}B$. Agora aplicamos a hipótese de indução e concluimos que essa derivação tem comprimento menor ou igual a $1 + Q_{k+2}(C, B) = Q_{k+1}(\lambda C, B) = Q_{k+1}(A, B)$.

□

Teorema 3.15 *O cálculo λs_e é mais adequado do que o cálculo λ_{Susp}.*

Demonstração.

Precisamos mostrar que para qualquer β-conversão $A \rightarrow_\beta B$, com $A, B \in \Lambda_{dB}$, toda λ_{Susp}-simulação $A \rightarrow_{\beta_s} B' \rightarrow_{Susp}^{m} Susp\text{-}norm(B') = B$, existe $n \leq m$ tal que $A \rightarrow_{\sigma-gen} C \rightarrow_{s}^{n} s\text{-}norm(C) = B$.

No λ_{Susp} para qualquer redex onde possamos aplicar a regra β_s temos $(\lambda D)\ E \to_{\beta_s} [\![D, 1, 0, (E, 0) :: nil]\!] \to^m_{Susp} Susp\text{-}norm([\![D, 1, 0, (E, 0) :: nil]\!])$, onde pela proposição 3.13, temos que

$$m \geq Q_1(D, E) \tag{3.1}$$

Por outro lado, no λs_e temos $(\lambda D)\ E \to_{\sigma-gen} D\sigma^1 E \to^n_s s\text{-}norm(D\sigma^1 E)$, onde pela proposição 3.14 temos que toda s_e-derivação de $D\sigma^1 E$ para a sua forma s_e-normal satisfaz

$$n \leq Q_1(D, E) \tag{3.2}$$

De 3.1 e 3.2 concluimos que para qualquer λ_{Susp}-simulação $A \to_{\beta_s} B' \to^m_{Susp} Susp\text{-}norm(B') = B$, toda λs_e-simulação $A \to_{\sigma-gen} C \to^n_s s_e\text{-}norm(C) = B$ tem comprimento $n \leq m$.

Para que a segunda condição de adequabilidade seja satisfeita precisamos apresentar uma β-conversão do λ-cálculo cuja λs_e-simulação seja estritamente menor do que sua respectiva λ_{Susp}-simulação. Considere, portanto, a seguinte β-conversão: $(\lambda \underline{\mathbf{2}})\ \underline{\mathbf{1}} \to_\beta \underline{\mathbf{1}}$.

No λ_{Susp} esta β-conversão pode ser simulada de maneira única, em 3 passos, como podemos ver abaixo:
$(\lambda \underline{\mathbf{2}})\ \underline{\mathbf{1}} \to_{\beta_s} [\![\underline{\mathbf{2}}, 1, 0, (\underline{\mathbf{1}}, 0) :: nil]\!] \to_{r_5} [\![\underline{\mathbf{1}}, 0, 0, nil]\!] \to_{r_2} \underline{\mathbf{1}}$.

Já no λs_e a mesma simulação se dá de forma única, em apenas 2 passos:
$(\lambda \underline{\mathbf{2}})\ \underline{\mathbf{1}} \to_{\sigma-gen} \underline{\mathbf{2}}\sigma^1 \underline{\mathbf{1}} \to_{\sigma-destr} \underline{\mathbf{1}}$. Isto finaliza a demonstração. □

Note que aqui demonstramos muito mais do que a noção de adequabilidade de [KR00]. De fato, provamos que o comprimento de qualquer λs_e-simulação é sempre menor ou igual que qualquer λ_{Susp}-simulação de um passo de β-conversão.

Vejamos um exemplo que nos mostra claramente como as simulações de um passo de β-conversão no λs_e são mais curtas que no λ_{Susp}. O exemplo que escolhemos constitui-se de um β-redex simples onde só é possível aplicar a regra que inicia a simulação de uma β-conversão, este redex é dado por $((\lambda(\lambda^n\ \underline{\mathbf{i}}))\ \underline{\mathbf{j}})$, onde $n \geq 0$ e $i, j \geq 1$. Veja que este é um exemplo adequado porque termos que contenham aplicações adicionais vão apenas duplicar o trabalho a ser feito e, termos com constantes, ou meta-variáveis, não nos interessam agora. Termos que possuem diversos abstratores são importantes porque ambos os cálculos trabalham da mesma forma ao entrar dentro de abstratores. A diferença se dá quando atingimos índices de De Bruijn. Vejamos, então, a simulação de um passo de β-conversão para o redex

$((\lambda(\lambda^n\ \underline{\mathbf{i}}))\ \underline{\mathbf{j}})$ no λ_{Susp}. No caso em que $i < n+1$, temos

$$
\begin{array}{l}
((\lambda(\lambda^n\ \underline{\mathbf{i}}))\ \underline{\mathbf{j}}) \to_{\beta_s} \\
[\![(\lambda^n\underline{\mathbf{i}}), 1, 0, (\underline{\mathbf{j}}, 0) :: nil]\!] \to_{r_7}^{n} \\
\lambda^n[\![\underline{\mathbf{i}}, n+1, n, @n-1 :: \ldots :: @0 :: (\underline{\mathbf{j}}, 0) :: nil]\!] \to_{r_5}^{i-1} \\
\lambda^n[\![\underline{\mathbf{1}}, n-i+2, n, @n-i :: \ldots :: @0 :: (\underline{\mathbf{j}}, 0) :: nil]\!] \to_{r_3} \lambda^n\underline{\mathbf{i}}.
\end{array}
$$

Esta redução tem um total de $n+i+1$ passos.

Quando $i = n+1$, temos

$$
\begin{array}{l}
((\lambda(\lambda^n\ \underline{\mathbf{i}}))\ \underline{\mathbf{j}}) \to_{\beta_s} [\![(\lambda^n\underline{\mathbf{i}}), 1, 0, (\underline{\mathbf{j}}, 0) :: nil]\!] \to_{r_7}^{n} \\
\lambda^n[\![\underline{\mathbf{i}}, n+1, n, @n-1 :: \ldots :: @0 :: (\underline{\mathbf{j}}, 0) :: nil]\!] \to_{r_5}^{i-1} \\
\lambda^n[\![\underline{\mathbf{1}}, 1, n, (\underline{\mathbf{j}}, 0) :: nil]\!] \to_{r_4} \lambda^n[\![\underline{\mathbf{j}}, 0, n, nil]\!] \to_{r_2} \lambda^n\underline{\mathbf{j}+\mathbf{n}}.
\end{array}
$$

Aqui a redução total tem comprimento $n+i+2$.

Por fim quando, $i > n+1$ temos

$$
\begin{array}{l}
((\lambda(\lambda^n\ \underline{\mathbf{i}}))\ \underline{\mathbf{j}}) \to_{\beta_s} [\![(\lambda^n\underline{\mathbf{i}}), 1, 0, (\underline{\mathbf{j}}, 0) :: nil]\!] \to_{r_7}^{n} \\
\lambda^n[\![\underline{\mathbf{i}}, n+1, n, @n-1 :: \ldots :: @0 :: (\underline{\mathbf{j}}, 0) :: nil]\!] \to_{r_5}^{n+1} \\
\lambda^n[\![\underline{\mathbf{i}-\mathbf{n}-\mathbf{1}}, 0, n, nil]\!] \to_{r_2} \lambda^n\underline{\mathbf{i}-\mathbf{1}}.
\end{array}
$$

Aqui a redução tem comprimento $2n+3$.

A mesma simulação no λs_e pode ser vista da seguinte forma:

$$((\lambda(\lambda^n\ \underline{\mathbf{i}}))\ \underline{\mathbf{j}}) \to_{\sigma\text{-}gen} (\lambda^n\underline{\mathbf{i}})\sigma^1\underline{\mathbf{j}} \to_{\sigma\text{-}\lambda}^{n} \lambda^n\underline{\mathbf{i}}\sigma^{n+1}\underline{\mathbf{j}}.$$

Agora se $i < n+1$ ou $i > n+1$, com uma aplicação de σ-*destruction* o termo anterior reduz-se, respectivamente, para $\lambda^n\underline{\mathbf{i}}$ e $\lambda^n\underline{\mathbf{i}-\mathbf{1}}$. Note que ambas reduções têm comprimento $n+2$.

No caso em que $i = n+1$, após uma aplicação de σ-*destruction* o termo $\lambda^n\underline{\mathbf{i}}\sigma^{n+1}\underline{\mathbf{j}}$ se reduz para $\lambda^n\varphi_0^{n+1}\underline{\mathbf{j}}$, que após uma aplicação de φ-*destruction* se reduz para $\lambda^n\underline{\mathbf{j}+\mathbf{n}}$. Isto nos dá um redução cujo comprimento total é igual a $n+3$.

Os comprimentos das derivações em cada caso podem ser melhor visualizados pela tabela abaixo:

	λs_e	λ_{Susp}
$i < n+1$	$n+2$	$n+i+1$
$i = n+1$	$n+3$	$n+i+2$
$i > n+1$	$n+2$	$2n+3$

Como $i \geq 1$ é fácil ver que qualquer simulação de um passo de β-conversão do redex dado acima tem sempre comprimento menor no λs_e, exceto quando $i = 1$, pois neste caso os comprimentos das reduções são iguais. Em todos os outros casos é evidente a vantagem do λs_e sobre o λ_{Susp} já que o primeiro consegue reduzir índices de De Bruijn de qualquer valor em um único passo (via regra σ-*destruction*), enquanto que o λ_{Susp} precisa reduzí-los um a um por meio da regra r_5 até que o contexto fique vazio e seja possível aplicar r_2, ou até que os índices de De Bruijn sejam decrementados até 1 e seja possível aplicar a regra r_3.

Capítulo 4

Uma Implementação para os Cálculos $\lambda\sigma$, λs_e e λ_{Susp} com a Eta-conversão

Uma outra contribuição deste trabalho é a implementação de um ambiente que simula β-conversões e η-conversões de λ-termos via cálculo $\lambda\sigma$, λs_e ou λ_{Susp}. A importância de uma tal implementação está, além do tempo e confiabilidade que se ganham em uma simulação de um passo de β-conversão, no fato de que muito se pode aprender sobre a estrutura interna dos cálculos utilizados. Adicionalmente, a construção dessa implementação foi um bom exercício sobre tipagem implícita, já que a linguagem utilizada para a mesma foi o Ocaml, da família ML, que utiliza este tipo de tipagem. A escolha dessa linguagem se deu basicamente pela facilidade de se trabalhar com λ-termos e pela simplicidade no manuseio de tipos para termos, substituições, contextos e listas de contextos dos cálculos tratados. Uma versão pré-compilada e o código dessa implementação podem ser obtidos em `http://www.mat.unb.br/~ayala/TCgroup/`.

Nossa implementação consiste basicamente de três ambientes, distintos e independentes entre si, onde foram implementadas as regras dos três cálculos citados acima. Assim, dado um λ-termo t qualquer podemos simular a redução de t a sua forma ξ-normal, onde $\xi \in \{\sigma, s_e, Susp\}$, passo a passo em qualquer um desses cálculos. A regra Eta também foi implementada nos três casos.

Observe que esta implementação foi construída com o objetivo de simular alguns exemplos simples de β-conversão e η-conversão. Portanto, como o mesmo não faz parte de um sistema maior, não nos preocupamos muito em construir um código que fosse o mais eficiente possível. Nossa preocupação fundamental foi em relação

à corretude do mesmo. Nesse sentido sabemos que algumas modificações podem ser feitas para melhorar a eficiência do programa.

Para o $\lambda\sigma$ considere, por exemplo, a regra *Abs*, que é dada por

$$(\lambda M)[S] \rightarrow \lambda M[\underline{\mathbf{1}}.(S \circ \uparrow)]$$

Devemos observar que o cálculo $\lambda\sigma$ trabalha com dois tipos de objetos, a saber, termos e substituições. Portanto nossa implementação deve diferenciar esses tipos e trabalhar de forma correta com cada um deles. Como sabemos, os termos do $\lambda\sigma$ são da forma $\underline{\mathbf{1}}$, λM, $(M\ N)$ ou $M[S]$ que representamos, na linguagem Ocaml, respectivamente por `One`, `L(M)`, `A(M,N)` ou `Sb(M,S)`. Já as substituições podem ser da forma id, $\uparrow$, $M.S$ ou $S \circ T$, representadas, respectivamente por `Id`, `Up`, `Pt(M,S)` ou `Cp(S,T)`. A implementação das regras de uma forma geral é constituída de dois passos. Primeiro fazemos uma busca sobre o termo dado a procura de possíveis rédices onde a regra possa ser aplicada. O segundo passo constitui da aplicação da regra propriamente dita ao redex encontrado e selecionado. No caso em que existam mais de um redex na mesma expressão, o usuário deve escolher em qual posição a redução deve ser feita. A busca por rédices para a regra *Abs* é feita pela função `matchingAbs` de tipo lista de posições, onde `exp` é a expressão, `l` acumula as posições dos rédices e `pos` é a posição atual.

```
let rec matchingAbs exp l pos =
match exp with
       Dummy       -> l |
       One         -> l |
       Vr c        -> l |
       A(e1,e2)    -> append (matchingAbs e1 l (append pos [1]))
                             (matchingAbs e2 [] (append pos [2])) |
       L(e1)       -> matchingAbs e1 l (append pos [1]) |
       Sb(L(e1),sb) -> pos :: append
                           (matchingAbs e1 l (append pos [1;1]))
                           (matchingAbsSb sb [] (append pos [2])) |
       Sb(e1,sb)   -> append (matchingAbs e1 l (append pos [1]))
                             (matchingAbsSb sb [] (append pos [2]))
and
matchingAbsSb subs l pos =
match subs with
       Up          -> l |
       Id          -> l |
       Pt(e1,sb) -> append (matchingAbs e1 l (append pos [1]))
                           (matchingAbsSb sb [] (append pos [2])) |
       Cp(s1,s2) -> append (matchingAbsSb s1 l (append pos [1]))
                           (matchingAbsSb s2 [] (append pos [2]));;
```

Note que a busca é dividida em duas etapas: a busca sobre expressões do tipo termo e a busca sobre expressões do tipo substituição.

A aplicação da regra *Abs* à expressão `exp` na posição `pr` é dada pela função `absreduction` abaixo:

```
let rec absreduction exp pr =
match pr with
       []   -> (match exp with
       Sb(L(e1),sb) -> L(Sb(e1,Pt(One,Cp(sb,Up)))) | _ -> exp)  |
       1 :: tail -> (match exp with
      Dummy      -> exp |
                    One        -> exp |
                    Vr c       -> exp |
                    A(e1,e2)   -> A((absreduction e1 tail),e2) |
                    L(e1)      -> L(absreduction e1 tail) |
                    Sb(e1,s2)  -> Sb((absreduction e1 tail),s2)) |
       2 :: tail -> (match exp with
      Dummy      -> exp |
                    One        -> exp |
                    Vr c       -> exp |
                    L(e1)      -> exp |
                    A(e1,e2)   -> A(e1,(absreduction e2 tail)) |
                    Sb(e1,s2)-> Sb(e1,(absreductionSb s2 tail))) |
       _ -> exp
and
absreductionSb subs pr =
match pr with
       []        -> subs |
       1 :: tail -> (match subs with
                    Id         -> subs |
                    Up         -> subs |
                    Cp(s1,s2) -> Cp((absreductionSb s1 tail),s2) |
                    Pt(e1,s2) -> Pt((absreduction e1 tail),s2)) |
       2 :: tail -> (match subs with
                    Id         -> subs |
                    Up         -> subs |
                    Cp(s1,s2) -> Cp(s1,(absreductionSb s2 tail)) |
                    Pt(e1,s2)-> Pt(e1,(absreductionSb s2 tail))) |
       _ -> subs ;;
```

Aqui, da mesma forma, o trabalho é dividido em duas partes, uma para redução sobre termos e outra sobre substituições. Isto ilustra a facilidade de se lidar com tipagem implícita no Ocaml. As outras regras são implementadas de forma análoga.

No λs_e a construção é feita de forma semelhante com a observação de que agora não temos mais dois tipos de expressões, mas apenas um. De fato, as expressões do λs_e são da forma $\underline{\mathbf{n}}$, $(M\ N)$, λM, $M\sigma^i N$ ou $\varphi_k^i M$ que representamos, em Ocaml, respectivamente, por `DB n`, `A(M,N)`, `L(M)`, `S(i,M,N)` ou `P(k,i,M)`. A estrutura da função para a busca de rédices para a regra σ-λ-*transition* é:

```
let rec matchingSLtransition exp l pos =
match exp with
       Dummy         -> l|
       DB i          -> l |
```

```
       Vr c          -> l |
       A(e1,e2)      -> append
                        (matchingSLtransition e1 l (append pos [1]))
                        (matchingSLtransition e2 [] (append pos [2])) |
       L(e1)         -> (matchingSLtransition e1 l (append pos [1])) |
       S(i,L(e1),e2) -> pos :: append
                        (matchingSLtransition e1 l (append pos [1;1]))
                        (matchingSLtransition e2 [] (append pos [2])) |
       S(i,e1,e2)    -> append
                        (matchingSLtransition e1 l (append pos [1]))
                        (matchingSLtransition e2 [] (append pos [2])) |
       P(j,k,e1)     -> (matchingSLtransition e1 l (append pos [1]));;
```

Já a função de aplicação da regra σ-λ-*transition* é:

```
let rec sltransition exp pr =
match pr with
       [] -> (match exp with
              S(i,L(e1),e2) -> L(S(i+1,e1,e2)) | _ -> exp) |
       1 :: tail -> (match exp with
                      A(e1,e2)  -> A((sltransition e1 tail),e2) |
                      L(e1)     -> L(sltransition e1 tail) |
                      S(i,e1,e2)-> S(i,(sltransition e1 tail),e2) |
                      P(j,k,e1) -> P(j,k,(sltransition e1 tail)) |
                      _ -> exp )  |
       2 :: tail -> (match exp with
                      A(e1,e2)  -> A(e1,(sltransition e2 tail)) |
                      S(i,e1,e2)-> S(i,e1,(sltransition e2 tail)) |
      _ -> exp ) |
       _ -> exp;;
```

O cálculo λ_{Susp} já possui uma estrutura mais complicada que a dos cálculos anteriores. De fato, as expressões no λ_{Susp} podem ser de três tipos distintos: termos suspensos, contextos ou termos de contextos. Os termos suspensos, por sua vez, podem ser C, $\underline{\mathbf{n}}$, $(M\ N)$, λM ou $[\![t, i, j, e]\!]$ que representamos, em Ocaml, respectivamente por `Vr c`, `DB n`, `A(M,N)`, `L(M)` ou `Sp(t,i,j,e)`. Os contextos podem ser nil, $et :: e$ ou $\{\!\{env1, i, j, env2\}\!\}$, que representamos, respectivamente, por `Nilen`, `Con(et,e)` ou `Ck(env1,i,j,env2)`. Os termos de contexto podem ser $@n$, (t, l) ou $\langle\!\langle envt, i, j, env\rangle\!\rangle$ que representamos, respectivamente, por `Ar(n)`, `Paar(t,l)` ou `LG(envt,i,j,env)`.

A função de busca de rédices para a regra (r_7) é dada por:

```
let rec matching_r7 exp l pos =
match exp with
       Dummy       -> l |
       DB i        -> l |
       Vr c        -> l |
       A(e1,e2)    -> append (matching_r7 e1 l (append pos [1]))
                             (matching_r7 e2 [] (append pos [2])) |
```

```
       L(e1)       -> (matching_r7 e1 l (append pos [1])) |
       Sp(L(e1),_,_,env)   -> pos :: append
                      (matching_r7 e1 l (append pos [1;1]))
                      (matchingEnv_r7 env [] (append pos [2])) |
       Sp(e1,_,_,env) -> append (matching_r7 e1 l (append pos [1]))
                          (matchingEnv_r7 env [] (append pos [2]))
and matchingEnv_r7 env l pos =
     match env with
       Nilen             -> l |
       Con(envt, env1)   -> append
                             (matchingEt_r7 envt l (append pos [1]))
                         (matchingEnv_r7 env1 [] (append pos [2])) |
       Ck(env1,_,_,env2) -> append
                            (matchingEnv_r7 env1 l (append pos [1]))
                           (matchingEnv_r7 env2 [] (append pos [2]))
and matchingEt_r7 envt l pos =
     match envt with
        Ar i ->  l |
        LG(envt1,_,_,env1) -> append
                            (matchingEt_r7 envt1 l (append pos [1]))
                          (matchingEnv_r7 env1 [] (append pos [2]))|
        Paar(e1,i)        -> (matching_r7 e1 l (append pos [1]));;
```

E a função para redução da expressão `exp` na posição `pr` da regra (r_7) é dada por:

```
let rec r7_reduction exp pr =
match pr with
       [] -> (match exp with
             Sp(L(e1),i,j,env) -> L(Sp(e1,i+1,j+1,Con(Ar(j),env)))|
                                              _ -> exp ) |
       1 :: tail -> (match exp with
     A(e1,e2)    -> A((r7_reduction e1 tail),e2) |
             L(e1)       -> L(r7_reduction e1 tail) |
             Sp(e1,i,j,env) -> Sp((r7_reduction e1 tail),i,j,env) |
                       _             -> exp) |
       2 :: tail -> (match exp with
                   A(e1,e2)       -> A(e1,(r7_reduction e2 tail)) |
          Sp(e1,i,j,env) -> Sp(e1,i,j,(r7_reductionEnv env tail)) |
          _  -> exp)
and
r7_reductionEnv env pr =
match pr with
       1 :: tail -> (match env with
                     Con(envt,env1)    ->
                     Con((r7_reductionEt envt tail),env1) |
                     Ck(env1,i,j,env2) ->
                     Ck((r7_reductionEnv env1 tail),i,j,env2) |
                     _                 -> env) |
       2 :: tail -> (match env with
                     Con(envt,env1)    ->
                     Con(envt,(r7_reductionEnv env1 tail)) |
                     Ck(env1,i,j,env2) ->
                     Ck(env1,i,j,(r7_reductionEnv env2 tail)) |
                     _             -> env)
and
r7_reductionEt envt pr =
match pr with
```

```
1 :: tail -> (match envt with
               Paar(e1,i)   ->
               Paar((r7_reduction e1 tail),i) |
               LG(envt1,i,j,env1) ->
               LG((r7_reductionEt envt1 tail),i,j,env1) |
               _  -> envt) |
2 :: tail -> (match envt with
               LG(envt1,i,j,env1) ->
               LG(envt1,i,j,(r7_reductionEnv env1 tail)) |
               _              -> envt);;
```

4.1 A implementação da regra Eta

Um aspecto interessante que vale a pena ser ressaltado é a maneira como a regra Eta foi implementada para os três cálculos. Essa etapa contribuiu muito para uma melhor compreensão da estrutura desses cálculos, pois que a implementação desta regra não é trivial. De fato, observe que dado um termo qualquer $t \in \Lambda_{dB}$ é sempre fácil decidir qual regra do sistema de reescrita $(\lambda\sigma, \lambda s_e, \lambda_{Susp})$ que devemos utilizar, pois essas regras são incondicionais ou possuem apenas uma condição aritmética simples que pode ser resolvida por algum algoritmo de dedução aritmética usualmente pré-construído nas linguagens de programação modernas. O mesmo, no entanto, não acontece com a regra Eta pois a condição de aplicação da mesma é mais complexa. Para contornar esse problema apresentamos um teste para implementação da regra Eta desenvolvido a partir das idéias de [Bor95] para o $\lambda\sigma$ e [ARK01a] para o λs_e. Essa idéia foi comentada superficialmente ao considerarmos a prova do lema 2.21.

Para entendermos melhor o que será feito, observe que, a η-conversão $\lambda(M\ \underline{1}) \to_\eta N$ deve nos fornecer um termo N, obtido de M decrementando todas as suas variáveis livres em 1. Mas isto, segundo a idéia de [Bor95], consiste exatamente em normalizar o termo $((\lambda M)\ \diamond)$, no caso em que $\diamond$ não ocorra na forma normal de $((\lambda M)\ \diamond)$, onde $\diamond$ é um símbolo de constante que não pertença à linguagem do cálculo considerado. Isto pode ser resumido de forma mais objetiva através da seguinte regra: Seja $gen_{\lambda_\xi}(M, pos)$ o resultado de se aplicar a regra que inicia a simulação de um passo de β-conversão ao termo M na posição pos. Então

$$\lambda(M\ \underline{1}) \to_{Eta} N \ \text{ se } N = norm(gen_{\lambda_\xi}(((\lambda M)\ \diamond), raiz))$$

e $\diamond$ não ocorre em N, para $\xi \in \{\sigma, s_e, Susp\}$.

Essa idéia já foi mostrada correta para o $\lambda\sigma$ [Bor95] e para o λs_e [ARK01a].

Vejamos agora que esta implementação também é correta para o λ_{Susp}.

Proposição 4.1 *A implementação descrita acima pela regra* (Eta_{Susp}) *é "correta" no sentido expresso na proposição 2.9.*

Antes de demonstrarmos essa proposição consideremos o seguinte:

Lema 4.2 *Sejam* A *e* B *termos bem-formados e* $k \geq 0$*. Então a rm-normalização do termo bem-formado* $[\![A, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!]$ *nos fornece um novo termo que decrementa em 1 todas as variáveis livres de* A *maiores do que* k*, substitui a* k*-ésima variável de* A *pelo termo* B *atualizado de forma correta de acordo com sua estrutura e, deixa inalteradas todas as variáveis de* A *menores do que* k*.*

Demonstração.

Por indução sobre a estrutura do termo A, temos:

1. Se $A = \underline{\mathbf{n}}$ então,

   (a) se $n > k$ então $[\![\underline{\mathbf{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_5}^{k}$ $[\![\underline{\mathbf{n-k}}, 0, k-1, nil]\!] \rightarrow_{r_2} \underline{\mathbf{n-1}}$;

   (b) se $n = k$ então $[\![\underline{\mathbf{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_5}^{k-1}$ $[\![\underline{\mathbf{1}}, 1, k-1, (B, l) :: nil]\!] \rightarrow_{r_4} [\![B, 0, k-l-1, nil]\!]$;

   (c) se $n < k$ então $[\![\underline{\mathbf{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_5}^{n-1}$ $[\![\underline{\mathbf{1}}, k-n+1, k-1, @k-n-1 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_3} \underline{\mathbf{n}}$;

2. Se $A = (C\ D)$ então aplicamos a hipótese de indução para os termos C e D;

3. Se $A = (\lambda C)$, como o termo C está limitado por um abstrator adicional, então apenas as variáveis livres de C maiores do que $k+1$ é que devem ser decrementadas em 1, a $k+1$-ésima variável deve ser substituida pelo termo B atualizado de forma correta de acordo com sua estrutura e, as outras variáveis devem permanecer inalteradas.

   Como $[\![(\lambda C), k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_7}$
   $\lambda[\![C, k+1, k, @k-1 :: \ldots :: @0 :: (B, l) :: nil]\!]$, aplicamos a hipótese de indução para o termo $[\![C, k+1, k, @k-1 :: \ldots :: @0 :: (B, l) :: nil]\!]$ e obtemos o resultado desejado.

4. Se $A = [\![t, ol, nl, e]\!]$ então primeiro determinamos a forma *rm*-normal do termo A. Pelo lema 2.7 temos que *rm-norm*(A) é um termo puro na notação de De Bruijn e, portanto a análise desse caso recai em um dos casos anteriores.

□

Demonstração da proposição 4.1. Inicialmente observe que o termo $[\![M, 1, 0, (\diamond, 0) :: nil]\!]$ é obtido a partir da aplicação da regra β_s na raiz do termo $((\lambda M)\ \diamond)$. Após a propagação do símbolo $\diamond$, todas as variáveis livres de M são decrementadas em 1 exceto aquelas que correspondam ao abstrator externo que deverão ser substituídas pelo símbolo $\diamond$. A demonstração desse fato é feita por indução sobre a estrutura do termo M:

- Se $M = \underline{\mathbf{n}}$ então,

  (a) se $n = 1$ temos $[\![\underline{\mathbf{n}}, 1, 0, (\diamond, 0) :: nil]\!] \rightarrow_{r_4} [\![\diamond, 0, 0, nil]\!] \rightarrow_{r_1} \diamond$, como era esperado pois $\diamond$ pode ser considerado como um símbolo de constante;

  (b) se $n > 1$ temos $[\![\underline{\mathbf{n}}, 1, 0, (\diamond, 0) :: nil]\!] \rightarrow_{r_5} [\![\underline{\mathbf{n-1}}, 0, 0, nil]\!] \rightarrow_{r_2} \underline{\mathbf{n-1}}$.

- Se $M = (A\ B)$ então aplicamos a hipótese de indução aos termos A e B e o resultado é imediato já que as ocorrências de variáveis no termo A não têm interferência junto as variáveis do termo B.

- Se $M = (\lambda A)$ então $[\![(\lambda A), 1, 0, (\diamond, 0) :: nil]\!] \rightarrow_{r_7} \lambda[\![A, 2, 1, @0 :: (\diamond, 0) :: nil]\!]$. Pelo lema 4.2 todas as variáveis livres de A maiores que 2 são decrementadas em 1, a segunda variável livre, que corresponde ao abstrator que foi eliminado na β_s-redução, é substituida pelo símbolo $\diamond$ enquanto que todas as outras variáveis permanecem inalteradas.

- Se $M = [\![t, ol, nl, e]\!]$ então primeiro *rm*-normalizamos o termo M. Como, pelo lema 2.7, o termo *rm-norm*(M) é um termo puro na notação de De Bruijn, temos que a análise desse caso recai em um dos casos anteriores.

Além disso, quando o termo *rm-norm*$([\![M, 1, 0, (\diamond, 0) :: nil]\!])$ não possui o símbolo $\diamond$, ao incrementarmos todas as suas variáveis livres, obteremos um termo *rm*-equivalente ao termo M. Isto corresponde a condição original para aplicação da regra η. A demonstração dessa *rm*-equivalência é feita por indução sobre a estrutura do termo M de maneira equivalente ao caso anterior. Assim, temos que ao

incrementarmos todas as variáveis livres de $rm\text{-}norm([\![M, 1, 0, (\diamond, 0) :: nil]\!])$, obteremos um termo rm-equivalente ao termo M. □

Vejamos agora como foram construídas as funções, em Ocaml, para implementação das regras Eta_ξ para $\xi \in \{\sigma, s_e, Susp\}$. Seja M um termo qualquer pertencente à linguagem do cálculo λ_ξ. Nossa proposta consiste em uma adaptação da idéia anteriormente apresentada onde a normalização do termo $((\lambda M)\ \diamond)$ deve ser feita de forma apropriada, isto é, para $\lambda(M\ \underline{\mathbf{1}}) \rightarrow_\eta N$, queremos uma normalização que nos forneça exatamente o termo N acima. Se simplesmente ξ-normalizarmos $((\lambda M)\ \diamond)$, obteremos provavelmente um termo mais simples do que N porque foram aplicadas regras adicionais que não eram necessárias. Nossa normalização deve apenas propagar o símbolo $\diamond$, e deixar inalterados subtermos que não contenham este símbolo. Como esta é uma normalização especial, a denominaremos a partir de agora de ξ-pseudo-normalização.

A implementação de nossas regras Eta_ξ, como todas as demais, se subdivide em duas etapas. A primeira busca um redex onde a regra se aplique. A segunda etapa faz a ξ-pseudo-normalização propriamente dita. Para que um subtermo da forma $\lambda(M\ \underline{\mathbf{1}})$ seja um redex para a regra Eta_ξ é necessário e suficiente que o termo M não contenha a variável livre $\underline{\mathbf{1}}$. Isto pode ser feito da seguinte forma: no caso em que essa forma ξ-pseudo-normal possua o símbolo $\diamond$, que caracteriza a ocorrência da variável livre $\underline{\mathbf{1}}$ no termo t_1, a regra Eta_ξ não se aplica. Caso contrário a mesma se aplica e o termo obtido após a aplicação dessa regra consiste exatamente da forma ξ-pseudo-normal de $((\lambda t_1)\ \diamond)$ como descrito acima.

Vejamos como isto ocorre em cada um dos cálculos que estamos considerando.

O primeiro ambiente de nossa implementação é o cálculo $\lambda\sigma$. Neste caso, $((\lambda t_1)\ \diamond) \rightarrow_{Beta} t_1[\diamond.id]$. Agora precisamos σ-pseudo-normalizar o termo $t_1[\diamond.id]$. Essa σ-pseudo-normalização, que denotaremos por `sig-norm` foi implementada da forma mostrada a seguir. O nome das regras foi colocado entre os símbolos ($*$ e $*$) para facilitar a leitura. As funções `occurdummy*` que aparecem nos códigos abaixo buscam a ocorrência de símbolos $\diamond$ na expressão dada como argumento e têm custo linear em função do tamanho da entrada.

```
let rec sig-norm exp =
match exp with
            Dummy  -> Dummy | One   -> One | Vr c   -> Vr c |
(* App *)   Sb(A(e1,e2),sb)-> (if (occurdummy1(e1)||occurdummy1(e2)||
            occurdummy1sb(sb)) then
            A(sig-norm(Sb(e1,sb)),sig-norm(Sb(e2,sb))) else exp) |
```

```
(* Abs *)   Sb(L(e1),sb)-> (if (occurdummy1(e1)||occurdummy1sb(sb))
            then L(sig-norm(Sb(e1,Pt(One,Cp(sb,Up))))) else exp) |
(* Clos *)  Sb(Sb(e1,s1),s2)-> sig-norm(Sb(e1,sig-normsb(Cp(s1,s2)))) |
(*VarCons*) Sb(One,Pt(e1,sb))-> (if (occurdummy1(e1)||occurdummy1sb(sb))
            then sig-norm(e1) else exp) |
(* Id *)    Sb(e1,Id) -> (if occurdummy1(e1)  then sig-norm(e1) else exp) |
            _ -> exp
and
sig-normsb subs =
match subs with
                Up -> Up | Id -> Id |
(*ShiftCons*)   Cp(Up,Pt(e1,sb)) ->
                (if (occurdummy1(e1) || occurdummy1sb(sb))
                then sig-normsb(sb) else subs) |
(* IdL *)       Cp(Id,sb) -> sig-normsb(sb) |
(* IdR *)       Cp(sb,Id) -> sig-normsb(sb) |
(* Map *)       Cp(Pt(e1,s1),s2) ->
                (if (occurdummy1(e1) || occurdummy1sb(s1) ||
                occurdummy1sb(s2)) then sig-normsb(Pt(sig-norm(Sb(e1,s2)),
                sig-normsb(Cp(s1,s2)))) else subs) |
(*Assoc*)       Cp(Cp(s1,s2),s3) ->
                (if (occurdummy1sb(s1) || occurdummy1sb(s2) ||
                occurdummy1sb(s3)) then
                sig-norm(Cp(sig-normsb(s1),sig-normsb(Cp(s2,s3))))
                else subs) |
(*SCons*)       Pt(Sb(One,s1),Cp(Up,s2)) ->
                (if ((s1 = s2)&&(occurdummy1sb(s1))) then
                sig-normsb(s1) else subs) | _ -> subs;;
```

Note que em `sig-norm` as reduções não-triviais estão sempre ligadas a um condicional `if`, com exceção das regras `IdL`, `IdR` e `Clos`. Uma das justificativas para esse fato pode ser vista na seguinte redução (feita via `sig-norm`):
$((\lambda \underline{\mathbf{1}}[\uparrow^2])\ \diamond) \rightarrow_{\beta_s} \underline{\mathbf{1}}[\uparrow^2][\diamond.id] \rightarrow_{Clos} \underline{\mathbf{1}}[\uparrow^2 \circ (\diamond.id)] \rightarrow_{Assoc} \underline{\mathbf{1}}[\uparrow \circ (\uparrow \circ (\diamond.id))] \rightarrow_{ShiftCons}$
$\underline{\mathbf{1}}[\uparrow \circ id] \rightarrow_{IdR} \underline{\mathbf{1}}[\uparrow]$.

Caso as regras IdR e IdL tivessem um condicional como as outras, não seria possível obter o resultado desejado. De fato, elas não poderiam ser aplicadas, por exemplo, em $\underline{\mathbf{1}}[\uparrow \circ id]$ já que esse termo não possui ocorrências do símbolo $\diamond$ mas precisa ser simplificado. Como nosso objetivo inicial era apenas propagar subtermos com alguma ocorrência do símbolo $\diamond$, a implementação de Eta_σ acaba por utilizar passos adicionais durante a derivação.

No λs_e temos, $((\lambda M)\ \diamond) \rightarrow_{\sigma-gen} M\sigma^1\diamond$. Precisamos, então, s_e-pseudo-normalizar $M\sigma^1\diamond$. Denotaremos essa s_e-pseudo-normalização por `se-norm` que é dada por:

```
let rec se-norm exp =
match exp with
       Dummy         -> Dummy   |
       DB i          -> DB i  |
       Vr c          -> Vr c  |
       S(i,Vr c,Dummy) -> exp |
```

```
(*sigma-destruction*)  S(i,DB j,Dummy) ->
                       (if j<i then DB j else
                       (if j>i then (DB (j-1))
                       else P(0,i,Dummy))) |
(*sigma-app*)       S(i,A(e1,e2),Dummy)   ->
                    A((se-norm (S(i,e1,Dummy))),
                     (se-norm (S(i,e2,Dummy)))) |
(*sigma-lambda*)    S(i,L(e1),Dummy)      ->
                    L(se-norm (S(i+1,e1,Dummy))) |
(*sigma-sigma*)     S(i,S(j,e1,e2),Dummy) ->
                     (if i >= j  then
                    S(j, (se-norm (S(i+1,e1,Dummy))),
                     (se-norm (S(i-j+1,e2,Dummy))))
                      else exp) |
(*sigma-phi*)       S(i,P(k,n,e),Dummy)   ->
                    (if i>=k+n then
                    P(k,n,(se-norm (S(i-n+1,e,Dummy))))
                    else (if i>k then
                    P(k,n-1,e) else exp))  |
                    _ -> exp  ;;
```

Como podemos observar, a implementação acima é mais simples que a anterior já que o λs_e preserva a estrutura do termo, ou seja, o símbolo $\diamond$ permanece sempre como o último argumento do subtermo a ser normalizado quando simulamos uma η-conversão de um λ-termo puro. Como consequência desta regularidade, a implementação dessas regras pôde ser feita sem a necessidade de nenhum condicional. Isto, claramente, é uma grande vantagem para o λs_e já que nos outros cálculos a verificação da função `occurdummy*` tem custo linear no tamanho do termo para cada aplicação da regra.

No λ_{Susp} a implementação se assemelha muito à implementação do $\lambda\sigma$. Aqui temos que $((\lambda M)\ \diamond) \rightarrow_{\beta_s} [\![M, 1, 0, (\diamond, 0) :: nil]\!]$. Denotaremos por `susp-norm` a $Susp$-pseudo-normalização do termo $[\![M, 1, 0, (\diamond, 0) :: nil]\!]$. A função `susp-norm` é dada por:

```
let rec susp-norm exp =
match exp with
         Dummy  -> Dummy |
         DB i   -> DB i |
         Vr c   -> Vr c |
(*r_1*)  Sp(Dummy,i,j,env) -> Dummy |
(*r_2*)  Sp(DB i,0,j,Nilen) -> DB (i+j) |
(*r_3*)  Sp(DB 1,i,j,Con(Ar(k),env)) -> DB (j-k) |
(*r_4*)  Sp(DB 1,i,j,Con(Paar(e1,k),env)) ->
           (if (occurdummy3 e1)
               then susp-norm(Sp(e1,0,j-k,Nilen))
               else exp) |
(*r_5*)  Sp(DB i,j,k,Con(envt,env)) ->
           (if ((occurdummy3_Et envt) ||
                (occurdummy3_Env env))
               then susp-norm(Sp(DB (i-1),j-1,k,env))
```

```
              else exp) |
(*r_6*)  Sp(A(e1,e2),i,j,env) -> (if ((occurdummy3 e1) ||
           (occurdummy3 e2) || (occurdummy3_Env env))
         then A(susp-norm(Sp(e1,i,j,env)),
                susp-norm(Sp(e2,i,j,env))) else exp) |
(*r_7*)  Sp(L(e1),i,j,env) -> (if ((occurdummy3 e1) ||
           (occurdummy3_Env env))
         then L(susp-norm(Sp(e1,i+1,j+1,Con(Ar(j),env))))
         else exp) |
         _ -> exp;;
```

As observações para a implementação da `susp-norm` são análogas às feitas para `sig-norm`. Ou seja, observamos aqui as mesmas dificuldades encontradas para o $\lambda\sigma$ pois as regras r_2 e r_3 são implementadas sem nenhum condicional `if`.

Em função de todas essas observações concluímos que a implementação da regra *Eta* no cálculo λs_e é mais eficiente que nos cálculos $\lambda\sigma$ e λ_{Susp}.

Apesar de estarem corretas, nós não temos a completude dessas implementações. Isto é, as implementações só funcionam da forma esperada para termos puros do λ-cálculo e não para termos quaisquer como acreditávamos inicialmente. Um contra-exemplo, para o λs_e pode ser visto na figura a seguir:

$$(\underline{\mathbf{4}}\sigma^1\underline{\mathbf{1}})\sigma^1\diamond$$

σ-destruction	*σ-σ-transition*
$\underline{\mathbf{3}}\sigma^1\diamond$	$(\underline{\mathbf{4}}\sigma^2\diamond)\sigma^1(\underline{\mathbf{1}}\sigma^1\diamond)$
σ-destruction	*σ-destruction* 2
$\underline{\mathbf{2}}$	$\underline{\mathbf{3}}\sigma^1(\varphi_0^1\diamond)$

O ramo esquerdo corresponde à redução segundo a idéia original de Borovanský[Bor95], enquanto que o ramo direito corresponde à redução via s_e-pseudo-normalização. Como podemos ver esses últimos termos obtidos não são juntáveis via s_e-pseudo-normalização.

Concluiremos este capítulo apresentando os resultados que mostram a corretude das implementações das regras Eta_ξ, onde $\xi \in \{\sigma, s_e, Susp\}$ mostradas acima, para termos puros.

Lema 4.3 *Seja $M \in \Lambda_{dB}$. Então a σ-pseudo-normalização de*

$$M[\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1}]$$

nos fornece um novo termo que mantém todas as variáveis do termo M menores que k inalteradas, substitui a k-ésima variável por $\Diamond[\uparrow^{k-1}]$ e decrementa em 1 todas as variáveis maiores do que k.

Demonstração.

Por indução sobre a estrutura de M temos que:

- Se $M = \underline{\mathbf{n}}$ e $n < k$ então $\underline{\mathbf{1}}[\uparrow^{n-1}][\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1}] \rightarrow_{Clos}$
  $\underline{\mathbf{1}}[\uparrow^{n-1} \circ (\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1})] \rightarrow^{n-2}_{Assoc}$
  $\underline{\mathbf{1}}[\uparrow \circ (\uparrow \circ (\ldots (\uparrow \circ (\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1})] \rightarrow^{n-1}_{ShiftCons} \underline{\mathbf{1}}[\uparrow^{n-1}].$

  Se $n = k$ então $\underline{\mathbf{1}}[\uparrow^{n-1}][\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1}] \rightarrow_{Clos}$
  $\underline{\mathbf{1}}[\uparrow^{n-1} \circ (\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1})] \rightarrow^{n-2}_{Assoc}$
  $\underline{\mathbf{1}}[\uparrow \circ (\uparrow \circ (\ldots (\uparrow \circ (\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1})] \rightarrow^{n-1}_{ShiftCons} \Diamond[\uparrow^{n-1}].$

  Se $n > k$ então $\underline{\mathbf{1}}[\uparrow^{n-1}][\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1}] \rightarrow_{Clos}$
  $\underline{\mathbf{1}}[\uparrow^{n-1} \circ (\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1})] \rightarrow^{n-2}_{Assoc}$
  $\underline{\mathbf{1}}[\uparrow \circ (\uparrow \circ (\ldots (\uparrow \circ (\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1})] \rightarrow^{n-1}_{ShiftCons}$
  $\underline{\mathbf{1}}[\uparrow^{n-1-k} \circ \uparrow^{k-1}] = \underline{\mathbf{1}}[\uparrow^{n-2}].$

- Para $M = (A\ B)$ aplicamos diretamente a hipótese de indução.

- Se $M = (\lambda A)$ então $(\lambda A)[\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1}] \rightarrow_{Abs}$
  $\lambda A[\underline{\mathbf{1}}.((\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}].\Diamond[\uparrow^{k-1}].\ \uparrow^{k-1}) \circ \uparrow)] \rightarrow^{k}_{Map}$
  $\lambda A[\underline{\mathbf{1}}.(\underline{\mathbf{1}}[\uparrow].\underline{\mathbf{1}}[\uparrow][\uparrow].\ldots.\underline{\mathbf{1}}[\uparrow^{k-2}][\uparrow].\Diamond[\uparrow^{k-1}][\uparrow].(\uparrow^{k-1} \circ \uparrow))] \rightarrow_{Clos}$
  $\lambda A[\underline{\mathbf{1}}.\underline{\mathbf{1}}[\uparrow].\underline{\mathbf{1}}[\uparrow^{2}].\ldots.\underline{\mathbf{1}}[\uparrow^{k-1}].\Diamond[\uparrow^{k}].\ \uparrow^{k}]$. Agora utilizamos a hipótese de indução para concluirmos o resultado desejado.

□

Proposição 4.4 *Seja $M \in \Lambda_{dB}$. Se $\lambda(M\ \underline{\mathbf{1}}) \rightarrow_{\eta} N$ então a implementação da regra Eta no cálculo $\lambda\sigma$ simula esta η-conversão, isto é, $\lambda(M\ \underline{\mathbf{1}}) \rightarrow_{Eta_\sigma} N$.*

Demonstração.

A caracterização da regra Eta_σ é dada pela função `sig-norm` mostrada anteriormente. A demonstração é feita por indução sobre a estrutura do termo M.

- Se $M = \underline{\mathbf{n}}$ e $n \neq 1$ então por um lado temos que $\lambda(\underline{\mathbf{n}}\ \underline{\mathbf{1}}) \rightarrow_{\eta} \underline{\mathbf{n-1}}$. Queremos mostrar que $\underline{\mathbf{n}}[\diamond.id] = \underline{\mathbf{1}}[\uparrow^{n-1}][\diamond.id]$ também σ-pseudo-reduz a $\underline{\mathbf{n-1}}$. De fato, $\underline{\mathbf{1}}[\uparrow^{n-1}][\diamond.id] \rightarrow_{Clos} \underline{\mathbf{1}}[\uparrow^{n-1} \circ(\diamond.id)] \rightarrow_{Assoc} \underline{\mathbf{1}}[\uparrow^{n-2} \circ(\uparrow \circ(\diamond.id))] \rightarrow_{ShiftCons} \underline{\mathbf{1}}[\uparrow^{n-2} \circ(id)] \rightarrow_{IdR} \underline{\mathbf{1}}[\uparrow^{n-2}] = \underline{\mathbf{n-1}}$.

- Se $M = (A\ B)$ com A e B sem ocorrências da variável livre $\underline{\mathbf{1}}$ então, pela condição de aplicação da regra η ao par $(A\ B)$, temos que $\lambda(A\ \underline{\mathbf{1}}) \rightarrow_{\eta} A'$ e $\lambda(B\ \underline{\mathbf{1}}) \rightarrow_{\eta} B'$, onde A' e B' são obtidos de A e B respectivamente, decrementando todas as suas variáveis livres em 1.

  Por outro lado, $(A\ B)[\diamond.id] \rightarrow_{App} A[\diamond.id]\ B[\diamond.id]$. Mas por hipótese de indução temos que a σ-pseudo-normalização de $A[\diamond.id]$ e $B[\diamond.id]$ corresponde, respectivamente, a A' e B'.

- Se $M = (\lambda A)$ e A não possui ocorrências da variável livre $\underline{\mathbf{2}}$ então $\lambda((\lambda A)\ \underline{\mathbf{1}}) \rightarrow_{\eta} \lambda A''$, onde A'' consiste do termo A com todas as suas variáveis livres decrementadas em 1.

  Por outro lado, aplicando o lema 4.3 ao termo $M[\diamond.id]$, e lembrando que $\uparrow^0 = id$, obtemos o resultado desejado.

□

Lema 4.5 *Seja $M \in \Lambda_{dB}$. Então a s_e-pseudo-normalização do termo $M\sigma^i\diamond$ nos fornece um novo termo tal que todas as variáveis de M menores do que i permanecem inalteradas, a i-ésima variável é substituída por $\varphi_0^i\diamond$ e todas as variáveis maiores do que i são decrementadas em 1.*

Demonstração.

Por indução sobre a estrutura do termo M temos:

- Se $M = \underline{\mathbf{n}}$ e $n < i$ então $\underline{\mathbf{n}}\sigma^i\diamond \rightarrow_{\sigma\text{-}destr} \underline{\mathbf{n}}$.

  Quando $n = i$ então $\underline{\mathbf{n}}\sigma^i\diamond \rightarrow_{\sigma\text{-}destr} \varphi_0^i\diamond$.

  Quando $n > i$ então $\underline{\mathbf{n}}\sigma^i\diamond \rightarrow_{\sigma\text{-}destr} \underline{\mathbf{n-1}}$.

- Se $M = (A\ B)$ então $(A\ B)\sigma^i\diamond \rightarrow_{\sigma\text{-}app} (A\sigma^i\diamond)\ (B\sigma^i\diamond)$. Agora pela hipótese de indução temos o resultado desejado.

- Se $M = (\lambda A)$ então $(\lambda A)\sigma^i\Diamond \rightarrow_{\sigma\text{-}\lambda} \lambda A\sigma^{i+1}\Diamond$. Agora utilizamos a hipótese de indução para obtermos o resultado desejado.

□

Proposição 4.6 *Seja $M \in \Lambda_{dB}$. Se $\lambda(M\ \underline{\mathbf{1}}) \rightarrow_\eta N$ então a implementação da regra Eta no cálculo λs_e simula esta η-conversão, isto é, $\lambda(M\ \underline{\mathbf{1}}) \rightarrow_{Eta_{s_e}} N$.*

Demonstração.

Por indução sobre a estrutura de M, temos que:

- Se $M = \underline{\mathbf{n}}$ e $n > 1$ então, de maneira análoga ao proposição anterior precisamos mostrar que $\underline{\mathbf{n}}\sigma^1\Diamond \rightarrow \underline{\mathbf{n-1}}$. De fato, isto acontece após a aplicação da regra *σ-destruction*.

- Se $M = (A\ B)$ com A e B sem ocorrências da variável livre $\underline{\mathbf{1}}$ então, pela condição de aplicação da regra η ao par $(A\ B)$, temos que $\lambda(A\ \underline{\mathbf{1}}) \rightarrow_\eta A'$ e $\lambda(B\ \underline{\mathbf{1}}) \rightarrow_\eta B'$, onde A' e B' são obtidos de A e B respectivamente, decrementando todas as suas variáveis livres em 1.

  Por outro lado, temos que $(A\ B)\sigma^1\Diamond \rightarrow_{\sigma-app} (A\sigma^1\Diamond)\ (B\sigma^1\Diamond)$, e por hipótese de indução temos que $(A\sigma^1\Diamond) \rightarrow_{Eta_{s_e}} A'$ e $(B\sigma^1\Diamond) \rightarrow_{Eta_{s_e}} B'$.

- Se $M = (\lambda A)$ e A não possui ocorrências da variável livre $\underline{\mathbf{2}}$ então $\lambda((\lambda A)\ \underline{\mathbf{1}}) \rightarrow_\eta \lambda A''$, onde A'' consiste do termo A com todas as suas variáveis livres, exceto $\underline{\mathbf{1}}$, decrementadas em 1.

  Por outro lado temos que $(\lambda A)\sigma^1\Diamond \rightarrow_{\sigma\text{-}\lambda} \lambda A\sigma^2\Diamond$. Utilizando o lema 4.5 temos que este último termo terá todas as suas variáveis livres decrementadas em 1.

□

Proposição 4.7 *Seja $M \in \Lambda_{dB}$. Se $\lambda(M\ \underline{\mathbf{1}}) \rightarrow_\eta N$ então a implementação da regra Eta no cálculo λ_{Susp} simula esta η-conversão, isto é, $\lambda(M\ \underline{\mathbf{1}}) \rightarrow_{Eta_{Susp}} N$.*

Demonstração.

Por indução sobre a estrutura de M, temos:

- Se $M = \underline{\mathbf{n}}$ e $n > 1$ então, de maneira análoga às proposições anteriores precisamos mostrar que $[\![\underline{\mathbf{n}}, 1, 0, (\diamond, 0) :: nil]\!] \rightarrow^* \underline{\mathbf{n-1}}$. De fato,
  $[\![\underline{\mathbf{n}}, 1, 0, (\diamond, 0) :: nil]\!] \rightarrow_{r_5} [\![\underline{\mathbf{n-1}}, 0, 0, nil]\!] \rightarrow_{r_2} \underline{\mathbf{n-1}}$.

- Se $M = (A\ B)$ com A e B sem ocorrências da variável livre $\underline{\mathbf{1}}$ então, pela condição de aplicação da regra η ao par $(A\ B)$, temos que $\lambda(A\ \underline{\mathbf{1}}) \rightarrow_\eta A'$ e $\lambda(B\ \underline{\mathbf{1}}) \rightarrow_\eta B'$, onde A' e B' são obtidos de A e B respectivamente, decrementando todas as suas variáveis livres em 1.

  Por outro lado temos que

  $$[\![(A\ B), 1, 0, (\diamond, 0) :: nil]\!] \rightarrow_{r_6} [\![A, 1, 0, (\diamond, 0) :: nil]\!]\ [\![B, 1, 0, (\diamond, 0) :: nil]\!]$$

  e, aplicando a hipótese de indução que nos diz que

  $$[\![A, 1, 0, (\diamond, 0) :: nil]\!] \rightarrow_{Eta_{Susp}} A' \text{ e } [\![B, 1, 0, (\diamond, 0) :: nil]\!] \rightarrow_{Eta_{Susp}} B'$$

  temos o resultado desejado.

- Se $M = (\lambda A)$ e A não possui ocorrências da variável livre $\underline{\mathbf{2}}$ então $\lambda((\lambda A)\ \underline{\mathbf{1}}) \rightarrow_\eta \lambda A''$, onde A'' consiste do termo A com todas as suas variáveis livres,exceto $\underline{\mathbf{1}}$, decrementadas em 1.

  Por outro lado, $[\![(\lambda A), 1, 0, (\diamond, 0) :: nil]\!] \rightarrow_{r_7} \lambda[\![A, 2, 1, @0 :: (\diamond, 0) :: nil]\!]$. Aplicando o lema 4.2 temos o resultado desejado.

□

Conclusão

A área de Substituições Explícitas tem crescido rapidamente nos últimos anos devido a um grande interesse em se poder controlar explicitamente as operações do λ-cálculo. Vários sabores de cálculos de substituições explícitas têm sido propostos devido à dificuldade em se conseguir um que seja confluente (tanto em termos abertos quanto em termos fechados), que simule a β-conversão, que preserve a terminação forte (PSN) e cujo cálculo de substituições explícitas subjacente seja terminante.

Aqui estudamos três cálculos de substituições explícitas: o $\lambda\sigma$, λs_e e o λ_{Susp}.

O $\lambda\sigma$ tem um papel importante por ter sido o primeiro cálculo de substituições explícitas a ser apresentado à comunidade científica. Vimos que ele é local-confluente (para termos abertos), e que simula a β-conversão. O cálculo de substituição subjacente σ é terminante, mas [Mel95] mostrou que $\lambda\sigma$ não é PSN.

O λ_{Susp} foi desenvolvido por Nadathur e Wilson [NW98], e até então não tinha sido comparado com nenhum outro cálculo de substituições explícitas. Aqui introduzimos uma regra *Eta* para esse cálculo e vimos que o sistema $Susp + Eta_{Susp}$ é terminante e confluente (para termos abertos e fechados). Até o presente momento não sabemos se esse cálculo é PSN ou não. Além disso, vimos também que λ_{Susp} é incomparável, no sentido de [KR00], com o $\lambda\sigma$.

O λs_e, uma extensão para termos abertos do λs, é confluente, simula a β-conversão, mas, como mostrou [Gui00] não é PSN. Uma questão em aberto consiste em saber se o cálculo de substituição subjacente, s_e, é terminante. Mostramos que o λs_e é mais adequado que o λ_{Susp}. Na verdade, obtivemos aqui muito mais do que a noção de adequabilidade de [KR00], pois qualquer simulação de um passo de β-conversão no λs_e tem comprimento menor ou igual à correspondente simulação no λ_{Susp}.

Paralelamente aos estudos teóricos, desenvolvemos uma implementação com o objetivo de simular as derivações de um λ-termo em qualquer dos cálculos aqui trata-

dos. Essa implementação foi feita na liguagem de programação funcional Ocaml, da família ML, que utiliza tipagem implícita. A escolha desta linguagem se deu basicamente pela sua facilidade em se trabalhar com λ-termos. No capítulo 4 apresentamos a idéia geral desse programa com exemplos das estruturas utilizadas na construção da implementação das regras de reescrita dos cálculos $\lambda\sigma$, λs_e e λ_{Susp}. A construção das regras Eta_σ, Eta_{s_e} e Eta_{Susp} foi uma etapa importante do trabalho haja vista que a condição de aplicação da η-conversão no λ-cálculo não é trivial. Apresentamos as idéias utilizadas para a implementação dessas regras e mostramos a corretude das mesmas. Pudemos observar também que a implementação da regra Eta no λs_e é muito mais eficiente do que no λ_{Susp} ou no $\lambda\sigma$. Essa eficiência se deve ao fato de a propagação do símbolo $\diamond$ ser feita apenas sobre termos da forma $M\sigma^i\diamond$. Como esta propagação mantém sempre o símbolo $\diamond$ como segundo argumento da função σ^i, não necessitamos estruturas com condicionais `if` que elevam o custo computacional da implementação.

Interessantes questões teóricas resultaram da experiência implementacional. Em particular é necessário um refinamento da implementação que permita uma análise detalhada de possíveis alternativas para uma implementação eficiente da regra Eta que não misture a normalização com passos isolados e independentes de reescrita do cálculo de substituições associado. Além disso, a interface com o usuário pode ser melhorada assim como podem ser acrescentadas algumas opções como, por exemplo, a possibilidade de se efetuar a mesma simulação automaticamente nos três cálculos. Um refinamento do código para o tratamento de exceções, isto é, dados incorretos fornecidos pelo usuário, também é necessário.

Outro trabalho futuro diz respeito a conjectura que Nadathur apresentou em [Nad99] referente à preservação da terminação forte (PSN) para o λ_{Susp}. Acreditamos que uma análise cuidadosa dos resultados obtidos nesta dissertação no que diz respeito a maior adequabilidade do λs_e na simulação de um passo de β-conversão pode ser um bom ponto de partida para se concluir ou não a preservação da terminação forte (PSN) para o λ_{Susp}.

Referências

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[AR00] M. Ayala-Rincón. Fundamentos da programação lógica e funcional - o princípio da resolução e a teoria da reescrita. Departamento de Matemática, Universidade de Brasília, 2000. Disponível em: http://www.mat.unb.br/˜ayala/academics.html.

[ARK01a] M. Ayala-Rincón and F. Kamareddine. On Applying the λs_e-Style of Unification for Simply-Typed Higher Order Unification in the Pure lambda Calculus. In R.J.G.B. de Queiroz and M. Ayala-Rincón, editors, *Pre-Proceedings Eighth Workshop on Logic, Language, Information and Computation - WoLLIC 2001*, pages 41–54, 2001.

[ARK01b] M. Ayala-Rincón and F. Kamareddine. Unification via the λs_e-Style of Explicit Substitution. *Logical Journal of the Interest Group in Pure and Applied Logics - IGPL*, 9(4):521–555, 2001.

[ARM00] M. Ayala-Rincón and C. Muñoz. Explicit Substitutions and All That. *Revista Colombiana de Computación*, 1(1):47–71, 2000.

[Bar97] Henk Barendregt. The impact of the lambda calculus in logic and computer science. *Bulletin of Simbolic Logic*, 3(3):181–215, 1997.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and* All That. Cambridge University Press, 1998.

[Bor95] P. Borovanský. Implementation of Higher-Order Unification Based on Calculus of Explicit Substitutions. In M. Bartošek, J. Staudek, and J. Wiedermann, editors, *Proceedings of the SOFSEM'95: Theory and*

Practice of Informatics, volume 1012 of *Lecture Notes on Computer Science*, pages 363–368. Springer Verlag, 1995.

[CF58] H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North Holland, 1958.

[Chu32] A. Church. A set of postulates for the foundation of logic. *Annals of Math.*, 33(2):346–366, 1932.

[Chu33] A. Church. A set of postulates for the foundation of logic (second paper). *Annals of Math.*, 34(2):839–864, 1933.

[CR36] A. Church and J. B. Rosser. Some properties of conversion. *Trans. Amer. Math. Soc.*, 39:472–482, 1936.

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Pitman, 1986. Revised edition : Birkhäuser (1993).

[dB72] N.G. de Bruijn. Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Indag. Mat.*, 34(5):381–392, 1972.

[DHK00] G. Dowek, T. Hardin, and C. Kirchner. Higher-order Unification via Explicit Substitutions. *Information and Computation*, 157(1/2):183–235, 2000.

[Gui00] B. Guillaume. The λs_e-calculus Does Not Preserve Strong Normalization. *Journal of Functional Programming*, 10(4):321–325, july 2000.

[KB70] D. Knuth and P. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[KR35] Stephen Kleene and Barkley Rosser. The inconsistency of certain formal logics. *Annals of Math.*, 36(2):630–636, 1935.

[KR95a] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with Explicit Substitutions. In *Proc. of PLILP'95*, volume 982 of *LNCS*, pages 45–62. Springer, 1995.

[KR95b] F. Kamareddine and A. Ríos. The λs-calculus: its typed and its extended versions. Technical report, Department of Computing Science, University of Glasgow, 1995.

[KR97] F. Kamareddine and A. Ríos. Extending a λ-calculus with Explicit Substitution which Preserves Strong Normalisation into a Confluent Calculus on Open Terms. *Journal of Functional Programming*, 7:395–420, 1997.

[KR00] F. Kamareddine and A. Ríos. Relating the $\lambda\sigma$- and λs-Styles of Explicit Substitutions. *Journal of Logic and Computation*, 10(3):349–380, 2000.

[Mel95] P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate in Proceedings of TLCA'95. *LNCS*, 902:328–349, 1995.

[Nad99] G. Nadathur. A Fine-Grained Notation for Lambda Terms and Its Use in Intensional Operations. *The Journal of Functional and Logic Programming*, 1999(2):1–62, 1999.

[New42] M. H. A. Newman. On theories with a combinatorial definition of equivalence. *Ann. of Math.*, 43(2):223–243, 1942.

[NM88] G. Nadathur and D. Miller. An overview of λProlog. In R.A. Kowalski (Eds.) In: K.A. Bowen, editor, *Proc. 5th Internat. Logic Programming Conf. Seattle, Washington*, Cambridge, MA, pages 810–827. MIT Press, 1988.

[NW98] G. Nadathur and D. S. Wilson. A Notation for Lambda Terms A Generalization of Environments. *Theoretical Computer Science*, 198:49–98, 1998.

[RW13] B. A. W. Russel and A. N. Whitehead. *Principia Mathematica*, volume 1 and 2. Cambridge University Press, 1910-1913.