Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Matemática

Um Estudo Comparativo em Unificação de Ordem Superior via Cálculos de Substituições Explícitas

por

Flávio Leonardo Cavalcanti de Moura

Brasília, 23 de março de 2006

Resumo

Neste trabalho estudamos unificação de ordem superior em cálculos de substituições explícitas. Em particular comparamos o algoritmo de Huet para o λ-cálculo simplesmente tipado com o método desenvolvido por Dowek, Hardin e Kirchner para o $\lambda\sigma$-cálculo e concluímos que existe uma correspondência estrutural entre as derivações realizadas por estes métodos para um dado problema. Para formalizar esta comparação, adaptamos o algoritmo de Huet para a notação *à la* de Bruijn, já que esta é a notação utilizada pelos cálculos de substituições explícitas aqui tratados e, introduzimos as noções de *árvore de unificação* e *pseudo-cozimento*.

Adicionalmente, mostramos que unificação no $\lambda\sigma$-cálculo e no λs_e-cálculo podem gerar árvores de derivação iguais, no sentido que os dois métodos de unificação sempre podem utilizar regras de unificação "correspondentes" para um dado problema. Esta comparação é feita utilizando-se duas funções T e L que traduzem termos do λs_e-cálculo para o $\lambda\sigma$-cálculo e vice-versa, respectivamente, mostrando que uma dada regra de unificação pode ser utilizada no $\lambda\sigma$-cálculo para um problema P se, e somente se a regra "correspondente" pode ser utilizada no λs_e-cálculo para a tradução de P.

Em seguida, utilizamos as idéias desenvolvidas em unificação para construir uma aplicação específica de um caso restrito de unificação conhecido como *matching*. Apresentamos um algoritmo que decide uma classe especial de problemas de segunda ordem no $\lambda\sigma$-cálculo.

O estudo comparativo aqui apresentado é importante para uma melhor compreensão dos métodos de unificação utilizados em sistemas computacionais baseados no λ-cálculo, e do papel dos cálculos de substituições explícitas em unificação de ordem superior. Além disto, a noção de pseudo-cozimento aqui introduzida pode ser útil na implementação de versões mais eficientes dos métodos de unificação desenvolvidos para cálculos de substituições explícitas já que esta combina a noção de pré-cozimento usual com as regras de unificação.

Abstract

In this work we study higher-order unification via explicit substitutions calculi. In particular, we compare Huet's algorithm for the simply typed λ-calculus with the method developed by Dowek, Hardin and Kirchner for the $\lambda\sigma$-calculus and we conclude that there exists a structural correspondence between the derivations in these two methods for a given unification problem. In order to formalize this comparison, we adapt Huet's algorithm for de Bruijn's notation since this is the notation used by the explicit substitutions calculi considered here. In addition, we introduce the notions of *unification trees* and *pseudo-precooking*.

In addition, we show that unification in both $\lambda\sigma$- and λs_e-calculus may generate identical derivation trees in the sense that, both unification methods can always apply "corresponding" rules for a given unification problem. This comparison is done by introducing the functions T and L that convert terms from the λs_e-calculus to the $\lambda\sigma$-calculus and vice-versa, respectively, and proving that a unification rule can be applied in the $\lambda\sigma$-calculus for a problem P, if and only if, we can apply the "corresponding" rule of the λs_e-calculus to the translation of P.

We use the ideas developed for unification to build an specific application for a particular case of unification called *matching*. We present an algorithm that decides a special class of second order problems in the $\lambda\sigma$-calculus.

The comparative study presented here is important for a better understanding of the unification methods used in computational systems based on the λ-calculus, as well as the role of explicit substitutions in the higher-order unification and shed some light on implementational questions. Moreover, the new notion of pseudo-precooking can be useful for optimizing implementations of the unification methods developed for explicit substitutions calculi since it combines the usual notion of precooking with applications of the unification rules.

Índice

Resumo **ii**

Abstract **iii**

Introdução **v**

1 O λ-cálculo Simplesmente Tipado **1**
1.1 O λ-cálculo com Nomes 1
1.2 O λ-cálculo em Notação *à la* de Bruijn 5
1.3 O $\lambda\sigma$-cálculo 14
1.4 O λs_e-cálculo 18

2 Unificação de Ordem Superior **21**
2.1 O Algoritmo de Huet para o λ-cálculo com Nomes 21
2.2 Árvores de Unificação 27
2.3 Uma Adaptação do Algoritmo de Huet para a Notação *à la* de Bruijn . . . 29
2.4 Unificação no $\lambda\sigma$-cálculo 34

3 Uma Relação Estrutural entre HOU no λ-cálculo e no $\lambda\sigma$-cálculo **41**
3.1 O Pseudo-cozimento 41
3.2 Relacionando Árvores de Unificação e Árvores de Derivação 44

4 Relacionando os Métodos de Unificação no $\lambda\sigma$-cálculo e no λs_e-cálculo **62**
4.1 Correspondência a partir do λs_e-cálculo 62
4.2 Correspondência a partir do $\lambda\sigma$-cálculo 71

5 Aplicação: *Matching* de Segunda Ordem **77**
5.1 *Matching* de Ordem Superior 77
5.2 *Matching* de Segunda Ordem no $\lambda\sigma$-cálculo 78
5.3 Notação de Unificação por Transformação 83
5.4 Algoritmo para *Matching* de Segunda Ordem 83

Conclusão **95**

Bibliografia **97**

Introdução

O λ-cálculo surgiu no início do século XX com os trabalhos de Alonzo Church [Chu32, Chu33]. Conhecido como o primeiro sistema de reescrita de ordem superior no contexto computacional, o λ-cálculo é uma teoria que modela funções computáveis, com uma notação compacta para algoritmos, e com uma estrutura bem simples contendo duas operações principais: a primeira é um mecanismo para aplicar funções abstratas a argumentos, chamada de β-conversão; e a segunda, uma abstração para a equivalência funcional, chamada de η-conversão. Além disso, uma terceira operação, denominada α-conversão, permite mudar nomes de variáveis.

A operação de *substituição* utilizada na definição da β-conversão do λ-cálculo é uma operação implícita que não está definida formalmente, pois assume por exemplo, que variáveis sejam renomeadas automaticamente para evitar possíveis capturas, isto é, para evitar que variáveis originalmente livres se tornem ligadas após a aplicação da β-conversão. A implementação da substituição para sistemas computacionais baseados no λ-cálculo constitui um problema que geralmente é resolvido utilizando-se soluções *ad-hoc* (geralmente obscuras). Dentre os primeiros trabalhos que têm sido desenvolvidos com o intuito de controlar melhor essa operação, podemos citar [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 $\lambda\sigma$-cálculo. Os cálculos de substituições explícitas são formalismos que estendem a linguagem do λ-cálculo com operadores que tornam explícita a operação de substituição subdividindo-a em partes mais simples. O $\lambda\sigma$-cálculo utiliza a chamada notação *à la* de Bruijn [dB72]. Nesta notação, variáveis são representadas por números naturais que indicam o abstrator que liga cada variável. Cada λ-termo bem formado tem uma representação única em notação *à la* de Bruijn o que evita a necessidade de utilização da α-conversão, tornando assim, esta notação muito adequada para implementações.

Neste trabalho estudamos *unificação de ordem superior* (que abreviaremos por

HOU) em ambientes de substituições explícitas. Originalmente, o estudo de HOU foi motivado no sentido de se obter um algoritmo fundamental para a automatização de lógicas de ordem superior e, ainda hoje esta continua sendo sua principal aplicação [Hue02]. Em linhas gerais, o problema de HOU para dois λ-termos a e b do mesmo tipo, denotado pela equação de unificação $a =^? b$, consiste em encontrar todas as substituições σ tais que $a\sigma =_{\beta\eta} b\sigma$. Apesar de parecer um problema simples, HOU é um problema indecidível em geral [Luc72, Hue73, Gol81]. Em [Hue75], Huet desenvolveu um procedimento de semi-decisão para HOU que se mostrou eficiente na prática e, tem sido extensivamente utilizado em linguagens de programação e assistentes de provas baseados no λ-cálculo, como por exemplo λProlog [NM88], Isabelle/HOL [NPW02], Twelf [PS99] e Coq [CH85]. O sucesso do procedimento de semi-decisão de Huet está baseado na observação de que um tipo particular de equações (conhecidas como flexível-flexível) sempre têm solução e, portanto não é necessário considerá-las durante o processo de unificação. Por esta razão, este procedimento é conhecido como um *algoritmo de pré-unificação* e, apesar de não ser um algoritmo no sentido estrito da palavra, o mesmo é chamado de *o algoritmo de Huet*, nomenclatura que também utilizaremos aqui.

Em [DHK00], Dowek, Hardin e Kirchner desenvolveram um procedimento para o tratamento de problemas de HOU baseado em substituições explícitas. A grande vantagem de se atacar HOU com substituições explícitas está no fato de se poder reduzir um problema de ordem superior para primeira ordem módulo uma teoria equacional adequada e assim utilizar substituição de primeira ordem, também conhecida como *grafting*, ao invés da substituição de ordem superior usual. O $\lambda\sigma$-cálculo é o cálculo de substituições explícitas utilizado em [DHK00], mas o método desenvolvido é de aplicabilidade geral e adaptável para outros cálculos [ARK01], como por exemplo o λs_e [KR97]. Essencialmente, HOU baseado em substituições explícitas consiste em:

- Transcrever o problema da linguagem do λ-cálculo simplesmente tipado para a linguagem do cálculo de substituições explícitas de forma adequada. Esta translação é conhecida como *pré-cozimento*.

- Em seguida o problema é resolvido na linguagem do cálculo de substituições explícitas utilizando-se regras específicas;

- Finalmente as possíveis soluções são traduzidas de volta para a linguagem do

λ-cálculo.

Um dos resultados mais importantes de [DHK00] diz que o problema de unificação $a =^? b$ tem solução no λ-cálculo simplesmente tipado se e, somente se sua forma pré-cozida $a_F =^?_{\lambda\sigma} b_F$ tem solução no $\lambda\sigma$-cálculo. Neste trabalho refinamos este resultado estabelecendo uma relação estrutural entre subproblemas do problema original no λ-cálculo e os subproblemas obtidos a partir da forma pré-cozida do problema original no $\lambda\sigma$-cálculo. Para formalizar esta relação estrutural, inicialmente apresentamos uma adaptação do algoritmo de Huet para a notação *à la* de Bruijn já que esta é a notação utilizada pelo $\lambda\sigma$-cálculo. Em seguida, introduzimos a noção de *árvores de unificação* que simplifica a apresentação do algoritmo de Huet já que cada passo do algoritmo é explicitamente representado por um arco nesta árvore. De forma similar utilizamos uma estrutura de árvores para representar derivações no $\lambda\sigma$-cálculo que chamamos de *árvores de derivação*. Utilizando estas estruturas de árvores, mostramos que para um problema de unificação P no λ-cálculo simplesmente tipado em notação *à la* de Bruijn, cada subproblema gerado em sua árvore de unificação $\mathcal{A}(P)$ possui um problema associado na árvore de derivação de sua forma pré-cozida P_F no $\lambda\sigma$-cálculo (cf. [dMARK06]).

Adicionalmente, comparamos os métodos de unificação do λs_e-cálculo [ARK01] e do $\lambda\sigma$-cálculo [DHK00] via funções que transformam λs_e-termos em $\lambda\sigma$-termos e vice-versa. Desta forma, estabelecemos uma correspondência entre as regras de unificação destes métodos e mostramos que as árvores de derivação no λs_e-cálculo coincidem com as árvores de derivação do $\lambda\sigma$-cálculo sempre que a mesma estratégia é utilizada na construção de ambas as árvores.

Uma aplicação específica de unificação que tratamos aqui é conhecida como *matching*. *Matching* também é extensivamente utilizado em sistemas computacionais e pode ser informalmente definido por: se a e b são λ-termos do mesmo tipo então a equação de *matching* correspondente é denotada por $a \ll^? b$ e, neste caso estamos interessados em encontrar todas as substituições σ tais que $a\sigma =_{\beta\eta} b$. A partir das idéias utilizadas em unificação desenvolvemos um algoritmo específico para *matching* de segunda ordem [HL78] baseado no $\lambda\sigma$-cálculo. Este algoritmo é capaz de decidir os problemas de *matching* de segunda ordem que estão na imagem da função de pré-cozimento e, desta forma engloba os problemas realmente importantes, a saber: os problemas de *matching* de segunda ordem gerados a partir do λ-cálculo (cf. [dMKAR05a]).

No Capítulo 1 apresentamos o λ-cálculo simplesmente tipado com nomes e em notação *à la* de Bruijn. Esta apresentação é importante para fixar a notação utilizada em todo o trabalho. O $\lambda\sigma$-cálculo também é apresentado neste capítulo. No Capítulo 2 apresentamos o algoritmo de Huet para o λ-cálculo com nomes e, em seguida a nossa adaptação para a notação *à la* de Bruijn. Introduzimos também neste capítulo a noção de *árvores de unificação* que é de fundamental importância na formalização dos resultados estabelecidos. O procedimento de unificação para o $\lambda\sigma$-cálculo desenvolvido por Dowek, Hardin e Kirchner também é apresentado brevemente neste capítulo. O Capítulo 3 formaliza a relação estrutural que estabelecemos entre unificação no λ-cálculo e no $\lambda\sigma$-cálculo simplesmente tipados. Neste capítulo também apresentamos a noção de *árvores de derivação*. No Capítulo 4 comparamos os métodos de unificação do λs_e-cálculo e do $\lambda\sigma$-cálculo via funções que traduzem diretamente termos de um cálculo para o outro. O Capítulo 5 apresenta um algoritmo que decide *matching* de segunda ordem para formas pré-cozidas no $\lambda\sigma$-cálculo. Este algoritmo é uma adaptação do procedimento de unificação de [DHK00] construída a partir da caracterização da classe dos problemas de segunda ordem gerados a partir do λ-cálculo simplesmente tipado.

Capítulo 1

O λ-cálculo Simplesmente Tipado

Neste capítulo apresentamos o λ-cálculo simplesmente tipado com nomes e em notação *à la* de Bruijn. A versão simplesmente tipada dos cálculos de substituições explícitas $\lambda\sigma$ e λs_e também são apresentados neste capítulo.

1.1 O λ-cálculo com Nomes

Nesta seção apresentamos o λ-cálculo simplesmente tipado com nomes. Para maiores detalhes veja [Bar84, Bar92]. Termos do λ-cálculo, também chamados de λ-termos (ou simplesmente termos), são construídos a partir de constantes e variáveis ligadas ($x, y, z, \ldots \in \mathcal{V}$), de meta-variáveis ($X, Y, Z, \ldots \in \mathcal{X}$), de aplicações e abstrações de acordo com a seguinte gramática:

$$a ::= x \mid X \mid (a\ a) \mid \lambda_x.a, \text{ onde } x \in \mathcal{V} \text{ e } X \in \mathcal{X}.$$

Observação 1.1 *Como usual, parênteses são utilizados para evitar ambiguidades. Assim assumiremos que aplicações se associam à esquerda, ou seja,* $(a_1\ a_2 \ldots a_n)$ *significa* $((\ldots(a_1\ a_2)\ldots)a_n)$ *assim como* $\lambda_{x_1}\lambda_{x_2}\ldots\lambda_{x_n}.a$ *é uma abreviação para* $\lambda_{x_1}.(\lambda_{x_2}.(\ldots(\lambda_{x_n}.a)\ldots))$. *Além disto, assumiremos que aplicações têm maior prioridade que as abstrações.*

Definição 1.2 *O conjunto das* variáveis livres *do termo a, denotado por VL(a), é definido indutivamente por:*

(i) $VL(x) = \{x\}$, *para qualquer* $x \in \mathcal{V}$;

(ii) $VL(X) = \emptyset$, *para qualquer meta-variável* X;

(iii) $VL(a\ b) = VL(a) \cup VL(b)$;

(iv) $VL(\lambda_x.a) = VL(a) \backslash \{x\}$.

As ocorrências da variável x *na expressão* $\lambda_x.a$ *são ditas* ligadas.

No λ-cálculo com nomes, termos são interpretados módulo α-conversão, o que significa que nomes de variáveis ligadas são irrelevantes. Por exemplo, $\lambda_x.(x\ z)$ e $\lambda_y.(y\ z)$ representam o mesmo termo.

As operações básicas do λ-cálculo são a β-redução, que implementa a aplicação de termos funcionais à argumentos, e a η-redução, que representa a equivalência funcional. Estas operações são "implicitamente" definidas por:

$(\lambda_x.a)\ b \to a\{x/b\}$ (β)
$\lambda_x.(a\ x) \to a$, se x não ocorre livre em a. (η)

Na regra (β) acima, o termo $a\{x/b\}$ representa o termo obtido de a após substituir todas as suas ocorrências livres de x por b. Assim, a "implicitude" da definição da β-redução é consequência desta pseudo-definição de substituição. Este fato constitui o principal "defeito" teórico do λ-cálculo quando este é utilizado em implementações de sistemas computacionais baseados no λ-cálculo. De fato, tais implementações em geral utilizam mecanismos *ad-hoc* para contornar este problema que é ajustado durante a própria implementação. Como veremos adiante, cálculos de substituições explícitas atacam este problema via uma formalização (em diferentes estilos) da noção de substituição, o que faz com que este formalismo esteja mais próximo e mais ligado aos aspectos implementacionais.

A noção de η-redução corresponde à de equivalência funcional e, pode ser melhor compreendida se observarmos que, supondo que a variável x não ocorra livre no termo a, então vale que:

$$\underline{(\lambda_x.a\ x)}y \to_\beta (a\ x)\{x/y\} = (\underline{a}\ y)$$

η-conversão

Definição 1.3 (Forma $\beta\eta$-normal) *Um termo da forma* $(\lambda_x.b)c$ *(resp.* $\lambda_x.(b\ x)$*) é chamado um* β-rédice *(resp.* η-rédice*). Um* λ*-termo* a *está em forma* β*-normal*

*(resp. η-normal) se a não contém β-rédices (resp. η-rédices) como subtermos. Além disso, dizemos que um λ-termo a está em forma $\beta\eta$-*normal *se a não contém nem β- nem η-rédices.*

O ambiente adequado para HOU é o λ-cálculo simplesmente tipado, que apresentamos brevemente a seguir. Iniciamos assumindo que existe um conjunto infinito $\mathbb{T}$ de variáveis de tipos chamadas de *tipos atômicos.* Os *tipos simples* são indutivamente definidos por:

$$A ::= K \mid A \to A, \text{ onde } K \in \mathbb{T}.$$

Assim um tipo atômico é uma variável $K \in \mathbb{T}$ e, se A e B representam tipos, então $A \to B$ é também um tipo, chamado de *tipo funcional* cuja *ordem*, denotada por $|A \to B|$, é definida como sendo o máximo entre os valores $1+|A|$ e B. A ordem de um tipo atômico é definido como igual a 1. O construtor de tipos $\to$ é associa-se à direita, isto é, $A_1 \to \ldots A_{n-1} \to A_n$ $(n > 0)$ significa $(A_1 \to (\ldots (A_{n-1} \to A_n) \ldots))$. Se $A_1 \to \ldots \to A_n \to A$ representa um tipo, então dizemos que A é o seu *tipo alvo.*

Apresentaremos agora o λ-cálculo simplemente tipado. Adotaremos o estilo *à la* Church de tipagem de termos. Neste estilo, em oposição ao estilo *à la* Curry (conhecido como sistema de designação de tipos), termos tipados são indutivamente definidos por:

$$a \ ::= \ x \mid X \mid (a\ a) \mid \lambda_{x:A}.a, \text{ onde } x \in \mathcal{V} \text{ e } X \in \mathcal{X}.$$

A *ordem* de uma meta-variável é definida como sendo a ordem do seu tipo, e a ordem de um termo é dada pela mais alta ordem dentre as meta-variáveis que ocorrem no termo.

A separação de constantes e variáveis ligadas de um lado e meta-variáveis (também conhecidas como variáveis de unificação) de outro é importante para que tenhamos uma clara distinção entre as substituições geradas por aplicações de β-reduções e substituições geradas por regras de unificação.

Uma *designação de tipo* é uma expressão da forma $a : A$, onde a é um termo e A é um tipo. *Contextos de tipos* ou simplesmente contextos são utilizados para armazenar informações sobre os tipos das variáveis livres que ocorrem em um termo. Um *contexto* é definido como sendo um conjunto finito de designações de tipos para variáveis. Utilizaremos as letras gregas maiúsculas $\Gamma, \Delta, \ldots$ para denotar contextos.

Um contexto Γ é dito ser *consistente* se cada variável em Γ não possui mais do que uma designação. Daqui por diante chamaremos contextos consistentes simplesmente de contextos.

As regras de tipagem do λ-cálculo simplesmente tipado são dadas por:

(var) $\dfrac{}{\Gamma \cup \{x : A\} \vdash x : A}$

(meta) $\dfrac{}{\Gamma \vdash X : A}$, onde Γ é qualquer contexto (consistente).

(app) $\dfrac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\ b) : B}$

(lambda) $\dfrac{\Gamma \vdash a : B}{\Gamma - x \vdash \lambda_{x:A}.a : A \to B}$, se Γ é consistente com $x : A$.

O *julgamento de tipo* $\Gamma \vdash a : A$ é dito ser *derivável* se puder ser deduzido a partir das regras de tipagem acima.

Na regra (lambda) acima, a notação $\Gamma - x$ significa que a designação de tipo para x em Γ foi removida (se tal designação existia). A condição "Γ é consistente com $x : A$" significa que Γ contém apenas a designação $x : A$ para x, ou não contém nenhuma designação para x. No primeiro caso, dizemos que x foi *descarregado* de Γ, e no último caso dizemos que x foi descarregado de Γ por vacuidade.

A regra (meta) tem como consequência que o tipo das meta-variáveis é independente do contexto onde elas aparecem. Isto é necessário para que possamos tipar termos que contenham meta-variáveis ocorrendo em diferentes níveis de abstração. Por exemplo, considere o julgamento de tipo:

$$\vdash \lambda_{z:A\to(A\to A)\to A}.(z\ Y\ \lambda_{x:A}.Y) : (A \to (A \to A) \to A) \to A$$

Neste caso, a meta-variável Y ocorre em dois diferentes níveis de abstração e, é possível verificar que o julgamento acima pode ser obtido utilizando-se a regra (meta) duas vezes para se obter os julgamentos de tipos $\vdash Y : A$ e $x : A \vdash Y : A$.

As versões tipadas das regras β- e η-redução são dadas, respectivamente, por:

$(\lambda_{x:A}.a)\ b \to a\{x/b\}$ $\qquad (\beta)$

$\lambda_{x:A}.(a\ x) \to a$, se x não ocorre livre em a. $\qquad (\eta)$

Um λ-termo a está *bem tipado* se, e somente se, existe um contexto Γ e um tipo A tais que $\Gamma \vdash a : A$ é derivável. É bem conhecido que o λ-cálculo restrito a termos bem tipados é fechado para subtermos e $\beta\eta$-redução. Mais ainda, o λ-cálculo simplesmente tipado é fortemente terminante, isto é, qualquer $\beta\eta$-redução iniciando a partir de um termo bem tipado é finita.

Neste ponto é importante fazer alguns comentários sobre tipos e contextos para estas regras. Na regra (β) devemos observar que os termos $(\lambda_{x:A}.a)$ e b são bem tipados, com o mesmo contexto e, além disto b tem que ter tipo A a fim de que esta aplicação esteja bem formada (lembre-se da regra (app)). Mais ainda, a variável x e o termo b têm o mesmo tipo, mas os contextos em que eles aparecem não precisam coincidir. De fato, se x ocorre livre no termo a então esta variável se encontra no escopo de (pelo menos) um abstrator e, portanto x está bem tipada sob um contexto que contenha uma designação para x. Por outro lado, o termo b é independente da abstração $\lambda_{x:A}$ no sentido que não precisa que seu contexto contenha uma designação para x. Como um exemplo simples, considere o seguinte β-rédice:

$$(\lambda_{x:A}.x)\ y$$

que é um termo bem tipado como podemos ver através da seguinte derivação:

$$\dfrac{\dfrac{\dfrac{}{\{y:A, x:A\} \vdash x:A}\,(var)}{\{y:A\} \vdash (\lambda_{x:A}.x) : A \to A}\,(lambda) \quad \dfrac{}{\{y:A\} \vdash y:A}\,(var)}{\{y:A\} \vdash ((\lambda_{x:A}.x)\ y) : A}\,(app)$$

A partir desta derivação podemos ver que a variável x está bem tipada no contexto $\{y:a, x:A\}$, enquanto que a variável y está bem tipada no contexto $\{y:A\}$ que não possui uma designação para x.

1.2 O λ-cálculo em Notação *à la* de Bruijn

Nesta seção apresentaremos o λ-cálculo simplesmente tipado em notação *à la* de Bruijn desenvolvida pelo matemático holandês N. G. de Bruijn [dB72]. A filosofia da notação *à la* de Bruijn é baseada no fato de que a ligação entre uma variável e seu correspondente abstrator, usualmente feita através do nome da variável, também pode ser feita considerando-se uma contagem sobre o número de abstratores que englobam esta variável. Para se fazer isto, variáveis e constantes são representadas por inteiros positivos chamados de *índices* à la *de Bruijn*. As variáveis livres (ou meta-variáveis) serão representadas por letras latinas maiúsculas $X, Y, Z, \ldots$. A apresentação original de N. G. de Bruijn não contém meta-variáveis, mas esta separação de variáveis em duas classes será importante por duas razões: primeiro porque

isto vai propiciar uma melhor compreensão dos métodos de unificação tratados aqui já que assim conseguiremos manter uma clara distinção entre as substituições geradas por aplicações de β-redução e as geradas por aplicações de regras de unificação; segundo, porque desta forma manteremos uma gramática semelhante à da apresentação que fizemos do λ-cálculo com nomes permitindo assim ver diretamente as diferenças e semelhanças das duas abordagens.

Contextos no λ-cálculo em notação *à la* de Bruijn são listas de tipos. Lembre-se que na versão com nomes, contextos são representados por conjuntos finitos de designações de tipos para variáveis. Contextos em notação *à la* de Bruijn precisam ser ordenados porque os índices *à la* de Bruijn livres agora se referem a posições específicas no contexto como veremos a seguir.

O λ-cálculo com nomes e em notação *à la* de Bruijn são isomorfos [Mau85] e, termos podem ser traduzidos de uma linguagem para a outra facilmente. Por exemplo, para traduzirmos o λ-termo $\lambda_x\lambda_y\lambda_z.(x\ (y\ z)\ z)$ para a notação *à la* de Bruijn basta observarmos a que abstrator corresponde cada variável:

$$\lambda_x\lambda_y\lambda_z.(x\ (y\ z)\ z) \qquad \begin{array}{c}\text{conversão para a}\\ \text{notação de de Bruijn}\end{array} \qquad \lambda\lambda\lambda.(\underline{3}\ (\underline{2}\ \underline{1})\ \underline{1})$$

Termos contendo constantes e meta-variáveis também podem ser traduzidos para a notação *à la* de Bruijn. As meta-variáveis permanecem inalteradas durante a conversão, mas precisamos criar um referencial contendo todas as constantes que ocorrem no termo. Por exemplo, considerando o termo $\lambda_x\lambda_y\lambda_z.(y\ (X\ u\ z)\ v)$ segundo o referencial u, v, consiste em considerar este termo sob o escopo dos abstratores $\lambda_v\lambda_u$, o que nos dá $\lambda\lambda\lambda.(\underline{2}\ (X\ \underline{4}\ \underline{1})\ \underline{5})$. Se o referencial adotado fosse v, u, o termo obtido seria $\lambda\lambda\lambda.(\underline{2}\ (X\ \underline{5}\ \underline{1})\ \underline{4})$ e, assim referenciais distintos resultam em representações diferentes. Para termos tipados, tais referenciais correspondem aos contextos.

Definição 1.4 *O conjunto de λ-termos (sem tipos) em notação* à la *de Bruijn é definido indutivamente por:*

$$a ::= \underline{n} \mid X \mid (a\ a) \mid \lambda.a \qquad \textit{onde } n \in \mathbb{N} \textit{ e } X \in \mathcal{X}.$$

Tipos, contextos e termos no λ-cálculo simplemente tipado são definidos por:

Tipos	$A ::= K \mid A \to A$	*onde* $K \in \mathbb{T}$.
Contextos	$\Gamma ::= nil \mid A \cdot \Gamma$	
Termos	$a ::= \underline{n} \mid X \mid (a\ a) \mid \lambda_A.a$	*onde* $n \in \mathbb{N}$ *and* $X \in \mathcal{X}$.

O conjunto de λ-termos bem tipados em notação à la *de Bruijn é denotado por $\Lambda_{dB}(\mathcal{X})$ e, as regras de tipagem para este cálculo são dadas por:*

$$\textit{(var)} \quad \frac{}{A \cdot \Gamma \vdash \underline{1} : A} \qquad \textit{(var+)} \quad \frac{\Gamma \vdash \underline{n} : B}{A \cdot \Gamma \vdash \underline{n+1} : B}$$

$$\textit{(lambda)} \quad \frac{A \cdot \Gamma \vdash a : B}{\Gamma \vdash \lambda_A.a : A \to B} \qquad \textit{(app)} \quad \frac{\Gamma \vdash a : A \to B \;\; \Gamma \vdash b : A}{\Gamma \vdash \;\; (a\ b) : B}$$

Adicionalmente, para cada meta-variável X, associamos um único tipo e, para cada tipo assumimos que existe um número infinito de meta-variáveis distintas com este tipo. A regra de tipagem para meta-variáveis é dada por:

$$\textit{(meta)} \quad \frac{}{\Gamma \vdash X : A}, \qquad \textit{onde } \Gamma \textit{ é um contexto qualquer.}$$

Como no λ-cálculo com nomes, o tipo das meta-variáveis é independente do contexto. No entanto, note que o tipo dos termos, em geral, depende do contexto.

Se a é um λ-termo, escreveremos a^{Γ}_{A} como uma abreviação para o julgamento de tipos $\Gamma \vdash a : A$.

A regra (β) para o λ-cálculo em notação *à la* de Bruijn é definida por:

$$(\lambda_A.a)\ b \to a\{\underline{1}/b\} \qquad (\beta)$$

enquanto que a regra (η) é definida por:

$$\lambda_A.(a\ \underline{1}) \to b \text{ if } a = b^{+} \qquad (\eta)$$

onde o operador $^{+}$ é dado na Definição 1.5.

Dizemos que um λ-termo a, em notação *à la* de Bruijn, está em forma β-normal se a não possui um subtermo da forma $(\lambda_A.b)\ c$. Esta definição requer regras específicas para que possamos propagar a substituição $\{\underline{1}/b\}$ no termo a. As definições a seguir formalizam o processo de propagação da substituição.

Definição 1.5 *Sejam $a \in \Lambda_{dB}(\mathcal{X})$ e $i \geq 0$. O termo a^{+}, chamado a* elevação *de a, é definido por $a^{+} = a^{+0}$, onde a^{+i} é indutivamente definido por:*

(a) $X^{+i} = X$, *para* $X \in \mathcal{X}$.

(b) $\underline{n}^{+i} = \begin{cases} \underline{n+1}, & \textit{se } n > i; \\ \underline{n}, & \textit{se } n \leq i. \end{cases}$

(c) $(a\ b)^{+i} = (a^{+i}\ b^{+i})$.

(d) $(\lambda_A.a)^{+i} = \lambda_A.a^{+(i+1)}$.

Definição 1.6 *Sejam $\underline{n}, a, b$ λ-termos bem tipados em notação* à la *de Bruijn tais que $\underline{n}$ e a são dois λ-termos com o mesmo tipo. A substituição de $\underline{n}$ por a em b, denotada por $b\{\underline{n}/a\}$, é definida por indução sobre a estrutura de b como segue:*

$(a)\ X\{\underline{n}/a\} = X.$ $\qquad$ $(b)\ \underline{m}\{\underline{n}/a\} = \begin{cases} \underline{m} & se\ m < n; \\ a & se\ m = n; \\ \underline{m-1} & se\ m > n. \end{cases}$

$(c)\ (c\ d)\{\underline{n}/a\} = (c\{\underline{n}/a\}\ d\{\underline{n}/a\})$ $\qquad$ $(d)\ (\lambda_A.c)\{\underline{n}/a\} = \lambda_A.(c\{\underline{n+1}/a^+\})$

A Definição 1.6 é específica para β-redução: de fato, no item (b) quando $m > n$ temos que $\underline{m}\{\underline{n}/a\}$ se reduz para $\underline{m-1}$ porque esta substituição assume (implicitamente) que um λ foi removido após uma aplicação de β-redução e, portanto todos os índices livres do termo atual precisam ser decrementados em um, já que as mesmas se encontram agora no escopo de um abstrator a menos. O item (d) nos diz que quando a substituição $\{\underline{n}/a\}$ é propagada para dentro de um abstrator, todos os seus índices livres precisam ser incrementados em 1.

Nas definições 1.5 e 1.6, a informação de tipos e contextos foi omitida para não carregar excessivamente a notação, mas é importante entender como os contextos são atualizados quando substituições são propagadas dentro dos termos.

A versão decorada com tipos e contextos da β-redução é dada por:

$$(\lambda_A.a_B^{A\cdot\Gamma})^{\Gamma}_{A\to B}\ b_A^{\Gamma} \to a_B^{A\cdot\Gamma}\{\underline{1}_A^{A\cdot\Gamma}/b_A^{\Gamma}\} \qquad (\beta)$$

Como podemos observar nesta versão decorada da regra (β), assim como no λ-cálculo com nomes, em notação *à la* de Bruijn as substituições são pares da forma $\{v/b\}$, onde v é um índice *à la* de Bruijn ou uma meta-variável e b é um λ-termo tais que v e b possuem o mesmo tipo, mas podem aparecer em contextos distintos. De fato, note que a variável v se refere a ocorrências em um termo que está dentro do escopo do abstrator que vai ser removido após a aplicação da (β), enquanto que b não se encontra dentro do escopo deste abstrator. Intuitivamente, a β-redução acima nos diz que cada ocorrência livre de $\underline{1}_A^{A\cdot\Gamma}$ em $a_B^{A\cdot\Gamma}$ deve ser substituída por b_A^{Γ} e, por esta razão $\underline{1}_A^{A\cdot\Gamma}$ e $a_B^{A\cdot\Gamma}$ precisam ter o mesmo contexto.

A seguir discutimos a versão decorada com tipos e contextos das definições 1.5 e 1.6.

Observação 1.7 *Sejam Δ um contexto e, $A, B, A_1, \ldots, A_n$ tipos quaisquer. A versão decorada da Definição 1.6 é dada como segue, onde assumimos que o abstrator do β-redex que originou a substituição é de tipo A_1:*

(a) $X_A^{A_n\cdot\ldots\cdot A_1\cdot\Delta}\{\underline{n}_B^{A_n\cdot\ldots\cdot A_1\cdot\Delta}/a_B^{A_n\cdot\ldots\cdot A_2\cdot\Delta}\} = X_A^{A_n\cdot\ldots\cdot A_2\cdot\Delta}$, *isto é, meta-variáveis não são afetadas pelas substituições geradas por* β*-reduções, exceto que o contexto precisa ser atualizado de acordo com a substituição.*

(b) Analisamos separadamente cada caso:

- *Se* $m < n$ *então* $\underline{m}$ *é um índice* à la *de Bruijn ligado e, portanto não deve mudar:*
$$\underline{m}_A^{A_n\cdot\ldots\cdot A_1\cdot\Delta}\{\underline{n}_B^{A_n\cdot\ldots\cdot A_1\cdot\Delta}/a_B^{A_n\cdot\ldots\cdot A_2\cdot\Delta}\} = \underline{m}_A^{A_n\cdot\ldots\cdot A_2\cdot\Delta}.$$
- *Se* $m = n$ *então temos:*
$$\underline{n}_A^{A_n\cdot\ldots\cdot A_1\cdot\Delta}\{\underline{n}_A^{A_n\cdot\ldots\cdot A_1\cdot\Delta}/a_A^{A_n\cdot\ldots\cdot A_2\cdot\Delta}\} = a_A^{A_n\cdot\ldots\cdot A_2\cdot\Delta}.$$
- *Se* $m > n$ *então* $\underline{m}$ *representa uma constante que agora está inserida em um* λ*-termo com um abstrator a menos (lembre-se que um abstrator é eliminado ao se aplicar uma* β*-redução) e, desta forma o elemento do contexto que corresponde ao tipo do abstrator eliminado precisa ser removido:*
$$\underline{m}_A^{A_n\cdot\ldots\cdot A_1\cdot\Delta}\{\underline{n}_B^{A_n\cdot\ldots\cdot A_1\cdot\Delta}/a_B^{A_n\cdot\ldots\cdot A_2\cdot\Delta}\} = \underline{m-1}_B^{A_n\cdot\ldots\cdot A_2\cdot\Delta}.$$

(c) Trivial.

(d) Ao propagarmos uma substituição dentro de um abstrator precisamos atualizar o índice $\underline{n}$ *que define a substituição assim como o termo a ser substituído e incluir o tipo deste abstrator nos contextos:*
$$((\lambda_B.b_A^{B\cdot A_n\cdot\ldots\cdot A_1\cdot\Delta})_{B\to A}^{A_n\cdot\ldots\cdot A_1\cdot\Delta}\{\underline{n}_{A_1}^{A_n\cdot\ldots\cdot A_1\cdot\Delta}/a_{A_1}^{A_n\cdot\ldots\cdot A_2\cdot\Delta}\})_{B\to A}^{A_n\cdot\ldots\cdot A_2\cdot\Delta} =$$
$$(\lambda_B.(b_A^{B\cdot A_n\cdot\ldots\cdot A_1\cdot\Delta}\{\underline{n+1}_{A_1}^{B\cdot A_n\cdot\ldots\cdot A_1\cdot\Delta}/(a^+)_{A_1}^{B\cdot A_n\cdot\ldots\cdot A_2\cdot\Delta}\}))_{B\to A}^{A_n\cdot\ldots\cdot A_2\cdot\Delta}.$$

As atualizações de contextos na Definição 1.5, são induzidas pelo item (d) acima. De fato, a *elevação* de um termo é introduzida apenas quando uma substituição é propagada dentro de abstratores cujos tipos são essenciais para que possamos determinar o contexto resultante dos termos. Assumiremos que B é o tipo da abstração que originou a *elevação*. A versão decorada da Definição 1.5 é dada por:

(a) Para todo $i \geq 0$, $((X_A^{A_1\cdot\ldots\cdot A_i\cdot\Delta})^{+i})_A^{A_1\cdot\ldots\cdot A_i\cdot B\cdot\Delta} = X_A^{A_1\cdot\ldots\cdot A_i\cdot B\cdot\Delta}$.

(b) Analisamos cada caso separadamente:

– Se $n > i$ então o índice $\underline{n}$ representa uma constante que está no escopo de i abstratores e, que agora foi inserida no escopo de um novo abstrator de tipo B. Assim, este índice precisa ser atualizado e deve conter em seu contexto informação referente a este novo abstrator:

$$((\underline{n}^{A_1\cdot\ldots\cdot A_i\cdot\Delta}_{A_n})^{+i})^{A_1\cdot\ldots\cdot A_i\cdot B\cdot\Delta}_{A_n} = \underline{n+1}^{A_1\cdot\ldots\cdot A_i\cdot B\cdot\Delta}_{A_n}$$

– Se $n \leq i$ então $\underline{n}$ representa um índice que está ligado a um abstrator e, portanto deve permanecer inalterado. No entanto o contexto resultante depende da *elevação*:

$$((\underline{n}^{A_1\cdot\ldots\cdot A_i\cdot\Delta}_{A_n})^{+i})^{A_1\cdot\ldots\cdot A_i\cdot B\cdot\Delta}_{A_n} = \underline{n}^{A_1\cdot\ldots\cdot A_i\cdot B\cdot\Delta}_{A_n}$$

(c) Trivial.

(d) Para propagarmos a *elevação* dentro de uma nova abstração é necessário incluir o tipo B da abstração que originou esta *elevação*:

$$(((\lambda_A.a^{A\cdot A_1\cdot\ldots\cdot A_i\cdot\Delta}_C)^{A_1\cdot\ldots\cdot A_i\cdot\Delta}_{A\to C})^{+i})^{A_1\cdot\ldots\cdot A_i\cdot B\cdot\Delta}_{A\to C} = (\lambda_A.(a^{i+1})^{A\cdot A_1\cdot\ldots\cdot A_i\cdot B\cdot\Delta}_C)^{A_1\cdot\ldots\cdot A_i\cdot B\cdot\Delta}_{A\to C}.$$

A regra de η-redução é definida por:

$$\lambda_A.(a\ \underline{1}) \to b \text{ se } a = b^+ \qquad (\eta)$$

e sua versão decorada com tipos e contextos é dada por:

$$\lambda_A.(a^{A\cdot\Gamma}_B\ \underline{1}^{A\cdot\Gamma}_A) \to b^\Gamma_B \text{ se } a^{A\cdot\Gamma}_B = (b^+)^{A\cdot\Gamma}_B \qquad (\eta)$$

A definição de η-redução tenta capturar a semântica operacional da η-redução do λ-cálculo com nomes, mas na verdade ela falha já que esta contrução não nos diz como obter o termo b a partir do termo a. No entanto, implementações da η-redução baseadas na detecção de ocorrências livres do índice $\underline{1}$ em a são consideradas adequadas [ARdMK05, Bor95, Ven06]. Sempre que possível evitaremos utilizar a versão decorada dos termos para simplificar a notação.

Como mencionado anteriormente, a separação entre variáveis livres (as chamadas meta-variáveis) de um lado e, variáveis ligadas e constantes, de outro, nos permite distinguir entre as substituições geradas por β-reduções e as geradas pelos procedimentos de unificação. A definição a seguir formaliza a noção de substituição gerada pelo procedimento de unificação, isto é, substituição para meta-variáveis:

Definição 1.8 *Seja θ uma função de $\mathcal{X}$ para $\Lambda_{dB}(\mathcal{X})$. Uma* substituição *$\theta'$ é uma extensão da função θ definida por:*

(a) $X\theta' = X\theta$;

(b) $\underline{n}\theta' = \underline{n}$;

(c) $(a\ b)\theta' = (a\theta'\ \ b\theta')$;

(d) $(\lambda_A.a)\theta' = \lambda_A.(a\theta'^{+})$, *onde* $\theta'^{+} := \{X_1^+/a_1^+, \ldots, X_n^+/a_n^+\}$
sendo $\theta' = \{X_1/a_1, \ldots, X_n/a_n\}$.

A principal diferença entre substituições geradas por β-reduções e substituições geradas pelo algoritmo de unificação é que, as últimas sempre substituem meta-variáveis por um termo de mesmo tipo e contexto, isto é, as substituições da forma X/a são tais que X e a são λ-termos que possuem o mesmo tipo e o mesmo contexto, enquanto que as substituições geradas a partir de β-reduções substituem índices por termos de mesmo tipo mas com contextos diferentes.

Na Definição 1.8 apenas a transformação indicada no item (d) exige uma atualização de contextos devido à elevação que é introduzida ao se propagar a substituição dentro de uma nova abstração. Por exemplo, a versão decorada do item (d) é dada por $(\lambda_B.c_C^{B\cdot\Delta})\{X_A^\Gamma/a_A^\Gamma\} = \lambda_B.(c_C^{B\cdot\Delta}\{X_A^{B\cdot\Gamma}/(a^+)_A^{B\cdot\Gamma}\})$.

O exemplo a seguir, ilustra o processo de propagação de substituições geradas pelo algoritmo de unificação e também a noção de elevação.

Exemplo 1.9 *Sejam $\Gamma = (A \to A) \to A \cdot nil$ um contexto e X uma meta-variável de tipo $(A \to A) \to A$ e contexto Γ. Mostraremos como a substituição:*

$$\{X^\Gamma_{(A\to A)\to A}/\underline{1}^\Gamma_{(A\to A)\to A}\}$$

deve ser aplicada ao termo:

$$\lambda_{A\to A}.(X\ \lambda_A.(X\ \underline{2}))$$

que tem tipo $(A \to A) \to A$ e contexto Γ. Para não carregar a notação utilizaremos decoração com tipos e contextos apenas nos subtermos onde esta informação for relevante:
$(\lambda_{A\to A}.(X\ \lambda_A.(X\ \underline{2})))^\Gamma_{(A\to A)\to A}\{X^\Gamma_{(A\to A)\to A}/\underline{1}^\Gamma_{(A\to A)\to A}\} =$

$\lambda_{A\to A}.(X\ \lambda_A.(X\ \underline{2}))^{A\to A\cdot\Gamma}_{A}\{X^{A\to A\cdot\Gamma}_{(A\to A)\to A}/\underline{2}^{A\to A\cdot\Gamma}_{(A\to A)\to A}\} =$
$\lambda_{A\to A}.(\underline{2}^{A\to A\cdot\Gamma}_{(A\to A)\to A}\ (\lambda_A.(X\ \underline{2}))^{A\to A\cdot\Gamma}_{A\to A}\{X^{A\to A\cdot\Gamma}_{(A\to A)\to A}/\underline{2}^{A\to A\cdot\Gamma}_{(A\to A)\to A}\}) =$
$\lambda_{A\to A}.(\underline{2}^{A\to A\cdot\Gamma}_{(A\to A)\to A}\ \lambda_A.(X\ \underline{2})^{A\cdot A\to A\cdot\Gamma}_{A}\{X^{A\cdot A\to A\cdot\Gamma}_{(A\to A)\to A}/\underline{3}^{A\cdot A\to A\cdot\Gamma}_{(A\to A)\to A}\}) =$
$\lambda_{A\to A}.(\underline{2}^{A\to A\cdot\Gamma}_{(A\to A)\to A}\ \lambda_A.(\underline{3}^{A\cdot A\to A\cdot\Gamma}_{(A\to A)\to A}\ \underline{2}^{A\cdot A\to A\cdot\Gamma}_{A\to A}))$.

Neste exemplo, a elevação foi utilizada duas vezes: na segunda e quarta linhas (de cima para baixo) devido à necessidade de se propagar a substituição dentro das duas abstrações existentes no termo dado.

A seguir definimos *funções de atualização* para índices *à la* de Bruijn. Estas funções são importantes na definição de forma η-longa em notação *à la* de Bruijn.

Definição 1.10 *As* funções de atualização $U^i_k : \Lambda_{dB}(\mathcal{X}) \to \Lambda_{dB}(\mathcal{X})$, *para* $k \geq 0$ *e* $i \geq 1$ *são indutivamente definidas por:*

(a) $U^i_k(X) = X$, *para* $X \in \mathcal{X}$;

(b) $U^i_k(a\ b) = (U^i_k(a)\ U^i_k(b))$;

(c) $U^i_k(\lambda_A.a) = \lambda_A.U^i_{k+1}(a)$;

(d) $U^i_k(\underline{n}) = \begin{cases} \underline{n+i-1}, & \text{se } n > k; \\ \underline{n}, & \text{se } n \leq k. \end{cases}$

As formas η-longas desempenham um papel importante em unificação de ordem superior e, são definidas por:

Definição 1.11 (Forma η-longa) *Seja* $a \in \Lambda_{dB}(\mathcal{X})$ *um* λ*-termo em notação* à la *de Bruijn, em forma* β*-normal e com tipo dado por* $A_1 \to \ldots \to A_m \to B$ *(*B *atômico) no contexto* Γ. *A forma* η-longa *do termo* a, *denotada por* a', *é indutivamente definida por:*

- *Se* $a = \lambda_A.b$ *então* $a' = \lambda_A.b'$.
- *Se* $a = (\underline{n}\ b_1 \ldots b_q)$ $q \geq 0$, *então* $a' = \lambda_{A_1} \ldots \lambda_{A_m}.(\underline{n+m}\ c_1 \ldots c_q\ \underline{m}' \ldots \underline{1}')$, *onde* $c_1, \ldots, c_q$ *correspondem a forma* η*-longa da forma* β*-normal de* $U^{m+1}_0(b_1)$, $\ldots$, $U^{m+1}_0(b_q)$, *respectivamente.*
- *Se* $a = (X\ b_1 \ldots b_q)$, *com* $q \geq 0$, *então* $a' = \lambda_{A_1} \ldots \lambda_{A_m}.(X\ c_1 \ldots c_q\ \underline{m}' \ldots \underline{1}')$, *onde* $c_1, \ldots, c_q$ *correspondem a forma* η*-longa da forma* β*-normal de* $U^{m+1}_0(b_1)$, $\ldots$, $U^{m+1}_0(b_q)$, *respectivamente.*

Exemplo 1.12 *Considere o seguinte julgamento de tipos:*

$$A \to A \cdot nil \vdash \underline{1} : A \to A$$

A forma η-longa do índice $\underline{1}$ neste julgamento de tipo é dada passo a passo da seguinte forma:

$A \to A \cdot nil \vdash \underline{1} : A \to A$

$A \to A \cdot nil \vdash \lambda_A.(\underline{2}\ \underline{1}') : A \to A$. *Agora como a forma η-longa de um índice de tipo atômico é o próprio índice, obtemos a seguinte forma η-longa:*

$A \to A \cdot nil \vdash \lambda_A.(\underline{2}\ \underline{1}) : A \to A$

Exemplo 1.13 *Um exemplo mais interessante consiste em calcular a forma η-longa do termo contido no seguinte julgamento de tipos:*

$$(A \to A) \to A \cdot nil \vdash \underline{1} : (A \to A) \to A.$$

De acordo com a definição de forma η-longa, num primeiro passo obtemos:

$$(A \to A) \to A \cdot nil \vdash \lambda_{A\to A}.(\underline{2}\ \underline{1}') : (A \to A) \to A.$$

Agora o problema se reduz a calcular a forma η-longa do termo $\underline{1}'$ que tem tipo $A \to A$ no contexto $A \to A \cdot (A \to A) \to A \cdot nil$. De acordo com o exemplo anterior, sabemos que a forma η-longa de $A \to A \cdot (A \to A) \to A \cdot nil \vdash \underline{1}' : A \to A$ é dada por $A \to A \cdot (A \to A) \to A \cdot nil \vdash \lambda_A.(\underline{2}\ \underline{1}) : A \to A$. Portanto temos que a forma η-longa do termo original é dada por:

$$(A \to A) \to A \cdot nil \vdash \lambda_{A\to A}.(\underline{2}\ \lambda_A.(\underline{2}\ \underline{1})) : (A \to A) \to A.$$

Para provar que a noção de forma η-longa está bem formulada, definiremos inicialmente o comprimento de um λ-termo em notação *à la* de Bruijn.

Definição 1.14 *O comprimento de um λ-termo $a \in \Lambda_{dB}(\mathcal{X})$, denotado por $|a|$, é definido indutivamente por:*

- *se $a = \underline{n}$ ou $a = X$ então $|a| = 1$;*
- *se $a = b\ c$ então $|a| = 1 + |b| + |c|$;*

- *se* $a = \lambda_A.b$ *então* $|a| = 1 + |b|$.

Proposição 1.15 *A noção de forma η-longa para λ-termos em notação* à la *de Bruijn está bem definida.*

Prova. A prova é por indução baseada na ordem lexicográfica sobre triplas contendo o número de ocorrências de meta-variáveis, o tamanho do λ-termo considerado e o tamanho do tipo deste λ-termo, respectivamente. A ordem do tipo A, denotado por $|A|$, é definido da forma usual: se A é um tipo atômico então $|A| = 1$ e, se B e C são tipos então $|B \to C| = max(1 + |B|, |C|)$.

Se $a = \lambda_A.b$ então o número de meta-variáveis permanece inalterado, mas o comprimento do termo decresce. Se $a = (\underline{n}\ b_1 \ldots b_q)$ e $q = 0$ então o número de meta-variáveis e o comprimento do termo permanecem inalterados, mas o comprimento do tipo decresce. Se $q \neq 0$ então o número de meta-variáveis permanece inalterado, mas o comprimento do termo decresce. Se $a = (X\ b_1 \ldots b_q)$ então o número de meta-variáveis decresce. □

1.3 O $\lambda\sigma$-cálculo

O λ-cálculo é baseado em uma noção de substituição que pertence à meta-linguagem e isto é necessário porque o processo de substituição utiliza renomeamento para evitar captura de variáveis. Uma solução para este tipo de problema é estender a linguagem considerada incorporando operadores que tornam explícita a operação de substituição. O primeiro mecanismo a fazer isto foi o $\lambda\sigma$-cálculo [ACCL91] que apresentaremos a seguir.

Definição 1.16 *A sintaxe do λσ-cálculo simplesmente tipado é dada por:*

Tipos	A	$::=$	$K \mid A \to A$	*onde* $K \in \mathbb{T}$
Contextos	Γ	$::=$	$nil \mid A \cdot \Gamma$	
Termos	a	$::=$	$\underline{1} \mid X \mid (a\ a) \mid \lambda_A.a \mid a[s]$	*onde* $X \in \mathcal{X}$
Substituições	s	$::=$	$id \mid \uparrow \mid a \cdot s \mid s \circ s$	

O sistema de regras de reescrita do $\lambda\sigma$-cálculo é apresentado na Tabela 1.1.

No $\lambda\sigma$-cálculo, quando uma substituição s é aplicada a um termo a, escrevemos $a[s]$. Substituições são representadas por listas de termos construídas com o operador *cons* (denotado por $\cdot$), um operador para a lista vazia (denotado por *id* que representa

(Beta)	$(\lambda.a)\ b$	$\longrightarrow$	$a[b \cdot id]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
(Abs)	$(\lambda.a)[s]$	$\longrightarrow$	$\lambda.a[\underline{1} \cdot (s \circ \uparrow)]$
(Clos)	$(a[s])[t]$	$\longrightarrow$	$a[s \circ t]$
(VarCons)	$\underline{1}[a \cdot s]$	$\longrightarrow$	a
(Id)	$a[id]$	$\longrightarrow$	a
(Assoc)	$(s \circ t) \circ u$	$\longrightarrow$	$s \circ (t \circ u)$
(Map)	$(a \cdot s) \circ t$	$\longrightarrow$	$a[t] \cdot (s \circ t)$
(IdL)	$id \circ s$	$\longrightarrow$	s
(IdR)	$s \circ id$	$\longrightarrow$	s
(ShiftCons)	$\uparrow \circ (a \cdot s)$	$\longrightarrow$	s
(VarShift)	$\underline{1} \cdot \uparrow$	$\longrightarrow$	id
(SCons)	$\underline{1}[s] \cdot (\uparrow \circ s)$	$\longrightarrow$	s
(Eta)	$\lambda.a\ \underline{1}$	$\longrightarrow$	b if $a =_\sigma b[\uparrow]$

Tabela 1.1: O sistema de reescrita do $\lambda\sigma$-cálculo

a substituição identidade) e o operador $\uparrow$ que representa a substituição simultânea $\underline{1}/\underline{2}, \underline{2}/\underline{3}, \ldots, \underline{n}/\underline{n+1}, \ldots$ que é abreviada por $\underline{2} \cdot \underline{3} \cdot \ldots$. Abreviaremos a composição $\uparrow \circ \ldots \circ \uparrow$ contendo n ocorrências de $\uparrow$ por $\uparrow^n$ e, em particular $\uparrow^0 = id$. Como substituições são representadas por listas de termos, o tipo de uma substituição é dado por uma lista de tipos, isto é, por um contexto. Neste caso, escrevemos $s \triangleright \Gamma$ para denotar que a substituição s tem tipo Γ.

As regras de tipagem para o $\lambda\sigma$-cálculo são dadas por:

$$\text{(var)}\quad \frac{}{A \cdot \Gamma \vdash \underline{1} : A} \qquad \text{(lambda)}\quad \frac{A \cdot \Gamma \vdash a : B}{\Gamma \vdash \lambda_A.a : A \to B}$$

$$\text{(app)}\quad \frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash (a\ b) : B} \qquad \text{(clos)}\quad \frac{\Gamma \vdash s \triangleright \Gamma' \quad \Gamma' \vdash a : A}{\Gamma \vdash a[s] : A}$$

$$\text{(id)}\quad \frac{}{\Gamma \vdash id \triangleright \Gamma} \qquad \text{(shift)}\quad \frac{}{A \cdot \Gamma \vdash \uparrow \triangleright \Gamma}$$

$$\text{(cons)}\quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash s \triangleright \Gamma'}{\Gamma \vdash a \cdot s \triangleright A \cdot \Gamma'} \qquad \text{(comp)}\quad \frac{\Gamma \vdash s'' \triangleright \Gamma'' \quad \Gamma'' \vdash s' \triangleright \Gamma'}{\Gamma \vdash s' \circ s'' \triangleright \Gamma'}$$

Adicionalmente, a cada meta-variável X associamos um único tipo T_X e um único

contexto Γ_X e adicionamos a seguinte regra para meta-variáveis:

$$(\text{meta})\ \frac{}{\Gamma_X \vdash X : T_X}$$

Exemplo 1.17 *Considere a substituição* $\underline{2}\cdot\uparrow$ *que pode ser tipada de acordo com a seguinte derivação:*

$$\dfrac{\dfrac{\dfrac{}{A_1\cdot A_2\cdot nil\vdash\uparrow\triangleright A_2\cdot nil}(shift)\quad\dfrac{}{A_2\cdot nil\vdash\underline{1}:A_2}(var)}{A_1\cdot A_2\cdot nil\vdash\underline{1}[\uparrow]:A_2}(clos)\qquad\dfrac{}{A_1\cdot A_2\cdot nil\vdash\uparrow\triangleright A_2\cdot nil}(shift)}{A_1\cdot A_2\cdot nil\vdash\underline{1}[\uparrow]\cdot\uparrow\triangleright A_2\cdot A_2\cdot nil}(cons)$$

De uma forma geral, um julgamento de tipos da forma:

$$\Gamma\vdash a_1\cdot\ldots\cdot a_p\cdot\uparrow^n\triangleright A_1\cdot\ldots\cdot A_p\cdot\Delta$$

nos diz que o i-ésimo elemento da substituição $a_1\cdot\ldots\cdot a_p\cdot\uparrow^n$ *tem tipo correspondente ao i-ésimo elemento do contexto* $A_1\cdot\ldots\cdot A_p\cdot\Delta$*, onde* Δ *é o tipo da substituição* $\uparrow^n$*.*

Em contraste ao que acontece com a regra (*meta*) no λ-cálculo (em notação *à la* de Bruijn), a regra (*meta*) para o $\lambda\sigma$-cálculo nos mostra que o tipo dos $\lambda\sigma$-termos depende do contexto. Isto é necessário porque unificação no $\lambda\sigma$-cálculo utiliza substituição de primeira ordem, conhecida com *grafting*, que aplicaremos pela esquerda para diferenciar da operação de substituição (de ordem superior) usual. Por exemplo, se a é um termo e σ é um *grafting* então a aplicação de σ ao termo a será denotada por σa. Além disto, precisamos que o *grafting* seja compatível com as regras de tipagem. A restrição imposta pela regra (*meta*) para meta-variáveis evita, por exemplo, que as duas ocorrências distintas da meta-variável X no $\lambda\sigma$-termo $(X\ \lambda_A.X)$ sejam substituídas pelo mesmo $\lambda\sigma$-termo, o que acarretaria capturas de variáveis como em $\{X\mapsto\underline{1}\}(X\ \lambda_A.X) = (\underline{1}\ \lambda_A.\underline{1})$, onde o correto seria $(\underline{1}\ \lambda_A.\underline{2})$. Como veremos posteriormente, para evitar este tipo de problema, λ-termos serão traduzidos para a linguagem do $\lambda\sigma$-cálculo de uma forma especial. Para uma explicação mais detalhada sobre este assunto veja a Observação 2.13.

O $\lambda\sigma$-cálculo codifica o índice *à la* de Bruijn $\underline{n}$ como $\underline{1}[\uparrow^{n-1}]$, mas para facilitar a leitura deste trabalho faremos como em [DHK00] e utilizaremos a notação usual para os índices *à la* de Bruijn.

As formas normais para termos e substituições no $\lambda\sigma$-cálculo são caracterizadas como:

Proposição 1.18 ([Río93]) *Qualquer $\lambda\sigma$-termo em forma* normal *é de uma das seguinte formas:*

1. $\lambda_A.a$, *onde* a *está em forma normal.*
2. $(a\ b_1 \ldots b_q)$, *onde* a *e* b_i *estão em forma normal e* a *é da forma* $\underline{1}$, $\underline{1}[\uparrow^n]$, X *ou* $X[s]$ *onde* s *é uma substituição em forma normal diferente de id.*
3. $a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n$, *onde* $a_1, \ldots, a_p$ *são* $\lambda\sigma$*-termos em forma normal e* $a_p \neq \underline{n}$.

Dizemos que uma substituição em forma $\lambda\sigma$-normal dada por $a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n$ $(n, p \geq 0)$ tem *comprimento* igual a p.

Definição 1.19 (Forma η-longa [DHK00]) *Seja* a *um* $\lambda\sigma$*-termo de tipo* $A_1 \to \ldots \to A_n \to B$ *no contexto* Γ *em forma normal. A forma* η-longa *de* a, *denotada por* a', *é definida por:*

1. *se* $a = \lambda_A.b$ *então* $a' = \lambda_A.b'$;
2. *se* $a = (\underline{k}\ b_1 \ldots b_q)$ *então* $a' = \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{k+n}\ c_1 \ldots c_q\ \underline{n}' \ldots \underline{1}')$, *onde* c_i *é a forma* η*-longa da forma normal de* $b_i[\uparrow^n]$;
3. *se* $a = (X[s]\ b_1 \ldots b_q)$ *então* $a' = \lambda_{A_1} \ldots \lambda_{A_n}.(X[s']\ c_1 \ldots c_q\ \underline{n}' \ldots \underline{1})$, *onde* c_i *é a forma* η*-longa da forma normal de* $b_i[\uparrow^n]$ *e se* $s = d_1 \cdot \ldots \cdot d_r \cdot \uparrow^k$ *então* $s' = e_1 \cdot \ldots \cdot e_r \cdot \uparrow^{k+n}$ *onde* e_i *é a forma* η*-longa de* $d_i[\uparrow^n]$.

Observação 1.20 *A prova de que a noção apresentada em 1.19 está bem definida pode ser encontrada em [DHK00]. Adicionalmente, devemos observar que "no* $\lambda\sigma$*-cálculo, a redução de um* η*-rédice pode criar um* σ*-rédice. Por exemplo, o* $\lambda\sigma$*-termo*

$$X[(\lambda_A.\underline{2}\ \underline{1}) \cdot \uparrow]$$

pode ser reduzido para $X[\underline{1} \cdot \uparrow]$ *e depois para* $X[id]$ *e, finalmente para* X. *Assim, para computarmos uma forma* η*-longa precisamos inicialmente reduzir todos os rédices (inclusive os* η*-rédices) para então expandir o termo".*

1.4 O λs_e-cálculo

O λs_e-cálculo foi desenvolvido por Kamaraddine e Ríos [KR97] e, diferentemente do $\lambda\sigma$, suas expressões são de apenas um tipo, denominado **termos**. Isto faz com que o λs_e mantenha uma estrutura sintática mais próxima do λ-cálculo. Os termos do cálculo λs_e são definidos por:

$$\text{Termos} \quad \text{M, N} \quad ::= \quad \underline{n} \mid \lambda_A.M \mid (M\ N) \mid M\ \sigma^i\ N \mid \varphi_k^i\ M, \text{ para } k \geq 0 \text{ e } i \geq 1$$

O λs_e-cálculo utiliza aritmética e dois operadores que manipulam explicitamente a operação de substituição. O operador σ é responsável pelas substituições propriamente ditas, enquanto que o operador φ faz a atualização de índices. O conjunto de regras do λs_e-cálculo é dado na Figura 1.1. Note que em contraste com o $\lambda\sigma$-cálculo, o λs_e-cálculo possui um sistema de regras infinito.

As regras de tipagem de termos são dadas por:

$$\text{(var)} \quad \frac{}{A.\Gamma \vdash \underline{1} : A} \qquad \text{(varn)} \quad \frac{\Gamma \vdash \underline{n} : B}{A.\Gamma \vdash \underline{n+1} : B}$$

$$\text{(app)} \quad \frac{\Gamma \vdash a : A \to B \ \ \Gamma \vdash b : A}{\Gamma \vdash \ (a\ b) : B} \qquad \text{(lambda)} \quad \frac{A \cdot \Gamma \vdash a : B}{\Gamma \vdash \lambda_A.a : A \to B}$$

$$\text{(sigma)} \quad \frac{\Gamma_{\geq i} \vdash b : B \ \ \Gamma_{<i}.B \cdot \Gamma_{\geq i} \vdash a : A}{\Gamma \vdash a\sigma^i b : A} \qquad \text{(phi)} \quad \frac{\Gamma_{\leq k} \cdot \Gamma_{\geq k+i} \vdash a : A}{\Gamma \vdash \varphi_k^i a : A}$$

onde $\Gamma_{>i}$ representa o contexto obtido de Γ removendo seus primeiros i elementos e, $\Gamma_{<i}$ representa o contexto obtido de Γ considerando apenas seus primeiros $i-1$ elementos. $\Gamma_{\leq i}$ e $\Gamma_{\geq i}$ são definidos de forma óbvia. Adicionalmente, escrevemos $\Gamma_{[i]}$ para denotar o i-ésimo elemento do contexto Γ.

Para cada meta-variável X, associamos um único tipo T_X e um único contexto Γ_X. Assumimos que para cada tipo existe um número infinito de meta-variáveis distintas com aquele tipo. Adicionamos a seguinte regra de tipagem para meta-variáveis:

$$\text{(meta)} \quad \frac{}{\Gamma_X \vdash X : T_X}$$

Definição 1.21 (Formas λs_e-normais [ARK03]) *Seja a um termo do λs_e-cálculo. Então a está em forma λs_e-*normal *se:*

1. $a \in \mathcal{X} \cup \mathbb{N}$

(σ-*generation*)	$(\lambda M\ N) \rightarrow M\ \sigma^1\ N$
(σ-λ-*transition*)	$(\lambda M)\ \sigma^i\ N \rightarrow \lambda(M\ \sigma^{i+1}\ N)$
(σ-*app-transition*)	$(M_1\ M_2)\ \sigma^i\ N \rightarrow ((M_1\ \sigma^i\ N)(M_2\ \sigma^i\ N))$
(σ-*destruction*)	$\underline{n}\sigma^i N \rightarrow \begin{cases} \underline{n-1} & \text{se } n > i \\ \varphi_0^i N & \text{se } n = i \\ \underline{n} & \text{se } n < i \end{cases}$
(φ-λ-*transition*)	$\varphi_k^i(\lambda M) \rightarrow \lambda(\varphi_{k+1}^i M)$
(φ-*app-transition*)	$\varphi_k^i(M_1\ M_2) \rightarrow ((\varphi_k^i M_1)\ (\varphi_k^i M_2))$
(φ-*destruction*)	$\varphi_k^i \underline{n} \rightarrow \begin{cases} \underline{n+i-1} & \text{se } n > k \\ \underline{n} & \text{se } n \leq k \end{cases}$
(σ-σ-*transition*)	$(M_1\ \sigma^i\ M_2)\sigma^j N \rightarrow (M_1\ \sigma^{j+1}\ N)\sigma^i(M_2\ \sigma^{j-i+1}\ N)$ se $i \leq j$
(σ-φ-*transition*1)	$(\varphi_k^i M)\sigma^j N \rightarrow \varphi_k^{i-1} M$ se $k < j < k+i$
(σ-φ-*transition*2)	$(\varphi_k^i M)\sigma^j N \rightarrow \varphi_k^i(M\sigma^{j-i+1} N)$ se $k+i \leq j$
(φ-σ-*transition*)	$\varphi_k^i(M\sigma^j N) \rightarrow (\varphi_{k+1}^i M)\sigma^j(\varphi_{k+1-j}^i N)$ se $j \leq k+1$
(φ-φ-*transition*1)	$\varphi_k^i(\varphi_l^j M) \rightarrow \varphi_l^j(\varphi_{k+1-j}^i M)$ se $l+j \leq k$
(φ-φ-*transition*2)	$\varphi_k^i(\varphi_l^j M) \rightarrow \varphi_l^{j+i-1} M$ se $l \leq k < l+j$
(Eta_{s_e})	$\lambda(M\ \underline{1}) \rightarrow N$ se $M =_{S_e} \varphi_0^2 N$

Figura 1.1: Regras para manipulação de termos no λs_e

2. *$a = (b\ c)$, onde b e c são formas λs_e-normais e b não é uma abstração da forma $\lambda.d$*

3. *$a = \lambda.b$, onde b é uma forma λs_e-normal excluindo aplicações da forma $(c\ \underline{1})$ tal que exista um d com $\varphi_0^2 d =_{s_e} c$*

4. *$a = b\sigma^j c$, onde c é uma forma λs_e-normal e b é uma forma λs_e-normal de uma das seguintes formas:*
   a) X b)$d\sigma^i e$, com $j < i$ ou c)$\varphi_k^i d$, com $j \leq k$.

5. $a = \varphi_k^i b$, *onde* b *é uma forma* λs_e*-normal de uma das seguintes formas:*
   a) X *b)* $c\sigma^j d$, *com* $j > k+1$ *ou* *c)* $\varphi_l^j c$, *com* $k < l$.

Definição 1.22 (Formas η-longas [ARK03]) *Sejam* Γ *um contexto e* a *um termo em forma* λs_e*-normal de tipo* $A_1 \to \ldots A_n \to B$ *no contexto* Γ. *A forma* η-longa *de* a, *denotada por* a', *é indutivamente definida por:*

- *se* $a = \lambda_A.b$ *então* $a' = \lambda_A.b'$
- *se* $a = (b_1 \ldots b_p)$ *então* $a' = \lambda_{A_1} \ldots \lambda_{A_n}.(c_1 \ldots c_p\ \underline{n}' \ldots \underline{1}')$, *onde* c_i *é a forma* η*-longa da forma* λs_e*-normal de* $\varphi_0^{n+1} b_i$.
- *se* $a = b\sigma^i c$ *então* $a' = \lambda_{A_1} \ldots \lambda_{A_n}.(d'\sigma^{i+n} e' \underline{n}' \ldots \underline{1}')$, *onde* d' *e* e' *correspondem as forma* η*-longas da forma* λs_e*-normal de* $\varphi_0^{n+1} b$ *e* $\varphi_0^{n+1} c$, *respectivamente.*
- *se* $a = \varphi_k^i b$ *então* $a' = \lambda_{A_1} \ldots \lambda_{A_n}.(\varphi_k^i c' \underline{n}' \ldots \underline{1}')$, *onde* c' *é a forma* η*-longa da forma* λs_e*-normal de* $\varphi_0^{n+1} b$.

Capítulo 2

Unificação de Ordem Superior

Unificação de Ordem Superior (HOU) é um problema em geral indecidível [Gol81] que aparece frequentemente em Ciência da Computação e Matemática. Há aproximadamente trinta anos, G. Huet [Hue75, Hue02] desenvolveu um algoritmo de semi-decisão para HOU, conhecido como o *algoritmo de Huet*. Este algoritmo sempre encontra soluções para problemas unificáveis mas pode entrar em *loop* caso o problema não tenha solução. Devido ao alto poder de expressividade de linguagens de ordem superior, diversos sistemas computacionais baseados no λ-cálculo, necessitam implementar mecanismos de HOU [NM88, NPW02, PS99, CH85]. Neste capítulo apresentaremos o algoritmo de Huet para o λ-cálculo com nomes e uma adaptação deste algoritmo para o λ-cálculo em notação *à la* de Bruijn. Adicionalmente, introduzimos a noção de árvores de unificação que será utilizada no restante deste trabalho.

2.1 O Algoritmo de Huet para o λ-cálculo com Nomes

Nesta seção apresentaremos o algoritmo de Huet para o λ-cálculo simplesmente tipado com nomes [Hue75]. Iniciaremos com algumas definições:

Definição 2.1 (Estrutura das formas normais) *Todo λ-termo bem tipado a em forma β-normal tem a forma:*

$$\lambda_{x_1:A_1} \ldots \lambda_{x_n:A_n}.(h\ e_1 \ldots e_p)$$

onde

- $n, p \geq 0$;
- h *é uma constante, uma variável ligada ou uma meta-variável, chamada de* cabeça *do termo;*
- $e_1, \ldots, e_p$ *são* λ*-termos em forma* β*-normal chamados de* argumentos *de* h;
- *os abstratores* $\lambda_{x_1:A_1}, \ldots, \lambda_{x_n:A_n}$ *são chamados de* abstratores externos *de* a;
- *o termo* $\lambda_{x_1:A_1} \ldots \lambda_{x_n:A_n}.h$ *é o* cabeçalho *de* a.

Definição 2.2 *Um* λ*-termo em forma* β*-normal é dito ser* rígido *se sua cabeça é uma constante ou uma variável ligada. Caso contrário, isto é, se sua cabeça é uma meta-variável, o termo é dito* flexível.

Definição 2.3 (Forma η-longa [Hin97]) *Um* λ*-termo a bem tipado e em forma* β*-normal está em forma* η-longa *se cada ocorrência de variável em a é seguida pela mais longa sequência de argumentos que seu tipo permite, isto é, se cada componente da forma* $(u\ e_1 \ldots e_p)$ *com* $p \geq 0$ *que não está em posição funcional possui tipo atômico.*

De agora em diante, assumiremos que os λ-termos estão em forma η-longa.

Definição 2.4 (Problema de Unificação) *Um* problema de unificação *no* λ*-cálculo simplesmente tipado é dado por uma conjunção de equações da forma* $a =^? b$, *onde a e b são* λ*-termos em forma* η*-longa com mesmo tipo e, onde todas as equações do problema estão tipadas sob o mesmo contexto. Uma tal equação é chamada* rígida-rígida *(resp.* flexível-flexível*) se ambos os termos a e b são rígidos (resp. flexíveis) e,* flexível-rígida *se a é flexível e b é rígido ou vice-versa. Uma equação da forma* $a =^? a$ *é chamada* trivial.

A exigência para que todas as equações em um problema de unificação estejam tipadas sob o mesmo contexto surge da necessidade de que as designações de tipos para constantes coincidam quando estas ocorrem em mais de uma equação, como ilustrado no seguinte exemplo:

Exemplo 2.5 *Seja* $\Gamma = \{x : A, f : A \to A\}$. *Considere o seguinte problema de unificação:*

$$X(f\ x) =^? f\ x \wedge X(f\ x) =^? f(X\ x)$$

Note que os julgamentos de tipos $\Gamma \vdash x : A$, $\Gamma \vdash f : A \to A$ *e* $\Gamma \vdash X : A \to A$ *são deriváveis e, consequentemente todos os termos envolvidos nas equações deste problema têm tipo* A *no contexto* Γ.

O algoritmo de Huet baseia-se em dois procedimentos conhecidos como SIMPL e MATCH que apresentamos a seguir:

O Procedimento SIMPL

Este é utilizado para "simplificar" problemas de unificação que contenham equações da forma rígida-rígida. Intuitivamente, se Γ é um contexto e P é um problema de unificação bem tipado no contexto Γ contendo uma equação rígida-rígida da forma:

$$\lambda_{x_1:A_1} \ldots \lambda_{x_n:A_n}.(h\ e_1^1 \ldots e_p^1) =^? \lambda_{y_1:A_1} \ldots \lambda_{y_n:A_n}.(h\ e_1^2 \ldots e_p^2) \tag{2.1}$$

onde $n, p \geq 0$ e h é uma variável ligada ou uma constante, então qualquer substituição σ que seja solução desta equação não afeta o cabeçalho dos termos da mesma. Além disto, σ precisa igualar argumentos correspondentes, isto é, σ é tal que:

$$\lambda_{x_1:A_1} \ldots \lambda_{x_n:A_n}.(h\ e_1^1\sigma \ldots e_p^1\sigma) =_{\beta\eta} \lambda_{y_1:A_1} \ldots \lambda_{y_n:A_n}.(h\ e_1^2\sigma \ldots e_p^2\sigma) \tag{2.2}$$

Desta forma, o problema de resolver a equação (2.1) pode ser reduzido ao problema de resolver a seguinte conjunção de (sub)equações:

$$\begin{gathered} \lambda_{x_1:A_1} \ldots \lambda_{x_n:A_n}.e_1^1 =^? \lambda_{y_1:A_1} \ldots \lambda_{y_n:A_n}.e_1^2 \\ \wedge \ldots \wedge \\ \lambda_{x_1:A_1} \ldots \lambda_{x_n:A_n}.e_p^1 =^? \lambda_{y_1:A_1} \ldots \lambda_{y_n:A_n}.e_p^2 \end{gathered} \tag{2.3}$$

que são bem tipadas no contexto Γ.

O procedimento SIMPL substitui equações da forma (2.1) por conjunções da forma (2.3). Quando o problema P contém uma equação rígida-rígida cujas cabeças não correspondem ao mesmo elemento, o procedimento pára dizendo que o problema não é unificável. Este processo é repetido até que o problema atual não contenha mais nenhuma equação da forma rígida-rígida, e o problema resultante é dito estar em forma *simplificada*.

Exemplo 2.6 *Seja* $\Gamma = \{w : A, u : A \to B, v : A \to A\}$ *um contexto,* X *uma meta-variável de tipo* $A \to B$ *e considere o seguinte problema de unificação:*

$$\lambda_{y:B\to B}.(y\ (X\ w)) =^{?} \lambda_{x:B\to B}.(x\ (u\ (v\ w)))$$

bem tipada no contexto Γ*. Uma aplicação de SIMPL a este problema gerará o seguinte problema de unificação simplificado:*

$$\lambda_{y:B\to B}.(X\ w) =^{?} \lambda_{x:B\to B}.(u\ (v\ w))$$

bem tipado no contexto Γ*.*

O Procedimento MATCH

Para cada problema de unificação simplificado que contenha pelo menos uma equação flexível-rígida, o algoritmo de Huet chama o procedimento MATCH. Este procedimento recebe como argumento uma equação flexível-rígida e retorna um conjunto finito Σ de substituições para a cabeça do termo flexível da equação dada. As equações geradas pelo procedimento MATCH são baseadas em duas regras chamadas de *imitação* e *projeção*.

1. A Regra de Imitação

Seja Γ um contexto e considere a seguinte equação flexível-rígida:

$$\lambda_{x_1:A_1} \ldots \lambda_{x_n:A_n}.(X\ e_1^1 \ldots e_{p_1}^1) =^{?} \lambda_{y_1:A_1} \ldots \lambda_{y_n:A_n}.(h\ e_1^2 \ldots e_{p_2}^2) \tag{2.4}$$

bem tipada no contexto Γ, onde

- $n, p_1, p_2 \geq 0$;
- X é uma meta-variável de tipo $B_1 \to \ldots \to B_{p_1} \to A$ com A atômico;
- h é uma variável ligada ou uma constante de tipo $C_1 \to \ldots \to C_{p_2} \to A$ com A atômico;
- se $p_1 \neq 0$ então e_i^1 é um λ-termo em forma η-longa de tipo B_i para todo $1 \leq i \leq p_1$;
- se $p_2 \neq 0$ então e_j^2 é um λ-termo em forma η-longa de tipo C_j para todo $1 \leq j \leq p_2$.

Uma substituição de imitação consiste em substituir a cabeça do termo flexível por um outro termo cuja cabeça coincida com a cabeça do termo rígido. Desta forma, para uma dada equação flexível-rígida, uma substituição de imitação só é possível se a cabeça do termo rígido for uma constante. Esta restrição se deve ao fato de que captura de variáveis é proibida durante as substituições já que isto, em geral, muda a semântica dos termos. No caso da equação (2.4), temos que se h é uma constante então a substituição de imitação gerada é dada por:

$$X/\lambda_{z_1:B_1} \ldots \lambda_{z_{p_1}:B_{p_1}}.(h\ (H_1\ z_1 \ldots z_{p_1}) \ldots (H_{p_2}\ z_1 \ldots z_{p_1})) \tag{2.5}$$

onde H_i é uma meta-variável nova de tipo $B_1 \to \ldots \to B_{p_1} \to C_i$ para cada $1 \leq i \leq p_2$ (se $p_2 > 0$, caso contrário nenhuma meta-variável nova é introduzida).

Exemplo 2.7 *Considere novamente a equação flexível-rígida gerada no Exemplo 2.6. Uma substituição de imitação é possível porque a cabeça u do termo rígido é uma constante. Esta substituição é dada por:*

$$X/\lambda_{z:A}.(u\ (H_1\ z))$$

onde H_1 é uma meta-variável nova de tipo $A \to A$. Veja que a substituição acima não é uma solução do problema original. De fato, esta substituição constitui apenas parte de uma possível solução. Neste sentido dizemos que as soluções são incrementalmente geradas pelas substituições, no sentido de que cada aplicação de MATCH determina apenas parte da solução do problema original que é obtida pela composição das substituições geradas ao longo de um caminho de sucesso (veja Fig. 2.1).

2. A Regra de Projeção

Considere novamente a estrutura da equação (2.4). Neste caso a meta-variável X possui p_1 argumentos, a saber $e^1_1, \ldots e^1_{p_1}$. Uma substitutição de projeção substitui X por algum de seus argumentos e, neste sentido X é "projetado" sobre um de seus argumentos. Uma condição suficiente para que uma substituição de projeção seja gerada é que o tipo alvo de X coincida com o tipo alvo do argumento a ser projetado. Assim, h pode ser tanto uma variável ligada quanto uma constante.

Temos que o tipo de X é $B_1 \to \ldots \to B_{p_1} \to A$ (A atômico). Agora suponha, a título de ilustração que o i-ésimo argumento de X ($1 \leq i \leq p_1$) tenha A como tipo alvo, isto é, estamos supondo que o tipo B_i é da forma $D_1 \to \ldots \to D_q \to A$, onde

$q \geq 0$. Neste caso, a substituição de projeção é dada por:

$$X/\lambda_{z_1:B_1} \ldots \lambda_{z_{p_1}:B_{p_1}}.(z_i\ (H_1\ z_1 \ldots z_{p_1}) \ldots (H_q\ z_1 \ldots z_{p_1}))$$

onde H_i é uma meta-variável nova de tipo $B_1 \rightarrow \ldots \rightarrow B_{p_1} \rightarrow C_j$ para cada $j = 1, \ldots, q$.

Como pudemos observar, para cada argumento de X que possui tipo alvo coincidente com o tipo alvo de X uma substituição de projeção é gerada. Isto nos dá um máximo de p_1 projeções possíveis, enquanto que, no máximo, uma imitação é gerada. No entanto, pode acontecer que nenhuma substituição seja gerada após uma aplicação de MATCH e, neste caso o algoritmo pára e responde que o problema dado não é unificável.

Exemplo 2.8 *Considere novamente a equação flexível-rígida gerada no Exemplo 2.6. Neste caso, nenhuma projeção é possível porque o tipo alvo de X é B enquanto que o tipo alvo do único argumento de X, a saber w, é A. Note que, como uma substituição de imitação foi gerada não podemos concluir ainda se o problema original é unificável ou não. De fato, a Figura 2.1 nos mostra que este problema possui duas soluções distintas.*

O algoritmo de Huet consiste de um procedimento principal que coordena as chamadas aos procedimentos SIMPL e MATCH. Este procedimento principal inicia com o problema original e, se o mesmo contém alguma equação rígida-rígida então o procedimento SIMPL é chamado. Se o resultado de SIMPL não é um terminal (de sucesso ou falha) então o procedimento principal busca por uma equação flexível-rígida. Se uma tal equação existe então o procedimento MATCH é chamado; caso contrário, o algoritmo pára e responde que o problema tem solução. Toda a execução do algoritmo de Huet para um problema dado pode ser visto por meio de uma estrutura de árvore. A apresentação original de Huet utiliza o que ele chama de *árvore de matching*. Formalmente, uma árvore de *matching* consiste em uma árvore cujos nós contém problemas simplificados, que em particular podem ser terminais de sucesso ou falha e, cujos arcos são marcados com substituições geradas por aplicações de MATCH. O exemplo a seguir nos mostra a árvore de *matching* para o problema de unificação apresentado no Exemplo 2.6.

Exemplo 2.9 *A árvore de* matching *de problema apresentado no Exemplo 2.6 é dada na Figura 2.1. A raiz desta árvore contém o problema original simplificado,*

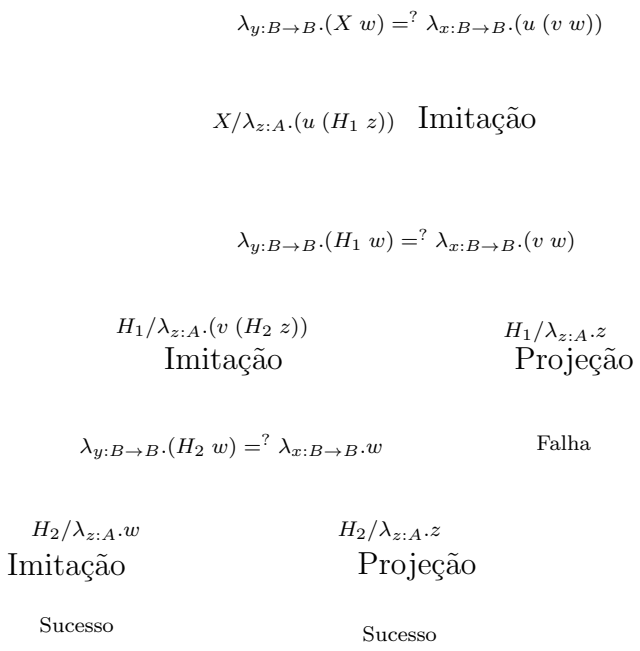


Figura 2.1: Árvore de *Matching* de Huet

isto é, o problema original após uma aplicação de SIMPL. O único arco partindo da raiz corresponde à substituição de imitação que é gerada por uma aplicação de MATCH. O nó em seguida contém o problema resultante já simplificado. Uma aplicação de MATCH a este novo problema gera duas substituições: uma imitação que vai dar origem a dois ramos de sucesso e, uma projeção que vai dar origem a um ramo de falha. As soluções do problema original são computadas pela composição das substituições obtidas ao longo de um ramo de sucesso. Para este exemplo, as soluções obtidas são $X/\lambda_{z:A}.(u\ (v\ w))$ *e* $X/\lambda_{z:A}.(u\ (v\ z))$.

Exemplo 2.10 (Continuação do Exemplo 2.5) *É fácil observar que a única solução do problema de unificação* $(X(f\ x)) =^? (f\ x) \wedge (X(f\ x)) =^? (f(X\ x))$ *é a função identidade:* $X/\lambda_{z:A}.z$. *No entanto, as soluções para a segunda equação incluem a função identidade e todas as composições de* f *dadas por:*

$$X/\lambda_{z:A}.(f\ z), X/\lambda_{z:A}.(f(f\ z)), X/\lambda_{z:A}.(f(f(f\ z))), \ldots$$

2.2 Árvores de Unificação

Nesta seção introduziremos uma nova notação denominada *árvores de unificação*. A idéia das árvores de unificação surgiu a partir da necessidade de se ter uma estrutura

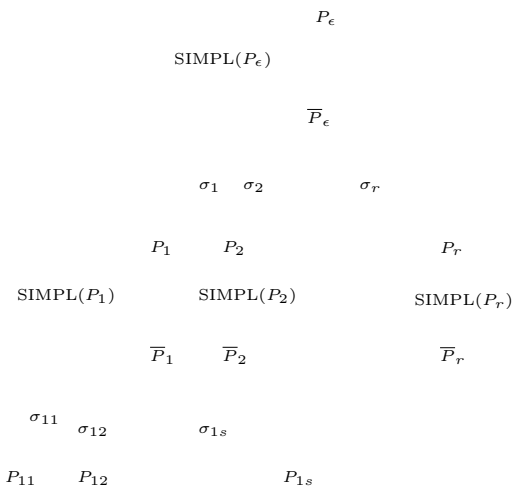


Figura 2.2: Árvore de Unificação

mais rica do que as árvores de *matching* para ilustrar melhor o funcionamento do algoritmo de Huet. Assim, gostaríamos de ter uma estrutura que, ao contrário das árvores de *matching*, não escondesse os passos de simplificação. Informalmente, uma árvore de unificação é obtida a partir da árvore de *matching* adicionando novos arcos que correspondem aos passos de simplificação e marcas para subproblemas e substituições. Estas marcas nos fornecem informações sobre as posições dos subproblemas e substituições (veja Figura 2.2).

A seguir fornecemos a descrição formal de como construir uma árvore de unificação para um dado problema P.

A árvore de unificação de um dado problema P, denotada por $\mathcal{A}(P)$, é construída da seguinte forma:

1. Marque P com subíndice ϵ (a posição vazia), isto é, P_ϵ. Este subíndice significa que o problema está na raiz da árvore de unificação.

2. Para um nó marcado com P_α, denotaremos por $\overline{P}_\alpha$ o filho de P_α obtido por uma aplicação de SIMPL. Neste caso, arco que liga P_α a $\overline{P}_\alpha$ é representado por uma linha curva.

3. Para um nó marcado com $\overline{P}_\alpha$ e contendo uma equação flexível-rígida eq, sejam $\sigma_{\alpha 1}, \ldots, \sigma_{\alpha k}$ $(k > 0)$ as substituições geradas após uma aplicação de MATCH à equação eq. Os descendentes de $\overline{P}_\alpha$, denotados por $P_{\alpha 1}, \ldots, P_{\alpha k}$, são definidos por $P_{\alpha i} := \overline{P}_\alpha \sigma_{\alpha i}$, para $1 \leq i \leq k$.

A partir de uma árvore de unificação fica fácil organizar e referenciar as substituições e os subproblemas que as geraram. Por exemplo, uma substituição com marca σ_{12315} só pode ter sido obtida por uma aplicação de MATCH ao problema com marca $\overline{P}_{1231}$. As soluções dos problemas de unificação também podem ser facilmente computadas via composição das substituições geradas ao longo de um ramo de sucesso. Por exemplo, se P_{1223} é um nó de sucesso, mas P_{122} não é, então a solução correspondente a este nó é obtida pela composição $\sigma_1\sigma_{12}\sigma_{122}\sigma_{1223}$.

2.3 Uma Adaptação do Algoritmo de Huet para a Notação *à la* de Bruijn

Nesta seção apresentaremos o algoritmo de Huet em notação *à la* de Bruijn. As definições de formas normais, termos rígidos, termos flexíveis e problemas de unificação em notação *à la* de Bruijn são obtidas de forma imediata a partir das definições correspondentes referentes ao λ-cálculo com nomes dadas na Seção 2.1.

Na próxima subseção apresentaremos os procedimentos SIMPL e MATCH do algoritmo de Huet em notação *à la* de Bruijn utilizando árvores de unificação.

O Procedimento SIMPL

O procedimento SIMPL recebe como argumento um problema de unificação P_α contendo pelo menos uma equação rígida-rígida (caso contrário o problema já estaria simplificado) e retorna um problema de unificação equivalente, denotado por $\overline{P}_\alpha$, e que pode ser tanto um terminal (Sucesso ou Falha) quanto um problema contendo pelo menos uma equação flexível-rígida. A seguir fornecemos uma descrição algoritmica de SIMPL:

O Procedimento SIMPL

`Entrada`: Um problema de unificação P_α contendo pelo menos uma equação rígida-rígida.

`Saída`: O problema de unificação $\overline{P}_\alpha$.

`Enquanto` existir uma equação rígida-rígida em P_α, digamos:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(h_1\ e_1^1 \ldots e_{p_1}^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(h_2\ e_1^2 \ldots e_{p_2}^2) \wedge P' \tag{2.6}$$

onde $n, p_1, p_2 \geq 0$ e h_1 e h_2 são índices *à la* de Bruijn `faça`

> `Se` h_1 e h_2 são índices diferentes `então` pare e retorne o terminal Falha `senão` substitua a equação (2.6), onde agora $p_1 = p_2$ porque os termos têm o mesmo tipo, pela conjunção:
>
> $$\lambda_{A_1} \ldots \lambda_{A_n}.e_1^1 =^? \lambda_{A_1} \ldots \lambda_{A_n}.e_1^2 \wedge \ldots \wedge \lambda_{A_1} \ldots \lambda_{A_n}.e_{p_1}^1 =^? \lambda_{A_1} \ldots \lambda_{A_n}.e_{p_1}^2$$
>
> em P_α e denote o problema resultante por $\overline{P}_\alpha$.

`Se` $\overline{P}_\alpha$ contém alguma equação flexível-rígida `então` retorne $\overline{P}_\alpha$, `senão` pare e retorne um terminal de sucesso.

O Procedimento MATCH

O procedimento MATCH recebe como argumento uma equação flexível-rígida e retorna um conjunto finito de substituições. Como já vimos anteriormente, este procedimento é baseado em duas regras: a imitação e a projeção que descrevemos a seguir:

A Regra de Imitação: Na Seção 2.1 apresentamos a regra de imitação para o cálculo-λ com nomes. Para apresentarmos a regra de imitação em notação *à la* de Bruijn, considere um contexto Γ e uma equação flexível-rígida bem tipada em Γ e dada por:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(X\ e_1^1 \ldots e_{p_1}^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e_1^2 \ldots e_{p_2}^2) \tag{2.7}$$

onde:

- $n, p_1, p_2 \geq 0$;
- X é uma meta-variável de tipo $B_1 \to \ldots \to B_{p_1} \to A$ (A atômico);
- $\underline{h}$ é um índice *à la* de Bruijn de tipo $C_1 \to \ldots \to C_{p_2} \to A$ (A atômico);
- se $p_1 \neq 0$ então e_i^1 é um λ-termo em forma η-longa de tipo B_i para todo $1 \leq i \leq p_1$;
- se $p_2 \neq 0$ então e_j^2 é um λ-termo em forma η-longa de tipo C_j para todo $1 \leq j \leq p_2$.

Para evitar captura de variáveis, uma substituição de imitação só é possível quando $\underline{h}$ representa uma constante, isto é, quando $h > n$ e, neste caso a substituição

é dada por:

$$X/\lambda_{B_1} \ldots \lambda_{B_{p_1}}.(\underline{p_1 + h - n}\ (X_1\ \underline{p_1} \ldots \underline{1}) \ldots (X_{p_2}\ \underline{p_1} \ldots \underline{1}))$$

onde X_i é uma meta-variável nova de tipo $B_1 \to \ldots \to B_{p_1} \to C_i$ no contexto Γ, para todo $1 \leq i \leq p_2$.

A Regra de Projeção:

Quando a cabeça do termo rígido na equação (2.7) é uma constante ou uma variável ligada então uma projeção é possível sempre que o tipo alvo de X (cabeça do termo flexível) coincida com o tipo alvo do argumento sobre o qual se deseja projetar X. Por exemplo, se o i-ésimo argumento e_i^1 de X tem tipo da forma $D_1 \to \ldots \to D_q \to A$ com $q \geq 0$, então a seguinte projeção é gerada:

$$X/\lambda_{B_1} \ldots \lambda_{B_{p_1}}.(\underline{p_1 - i + 1}\ (H_1\ \underline{p_1} \ldots \underline{1}) \ldots (H_q\ \underline{p_1} \ldots \underline{1}))$$

onde cada H_j é uma meta-variável nova com tipo $B_1 \to \ldots \to B_{p_1} \to D_j$, para todo $1 \leq j \leq q$.

O procedimento MATCH pode então ser formalizado da seguinte forma:

O procedimento MATCH

`Entrada`: Uma equação flexível rígida, digamos eq.

`Saída`: Um conjunto finito de substituições para a cabeça do termo flexível de eq.

1. Aplique não-deterministicamente as regras de imitação e projeção à equação eq e denote por Σ o conjunto de substituições resultante.

O Procedimento Principal

O procedimento principal do algoritmo de Huet é quem faz as chamadas dos procedimentos SIMPL e MATCH. Formalmente, podemos apresentar este procedimento principal da seguinte forma:

O Procedimento Principal

`Entrada`: Um problema de unificação P_ϵ.

`Saída`: A árvore de unificação do problema P_ϵ.

1. Se $P_{i_1 \ldots i_k}$ contém uma equação rígida-rígida então chame SIMPL e vá para o próximo passo, caso contrário, se $P_{i_1 \ldots i_k}$ contém uma equação flexível-rígida então renomeie $P_{i_1 \ldots i_k}$ para $\overline{P}_{i_1 \ldots i_k}$ e vá para o próximo passo, caso contrário vá para o passo 4.

2. Seja eq uma equação flexível rígida em $\overline{P}_{i_1 \ldots i_k}$. Aplique MATCH para eq, denote por $\Sigma_{i_1 \ldots i_k}$ o conjunto de substituições gerado e vá para o passo 3.

3. Se $\Sigma_{i_1 \ldots i_k}$ é um conjunto vazio então pare e retorne um terminal de falha, caso contrário seja $\Sigma_{i_1 \ldots i_k} = \{\sigma_{i_1 \ldots i_k 1}, \ldots, \sigma_{i_1 \ldots i_k r}\}$ onde $r > 0$ e, **para cada** substituição $\sigma_{i_1 \ldots i_k j} \in \Sigma_{i_1 \ldots i_k}$ denote $P_{i_1 \ldots i_k j} := P_{i_1 \ldots i_k}\sigma_{i_1 \ldots i_k j}$ o novo problema de unificação e vá para o passo 1.

4. Pare e retorne um terminal de sucesso. A solução correspondente a este terminal de sucesso assumindo que a posição atual é $i_1 \ldots i_k$ é dada pela seguinte composição:
   $\sigma_{i_1}\sigma_{i_1 i_2} \ldots \sigma_{i_1 i_2 \ldots i_{k-1}}\sigma_{i_1 i_2 \ldots i_k}$

Exemplo 2.11 *Considere o problema de unificação apresentado no Exemplo 2.6. Iniciaremos convertendo-o para a notação de de Bruijn. Para isto consideraremos o seguinte referencial:* $u : A \to B, w : A, v : A \to A$. *Este referencial corresponde a considerar o seguinte contexto:* $\Gamma = A \to B \cdot A \cdot A \to A \cdot nil$. *Conforme explicamos na Seção 1.2, considerando a equação:*

$$\lambda_{y:B\to B}.(y\ (X\ w)) =^{?} \lambda_{x:B\to B}.(x\ (u\ (v\ w)))$$

no escopo dos abstratores $\lambda_{v:A\to A}\lambda_{w:A}\lambda_{u:A\to B}$ *obtemos a seguinte equação:*

$$\lambda_{B\to B}.(\underline{1}(X\ \underline{3})) =^{?} \lambda_{B\to B}.(\underline{1}(\underline{2}(\underline{4}\ \underline{3})))$$

bem tipada no contexto Γ.

A Figura 2.3 nos fornece a árvore de unificação gerada para este problema. As soluções para o problema original são dadas pela composição das substituições ao longo de um caminho cuja folha é um terminal de sucesso. Note que depois da composição, os termos precisam ser normalizados. Por exemplo, a composição $\sigma_1\sigma_{11}\sigma_{111}$ *é computada inicialmente pela composição usual de substituições:*

$$\{X/\lambda_A.(\underline{2}((\lambda_A.\underline{5}((\lambda_A.\underline{1})\ \underline{1}))\ \underline{1})), X_1/\lambda_A.(\underline{4}((\lambda_A.\underline{1})\ \underline{1})), X_2/\lambda_A.\underline{1}\}$$

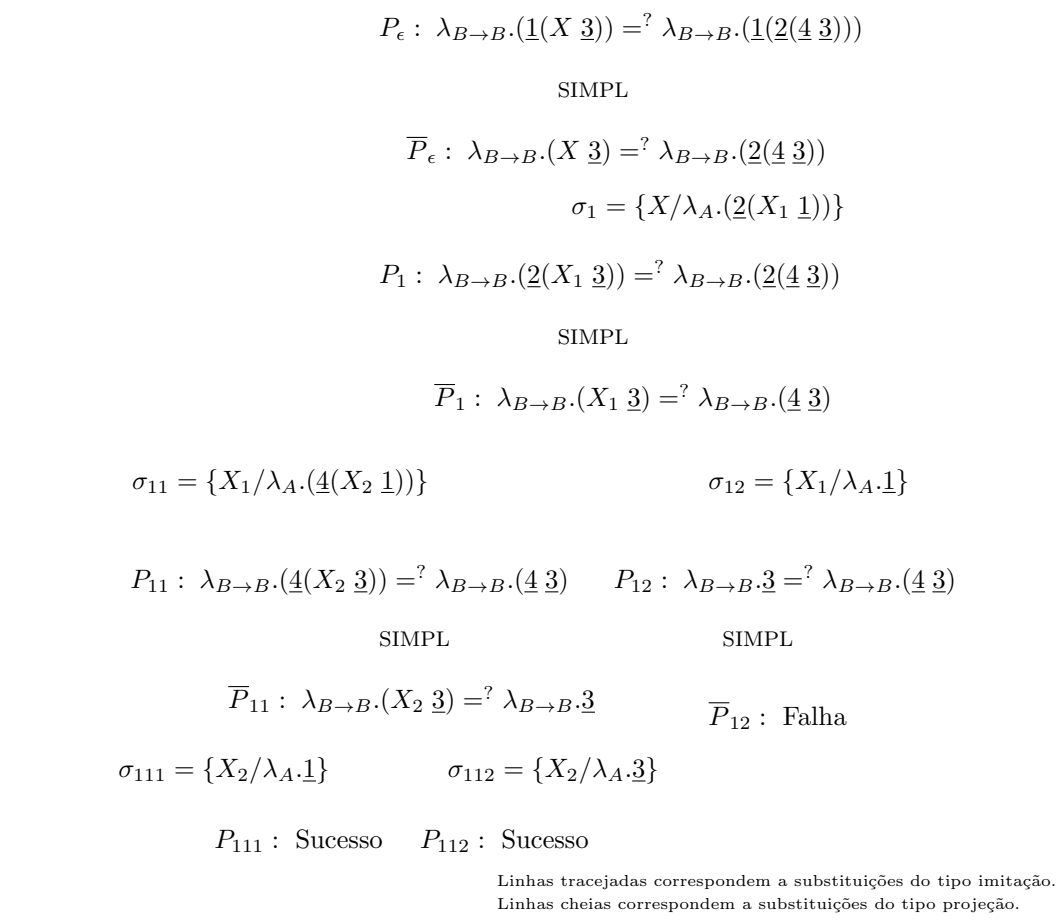


Figura 2.3: Um exemplo de árvore de unificação.

e, em seguida obtemos o resultado desejado por β-normalização:

$$\{X/\lambda_A.(\underline{2}(\underline{4}\ \underline{1})), X_1/\lambda_A.(\underline{4}\ \underline{1}), X_2/\lambda_A.\underline{1}\}.$$

Da mesma forma, podemos calcular a substituição:

$$\sigma_1\sigma_{11}\sigma_{112} = \{X/\lambda_A.(\underline{2}(\underline{4}\ \underline{3})), X_1/\lambda_A.(\underline{4}\ \underline{3}), X_2/\lambda_A.\underline{3}\}.$$

As soluções do problema original são dadas pelas substituições para as meta-variáveis que aparecem no problema original: $X/\lambda_A.(\underline{2}(\underline{4}\ \underline{3}))$ e $X/\lambda_A.(\underline{2}(\underline{4}\ \underline{1}))$.

2.4 Unificação no $\lambda\sigma$-cálculo

Seja J um subconjunto finito de inteiros positivos. Um problema de unificação no $\lambda\sigma$-cálculo é dado por uma conjunção de um número finito de equações da forma:

$$\bigwedge_i a_i =^?_{\lambda\sigma} b_i$$

onde a_i e b_i são $\lambda\sigma$-termos do mesmo tipo no mesmo contexto. Note que para um problema de unificação estar bem tipado precisamos que todas as ocorrências de uma dada meta-variável, digamos X, ocorram sempre sob o mesmo e único contexto Γ_X.

Os problemas de unificação que estamos interessados em resolver são obviamente os problemas gerados no λ-cálculo simplesmente tipado. O $\lambda\sigma$-cálculo é uma extensão do λ-cálculo que vai nos permitir utilizar *grafting* ao invés de substituição (de ordem superior) durante o processo de unificação. Para que possamos então resolver um problema P do λ-cálculo simplesmente tipado no $\lambda\sigma$-cálculo precismos inicialmente escrever P na linguagem do $\lambda\sigma$-cálculo. Esta conversão é feita por meio de uma função chamada de *pré-cozimento* que definimos a seguir:

Definição 2.12 (Pré-cozimento [DHK00]) *Seja $a \in \Lambda_{dB}(\mathcal{X})$ tal que $\Gamma \vdash a : A$. A cada meta-variável X de tipo U em a, associamos o tipo U e o contexto Γ no $\lambda\sigma$-cálculo. O* pré-cozimento *de a de $\Lambda_{dB}(\mathcal{X})$ para o conjunto de $\lambda\sigma$-termos, denotado por $\Lambda_{\lambda\sigma}(\mathcal{X})$, é definido por $a_F = F(a, 0)$ onde $F(a, n)$ $(n \geq 0)$ é definido por:*

(a) $F((\lambda_B.a), n) = \lambda_B(F(a, n + 1))$;

(b) $F(\underline{k}, n) = \underline{1}[\uparrow^{k-1}]$;

(c) $F((a\ b), n) = (F(a, n)\ F(b, n))$;

(d) $F(X, n) = X[\uparrow^n]$.

A função de pré-cozimento definida acima é injetiva e, portanto sua inversa está bem definida. Esta observação é importante para traduzirmos as soluções obtidas no $\lambda\sigma$-cálculo de volta para o λ-cálculo.

Agora estamos prontos para discutir um detalhe técnico que é essencial para entendermos claramente as diferenças entre unificação no λ-cálculo e no $\lambda\sigma$-cálculo.

Observação 2.13 *Pretendemos enfatizar uma importante diferença existente entre as regras (meta) do λ-cálculo e do λσ-cálculo. No λ-cálculo em notação* à la *de Bruijn o tipo das meta-variáveis é independente do contexto. De fato, podemos tipar um λ-termo que contenha várias ocorrências da mesma meta-variável em diferentes níveis de abstração: por exemplo, seja Γ um contexto qualquer e X uma meta-variável de tipo* $(A \to A) \to A$. *Considere a seguinte dedução:*

$$\dfrac{\boxtimes \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \cdot A \cdot (A \to A) \to A \cdot nil \vdash \underline{1} : A}\,(var)}{A \cdot (A \to A) \to A \cdot nil \vdash \lambda_A.\underline{1} : A \to A}\,(lambda) \qquad \boxplus}{A \cdot (A \to A) \to A \cdot nil \vdash (X\ \lambda_A.\underline{1}) : A}\,(app)}{(A \to A) \to A \cdot nil \vdash \lambda_A.(X\ \lambda_A.\underline{1}) : A \to A}\,(lambda)}{(A \to A) \to A \cdot nil \vdash (X\ \lambda_A.(X\ \lambda_A.\underline{1})) : A}\,(app)$$

onde $\boxplus$ *corresponde a:*

$$\dfrac{}{A \cdot (A \to A) \to A \cdot nil \vdash X : (A \to A) \to A}\,(meta)$$

e $\boxtimes$ *corresponde a:*

$$\dfrac{}{(A \to A) \to A \cdot nil \vdash X : (A \to A) \to A}\,(meta)$$

Por outro lado, o termo $(X\ \lambda_A.(X\ \lambda_A.\underline{1}))$ *apesar de estar bem formado, não pode ser tipado no λσ-cálculo! De fato, na derivação acima, precisamos utilizar a regra (meta) duas vezes em diferentes contextos para a mesma meta-variável X, fato que é possível no λ-cálculo mas não no λσ-cálculo. No λσ-cálculo cada meta-variável está associada a um único contexto e, portanto a mesma meta-variável não pode ser tipada em diferentes níveis de abstração. Esta restrição que parece ser por demais severa é necessária porque no λσ-cálculo temos* grafting *ao invés de substitutição e, a aplicação do* grafting $\{X \mapsto \underline{1}\}$ *ao λσ-termo* $(X\ \lambda_A.(X\ \lambda_A.\underline{1}))$ *resulta em* $(\underline{1}\ \lambda_A.(\underline{1}\ \lambda_A.\underline{1}))$ *que claramente está incorreto devido a captura da segunda ocorrência do índice* $\underline{1}$. *A função de pré-cozimento foi desenvolvida justamente para resolver este problema. A forma pré cozida do termo* $(X\ \lambda_A.(X\ \lambda_A.\underline{1}))$ *é dada por* $(X\ \lambda_A.(X[\uparrow]\ \lambda_A.\underline{1}))$ *que é bem tipada no λσ-cálculo como podemos ver pela seguinte*

derivação:

$$\dfrac{\boxtimes \qquad \dfrac{\dfrac{\dfrac{}{A \cdot A \cdot (A \to A) \to A \cdot nil \vdash \underline{1} : A}\,(var)}{A \cdot (A \to A) \to A \cdot nil \vdash \lambda_A.\underline{1} : A \to A}\,(lambda) \quad \boxdot}{A \cdot (A \to A) \to A \cdot nil \vdash (X[\uparrow]\ \lambda_A.\underline{1}) : A}\,(app)}{(A \to A) \to A \cdot nil \vdash \lambda_A.(X[\uparrow]\ \lambda_A.\underline{1}) : A \to A}\,(lambda)}{(A \to A) \to A \cdot nil \vdash (X\ \lambda_A.(X[\uparrow]\ \lambda_A.\underline{1})) : A}\,(app)$$

onde $\boxdot$ *corresponde a:*

$$\dfrac{\boxplus \qquad \dfrac{}{(A \to A) \to A \cdot nil \vdash X : (A \to A) \to A}\,(meta)}{A \cdot (A \to A) \to A \cdot nil \vdash X[\uparrow] : (A \to A) \to A}\,(clos)$$

onde $\boxplus$ *corresponde a:*

$$\dfrac{}{A \cdot (A \to A) \to A \cdot nil \vdash \uparrow \rhd (A \to A) \to A \cdot nil}\,(shift)$$

e $\boxtimes$ *corresponde a:*

$$\dfrac{}{(A \to A) \to A \cdot nil \vdash X : (A \to A) \to A}\,(meta)$$

Como pudemos observar, a função de pré-cozimento realiza os ajustes necessários para que λ-termos sejam convertidos em λσ-termos onde se pode, de forma correta, utilizar grafting *ao invés de substituição.*

Definição 2.14 (Formas resolvidas [DHK00]) *Um problema de unificação P é dito estar em* forma resolvida *se for formado por uma conjunção de equações não-triviais das seguintes formas:*

- **Resolvida***: $X =^{?}_{\lambda\sigma} a$ onde a meta-variável X não aparece em nenhum outro lugar em P e a está em forma η-longa. Dizemos que uma tal equação está* resolvida *em P e, neste caso a meta-variável X também está resolvida.*
- **Flexível-flexível***: $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] =^{?}_{\lambda\sigma} Y[b_1 \cdot \ldots \cdot b_q \cdot \uparrow^m]$, onde $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ e $Y[b_1 \cdot \ldots \cdot b_q \cdot \uparrow^m]$ estão em forma η-longa e a equação não está resolvida.*

As regras de unificação para o $\lambda\sigma$-cálculo são dadas na Tabela 2.1 originalmente encontrada em [DHK00]. Este conjunto de regras será chamado de **Unif** que assumimos serem sempre aplicadas de uma forma "honesta" que significa que aplicações de **Exp-λ** (a regra que introduz novas meta-variáveis de tipo mais simples) são sempre seguidas de aplicações de **Replace** para evitar aplicações infinitas de **Exp-λ**. De fato, como uma aplicação de **Exp-λ** apenas adiciona uma nova equação flexível-flexível e não muda mais nada no problema atual, sem uma aplicação de **Replace** esta regra poderia ser aplicada indefinidamente.

Aplicações das regras de unificação a um dado problema do $\lambda\sigma$-cálculo serão representadas por meio de árvores, que chamaremos de *árvores de derivação*. Uma árvore de derivação é uma árvore cujos nós contêm problemas de unificação e arcos contêm os nomes das regras aplicadas como marcas. Como usual, disjunções são representadas por ramificações na árvore de derivação.

Exemplo 2.15 *Considere novamente o contexto* $\Gamma = A \to B \cdot A \cdot A \to A \cdot nil$ *e o problema de unificação dado por:*

$$\lambda_{B\to B}.(\underline{1}(X\ \underline{3})) =^{?} \lambda_{B\to B}.(\underline{1}(\underline{2}(\underline{4}\ \underline{3})))$$

bem tipado no contexto Γ*. A árvore de unificação deste problema é dada na Figura 2.3. Para resolvermos este problema de unificação no $\lambda\sigma$-cálculo precisamos aplicar a função de pré-cozimento, que gera o seguinte problema de unificação:*

$$\lambda_{B\to B}.(\underline{1}(X[\uparrow]\ \underline{3})) =^{?}_{\lambda\sigma} \lambda_{B\to B}.(\underline{1}(\underline{2}(\underline{4}\ \underline{3}))) \tag{2.8}$$

que é bem tipado no contexto Γ*. A árvore de derivação para este problema é apresentada nas figuras 2.4, 2.5, 2.6, 2.7 e 2.8. Note que esta árvore de derivação e a árvore de unificação da Figura 2.3 possuem uma estrutura similar: ambas têm exatamente um nó de falha e dois nós de sucesso. As soluções da equação (2.8) são dadas pelos* graftings $\{X \mapsto \lambda_A.(\underline{2}(\underline{4}\ \underline{1}))\}$ *e* $\{X \mapsto \lambda_A.(\underline{2}(\underline{4}\ \underline{3}))\}$ *que correspondem respectivamente, às substituições* $\{X/\lambda_A.(\underline{2}(\underline{4}\ \underline{1}))\}$ *e* $\{X/\lambda_A.(\underline{2}(\underline{4}\ \underline{3}))\}$ *obtidas no Exemplo 2.11.*

Dec-λ	$\dfrac{P \wedge \lambda_A.e_1 =^?_{\lambda\sigma} \lambda_A.e_2}{P \wedge e_1 =^?_{\lambda\sigma} e_2}$
Dec-App	$\dfrac{P \wedge (\underline{n}\ e_1^1 \dots e_p^1) =^?_{\lambda\sigma} (\underline{n}\ e_1^2 \dots e_p^2)}{P \wedge e_1^1 =^?_{\lambda\sigma} e_1^2 \wedge \dots \wedge e_p^1 =^?_{\lambda\sigma} e_p^2}$
Dec-Fail	$\dfrac{P \wedge (\underline{n}\ e_1^1 \dots e_{p_1}^1) =^?_{\lambda\sigma} (\underline{m}\ e_1^2 \dots e_{p_2}^2)}{Falha}$, se $m \neq n$.
Exp-λ	$\dfrac{P}{\exists(A \cdot \Gamma \vdash Y : B), P \wedge X =^?_{\lambda\sigma} \lambda_A.Y}$ se $(\Gamma \vdash X : A \to B) \in \mathcal{TV}ar(P)$, $Y \notin \mathcal{TV}ar(P)$, e X é uma meta-variável não resolvida.
Exp-App	$\dfrac{P \wedge X[a_1 \cdot \dots \cdot a_p \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \dots b_q)}{P \wedge X[a_1 \cdot \dots \cdot a_p \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \dots b_q) \wedge \bigvee_{r \in R_p \cup R_i} \exists H_1 \dots \exists H_k : X =^?_{\lambda\sigma} (\underline{r}\ H_1 \dots H_k)}$ se X tem tipo atômico e não é uma meta-variável resolvida. onde $H_1, \dots, H_k$ são meta-variáveis novas de tipo apropriado, que não ocorram em P,com contextos $\Gamma_{H_i} = \Gamma_X$, R_p é um subconjunto de $\{1, \dots, p\}$ tal que $(\underline{r}\ H_1 \dots H_k)$ tem o tipo adequado, R_i =se $m \geq n+1$ então $\{m - n + p\}$ caso contrário $\emptyset$.
Normalise	$\dfrac{P \wedge e_1 =^?_{\lambda\sigma} e_2}{P \wedge e_1' =^?_{\lambda\sigma} e_2'}$ se e_1 ou e_2 não está em forma η-longa. onde e_1' (resp. e_2') é a forma η-longa de e_1 (resp. e_2) se e_1 (resp. e_2) não é uma variável resolvida e e_1 (resp. e_2) caso contrário.
Replace	$\dfrac{P \wedge X =^?_{\lambda\sigma} t}{\{X \mapsto t\}(P) \wedge X =^?_{\lambda\sigma} t}$ se $X \in \mathcal{TV}ar(P), X \notin \mathcal{TV}ar(t)$ e se t é uma meta-variável então $t \in \mathcal{TV}ar(P)$.

Tabela 2.1: Regras de unificação para o $\lambda\sigma$-cálculo

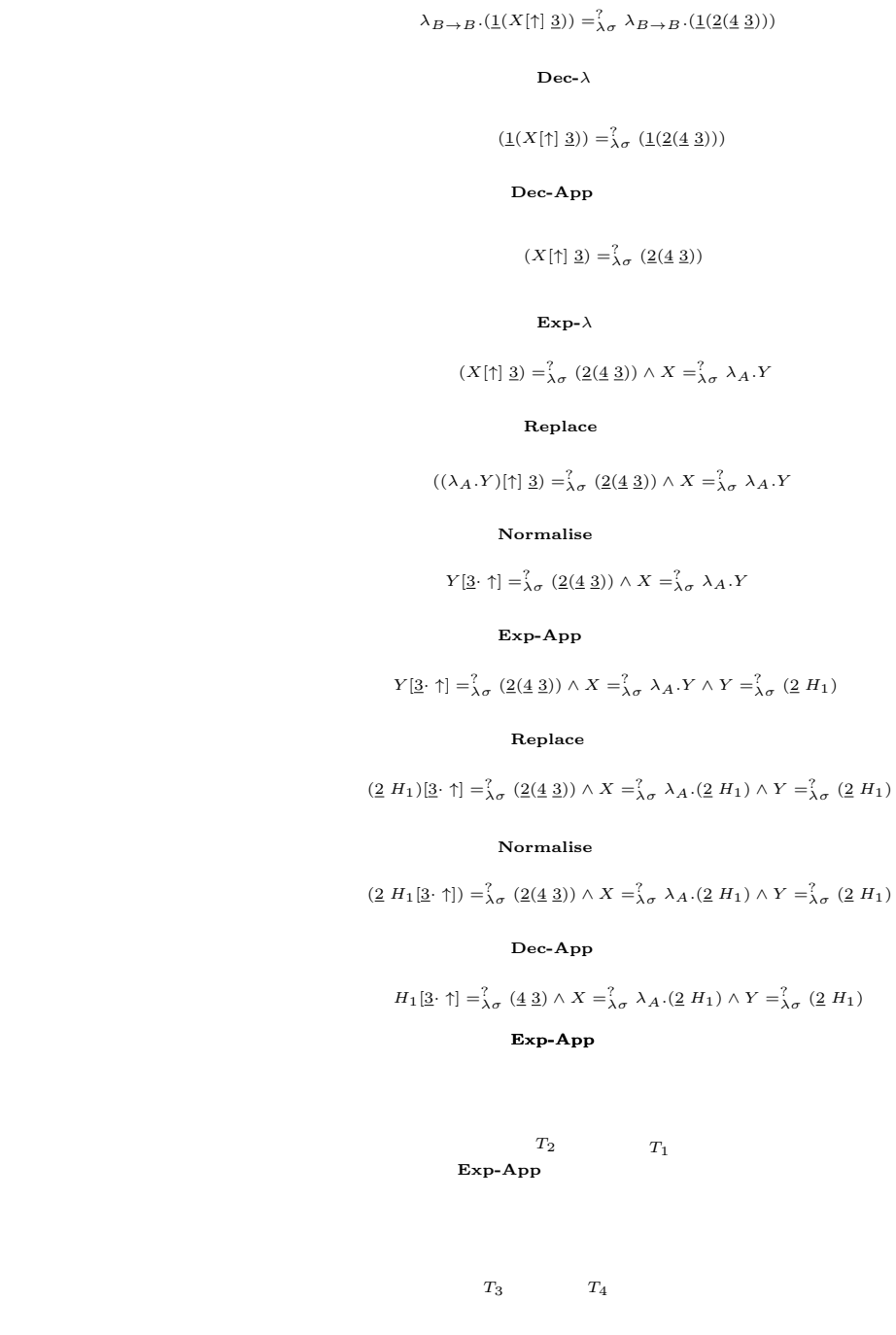


Figura 2.4: Árvore de derivação no $\lambda\sigma$-cálculo.

$$H_1[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} (\underline{4}\ \underline{3}) \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ H_1) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ H_1) \wedge H_1 =^?_{\lambda\sigma} \underline{1}$$

Replace

$$\underline{1}[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} (\underline{4}\ \underline{3}) \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ \underline{1}) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ \underline{1}) \wedge H_1 =^?_{\lambda\sigma} \underline{1}$$

Normalise

$$\underline{3} =^?_{\lambda\sigma} (\underline{4}\ \underline{3}) \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ \underline{1}) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ \underline{1}) \wedge H_1 =^?_{\lambda\sigma} \underline{1}$$

Dec-Fail

Falha

Figura 2.5: A árvore T_1 no $\lambda\sigma$-cálculo.

$$H_1[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} (\underline{4}\ \underline{3}) \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ H_1) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ H_1) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ H_2)$$

Replace

$$(\underline{4}\ H_2)[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} (\underline{4}\ \underline{3}) \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ (\underline{4}\ H_2)) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ H_2)$$

Normalise

$$(\underline{4}\ H_2[\underline{3}\cdot\uparrow]) =^?_{\lambda\sigma} (\underline{4}\ \underline{3}) \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ (\underline{4}\ H_2)) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ H_2)$$

Dec-App

$$H_2[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} \underline{3} \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ (\underline{4}\ H_2)) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ H_2)$$

Figura 2.6: A árvore T_2 no $\lambda\sigma$-cálculo.

$$H_2[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} \underline{3} \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ (\underline{4}\ H_2)) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ H_2) \wedge H_2 =^?_{\lambda\sigma} \underline{3}$$

Replace

$$\underline{3}[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} \underline{3} \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (\underline{4}\ \underline{3})) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ (\underline{4}\ \underline{3})) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ \underline{3}) \wedge H_2 =^?_{\lambda\sigma} \underline{3}$$

Normalise

$$X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (\underline{4}\ \underline{3})) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ (\underline{4}\ \underline{3})) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ \underline{3}) \wedge H_2 =^?_{\lambda\sigma} \underline{3}$$

Sucesso

Figura 2.7: A árvore T_3 no $\lambda\sigma$-cálculo.

$$H_2[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} \underline{3} \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ (\underline{4}\ H_2)) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ H_2) \wedge H_2 =^?_{\lambda\sigma} \underline{1}$$

Replace

$$\underline{1}[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} \underline{3} \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (\underline{4}\ \underline{1})) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ (\underline{4}\ \underline{1})) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ \underline{1}) \wedge H_2 =^?_{\lambda\sigma} \underline{1}$$

Normalise

$$X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (\underline{4}\ \underline{1})) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ (\underline{4}\ \underline{1})) \wedge H_1 =^?_{\lambda\sigma} (\underline{4}\ \underline{1}) \wedge H_2 =^?_{\lambda\sigma} \underline{1}$$

Sucesso

Figura 2.8: A árvore T_4 no $\lambda\sigma$-cálculo.

Capítulo 3

Uma Relação Estrutural entre HOU no λ-cálculo e no $\lambda\sigma$-cálculo

O objetivo deste capítulo é apresentar uma relação estrutural entre o algoritmo de Huet para o λ-cálculo simplesmente tipado em notação *à la* de Bruijn e o método de unificação apresentado na Seção 2.4 para o $\lambda\sigma$-cálculo. Como veremos, a relação estrutural que apresentaremos refina o resultado de Dowek, Hardin and Kirchner [DHK00] que estabelece que um problema da forma $a =^{?} b$ tem solução no λ-cálculo simplesmente tipado se, e somente se, $a_F =^{?}_{\lambda\sigma} b_F$ tem solução no $\lambda\sigma$-cálculo.

3.1 O Pseudo-cozimento

Iniciaremos esta seção com a noção de *pseudo-cozimento* que estende a noção usual de pré-cozimento combinando-a com idéias ligadas às regras de unificação.

Definição 3.1 (Pseudo-cozimento) *Seja Γ um contexto e a um λ-termo bem tipado no contexto Γ. A cada meta-variável X de tipo B ocorrendo em a, associamos o tipo B e o contexto Γ. O* pseudo-cozimento *do termo a de $\Lambda_{dB}(\mathcal{X})$ para $\Lambda_{\lambda\sigma}(\mathcal{X})$, denotado por $\overline{a}$, é definido por $\overline{a} = p(a, 0)$, onde $p(a, n)$ é definido indutivamente, para todo $m \geq 0$, como segue:*

- $p(\lambda_{A_1} \ldots \lambda_{A_m}.a, n) = \lambda_{A_1} \ldots \lambda_{A_m}.p(a, n + m)$.
- $p((\underline{k}\ a_1 \ldots a_m), n) = (\underline{1}[\uparrow^{k-1}]\ p(a_1, n) \ldots p(a_m, n))$.

- $p((X\ a_1 \ldots a_m), n) = Y[p(a_m, n)\cdot \ldots \cdot p(a_1, n)\cdot \uparrow^n]$, *onde* Y *é uma meta-variável nova com tipo igual ao tipo alvo de* X.

O pseudo-cozimento é uma função que toma como argumento um λ-termo bem tipado a e retorna um $\lambda\sigma$-termo $\overline{a}$ bem tipado com o mesmo tipo e contexto de a.

Intuitivamente, $\overline{a}$ pode ser obtido a partir de a_F (a forma pré-cozida de a) após algumas aplicações de **Exp-λ**, **Replace** e **Normalise** às meta-variáveis de tipo funcional que ocorrem no problema de unificação que contém a_F. Aplicações de **Exp-λ** introduzem novas equações que são ignoradas pelo pseudo-cozimento porque neste ponto estamos interessados apenas na estrutura de algumas equações obtidas no $\lambda\sigma$-cálculo. A noção de pseudo-cozimento é essencial para relacionarmos unificação no λ- e no $\lambda\sigma$-cálculo. Além disto, esta noção pode ser útil na implementação de uma versão melhorada do procedimento de Dowek, Hardin e Kirchner, no sentido que o pseudo-cozimento é capaz de combinar o pré-cozimento usual com aplicações de regras de unificação em um único passo.

O exemplo a seguir nos dá uma idéia de como o pseudo-cozimento relaciona o pré-cozimento com as regras de unificação

Exemplo 3.2 *Considere a árvore de unificação dada na Figura 2.3 e, tome o sub-problema* $\overline{P}_\epsilon$*:*

$$\lambda_{B\to B}.(X\ \underline{3}) =^? \lambda_{B\to B}.(\underline{2}(\underline{4}\ \underline{3})) \tag{3.1}$$

cuja forma pseudo-cozida é dada por $\lambda_{B\to B}.Z[\underline{3}\cdot \uparrow] =^?_{\lambda\sigma} \lambda_{B\to B}.\underline{2}(\underline{4}\ \underline{3})$*, que pode ser encontrada na Figura 2.4, após a primeira aplicação de* **Normalise** *a menos de renomeamento de meta-variáveis e sem os abstratores externos. Os abstratores externos são removidos por aplicações de* **Dec**-λ *no início da derivação e, por uma simples inspeção das regras* **Unif** *é fácil notar que nenhuma delas insere novos abstratores. A forma pseudo-cozida dada acima pode ser obtida a partir da forma pré-cozida da equação (3.1) da seguinte forma:*

$$\dfrac{\dfrac{\dfrac{\lambda_{B\to B}.(X[\uparrow]\ \underline{3}) =^?_{\lambda\sigma} \lambda_{B\to B}.(\underline{2}(\underline{4}\ \underline{3}))}{\lambda_{B\to B}.(X[\uparrow]\ \underline{3}) =^?_{\lambda\sigma} \lambda_{B\to B}.(\underline{2}(\underline{4}\ \underline{3})) \wedge X =^?_{\lambda\sigma} \lambda_A.Z}\ \textbf{Exp-}\lambda}{\lambda_{B\to B}.((\lambda_A.Z)[\uparrow]\ \underline{3}) =^?_{\lambda\sigma} (\underline{2}(\underline{4}\ \underline{3})) \wedge X =^?_{\lambda\sigma} \lambda_{B\to B}.\lambda_A.Z}\ \textbf{Replace}}{\lambda_{B\to B}.Z[\underline{3}\cdot \uparrow] =^?_{\lambda\sigma} \lambda_{B\to B}.(\underline{2}(\underline{4}\ \underline{3})) \wedge X =^?_{\lambda\sigma} \lambda_A.Z}\ \textbf{Normalise}$$

A proposição a seguir nos mostra que o pseudo-cozimento preserva os tipos e os contextos dos termos.

Proposição 3.3 *Seja Γ um contexto e a um λ-termo em $\Lambda_{dB}(\mathcal{X})$ bem tipado no contexto Γ. Se a' é um subtermo de a tal que $A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash a' : A$ então $A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash p(a', n) : A$, para todo $n \geq 0$.*

Prova. A prova é por indução na estrutura de a':

- Se a' é um índice *à la* de Bruijn ou uma aplicação então o resultado é trivial.
- Se $a' = \lambda_B.b$ é um termo de tipo $B \to C$ então, por hipótese temos que $A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash \lambda_B.b : B \to C$, e portanto $B \cdot A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash b : C$. Por hipótese de indução temos que $B \cdot A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash p(b, n+1) : C$, para todo $n \geq 0$. Após uma aplicação de $(lambda)$ temos que $A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash \lambda_B.p(b, n+1) : B \to C$ o que é equivalente a $A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash p(\lambda_B.b, n) : B \to C$.
- Se $a' = (X\ b_1 \ldots b_m)$, onde X é uma meta-variável de tipo $B_1 \to \ldots \to B_m \to A$ então por hipótese temos que $A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash (X\ a_1 \ldots a_m) : A$. Por hipótese de indução, temos para cada $1 \leq i \leq m$: $A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash p(a_i, n) : B_i$, para todo $n \geq 0$. Seja Y uma meta-variável nova de tipo A e, considere a seguinte derivação:

$$\dfrac{\boxdot \qquad \dfrac{}{B_m \cdot \ldots \cdot B_1 \cdot \Gamma \vdash Y : A}\,(meta)}{A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash Y[p(b_m, n) \cdot \ldots \cdot p(b_1, n) \cdot \uparrow^n] : A}\,(clos)$$

onde $\boxdot$ corresponde a:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash p(b_1, n) : B_1}\,(IH) \qquad \dfrac{}{A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash \uparrow^n \rhd \Gamma}\,(shift)}{A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash p(b_1, n) \cdot \uparrow^n \rhd B_1 \cdot \Gamma}\,(cons)}{\vdots \qquad\qquad \vdots}\,(cons)}{A_1 \cdot \ldots \cdot A_n \cdot \Gamma \vdash p(b_m, n) \cdot \ldots \cdot p(b_1, n) \cdot \uparrow^n \rhd B_1 \cdot \Gamma}\,(clos)$$

□

3.2 Relacionando Árvores de Unificação e Árvores de Derivação

O lema a seguir nos mostra como a estrutura das equações em problemas de unificação do λ-cálculo estão relacionadas com os problemas de unificação no $\lambda\sigma$-cálculo a partir de suas formas pré-cozidas.

Lema 3.4 *Sejam Γ um contexto, P um problema de unificação com termos em $\Lambda_{dB}(\mathcal{X})$ tal que todas as suas equações estejam bem tipadas no contexto Γ e, $\mathcal{A}(P)$ a árvore de unificação de P. Para cada problema P_α ocorrendo em $\mathcal{A}(P)$, existe um problema de unificação P^* derivado de P_F via o sistema de regras* **Unif** *satisfazendo as seguintes condições:*

1. *Para cada equação em P_α da forma:*

$$\lambda_{A_1} \ldots \lambda_{A_n}.(h_1\ e_1^1 \ldots e_{p_1}^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e_1^2 \ldots e_{p_2}^2) \tag{3.2}$$

   bem tipada no contexto Γ, onde h_1 é um índice à la *de Bruijn ou uma meta-variável, existe uma equação em P^* da forma:*

$$(h_1\ \overline{e}_1^1 \ldots \overline{e}_{p_2}^1) =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2),\ \textit{se } h_1 \textit{ é um índice } \text{à la } \textit{de Bruijn}, \tag{3.3}$$

   ou

$$Y[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2),\ \textit{se } h_1 \textit{ é uma meta-variável;} \tag{3.4}$$

   bem tipadas no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$.

2. *Para cada equação da forma flexível-flexível eq em P_α, a equação eq_F está em P^*. Neste caso, assumimos que nenhuma regra de unificação é aplicada à equação da forma flexível-flexível já que o método de unificação não precisa lidar com equações deste tipo.*

Prova. A prova é por indução sobre o comprimento da derivação que gera P_α. Para a base de indução (bi), considere $\alpha = \epsilon$. Dividimos a análise em dois casos:
(bi.1): Se P_ϵ contém apenas equações da forma flexível-flexível então, como para cada equação eq em P, eq_F está em P_F, tome $P^* = P_F$.
(bi.2): Se P_ϵ contém equações da forma:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(h_1\ e_1^1 \ldots e_{p_1}^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e_1^2 \ldots e_{p_2}^2)$$

onde h_1 é uma meta-variável ou o índice *à la* de Bruijn $\underline{h}$ então consideramos dois casos:
(bi.2.1): Se h_1 é uma meta-variável, digamos X, então P_F contém a equação:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(X[\uparrow^n]\ e^1_{1_F} \ldots e^1_{p_{1_F}}) =^?_{\lambda\sigma} \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e^2_{1_F} \ldots e^2_{p_{2_F}}).$$

Considere a seguinte derivação:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\lambda_{A_1} \ldots \lambda_{A_n}.(X[\uparrow^n]\ e^1_{1_F} \ldots e^1_{p_{1_F}}) =^?_{\lambda\sigma} \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e^2_{1_F} \ldots e^2_{p_{2_F}})}{\vdots\ n \text{ vezes}}\ \textbf{Dec-}\lambda}{(X[\uparrow^n]\ e^1_{1_F} \ldots e^1_{p_{1_F}}) =^?_{\lambda\sigma} (\underline{h}\ e^2_{1_F} \ldots e^2_{p_{2_F}})}\ \textbf{Dec-}\lambda}{(X[\uparrow^n]\ e^1_{1_F} \ldots e^1_{p_{1_F}}) =^?_{\lambda\sigma} (\underline{h}\ e^2_{1_F} \ldots e^2_{p_{2_F}}) \wedge X =^?_{\lambda\sigma} \lambda_{B_1}.X_1}\ \textbf{Exp-}\lambda}{((\lambda_{B_1}.X_1)[\uparrow^n]\ e^1_{1_F} \ldots e^1_{p_{1_F}}) =^?_{\lambda\sigma} (\underline{h}\ e^2_{1_F} \ldots e^2_{p_{2_F}}) \wedge X =^?_{\lambda\sigma} \lambda_{B_1}.X_1}\ \textbf{Replace}}{\vdots\ (p_1 - 1) \text{ vezes}}\ \boxdot$$

$$\dfrac{((\lambda_{B_1} \ldots \lambda_{B_{p_1}}.Y)[\uparrow^n]\ e^1_{1_F} \ldots e^1_{p_{1_F}}) =^?_{\lambda\sigma} (\underline{h}\ e^2_{1_F} \ldots e^2_{p_{2_F}}) \wedge X =^?_{\lambda\sigma} \lambda_{B_1} \ldots \lambda_{B_{p_1}}.Y \wedge \ldots}{Y[e^1_{p_{1_F}} \cdot \ldots \cdot e^1_{1_F} \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{h}\ e^2_{1_F} \ldots e^2_{p_{2_F}}) \wedge X =^?_{\lambda\sigma} \lambda_{B_1} \ldots \lambda_{B_{p_1}}.Y \wedge \ldots}\ \boxtimes$$

onde Y é uma nova meta-variável, $\boxtimes$ corresponde a uma aplicação de **Normalise** e $\boxdot$ corresponde a uma aplicação de **Exp-λ** seguida de uma aplicação de **Replace**. A partir deste ponto, aplicamos **Exp-λ** seguida de **Replace** para cada subtermo de $e^1_{1_F}$, ..., $e^1_{p_{1_F}}$, $e^2_{1_F}$, ..., $e^2_{p_{2_F}}$ da forma $(V[\uparrow^m]\ f_1 \ldots f_r)$ $(m, r \geq 0)$ recursivamente. Depois de normalizados, estes subtermos passam a ter a forma $W[\overline{f}_r \cdot \ldots \cdot \overline{f}_1 \cdot \uparrow^m]$, onde W é uma meta-variável nova com tipo igual ao tipo alvo de V. Desta forma, obtemos o problema de unificação P^* que contém a equação $Y[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] =^?_{\lambda\sigma}$ $(\underline{h}\ \overline{e}^2_1 \ldots \overline{e}^2_{p_2})$. Todas as outras ocorrências de X no problema de unificação atual são substituídas por $\lambda_{B_1} \ldots \lambda_{B_{p_1}}.Y$ e, portanto qualquer outro termo que tenha X como cabeça permanece sendo flexível e, após uma aplicação de **Replace** a meta-variável X passa a ser resolvida. O mesmo ocorre para cada meta-variável de tipo funcional que ocorra no problema de unificação e, desta forma, meta-variáveis de tipo funcional aparecem apenas em equações resolvidas. Durante a derivação acima nenhuma equação trivial é gerada porque as aplicações de **Exp-λ** sempre introduzem meta-variáveis novas que claramente não podem ser eliminadas por aplicações de **Dec-λ**, **Replace** ou **Normalise** que correspondem às regras utilizadas.

A derivação acima nos mostra que após n aplicações de **Dec-λ** e um número finito de aplicações de **Exp-λ** sempre seguidas de uma aplicação de **Replace**, obtemos um problema (após o processo de normalização) contendo uma equação com a mesma estrutura da equação (3.4). A ordem em que as regras **Dec-λ** e **Exp-λ** são aplicadas

é irrelevante porque elas são aplicadas em posições distintas de um termo. A única restrição existente nas derivações aqui apresentadas é que uma aplicação de **Exp-λ** seja sempre seguida de uma aplicação de **Replace** para evitar reduções com infinitas aplicações de **Exp-λ** (cf. [DHK00]). O passo de normalização (**Normalise**), pode ser aplicado diversas vezes em passos intermediários da redução ou em um único passo ao final da redução sem que o resultado final seja alterado.

(bi.2.2): Se h_1 é um índice *à la* de Bruijn então P_F contém a equação:

$$\lambda_{A_1} \dots \lambda_{A_n}.(h_1\ e^1_{1_F} \dots e^1_{p_{1_F}}) =^?_{\lambda\sigma} \lambda_{A_1} \dots \lambda_{A_n}.(\underline{h}\ e^2_{1_F} \dots e^2_{p_{2_F}}).$$

Neste caso, tomamos P^* como sendo o problema obtido de P_F após n aplicações de **Dec-λ** seguidas de aplicações de **Exp-λ** e **Replace** a todo subtermo de $e^1_{1_F}$, ..., $e^1_{p_{1_F}}$, $e^2_{1_F}$, ..., $e^2_{p_{2_F}}$ da forma $(V[\uparrow^m]\ f_1 \dots f_r)$. Como no caso anterior, nenhuma equação é eliminada durante este processo.

Para o passo indutivo (pi), suponha que P_α seja um problema de unificação em $\mathcal{A}(P)$ com $\alpha \neq \epsilon$ e P^* o problema de unificação obtido de P_F de acordo com o enunciado desse lema. Para cada problema de unificação derivado de P_α, precisamos encontrar um sitema de unificação derivado de P^*, digamos P^{**}, que satisfaça as hipóteses desse lema. Dividiremos a análise em dois casos:
(pi.1): *O problema derivado de P_α é obtido após uma aplicação de SIMPL*:

Neste caso, o problema derivado de P_α é $\overline{P}_\alpha$ de acordo com a definição de árvores de unificação e, temos as seguintes possibilidades para $\overline{P}_\alpha$:

(pi.1.1): *$\overline{P}_\alpha$ ainda não está resolvido, isto é, contém equações da forma flexível-rígida ou rígida-rígida*: Então P_α contém (pelo menos) uma equação rígida-rígida da forma:

$$\lambda_{B_1} \dots \lambda_{B_m}.(\underline{k}\ f^1_1 \dots f^1_p) =^? \lambda_{B_1} \dots \lambda_{B_m}.(\underline{k}\ f^2_1 \dots f^2_p) \tag{3.5}$$

bem tipada no contexto Γ, e tal que:

$$f^1_i = \lambda_{C_1} \dots \lambda_{C_w}.(h_1\ e^1_1 \dots e^1_{p_1}) \text{ e } f^2_i = \lambda_{C_1} \dots \lambda_{C_w}.(\underline{h}\ e^2_1 \dots e^2_{p_2})$$

para algum $i = 1, \dots, p$, onde h_1 é um índice *à la* de Bruijn ou uma meta-variável.

Durante a aplicação de SIMPL ao problema atual, a equação (3.5) é substituída pela conjunção:
$\lambda_{B_1} \dots \lambda_{B_m}.f^1_1 =^? \lambda_{B_1} \dots \lambda_{B_m}.f^2_1 \wedge \dots \wedge \lambda_{B_1} \dots \lambda_{B_m}.f^1_i =^? \lambda_{B_1} \dots \lambda_{B_m}.f^2_i \wedge \dots$
$\wedge\ \lambda_{B_1} \dots \lambda_{B_m}.f^1_p =^? \lambda_{B_1} \dots \lambda_{B_m}.f^2_p$

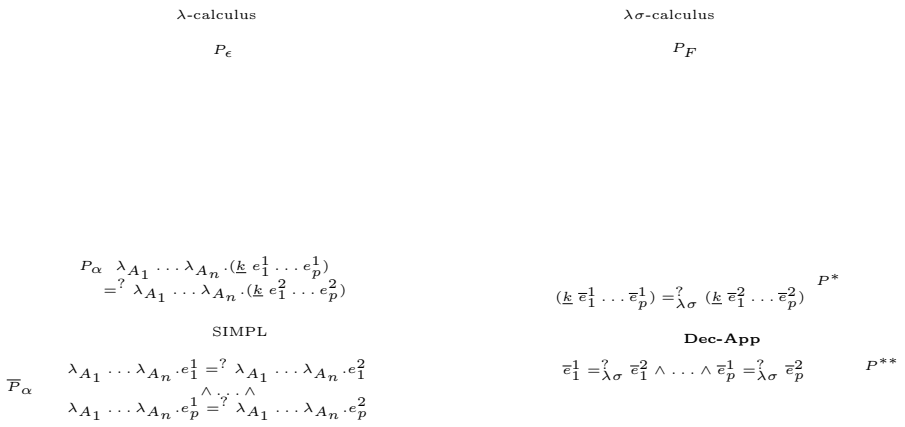


Figura 3.1: Um passo de simplificação

cujas equações são bem tipadas no contexto Γ e, onde para algum $1 \leq i \leq p$, a equação $\lambda_{B_1}\ldots\lambda_{B_m}.f_i^1 =^? \lambda_{B_1}\ldots\lambda_{B_m}.f_i^2$ tem a forma:

$$\lambda_{A_1}\ldots\lambda_{A_n}.(h_1\ e_1^1\ldots e_{p_1}^1) =^? \lambda_{A_1}\ldots\lambda_{A_n}.(\underline{h}\ e_1^2\ldots e_{p_2}^2)$$

onde $A_i = \begin{cases} B_i & \text{, for } 1 \leq i \leq m; \\ C_{i-m} & \text{, for } m < i \leq n(= m+w) \end{cases}$
e h_1 é um índice *à la* de Bruijn ou uma meta-variável.

Por hipótese, existe uma derivação de P_F que gera o problema P^* contendo a equação:

$$(\underline{j}\ \overline{f}_1^1\ldots\overline{f}_i^1\ldots\overline{f}_r^1) =^?_{\lambda\sigma} (\underline{j}\ \overline{f}_1^2\ldots\overline{f}_i^2\ldots\overline{f}_r^2) \tag{3.6}$$

que é bem tipada no contexto $B_1\cdot\ldots\cdot B_m\cdot\Gamma$.

Depois de uma aplicação de **Dec-App**, a equação (3.6) é substituída pela conjunção $\overline{f}_1^1 =^?_{\lambda\sigma} \overline{f}_1^2 \wedge\ldots\wedge \overline{f}_i^1 =^?_{\lambda\sigma} \overline{f}_i^2 \wedge\ldots\wedge \overline{f}_r^1 =^?_{\lambda\sigma} \overline{f}_r^2$.

Agora consideramos dois subcasos:

(*pi*.1.1.1) h_1 *é um índice* à la *de Bruijn*: Neste caso, a equação $\overline{f}_i^1 =^?_{\lambda\sigma} \overline{f}_i^2$ tem a forma:

$$\lambda_{C_1}\ldots\lambda_{C_w}.(h_1\ \overline{e}_1^1\ldots\overline{e}_{p_1}^1) =^?_{\lambda\sigma} \lambda_{C_1}\ldots\lambda_{C_w}.(\underline{h}\ \overline{e}_1^2\ldots\overline{e}_{p_2}^2).$$

Tome P^{**} como sendo o problema obtido após w aplicações de **Dec-**λ, isto é, o problema contendo a equação $(h_1\ \overline{e}_1^1\ldots\overline{e}_{p_1}^1) =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2\ldots\overline{e}_{p_2}^2)$, que é bem tipada no contexto $B_1\cdot\ldots\cdot B_m\cdot C_1\cdot\ldots\cdot C_w\cdot\Gamma$. Durante este processo todas as outras equações do problema permanecem inalteradas.

(*pi*.1.1.2) h_1 *é uma meta-variável*: Neste caso, a equação $\overline{f}_i^1 =^?_{\lambda\sigma} \overline{f}_i^2$ tem a forma $\lambda_{C_1}\ldots\lambda_{C_w}.Y[\overline{e}_{p_1}^1\cdot\ldots\cdot\overline{e}_1^1\cdot\uparrow^{m+w}] =^?_{\lambda\sigma} \lambda_{C_1}\ldots\lambda_{C_w}.(\underline{h}\ \overline{e}_1^2\ldots\overline{e}_{p_2}^2)$ e, tomamos P^{**} como

sendo o problema obtido após w aplicações de **Dec-λ**. Como no caso anterior, todas as outras equações permanecem inalteradas.

(pi.1.2) $\overline{P}_\alpha$ *já está resolvido, isto é, contém apenas equações da forma flexível-flexível*: Então P_α contém (pelo menos) uma equação da forma:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(\underline{k}\ e_1^1 \ldots e_p^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{k}\ e_1^2 \ldots e_p^2) \tag{3.7}$$

bem tipada no contexto Γ onde todos os termos $e_1^1, \ldots, e_p^1, \ldots, e_1^2, \ldots, e_p^2$ são flexíveis. Durante a aplicação de SIMPL a P_α, a equação (3.7) é substituída pela conjunção:

$$\lambda_{A_1} \ldots \lambda_{A_n}.e_1^1 =^? \lambda_{A_1} \ldots \lambda_{A_n}.e_1^2 \wedge \ldots \wedge \lambda_{A_1} \ldots \lambda_{A_n}.e_p^1 =^? \lambda_{A_1} \ldots \lambda_{A_n}.e_p^2$$

onde cada equação é bem tipada no contexto Γ e, todas as outras equações do problema permanecem inalteradas.

Por hipótese, existe uma derivação de P_F que gera o problema de unificação P^* contendo a equação $(\underline{k}\ \overline{e}_1^1 \ldots \overline{e}_p^1) =^?_{\lambda\sigma} (\underline{k}\ \overline{e}_1^2 \ldots \overline{e}_p^2)$ bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$ e, onde todos os termos $\overline{e}_1^1, \ldots, \overline{e}_p^1, \overline{e}_1^2, \ldots, \overline{e}_p^2$ são flexíveis porque a função de pré-cozimento leva termos flexíveis em termos flexíveis. Depois de uma aplicação de **Dec-App** obteremos a conjunção $\overline{e}_1^1 =^?_{\lambda\sigma} \overline{e}_1^2 \wedge \ldots \wedge \overline{e}_p^1 =^?_{\lambda\sigma} \overline{e}_p^2$ que possui todas as equações bem tipadas no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$ e todas as demais equações permanecem inalteradas (veja Figura 3.1).

Este processo é repetido exaustivamente para todas as equações da forma rígida-rígida de P_α. Observe que este processo de simplificação não altera nenhuma outra equação do problema já que não envolve substituições.

Tome P^{**} como sendo o problema de unificação obtido de P^* após exaustivas aplicações de **Dec-λ** e **Dec-App** às suas equações da forma rígida-rígida.

(pi.2): *O problema derivado de P_α é obtido após uma aplicação de MATCH*:

Seja $P_{\alpha k}$ $(k > 0)$ um problema de unificação gerado a partir desta aplicação de MATCH. Como um passo de imitação sempre gera um problema contendo uma equação da forma rígida-rígida e um passo de projeção sempre gera um problema que possui uma equação da forma flexível-rígida ou rígida-rígida concluímos que $P_{\alpha k}$ contém (pelo menos) uma equação flexível-rígida ou rígida-rígida.

Assumimos que P_α contém (pelo menos) uma equação flexível-rígida da forma:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(X\ e_1^1 \ldots e_{p_1}^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e_1^2 \ldots e_{p_2}^2) \tag{3.8}$$

bem tipada no contexto Γ e onde

- $n, p_1, p_2 \geq 0$;
- X é uma meta-variável de tipo $B_1 \to \ldots \to B_{p_1} \to A$ com A atômico;
- $\underline{h}$ é um índice de tipo $C_1 \to \ldots \to C_{p_2} \to A$ com A atômico;
- se $p_1 > 0$ então e_i^1 são termos de tipo B_i em forma η-longa para todo $1 \leq i \leq p_1$;
- se $p_2 > 0$ então e_j^2 são termos de tipo C_j em forma η-longa para todo $1 \leq j \leq p_2$;

Dividimos a análise em dois casos:
(pi.2.1): *Imitação*: Sabemos que uma imitação só é possível quando a cabeça do termo rígido é uma constante, isto é, quando $h > n$. Neste caso a substituição de imitação é dada por:

$$X/\lambda_{B_1} \ldots \lambda_{B_{p_1}}.(\underline{h - n + p_1}\ (H_1\ \underline{p_1} \ldots \underline{1}) \ldots (H_{p_2}\ \underline{p_1} \ldots \underline{1}))$$

onde $p_2 \geq 0$ e, se $p_2 > 0$ então as H_i's são meta-variáveis novas de tipo $B_1 \to \ldots \to B_{p_1} \to C_i$ para todo $1 \leq i \leq p_2$.

Aplicando esta substituição na equação (3.8), obtemos o problema $P_{\alpha k}$ que contém a seguinte equação:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ (H_1\ e_1^1 \ldots e_{p_1}^1) \ldots (H_{p_2}\ e_1^1 \ldots e_{p_1}^1)) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e_1^2 \ldots e_{p_2}^2)$$

que é bem tipada no contexto Γ.

Por hipótese de indução existe uma derivação de P_F que gera o problema de unificação P^* contendo uma equação da forma:

$$Y[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2) \tag{3.9}$$

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Como $h > n$ após uma aplicação **Exp-App** obteremos um problema de unificação contendo a seguinte conjunção:

$$Y[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2) \wedge Y =^?_{\lambda\sigma} (\underline{h - n + p_1}\ W_1 \ldots W_{p_2})$$

onde as W_i's são meta-variáveis novas com o mesmo tipo de $\overline{e}_i^2$ para todo $1 \leq i \leq p_2$. Após uma aplicação de **Replace** e **Normalise** ao problema atual obteremos um novo problema contendo a equação:

$$(\underline{h}\ W_1[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] \ldots W_{p_2}[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n]) =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2)$$

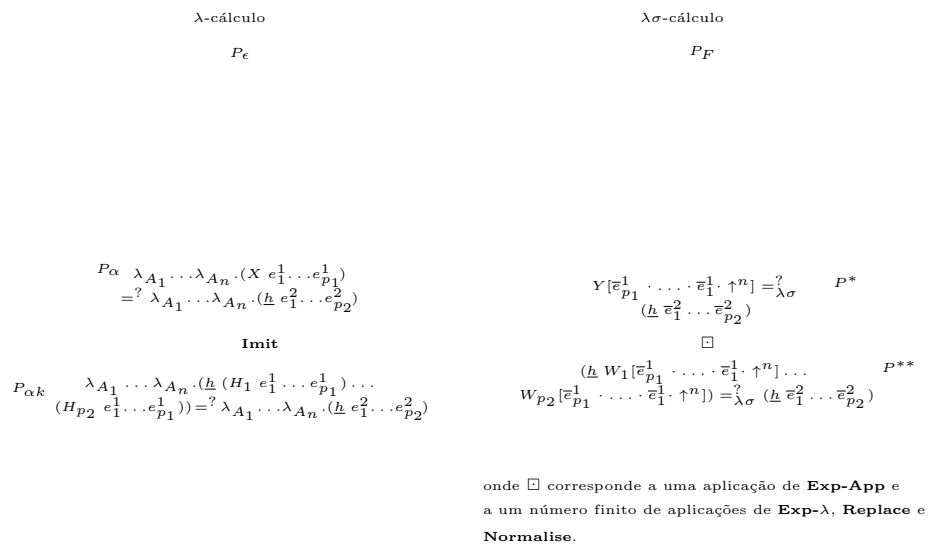


Figura 3.2: Um passo de imitação

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Tomamos este novo problema obtido como sendo P^{**}. A Figura 3.2 ilustra como é feito este passo de imitação. Observe que só existe uma possibilidade de alguma equação ser eliminada durante esta redução: quando o índice $\underline{h}$ na equação (3.9) for de tipo atômico pois neste caso nenhuma meta-variável nova é introduzida e uma equação trivial é gerada. Mas note que neste caso, um equação trivial é também gerada em $P_{\alpha k}$ e, o resultado vale.

(*pi*.2.2): *Projeção*: Neste caso a cabeça X do termo flexível da equação (3.8) é projetada sobre cada um de seus argumentos cujo tipo alvo coincida com o tipo alvo de X. Sem perda de generalidade suponha que X possa ser projetado sobre o seu l-ésimo argumento para algum $1 \leq l \leq p_1$, isto é, suponha que o tipo de e^1_l seja da forma $D_1 \to \ldots \to D_q \to A$ com $q \geq 0$. A substituição de projeção é dada por:

$$X/\lambda_{B_1}\ldots\lambda_{B_{p_1}}.(\underline{p_1 - l + 1}\ (H_1\ \underline{p_1}\ldots\underline{1})\ldots(H_q\ \underline{p_1}\ldots\underline{1}))$$

e se $q > 0$ então as H_i's são meta-variáveis novas de tipo $B_1 \to \ldots \to B_{p_1} \to D_i$ para todo $1 \leq i \leq q$. Após a aplicação desta substituição, obtemos um problema contendo a seguinte equação:

$$\lambda_{A_1}\ldots\lambda_{A_n}.(e^1_l\ (H_1\ e^1_1\ldots e^1_{p_1})\ldots(H_q\ e^1_1\ldots e^1_{p_1})) =^?\ \lambda_{A_1}\ldots\lambda_{A_n}.(\underline{h}\ e^2_1\ldots e^2_{p_2}) \quad (3.10)$$

bem tipada no contexto Γ. Consideramos agora dois subcasos:

(*pi*.2.2.1): *A cabeça do termo* e^1_l *é um índice* à la *de Bruijn*:

Sabendo que termos estão sempre em forma η-longa, podemos supor sem perda de generalidade, que e_l^1 é da forma $\lambda_{D_1} \ldots \lambda_{D_q}.(\underline{k}\ f_1 \ldots f_s)$ e, normalizando a equação (3.10) obteremos uma nova equação de alguma das duas formas seguintes:

- $k > q$:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(\underline{k-q}\ f_1^1 \ldots f_s^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e_1^2 \ldots e_{p_2}^2) \tag{3.11}$$

- $k \leq q$:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(H_{q-k+1}\ e_1^1 \ldots e_{p_1}^1\ f_1^1 \ldots f_s^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e_1^2 \ldots e_{p_2}^2) \tag{3.12}$$

bem tipadas no contexto Γ e, onde f_j^1 é obtido de f_j substituindo-se todas as suas ocorrências livres dos índices $\underline{1}, \ldots, \underline{q}$, respectivamente, por $(H_q\ e_1^1 \ldots e_{p_1}^1), \ldots, (H_1\ e_1^1 \ldots e_{p_1}^1)$.

Por hipótese de indução, existe um problema de unificação P^* derivado de P_F contendo a seguinte equação:

$$Y[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] =_{\lambda\sigma}^? (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2) \tag{3.13}$$

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Como a função de pré-cozimento preserva tipos, temos que o tipo alvo de $\overline{e}_l^1$ coincide com o tipo de Y que, por sua vez possui o tipo alvo de X da equação (3.8). Assim uma aplicação de **Exp-App** à equação (3.13) vai gerar um problema de unificação contendo a seguinte conjunção:

$$Y[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] =_{\lambda\sigma}^? (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2) \wedge Y =_{\lambda\sigma}^? (\underline{p_1 - l + 1}\ W_1 \ldots W_q)$$

onde W_i é uma meta-variável nova de tipo D_i para todo $1 \leq i \leq q$. Após uma aplicação de **Replace** e **Normalise** obteremos um problema de unificação que contém a seguinte equação:

$$(\overline{e}_l^1\ W_1[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] \ldots W_q[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n]) =_{\lambda\sigma}^? (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2) \tag{3.14}$$

que é bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Como $\overline{e}_l^1 = \lambda_{D_1} \ldots \lambda_{D_q}.(\underline{k}\ \overline{f}_1 \ldots \overline{f}_s)$ então temos as seguintes reduções:

- $k > q$:

$$((\lambda_{D_1} \ldots \lambda_{D_q}.\underline{k}\ \overline{f}_1 \ldots \overline{f}_s)\ W_1[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] \ldots W_q[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n]) \rightarrow_{\lambda\sigma}^*$$

$$(\underline{k-q}\ \overline{f}_1[W_q[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] \cdot \ldots \cdot W_1[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] \cdot id] \ldots$$

$$\overline{f}_s[W_q[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] \cdot \ldots \cdot W_1[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] \cdot id]).$$

• $k \leq q$:

$$((\lambda_{D_1} \ldots \lambda_{D_q}.\underline{k}\ \overline{f}_1 \ldots \overline{f}_s)\ W_1[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] \ldots W_q[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n]) \rightarrow^*_{\lambda\sigma}$$

$$(W_{q-k+1}[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n]\ \overline{f}_1[W_q[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] \cdot \ldots \cdot W_1[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] \cdot id] \ldots$$
$$\overline{f}_s[W_q[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] \cdot \ldots \cdot W_1[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] \cdot id])$$

Agora note que para todo $1 \leq j \leq q$, $W_j[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n]$ corresponde à forma pseudo-cozida do subtermo $(H_j\ e^1_1 \ldots e^1_{p_1})$ que ocorre na equação (3.10) no escopo de n abstratores. E assim, para todo $1 \leq i \leq s$, o termo:

$$\overline{f}_i[W_q[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] \cdot \ldots \cdot W_1[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] \cdot id]$$

corresponde a forma pseudo-cozida de f^1_i, denotada por $\overline{f}^1_i$, de forma que o problema de unificação até então obtido possui uma das seguintes equações de acordo com o valor de k:

• $k > q$

$$(\underline{k-q}\ \overline{f}^1_1 \ldots \overline{f}^1_r) =^?_{\lambda\sigma} (\underline{h}\ \overline{e}^2_1 \ldots \overline{e}^2_{p_2})$$

• $k \leq q$

$$M[\overline{f}^1_r \cdot \ldots \cdot \overline{f}^1_1 \cdot \overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{h}\ \overline{e}^2_1 \ldots \overline{e}^2_{p_2})$$

após aplicações de **Exp-λ**, **Replace** e **Normalise**, onde M é uma meta-variável nova de tipo A e ambas as equações são bem tipadas no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Tome P^{**} como sendo o problema de unificação atual.

(pi.2.2.2): *A cabeça do termo e^1_l é uma meta-variável*:

Neste caso, e^1_l tem a forma $\lambda_{D_1} \ldots \lambda_{D_q}.(Z\ f_1 \ldots f_s)$ e, normalizando a equação (3.10) obteremos o problema $P_{\alpha k}$ que contém a seguinte equação:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(Z\ f^1_1 \ldots f^1_s) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e^2_1 \ldots e^2_{p_2})$$

bem tipada no contexto Γ e, como no caso anterior, f^1_j é obtido de f_j substituindo-se todas as ocorrências livres dos índices $\underline{1}$, ..., $\underline{q}$ respectivamente por $(H_q\ e^1_1 \ldots e^1_{p_1})$, ..., $(H_1\ e^1_1 \ldots e^1_{p_1})$.

Por hipótese, existe um problema de unificação P^*, derivado de P_F, que contém a equação (3.13) e, que após uma aplicação de **Exp-App**, **Replace** e **Normalise** gera um novo problema contendo a equação (3.14). Como

$$\overline{e}^1_l = \lambda_{D_1} \ldots \lambda_{D_q}.(Z[\uparrow^{n+q}]\ \overline{f}_1 \ldots \overline{f}_s)$$

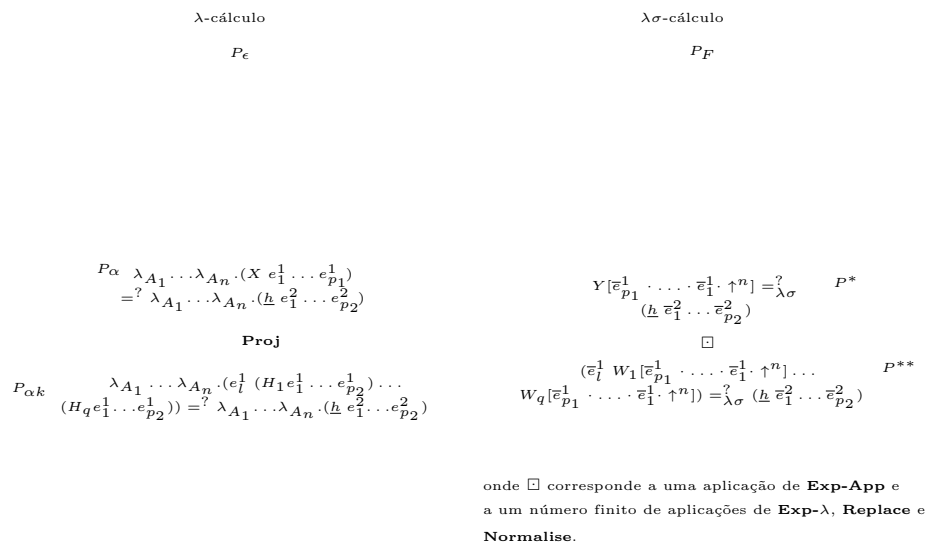


Figura 3.3: Um passo de projeção

concluímos que a forma normal da equação (3.14) é dada por:

$$U[\overline{f}^1_s\cdot\ldots\cdot\overline{f}^1_1\cdot\uparrow^n] =^?_{\lambda\sigma} (\underline{h}\ \overline{e}^2_1\ldots\overline{e}^2_{p_2}) \tag{3.15}$$

bem tipada no contexto $A_1\cdot\ldots\cdot A_n\cdot\Gamma$, onde U á uma meta-variável nova de tipo A. Tome P^{**} como sendo o problema de unificação que contém a equação (3.15). A Figura 3.3 ilustra um passo de projeção no λ-cálculo e no $\lambda\sigma$-cálculo. □

No Lema 3.4 estabelecemos uma relação entre a estrutura das equações que ocorrem em subproblemas gerados durante o processo de unificação no λ-cálculo simplesmente tipado em notação *à la* de Bruijn e suas contrapartes no $\lambda\sigma$-cálculo simplesmente tipado. Este lema é fundamental para os resultados que se seguem e relacionam soluções de subproblemas gerados pelos procedimentos de unificação. De fato, na proposição a seguir mostraremos que se $\mathcal{A}(P)$ é a árvore de unificação de um dado problema P, então para cada subárvore de $\mathcal{A}(P)$, existe uma subárvore da árvore de derivação de P_F com o mesmo número de ramos de sucesso e falha. Mais ainda, as versões pré-cozidas dos subproblemas de P podem ser obtidas como subproblemas de P_F.

Apresentaremos a seguir um novo conjunto de regras adicionalmente ao conjunto **Unif** e que de certa forma desfaz o trabalho feito pelas regras **Exp**-λ e **Dec**-λ. Este conjunto é chamado de **Back** e é dado na Tabela 3.1 (cf. [DHK00]). De fato, enquanto **Dec**-λ remove os abstratores externos de uma equação, a regra **Anti-Dec**-λ inclui novamente tais abstratores permitindo assim que a equação volte a ter o mesmo contexto do problema inicial. Da mesma forma a regra **Anti-Exp**-λ

Anti-Exp-λ	$\dfrac{P}{\exists Y(P \wedge X =^?_{\lambda\sigma} (Y[\uparrow]\ \underline{1}))}$ se $X \in Var(P)$ é tal que $\Gamma_X = A \cdot \Gamma'_X$ onde $Y \in \mathcal{X}, Y \notin Var(P)$ e $T_y = A \to T_X, \Gamma_Y = \Gamma'_X$
Anti-Dec-λ	$\dfrac{P \wedge a =^?_{\lambda\sigma} b}{P \wedge \lambda_A.a =^?_{\lambda\sigma} \lambda_A.b}$ se $a =^?_{\lambda\sigma} b$ está bem tipada no contexto $A \cdot \Delta$.

Tabela 3.1: Regras Back

recupera as meta-variáveis de tipo funcional que foram atomizadas por aplicações de **Exp**-λ. Desta forma, as regras definidas em **Back** são essenciais para que possamos construir a forma pré-cozida de subproblemas de P.

Proposição 3.5 (Correspondência entre Árvores) *Seja Γ um contexto, P um problema de unificação no λ-cálculo simplesmente tipado e $\mathcal{A}(P)$ sua árvore de unificação. Para cada problema P_α em $\mathcal{A}(P)$ existe um problema de unificação P^*, derivado de P_F via a estratégia* **Unif***, tal que:*

1. *Se P_α contém um ramo que termina em um nó de sucesso, então P^* contém um ramo que termina em uma forma resolvida.*

2. *Se P_α contém um ramo que termina em um nó de falha, então P^* contém um ramo que termina em um nó de falha.*

3. *Se P_α é formado pelas equações $eq_1, \ldots, eq_s$ bem tipadas no contexto Γ, então existe um problema de unificação P^*_B, derivado de P^* utilizando-se a estratégia* **Back***, contendo as equações $eq_{1_F}, \ldots, eq_{s_F}$ bem tipadas no contexto Γ a menos de renomeamento de meta-variáveis. Mais ainda, qualquer outra equação existente em P^*_B é da forma flexível-flexível ou resolvida.*

Prova. A prova é por análise de casos:

1. Suponha que a árvore de P_α contenha um ramo contendo um nó de sucesso e, seja $P_{\alpha\gamma}$ o pai deste nó de sucesso. Consideraremos dois casos:

1.1: *$P_{\alpha\gamma}$ é sucedido por uma aplicação de SIMPL.* Neste caso, $P_{\alpha\gamma}$ contém (pelo menos) uma equação rígida-rígida da forma:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(\underline{k}\ e_1^1 \ldots e_p^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{k}\ e_1^2 \ldots e_p^2) \tag{3.16}$$

bem tipada no contexto Γ e, tal que todos os termos $e_1^1, \ldots, e_p^1, e_1^2, \ldots, e_p^2$ são flexíveis ou, caso algum destes e_j^i não seja flexível (para algum $1 \leq i \leq 2, 1 \leq j \leq p$), então $e_j^1 = e_j^2$; caso contrário o problema resultante não estaria ainda solucionado e não poderia ser um nó de sucesso. Após a aplicação de SIMPL todas as equações da forma rígida-rígida são substituídas por novas equações da forma flexível-flexível de modo que o problema assim obtido, denotado por $\overline{P}_{\alpha\gamma}$, contém apenas equações da forma flexível-flexível e, portanto $\overline{P}_{\alpha\gamma}$ é um nó de sucesso.

De acordo com o Lema 3.4, existe um problema de unificação P^*, derivado de P_F e, que contém uma equação para cada equação existente em $P_{\alpha\gamma}$. Em particular, para cada equação da forma (3.16) existe uma equação em P^* da forma:

$$(\underline{k}\ \overline{e}_1^1 \ldots \overline{e}_p^1) =^?_{\lambda\sigma} (\underline{k}\ \overline{e}_1^2 \ldots \overline{e}_p^2) \tag{3.17}$$

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$ e, onde todos os termos $\overline{e}_1^1, \ldots, \overline{e}_p^1, \overline{e}_1^2, \ldots, \overline{e}_p^2$ são flexíveis, ou caso algum destes $\overline{e}_j^i$ não seja flexível (para algum $1 \leq i \leq 2, 1 \leq j \leq p$) então $\overline{e}_j^1 = \overline{e}_j^2$. Assim, após um número finito de aplicações de **Dec-App** todas as equações da forma (3.17) são substituídas por um número finito de equações da forma flexível-flexível. Assim o problema de unificação obtido contém apenas um número finito de equações da forma flexível-flexível e, portanto é uma forma resolvida.

1.2: *$P_{\alpha\gamma}$ é sucedido por uma aplicação de MATCH.* Neste caso, $P_{\alpha\gamma}$ contém (pelo menos) uma equação flexível-rígida e, o nó de sucesso será denotado por $P_{\alpha\gamma k}$ $(k > 0)$. No entanto, se $P_{\alpha\gamma}$ possui mais de uma equação flexível-rígida, então todas estas equações precisam ter a mesma meta-variável como cabeça do termo flexível; caso contrário, o problema resultante não seria um nó de sucesso. Além disto todas as possíveis equações flexível-rígidas têm a forma:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(X\ e_1 \ldots e_p) =^? \lambda_{A_1} \ldots \lambda_{A_n}.\underline{h} \tag{3.18}$$

bem tipada no contexto Γ.

1.2.1: Se a aplicação de MATCH consiste em um passo de imitação então temos que $h > n$ e, como a substituição gerada não introduz novas meta-variáveis a

equação resultante é trivial. Do Lema 3.4, sabemos que existe um problema de unificação P^*, derivado de P_F, e contendo para equação da forma:

$$Y[\overline{e}_p \cdot \ldots \cdot \overline{e}_1 \cdot \uparrow^n] =^?_{\lambda\sigma} \underline{h}$$

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Como $h > n$, após uma aplicação de **Exp-App** seguida de **Replace** e **Normalise** obtemos uma forma resolvida.

1.2.2: Se por outro lado, a aplicação de MATCH consiste em um passo de projeção, então todas as equações da forma (3.18) têm $\underline{h}$ como sendo o argumento a ser projetado, gerando após normalização, um nó de sucesso. De acordo com o Lema 3.4, existe um problema de unificação P^*, derivado de P_F, e contendo para cada equação da forma (3.18) uma equação da forma:

$$Y[\overline{e}_p \cdot \ldots \cdot \overline{e}_1 \cdot \uparrow^n] =^?_{\lambda\sigma} \underline{h}$$

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Após uma aplicação de **Exp-App** seguida de **Replace** e **Normalise**, todas estas equações flexível-rígidas se convertem em equações triviais, gerando assim uma forma resolvida.

2. Considere um nó de falha de P_α. O último passo para se obter este nó consiste em uma aplicação de SIMPL ao nó antecedente que denotaremos por $P_{\alpha\gamma}$ que contém uma equação rígida-rígida com cabeças distintas:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(\underline{k}\ e^1_1 \ldots e^1_{p_1}) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e^2_1 \ldots e^2_{p_2})$$

bem tipada no contexto Γ e tal que $h \neq k$. Pelo Lema 3.4, existe um problema de unificação P^*, derivado de P_F, contendo a equação:

$$(\underline{k}\ \overline{e}^1_1 \ldots \overline{e}^1_{p_1}) =^?_{\lambda\sigma} (\underline{h}\ \overline{e}^2_1 \ldots \overline{e}^2_{p_2})$$

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Após uma aplicação de **Dec-Fail** obteremos um nó de falha.

3. Suponha que $P_\alpha = eq_1 \wedge \ldots \wedge eq_s$ $(s > 0)$. É suficiente provar que para uma dada equação eq_j $(1 \leq j \leq s)$ de P_α conseguimos obter a equação eq_{j_F} a partir do problema P^* construído no Lema 3.4, utilizando-se a estratégia **Back**. De fato, a estratégia **Back** provoca mudanças apenas na equação que está sendo considerada, o que justifica esta análise local. Faremos a análise de acordo com a estrutura da equação eq_j considerada:

3.1: eq_j *é uma equação flexível-rígida*: Neste caso, eq_j tem a forma

$$\lambda_{A_1} \ldots \lambda_{A_n}.(X\ e_1^1 \ldots e_{p_1}^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e_1^2 \ldots e_{p_2}^2)$$

bem tipada no contexto Γ e, onde X é uma meta-variável de tipo $B_1 \to \ldots \to B_{p_1} \to A$ (A atômico). De acordo com o Lema 3.4, existe um problema de unificação P^*, derivado de P_F, contendo uma equação da forma:

$$Y[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2)$$

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$, onde Y é uma meta-variável de tipo A no contexto $B_{p_1} \cdot \ldots \cdot B_1 \cdot \Gamma$. Após uma aplicação de **Anti-Exp-**λ obteremos a conjunção:

$$Y[\overline{e}_{p_1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2) \wedge Y =^?_{\lambda\sigma} (Z[\uparrow]\ \underline{1})$$

onde Z é uma meta-variável nova de tipo $B_{p_1} \to A$ e contexto $B_{p_1-1} \cdot \ldots \cdot B_1 \cdot \Gamma$. Após uma aplicação de **Replace** e **Normalise**, obteremos um problema de unificação contendo a conjunção:

$$(Z[\overline{e}_{p_1-1}^1 \cdot \ldots \cdot \overline{e}_1^1 \cdot \uparrow^n]\ \overline{e}_{p_1}^1) =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2) \wedge Y =^?_{\lambda\sigma} (Z[\uparrow]\ \underline{1}).$$

Repetindo esta estratégia mais $p_1 - 1$ vezes, obteremos um problema de unificação contendo a seguinte equação:

$$(W[\uparrow^n]\ \overline{e}_1^1 \ldots \overline{e}_{p_1}^1) =^?_{\lambda\sigma} (\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2)$$

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Ao final de n aplicações de **Anti-Dec-**λ teremos:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(W[\uparrow^n]\ \overline{e}_1^1 \ldots \overline{e}_{p_1}^1) =^?_{\lambda\sigma} \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ \overline{e}_1^2 \ldots \overline{e}_{p_2}^2)$$

bem tipada no contexto Γ e, a partir deste ponto aplicamos esta estratégia sempre que possível a todos os subtermos $\overline{e}_j^i$ $(1 \leq i \leq 2; 1 \leq j \leq p_1)$ desta equação, obtendo assim $e_{j_F}^i$ a menos de renomeamento de meta-variáveis. Depois disto obteremos um problema de unificação contendo uma equação que corresponde a eq_{j_F} a menos de renomeamento de meta-variáveis.

3.2 eq_j *é uma equação rígida-rígida*: Neste caso, eq_j tem a forma:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(\underline{k}\ e_1^1 \ldots e_{p_1}^1) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(\underline{h}\ e_1^2 \ldots e_{p_2}^2)$$

bem tipada no contexto Γ. De acordo com o Lema 3.4, existe um problema de unificação P^*, derivado de P_F e contendo uma equação da forma:

$$(\underline{k}\ \overline{e}^1_1 \ldots \overline{e}^1_{p_1}) =^? (\underline{h}\ \overline{e}^2_1 \ldots \overline{e}^2_{p_2})$$

bem tipada no contexto $A_1 \cdot \ldots \cdot A_n \cdot \Gamma$. Utilizando-se a mesma estratégia do item anterior recursivamente a todos os subtermos $\overline{e}^1_1, \ldots, \overline{e}^1_{p_1}, \overline{e}^2_1, \ldots, \overline{e}^2_{p_2}$ obteremos respectivamente os termos $e^1_{1_F}, \ldots, e^1_{p_{1F}}, e^2_{1_F}, \ldots, e^2_{p_{2F}}$ e, após n aplicações de **Anti-Dec-λ** obteremos um problema de unificação contendo uma equação que corresponde a eq_{j_F} a menos de renomeamento de meta-variáveis.

3.3 *eq_j é uma equação flexível-flexível*: Neste caso, eq_j tem a forma:

$$\lambda_{A_1} \ldots \lambda_{A_n}.(X\ e^1_1 \ldots e^1_{p_1}) =^? \lambda_{A_1} \ldots \lambda_{A_n}.(Y\ e^2_1 \ldots e^2_{p_2})$$

bem tipada no contexto Γ, onde X é uma meta-variável de tipo $B_1 \to \ldots \to B_{p_1} \to A$ (A atômico) e Y é uma meta-variável de tipo $C_1 \to \ldots \to C_{p_2} \to A$ (A atômico). De acordo com o Lema 3.4, existe um problema de unificação P^*, derivado de P_F, e contendo a equação:

$$Z[\overline{e}^1_{p_1} \cdot \ldots \cdot \overline{e}^1_1 \cdot \uparrow^n] =^?_{\lambda\sigma} W[\overline{e}^2_{p_2} \cdot \ldots \cdot \overline{e}^2_1 \cdot \uparrow^n]$$

onde Z e W são meta-variáveis de tipos A. Repetindo a estratégia utilizada para equações flexível-rígidas acima obteremos uma equação que corresponde a eq_{j_F} a menos de renomeamento de meta-variáveis.

Durante o processo de unificação no $\lambda\sigma$-cálculo novas equações podem ser introduzidas a um dado problema de unificação. Isto pode ocorrer após uma aplicação de **Exp-λ**, **Exp-App** ou **Anti-Exp-λ**. Vejamos que cada uma destas novas equações que são introduzidas são da forma flexível-flexível ou resolvida e, assim permanecem durante todo o processo de unificação:

- Uma aplicação de **Exp-λ** introduz a equação $X =^?_{\lambda\sigma} \lambda_A.Y$ em um problema de unificação que contenha ocorrências da variável funcional X.
- Após uma aplicação de **Replace** ao problema corrente esta equação passa a ser resolvida já que X não vai ocorrer em nenhuma outra equação do problema. Posteriores substituições para Y (ou outras meta-variáveis que venham a ocorrer do lado direito desta equação) são irrelevantes já que a mesma permanece resolvida.

- Uma aplicação de **Exp-App** introduz uma equação flexível-rígida, digamos $X =^?_{\lambda\sigma} a$ ao problema corrente como resultado de uma imitação ou projeção. Da mesma forma, após uma aplicação de **Replace** ao problema corrente, esta equação passa a ter a forma resolvida porque X não vai mais ocorrer em nenhuma outra equação do problema. Posteriores substituições no termo a não mudam o status desta equação, isto é, ela permanece resolvida.

- Por fim, uma aplicação de **Anti-Exp-λ** introduz uma equação flexível-flexível que após uma aplicação de **Replace** também adquire o status de equação resolvida.

□

O exemplo a seguir ilustra o conteúdo da proposição anterior com a utilização da estratégia **Back**.

Exemplo 3.6 *Considere o problema de unificação apresentado no Exemplo 2.6 cuja árvore de unificação é dada na Figura 2.3. Considere o subproblema:*

$$\overline{P}_1 = \{\lambda_{B\to B}.(X_1\ \underline{3}) =^? \lambda_{B\to B}.(\underline{4}\ \underline{3})\}$$

de $\mathcal{A}(P)$ cuja árvore de unificação possui um ramo de falha e dois ramos de sucesso. O problema de unificação P^*, derivado de P_F, de acordo com o Lema 3.4 é dado por:

$$P^* = \{H_1[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} (\underline{4}\ \underline{3}) \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ H_1) \wedge Y =^?_{\lambda\sigma} (\underline{2}\ H_1)\}$$

cuja árvore de derivação, que também contém um ramo de falha e dois ramos de sucesso, é dada na Figura 2.4. Aplicando a estratégia **Back** *a P^* obteremos o problema:*

$$P^*_B = \{\lambda_{B\to B}.(N[\uparrow]\ \underline{3}) =^?_{\lambda\sigma} \lambda_{B\to B}.(\underline{4}\ \underline{3}) \wedge X =^?_{\lambda\sigma} \lambda_A.(\underline{2}\ (N[\uparrow]\ \underline{1}))\wedge$$

$$Y =^?_{\lambda\sigma} (\underline{2}\ (N[\uparrow]\ \underline{1})) \wedge H_1 =^?_{\lambda\sigma} (N[\uparrow]\ \underline{1}\})$$

onde a equação:

$$\lambda_{B\to B}.(N[\uparrow]\ \underline{3}) =^?_{\lambda\sigma} \lambda_{B\to B}.(\underline{4}\ \underline{3})$$

corresponde a forma pré cozida de:

$$\lambda_{B\to B}.(X_1\ \underline{3}) =^? \lambda_{B\to B}.(\underline{4}\ \underline{3})$$

a menos de renomeamento de meta-variáveis e todas as outras equações são flexível-flexível ou resolvidas. A solução σ *de* $\overline{P}_1$ *é dada por* $\sigma = \{X_1/\lambda_A.(\underline{4}\ \underline{1}), X_1/\lambda_A.(\underline{4}\ \underline{3})\}$. *É fácil verificar que o* grafting $\sigma_F = \{N \mapsto \lambda_A.(\underline{4}\ \underline{1}), N \mapsto \lambda_A.(\underline{4}\ \underline{3})\}$, *obtido de* σ *renomeando* X_1 *por* N, *é solução de* P_B^*.

O lema a seguir nos permite "recuperar" as formas pré-cozidas no $\lambda\sigma$-cálculo dos subproblemas que são gerados na árvore de unificação de um dado problema de unificação P do λ-cálculo.

Corolário 3.7 (Correspondência entre Soluções) *Seja* P *um problema de unificação no* λ*-cálculo simplesmente tipado e* $\mathcal{A}(P)$ *sua árvore de unificação. Para cada subproblema de unificação* P_α *em* $\mathcal{A}(P)$ *com solução* σ, *existe um problema de unificação* P_B^*, *derivado de* P_F *via as estratégias* **Unif** *e* **Back**, *que após um renomeamento adequado de meta-variáveis, tem como solução o grafting* σ_F.

Prova. Seja Γ um contexto e P_α um problema de unificação em $\mathcal{A}(P)$ bem tipado no contexto Γ. De acordo com a Proposição 3.5, existe uma derivação de P_F utilizando as estratégias **Back** e **Unif** que gera um problema de unificação denotado por P_B^* que contém todas as equações de P_{α_F} a menos de renomeamento de meta-variáveis. Além disto, todas as outras equações existentes em P_B^* são da forma flexível-flexível ou resolvidas. Em [DHK00], Dowek, Hardin e Kirchner provam que se a substituição (de ordem superior) σ é uma solução de $a =^? b$ então o *grafting* σ_F é solução de $a_F =^?_{\lambda\sigma} b_F$. Daí concluímos que σ_F é uma solução de P_B^* após um renomeamento adequado de meta-variáveis.

□

Neste capítulo apresentamos uma relação estrutural existente entre o algoritmo de unificação de Huet para o λ-cálculo simplesmente tipado e o método de unificação desenvolvido por Dowek, Hardin e Kirchner para o $\lambda\sigma$-cálculo simplesmente tipado. Em [DHK00], Dowek, Hardin e Kirchner concluem que unificação no $\lambda\sigma$-cálculo simplemente tipado é uma generalização do algoritmo de Huet já que cada solução de um problema de unificação computada no $\lambda\sigma$-cálculo é também computada pelo algoritmo de Huet e, que um problema P tem solução no λ-cálculo se, e somente se, sua forma pré-cozida P_F tem uma solução no $\lambda\sigma$-cálculo. Neste capítulo refinamos este resultado mostrando uma relação explícita existente entre possíveis soluções de subproblemas. Assim, para cada subproblema P_α, com solução σ, de um dado

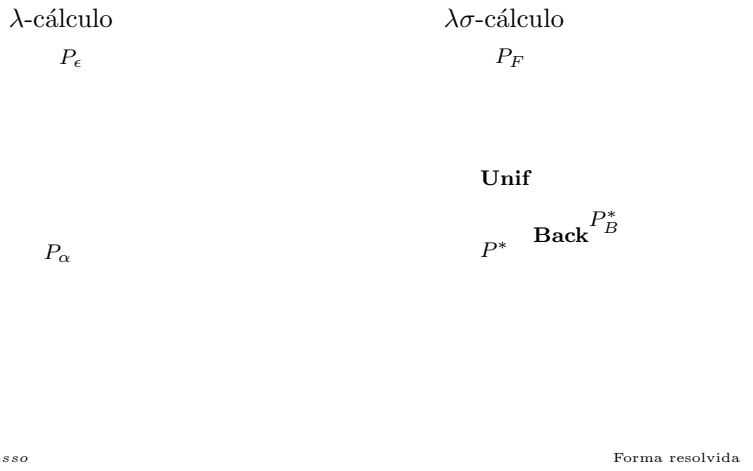


Figura 3.4: Esquema comparativo de unificação entre o λ-cálculo e λσ-cálculo

problema P, sempre podemos encontrar um subproblema da forma pré-cozida P_F que aceita o *grafting* σ_F como solução.

Capítulo 4

Relacionando os Métodos de Unificação no $\lambda\sigma$-cálculo e no λs_e-cálculo

Neste capítulo faremos uma comparação entre os métodos de unificação do $\lambda\sigma$-cálculo e do λs_e-cálculo. Em [ARK01] o método apresentado na Seção 2.4 foi adaptado para o λs_e-cálculo. Inicialmente contruiremos funções que traduzem termos do λs_e-cálculo para o $\lambda\sigma$-cálculo e vice-versa. Em seguida, mostraremos que uma regra de unificação pode ser aplicada no λs_e-cálculo para um problema P se, e somente se, a regra de unificação "correspondente" do $\lambda\sigma$-cálculo pode ser aplicada à tradução do problema P para o $\lambda\sigma$-cálculo.

4.1 Correspondência a partir do λs_e-cálculo

Nesta seção, mostraremos como a unificação no λs_e-cálculo corresponde a unificação no $\lambda\sigma$-cálculo. A seguir apresentaremos algumas definições importantes.

Definição 4.1 (Esqueleto de um termo em forma λs_e-normal) *O* esqueleto *de um termo a, denotado por $sk(a)$, em forma λs_e-normal cujo principal operador é igual a σ ou φ é definido recursivamente por:*

- *Se a tem a forma $X\sigma^i b$ ou $\varphi_j^i X$ então $sk(a) = \psi_i^0(X, b)$ ou $sk(a) = \psi_i^j(X)$, respectivamente.*
- *Se a é da forma $b\sigma^i c$ ou $\varphi_i^j b$, onde b e c são termos em forma λs_e-normal e*

$sk(b) = \psi_{i_k}^{j_k} \ldots \psi_{i_1}^{j_1}(X, b_1, \ldots, b_{k'})$, *onde* $k' \leq k$, *então*

$$sk(a) = \psi_i^0 \psi_{i_k}^{j_k} \ldots \psi_{i_1}^{j_1}(X, b_1, \ldots, b_{k'}, c)$$

ou $sk(a) = \psi_j^i \psi_{i_k}^{j_k} \ldots \psi_{i_1}^{j_1}(X, b_1, \ldots, b_{k'})$, *respectivamente.*

Exemplo 4.2 *O esqueleto do termo* $\varphi_{i_5}^{j_5}((\varphi_{i_3}^{j_3}(\varphi_{i_2}^{j_2}(X\sigma^{i_1}a)))\sigma^{i_4}b)$ *é dado por*

$$\varphi_{i_5}^{j_5}\sigma^{i_4}\varphi_{i_3}^{j_3}\varphi_{i_2}^{j_2}\sigma^{i_1}(X, a, b).$$

Lema 4.3 ([ARK01]) *Seja a um termo em forma* λs_e*-normal cujo operador raiz (isto é, o operador mais externo) seja* σ *ou* φ *e seja tal que*

$$sk(a) = \psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p).$$

Então subíndices sucessivos i_k *e* i_{k+1} *satisfazem as seguintes relações:*

1. $i_k > i_{k+1}$ *se* ψ_k *e* ψ_{k+1} *são ambos iguais a* σ *ou a* φ*;*
2. $i_k \geq i_{k+1}$ *se* ψ_k *e* ψ_{k+1} *são iguais a* φ *e* σ*, respectivamente;*
3. $i_k > i_{k+1} + 1$ *se* ψ_k *e* ψ_{k+1} *são iguais a* σ *e* φ*, respectivamente.*

As regras de unificação do λs_e-cálculo são dadas na Tabela 4.1 e, como podemos observar existe uma *correspondência* entre estas e as regras de unificação do $\lambda\sigma$-cálculo (ver Tabela 2.1): as regras **Dec-λ**, **Dec-App**, **App-Fail**, **Exp-λ**, **Exp-App**, **Normalise** e **Replace** do λs_e-cálculo correspondem respectivamente às regras **Dec-λ**, **Dec-App**, **Dec-Fail**, **Exp-λ**, **Exp-App**, **Normalise** e **Replace** do $\lambda\sigma$-cálculo. Para formalizarmos esta correspondência precisamos inicialmente definir funções que transformem termos do λs_e-cálculo em termos do $\lambda\sigma$-cálculo e vice-versa. A seguir definimos a função T que transforma termos do λs_e-cálculo em termos do $\lambda\sigma$-cálculo.

Definição 4.4 *A função* $T : \Lambda_{\lambda s_e}(\mathcal{X}) \to \Lambda_{\lambda\sigma}(\mathcal{X})$ *é definida indutivamente por:*

(a) $T(X) = X$*;*

(b) $T(\underline{n}) = \underline{1}[\uparrow^{n-1}]$*;*

(c) $T(a\ b) = T(a)\ T(b)$*;*

(d) $T(\lambda_A.a) = \lambda_A.T(a)$*;*

Dec-λ	$\dfrac{P \wedge \lambda_A.e_1 =^?_{\lambda s_e} \lambda_A.e_2}{P \wedge e_1 =^?_{\lambda s_e} e_2}$
Dec-App	$\dfrac{P \wedge (\underline{n}\ e_1^1 \dots e_p^1) =^?_{\lambda s_e} (\underline{n}\ e_1^2 \dots e_p^2)}{P \wedge e_1^1 =^?_{\lambda s_e} e_1^2 \wedge \dots \wedge e_p^1 =^?_{\lambda s_e} e_p^2}$
App-Fail	$\dfrac{P \wedge (\underline{n}\ e_1^1 \dots e_{p_1}^1) =^?_{\lambda s_e} (\underline{m}\ e_1^2 \dots e_{p_2}^2)}{Falha}$, se $m \neq n$.
Exp-λ	$\dfrac{P}{\exists(A \cdot \Gamma \vdash Y : B), P \wedge X =^?_{\lambda s_e} \lambda_A.Y}$ se $(\Gamma \vdash X : A \to B) \in \mathcal{TV}ar(P)$, $Y \notin \mathcal{TV}ar(P)$, e X é uma meta-variável não resolvida.
Exp-App	$\dfrac{P \wedge \psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\underline{m}\ b_1 \dots b_q)}{P \wedge \psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\underline{m}\ b_1 \dots b_q) \wedge \bigvee_{r \in R_p \cup R_i} \exists H_1 \dots \exists H_k, X =^?_{\lambda s_e} (\underline{r}\ H_1 \dots H_k)}$ se $\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)$ é o esqueleto de uma forma λs_e-normal, e X é uma meta-variável de tipo atômico não resolvida; onde $H_1, \dots, H_k$ são meta-variáveis novas com tipo apropriado, que não ocorrem em P, com contextos $\Gamma_{H_i} = \Gamma_X$, R_p é um subconjunto de $\{i_1, \dots, i_p\}$ de super-índices do operador σ tal que $(\underline{r}\ H_1 \dots H_k)$ tem o tipo correto, $R_i = \bigcup_{k=0}^{p}$ se $i_k \geq m + p - k - \Sigma_{l=k+1}^{p} j_l > i_{k+1}$ então $\{m + p - k - \Sigma_{l=k+1}^{p} j_l\}$ senão $\emptyset$, onde $i_0 = \infty$ e $i_{p+1} = 0$.
Normalise	$\dfrac{P \wedge e_1 =^?_{\lambda s_e} e_2}{P \wedge e_1' =^?_{\lambda s_e} e_2'}$ se e_1 ou e_2 não está em forma η-longa. onde e_1' (resp. e_2') é a forma η-longa de e_1 (resp. e_2) se e_1 (resp. e_2) não é uma variável resolvida e e_1 (resp. e_2) caso contrário.
Replace	$\dfrac{P \wedge X =^?_{\lambda s_e} t}{\{X \mapsto t\}(P) \wedge X =^?_{\lambda s_e} t}$ se $X \in \mathcal{TV}ar(P), X \notin \mathcal{TV}ar(t)$ e se t é uma meta-variável então $t \in \mathcal{TV}ar(P)$.

Tabela 4.1: Regras de unificação para o λs_e-cálculo

(e) $T(a\sigma^i b) = T(a)[\underline{1}\cdot\underline{2}\cdot\ldots\cdot\underline{i-1}\cdot T(b)[\uparrow^{i-1}]\cdot\uparrow^{i-1}]$, *onde* $i \geq 1$;

(f) $T(\varphi_k^i(a)) = T(a)[\underline{1}\cdot\underline{2}\cdot\ldots\cdot\underline{k}\cdot\uparrow^{k+i-1}]$, *onde* $k \geq 0$ *e* $i \geq 1$.

Exemplo 4.5 *Considere o seguinte termo do* λs_e*-cálculo:*

$$\varphi_1^4(((\varphi_7^3 X)\sigma^5 a)\sigma^3 b)$$

que pode ser reescrito utilizando-se a noção de esqueleto da seguinte forma:

$$\varphi_1^4\sigma^3\sigma^5\varphi_7^3(X,a,b).$$

O termo do $\lambda\sigma$*-cálculo correspondente é obtido por sucessivas aplicações da função* T *seguidas de possíveis passos de normalização, como mostramos a seguir:*
$T(\varphi_1^4\sigma^3\sigma^5\varphi_7^3(X,a,b)) \stackrel{def}{=} T(\sigma^3\sigma^5\varphi_7^3(X,a,b))[\underline{1}\cdot\uparrow^4] \stackrel{def}{=}$
$T(\sigma^5\varphi_7^3(X,a))[\underline{1}\cdot\underline{2}\cdot T(b)[\uparrow^2]\cdot\uparrow^2][\underline{1}\cdot\uparrow^4] \rightarrow_\sigma^*$
$T(\sigma^5\varphi_7^3(X,a))[\underline{1}\cdot\underline{5}\cdot T(b)[\uparrow^5]\cdot\uparrow^5] \stackrel{def}{=}$
$T(\varphi_7^3(X))[\underline{1}\cdot\underline{2}\cdot\underline{3}\cdot\underline{4}\cdot T(a)[\uparrow^4]\cdot\uparrow^4][\underline{1}\cdot\underline{5}\cdot T(b)[\uparrow^5]\cdot\uparrow^5] \rightarrow_\sigma^*$
$T(\varphi_7^3(X))[\underline{1}\cdot\underline{5}\cdot T(b)[\uparrow^5]\cdot\underline{6}\cdot T(a)[\uparrow^6]\cdot\uparrow^6] \stackrel{def}{=}$
$X[\underline{1}\cdot\underline{2}\cdot\underline{3}\cdot\underline{4}\cdot\underline{5}\cdot\underline{6}\cdot\underline{7}\cdot\uparrow^9][\underline{1}\cdot\underline{5}\cdot T(b)[\uparrow^5]\cdot\underline{6}\cdot T(a)[\uparrow^6]\cdot\uparrow^6] \rightarrow_\sigma^*$
$X[\underline{1}\cdot\underline{5}\cdot T(b)[\uparrow^5]\cdot\underline{6}\cdot T(a)[\uparrow^6]\cdot\underline{7}\cdot\underline{8}\cdot\uparrow^{10}]$.

A proposição a seguir nos mostra que a função T preserva o tipo e o contexto dos termos:

Proposição 4.6 *Seja* Γ *um contexto, e* a *um termo em forma* λs_e*-normal de tipo* A. *Se* $\Gamma \vdash a : A$ *então* $\Gamma \vdash T(a) : A$

Prova. Para tornar a notação mais compacta, escreveremos a_A^Γ para denotar o julgamento de tipo $\Gamma \vdash a : A$. A prova é por indução sobre a estrutura de a:

- Se $a = \underline{n}$ então temos que:

$$T(\underline{n}_A^{A_1\cdot\ldots\cdot A_{n-1}\cdot A\cdot\Gamma}) = \underline{1}_A^{A\cdot\Gamma}[(\uparrow^{n-1})_{A\cdot\Gamma}^{A_1\cdot\ldots\cdot A_{n-1}\cdot A\cdot\Gamma}]_A^{A_1\cdot\ldots\cdot A_{n-1}\cdot A\cdot\Gamma}$$

- Se $a = X$ então $T(X_A^\Gamma) = X_A^\Gamma$.

- Se $a = (b\ c)$ então o resultado segue por hipótese de indução.

- Se $a = \lambda_B.b$ então A é da forma $B \to C$ e, por hipótese de indução, temos que $B \cdot \Gamma \vdash T(b) : C$. Após uma aplicação de $(lambda)$ obtemos $\Gamma \vdash \lambda_B.T(b) : B \to C$ que, pela definição de T, nos permite concluir que $\Gamma \vdash T(\lambda_B.b) : B \to C$.

- Se $a = b\sigma^i c$ então pelas regras de tipagem do λs_e-cálculo temos que $a_A^\Gamma = (b_A^{\Gamma_{<i} \cdot B \cdot \Gamma_{\geq i}} \sigma^i c_B^{\Gamma_{\geq i}})_A^\Gamma$ e, por hipótese de indução, obtemos que $\Gamma_{<i} \cdot B \cdot \Gamma_{\geq i} \vdash T(b) : A$ e $\Gamma_{\geq i} \vdash T(c) : B$. Considere a seguinte derivação:

$$\dfrac{\dfrac{\vdots \qquad \dfrac{\dfrac{}{\Gamma \vdash \underline{i-1} : \Gamma_{[i-1]}} \qquad \dfrac{\dfrac{}{\Gamma \vdash T(c)[\uparrow^{i-1}] : B} \qquad \dfrac{}{\Gamma \vdash \uparrow^{i-1} \rhd \Gamma_{\geq i}}}{\Gamma \vdash T(c)[\uparrow^{i-1}] \cdot \uparrow^{i-1} \rhd B \cdot \Gamma_{\geq i}}\, cons}{\Gamma \vdash \underline{i-1} \cdot T(c)[\uparrow^{i-1}] \cdot \uparrow^{i-1} \rhd \Gamma_{[i-1]} \cdot B \cdot \Gamma_{\geq i}}\, cons}{\vdots}}{\Gamma \vdash \underline{1} \cdot \ldots \cdot \underline{i-1} \cdot \cdot T(c)[\uparrow^{i-1}] \cdot \uparrow^{i-1} \rhd \Gamma_{<i} \cdot B \cdot \Gamma_{\geq i}}\, cons$$

  e, portanto

$$\Gamma \vdash T(b)_A^{\Gamma_{<i} \cdot B \cdot \Gamma_{\geq i}}[\underline{1}_{\Gamma_{[1]}}^\Gamma \cdot \ldots \cdot \underline{i-1}_{\Gamma_{[i-1]}}^\Gamma \cdot T(c)_B^{\Gamma_{\geq i}}[(\uparrow^{i-1})_{\Gamma_{\geq i}}^\Gamma] \cdot (\uparrow^{i-1})_{\Gamma_{\geq i}}^\Gamma] : A$$

- Se $a = \varphi_k^i b$ então pelas regras de tipagem do λs_e-cálculo temos que

$$\Gamma \vdash \varphi_k^i(b_A^{\Gamma_{\leq k} \cdot \Gamma_{\geq k+i}}) : A.$$

  Por hipótese de indução, temos que $\Gamma_{\leq k} \cdot \Gamma_{\geq k+i} \vdash T(b) : A$ e, portanto

$$\Gamma \vdash T(b)_A^{\Gamma_{\leq k} \cdot \Gamma_{\geq k+i}}[\underline{1}_{\Gamma_{[1]}}^\Gamma \cdot \ldots \cdot \underline{k}_{\Gamma_{[k]}}^\Gamma \cdot (\uparrow^{k+i-1})_{\Gamma_{\geq k+i}}^\Gamma] : A$$

□

Os lemas a seguir são importantes para que possamos formalizar a relação existente entre as regras **Exp-App** do $\lambda\sigma$- e λs_e-cálculo, que denotaremos respectivamente por **Exp-App**$_{\lambda\sigma}$ e **Exp-App**$_{\lambda s_e}$ quando for necessário para diferenciá-las. Utilizaremos esta notação também para as outras regras quando necessário. Apesar de a função T transformar termos do λs_e-cálculo em termos do $\lambda\sigma$-cálculo, a transformação de uma equação flexível-rígida como a que ocorre na regra **Exp-App** não é imediata. De fato, a função T nos mostra como transformar cada operador por vez.

O lema a seguir fornece a estrutura geral dos termos flexíveis obtidos após sucessivas aplicações da função T.

Lema 4.7 *Seja $\psi_{i_p}^{j_p}\dots\psi_{i_1}^{j_1}(X,a_1,\dots,a_p)$ o esqueleto de um termo em forma λs_e-normal. A forma $\lambda\sigma$-normal de $T(\psi_{i_p}^{j_p}\dots\psi_{i_1}^{j_1}(X,a_1,\dots,a_p))$ é um termo da forma $X[\underline{1}\cdot\ldots\cdot\underline{i_p-1}\cdot c_1\cdot\ldots\cdot c_{i_1-i_p+1}\cdot\uparrow^n]$ onde c_1, …, $c_{i_1-i_p+1}$ são $\lambda\sigma$-termos bem tipados e $n=i_1-p+\sum_{l=1}^{p}j_l$. Além disto, para cada $1\leq k\leq p$ tal que o operador $\psi_{i_k}^{j_k}$ é igual a σ, temos que o i_k-ésimo elemento da substituição $\underline{1}\cdot\ldots\cdot\underline{i_p-1}\cdot c_1\cdot\ldots\cdot c_{i_1-i_p+1}\cdot\uparrow^n$ é um termo da forma $T(a_k)[\uparrow^m]$, para algum $m\geq 0$.*

Prova. A prova é por indução sobre p. Se $p=1$ então temos 2 casos a considerar:
• $\psi_{i_1}^{j_1}=\sigma^{i_1}$: Pela definição de T temos que

$$T(\sigma^{i_1}(X,a_1))=X[\underline{1}\cdot\ldots\cdot\underline{i_1-1}\cdot T(a_1)[\uparrow^{i_1-1}]\cdot\uparrow^{i_1-1}].$$

Neste caso, $j_1=0$ e, então $n=i_1-1$. Além disto, o i_1-ésimo termo da substituição $\underline{1}\cdot\ldots\cdot\underline{i_1-1}\cdot T(a_1)[\uparrow^{i_1-1}]\cdot\uparrow^{i_1-1}$ é igual a $T(a_1)[\uparrow^{i_1-1}]$.
• $\psi_{i_1}^{j_1}=\varphi_{i_1}^{j_1}$: Pela definição de T temos que

$$T(\varphi_{i_1}^{j_1}(X,a_1))=T(\varphi_{i_1}^{j_1}(X))=X[\underline{1}\cdot\ldots\cdot\underline{i_1}\cdot\uparrow^{i_1+j_1-1}].$$

Assuma o resultado válido para $p-1$ e provemos sua validade para p. Temos 2 casos a considerar:
• $\psi_{i_p}^{j_p}=\sigma^{i_p}$: Neste caso, note que $j_p=0$. Pela definição de T temos que:
$T(\sigma^{i_p}\psi_{i_{p-1}}^{j_{p-1}}\dots\psi_{i_1}^{j_1}(X,a_1,\dots,a_p))=$
$T(\psi_{i_{p-1}}^{j_{p-1}}\dots\psi_{i_1}^{j_1}(X,a_1,\dots,a_{p-1}))[\underline{1}\cdot\ldots\cdot\underline{i_p-1}\cdot T(a_p)[\uparrow^{i_p-1}]\cdot\uparrow^{i_p-1}]=$

$$X[\underline{1}\cdot\ldots\cdot\underline{i_{p-1}-1}\cdot t_1\cdot\ldots\cdot t_{i_1-i_{p-1}+1}\cdot\uparrow^n][\underline{1}\cdot\ldots\cdot\underline{i_p-1}\cdot T(a_p)[\uparrow^{i_p-1}]\cdot\uparrow^{i_p-1}] \quad (4.1)$$

onde $n=i_1-(p-1)+\sum_{l=1}^{p-1}j_l$ e, t_1, …, $t_{i_1-i_{p-1}+1}$ são $\lambda\sigma$-termos bem tipados. Temos do Lema 4.3 que $n=i_1-(p-1)+\sum_{l=1}^{p-1}j_l\geq i_p$ e além disto, $i_p<i_{p-1}$ sempre que $\psi_{i_{p-1}}^{j_{p-1}}=\sigma^{i_{p-1}}$ e, $t_1=\underline{i_{p-1}}$ quando $\psi_{i_{p-1}}^{j_{p-1}}=\varphi_{i_{p-1}}^{j_{p-1}}$. Em ambos os casos, o termo (4.1) se reduz para um termo da forma $X[\underline{1}\cdot\ldots\cdot\underline{i_p-1}\cdot T(a_p)[\uparrow^{i_p-1}]\cdot c_1\cdot\ldots\cdot c_{i_1-i_p}\cdot\uparrow^{n-1}]$, onde c_1 …, $c_{i_1-i_p}$ são $\lambda\sigma$-termos bem tipados e, desta forma o i_p-ésimo termo da substituição $\underline{1}\cdot\ldots\cdot\underline{i_p-1}\cdot T(a_p)[\uparrow^{i_p-1}]\cdot c_1\cdot\ldots\cdot c_{i_1-i_p}\cdot\uparrow^{n-1}$ é dado por $T(a_p)[\uparrow^{i_p-1}]$. Por hipótese de indução temos que $n=i_1-(p-1)+\sum_{l=1}^{p-1}j_l$ e, como $j_p=0$ concluímos que $n=i_1-(p-1)+\sum_{l=1}^{p}j_l$, ou seja, $n-1=i_1-p+\sum_{l=1}^{p}j_l$.

• $\psi_{i_p}^{j_p}=\varphi_{i_p}^{j_p}$: Pela definição de T temos que:
$T(\varphi_{i_p}^{j_p}\psi_{i_{p-1}}^{j_{p-1}}\dots\psi_{i_1}^{j_1}(X,a_1,\dots,a_p))=$
$T(\psi_{i_{p-1}}^{j_{p-1}}\dots\psi_{i_1}^{j_1}(X,a_1,\dots,a_{p-1}))[\underline{1}\cdot\ldots\cdot\underline{i_p}\cdot\uparrow^{i_p+j_p-1}]=$

$$X[\underline{1}\cdot\ldots\cdot\underline{i_{p-1}-1}\cdot t_1\cdot\ldots\cdot t_{i_1-i_{p-1}+1}\cdot\uparrow^n][\underline{1}\cdot\ldots\cdot\underline{i_p}\cdot\uparrow^{i_p+j_p-1}] \quad (4.2)$$

onde $n = i_1 - (p-1) + \sum_{l=1}^{p-1} j_l$ e, $t_1, \ldots, t_{i_1 - i_{p-1}+1}$ são $\lambda\sigma$-termos bem tipados. Como $n \geq i_p$, o termo (4.2) se reduz para um termo da forma:

$$X[\underline{1} \cdot \ldots \cdot \underline{i_p} \cdot c_1 \cdot \ldots \cdot c_{i_1 - i_p} \cdot \uparrow^{n-1+j_p}]$$

onde $c_1 \ldots, c_{i_1 - i_p}$ são $\lambda\sigma$-termos bem tipados. Por hipótese de indução temos que:
$n = i_1 - (p-1) + \sum_{l=1}^{p-1} j_l \Leftrightarrow n - 1 = i_1 - p + \sum_{l=1}^{p-1} j_l \Leftrightarrow$
$n - 1 + j_p = i_1 - p + \sum_{l=1}^{p} j_l$. □

O Lema 4.7 nos mostra que a forma $\lambda\sigma$-normal de um termo da forma

$$T(\psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p))$$

é um termo da forma $X[\underline{1} \cdot \ldots \cdot \underline{i_p - 1} \cdot c_1 \cdot \ldots \cdot c_{i_1 - i_p + 1} \cdot \uparrow^n]$ e, portanto a substituição $\underline{1} \cdot \ldots \cdot \underline{i_p - 1} \cdot c_1 \cdot \ldots \cdot c_{i_1 - i_p + 1} \cdot \uparrow^n$ tem comprimento i_1. Este resultado é de fundamental importância na prova da proposição seguinte que nos mostra que os *graftings* gerados para uma equação flexível-rígida 'e' no λs_e-cálculo também são geradas para a equação $T(e)$ no $\lambda\sigma$-cálculo a menos de renomeamento de meta-variáveis.

Proposição 4.8 *Seja X uma meta-variável de tipo atômico e,*

$$\psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p) =^?_{\lambda s_e} (\underline{m}\ b_1 \ldots b_q) \tag{4.3}$$

uma equação flexível-rígida no λs_e-cálculo. Se uma aplicação da regra **Exp-App**$_{\lambda s_e}$ *à equação (4.3) gera uma nova equação da forma $X =^?_{\lambda s_e} (\underline{r}\ H_1 \ldots H_k)$ então uma aplicação de* **Exp-App**$_{\lambda\sigma}$ *à equação:*

$$T(\psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p)) =^?_{\lambda\sigma} (\underline{m}\ T(b_1) \ldots T(b_q))$$

gera uma nova equação da forma $X =^?_{\lambda\sigma} (\underline{r}\ W_1 \ldots W_k)$.

Prova. Suponha que uma aplicação de **Exp-App**$_{\lambda s_e}$ à equação (4.3) gere uma nova equação da forma:

$$X =^?_{\lambda s_e} (\underline{r}\ H_1 \ldots H_k) \tag{4.4}$$

Existem dois casos a serem considerados:

1. $\underline{r} \in R_p$, onde R_p é um subconjunto de $\{i_1, \ldots, i_p\}$ o conjunto dos super-índices do operadores σ que ocorrem do lado esquerdo da equação (4.3). Então existe $1 \leq k \leq p$ tal que $r = i_k$ e, neste caso temos que o tipo alvo de a_k coincide

com o tipo de X. Pelo Lema 4.7, sabemos que a forma $\lambda\sigma$-normal do lado esquerdo da equação:

$$T(\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)) =^?_{\lambda\sigma} T(\underline{m}\ b_1 \dots b_q)$$

é um termo da forma $X[c_1 \cdot \dots \cdot c_{i_1} \cdot \uparrow^n]$ tal que o i_k-ésimo elemento da substituição $c_1 \cdot \dots \cdot c_{i_1} \cdot \uparrow^n$ tem a forma $T(a_k)[\uparrow^m]$ para algum $m \geq 0$. Como T é uma função que preserva tipos e pelas regras de tipagem do $\lambda\sigma$-cálculo, temos que o tipo de a_k é igual ao tipo de $T(a_k)[\uparrow^m]$ e, portanto uma aplicação de **Exp-App**$_{\lambda\sigma}$ à equação:

$$X[c_1 \cdot \dots \cdot c_{i_1} \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ T(b_1) \dots T(b_q))$$

gera uma equação da forma:

$$X =^?_{\lambda\sigma} (\underline{r}\ W_1 \dots W_k) \tag{4.5}$$

que também introduz k meta-variáveis já que as cabeças das equações (4.4) e (4.5) possuem o mesmo tipo.

2. $\underline{r} \in R_i$, onde $R_i = \bigcup_{k=0}^{p}$(se $i_k \geq m + p - k - \Sigma_{l=k+1}^{p} j_l > i_{k+1}$ então $\{m + p - k - \Sigma_{l=k+1}^{p} j_l\}$ senão $\emptyset$), e $i_0 = \infty$ e $i_{p+1} = 0$. Assim existem $p + 1$ possíveis valores para $\underline{r}$ e, dividiremos a análise em 2 casos:
   2.1: $k = 0$: Suponha que a desigualdade:

$$\infty \geq m + p - \sum_{l=1}^{p} j_l > i_1 \tag{4.6}$$

seja verdadeira, isto é, estamos assumindo que o caso $k = 0$ $(0 \leq k \leq p)$ gere um elemento em R_i. Isto significa que uma aplicação de **Exp-App**$_{\lambda s_e}$ à equação (4.3) gera uma nova equação da forma:

$$X =^?_{\lambda s_e} (\underline{m + p - \sum_{l=1}^{p} j_l}\ H_1 \dots H_q).$$

Aplicando a função T à equação (4.3), de acordo com o Lema 4.7, obteremos uma equação da forma:

$$X[\underline{1} \cdot \dots \cdot \underline{i_p - 1} \cdot c_1 \cdot \dots \cdot c_{i_1 - i_p + 1} \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ T(b_1) \dots T(b_q)) \tag{4.7}$$

onde n é tal que $n = i_1 - p + \sum_{l=1}^{p} j_l$ que, juntamente com a desigualdade (4.6) nos permite concluir que $n < m$ e, portanto uma aplicação de **Exp-App**$_{\lambda\sigma}$ à equação (4.7) gera uma nova equação da forma:

$$X =^{?}_{\lambda\sigma} (\underline{m + p - \sum_{l=1}^{p} j_l}\ W_1 \ldots W_q)$$

• $0 < k \leq p$: Pela definição do conjunto R_i temos que todos os seus possíveis elementos $\underline{r}$, gerados por valores de k tais que $0 < k \leq p$, são tais que $r \leq i_1$ e, portanto uma nova equação da forma $X =^{?}_{\lambda s_e} (\underline{r}\ H_1 \ldots H_k)$ é gerada somente se o tipo alvo de $\underline{r}$ coincide com o tipo de X, mas neste caso, como T é uma função que preserva o tipo dos termos, teremos que o r-ésimo elemento da substituição $\underline{1} \cdot \ldots \cdot \underline{i_p - 1} \cdot c_1 \cdot \ldots \cdot c_{i_1 - i_p + 1} \cdot \uparrow^n$ também é tal que seu tipo alvo coincide com o tipo de X o que constitui a condição necessária para que uma aplicação de **Exp-App**$_{\lambda\sigma}$ à equação (4.7) gere uma nova equação da forma $X =^{?}_{\lambda\sigma} (\underline{r}\ W_1 \ldots W_k)$. Observe que neste caso, elementos de R_i da regra **Exp-App**$_{\lambda s_e}$ correspondem a elementos de R_p da regra **Exp-App**$_{\lambda\sigma}$. Isto justifica o fato de, as regras **Exp-App**$_{\lambda\sigma}$ e **Exp-App**$_{\lambda s_e}$ gerarem o mesmo número de novas equações apesar de o conjunto R_i ser unitário apenas na regra **Exp-App**$_{\lambda\sigma}$.

□

Exemplo 4.9 *Considere o seguinte problema de unificação*

$$\varphi_1^4 \sigma^3 \sigma^5 \varphi_7^3 (X, a, b) =^{?}_{\lambda s_e} (\underline{12}\ b_1 \ldots b_q) \tag{4.8}$$

cujo lado esquerdo já foi estudado no Exemplo 4.5.

De acordo com a definição do conjunto R_i da regra **Exp-App**$_{\lambda s_e}$*, temos que uma equação da forma $X =^{?}_{\lambda s_e} (\underline{9}\ H_1 \ldots H_q)$ é gerada.*

Pelo Exemplo 4.5, sabemos que o problema de unificação correspondente à equação (4.8) no $\lambda\sigma$-cálculo é dado por:

$$X[\underline{1} \cdot \underline{5} \cdot T(b)[\uparrow^5] \cdot \underline{6} \cdot T(a)[\uparrow^6] \cdot \underline{7} \cdot \underline{8} \cdot \uparrow^{10}] =^{?}_{\lambda\sigma} (\underline{12}\ T(b_1) \ldots T(b_q)) \tag{4.9}$$

Uma aplicação da regra **Exp-App**$_{\lambda\sigma}$ *à equação (4.9) gera uma nova equação da forma $X =^{?}_{\lambda\sigma} (\underline{9}\ W_1 \ldots W_q)$, como afirma a Proposição 4.8.*

Na seção seguinte estabeleceremos uma relação semelhante a esta proposição, mas na outra direção, isto é, partindo do $\lambda\sigma$-cálculo.

4.2 Correspondência a partir do $\lambda\sigma$-cálculo

Nesta seção mostraremos como a unificação no $\lambda\sigma$-cálculo corresponde à unificação no λs_e-cálculo.

Definição 4.10 *A função* $L : \Lambda_{\lambda\sigma}(\mathcal{X}) \to \Lambda_{\lambda s_e}(\mathcal{X})$ *é definida indutivamente por:*

(a) $L(X) = X$;

(b) $L(\underline{1}[\uparrow^{m-1}]) = \underline{m}$, *onde* $m \in \mathbb{N}$;

(c) $L(a\ b) = (L(a)\ L(b))$;

(d) $L(\lambda_A.a) = \lambda_A.L(a)$;

(e) $L(a[a_1 \cdot a_2 \cdot \ldots \cdot a_p \cdot \uparrow^n]) = \sigma^1 \ldots \sigma^p \varphi_p^{n+1}(L(a), L(a_p), L(a_{p-1}), \ldots, L(a_2), L(a_1))$, *onde* $a_1 \cdot a_2 \cdot \ldots \cdot a_p \cdot \uparrow^n$ *é uma substituição em forma* $\lambda\sigma$*-normal, e* $n, p \geq 0$.

A proposição a seguir mostra que a função L preserva o tipo e o contexto dos termos:

Proposição 4.11 *Seja* Γ *um contexto, e* a *um termo em forma* $\lambda\sigma$*-normal de tipo* A. *Se* $\Gamma \vdash a : A$ *então* $\Gamma \vdash L(a) : A$

Prova. A prova é por indução sobre a estrutura de a. Os casos em que a é uma meta-variável, um índice *à la* de Bruijn ou uma aplicação são triviais.

- Se $a = \lambda_B.b$ então A é da forma $B \to C$ e, por hipótese de indução, temos que $B \cdot \Gamma \vdash L(b) : C$. Após uma aplicação de $(lambda)$ obtemos $\Gamma \vdash \lambda_B.L(b) : B \to C$ que, pela definição de L, nos permite concluir que $\Gamma \vdash L(\lambda_B.b) : B \to C$.

- Se $a = b[c_1 \cdot \ldots \cdot c_p \cdot \uparrow^n]$ então temos que $a_A^\Gamma = b_A^{C_1 \cdot \ldots \cdot C_p \cdot \Gamma_{>n}}[c_{1\,C_1}^{\ \Gamma} \cdot \ldots \cdot c_{p\,C_p}^{\ \Gamma} \cdot (\uparrow^n)_{\Gamma_{>n}}^{\Gamma}]$ e, pela definição de L sabemos: $L(a_A^\Gamma) = L(b_A^{C_1 \cdot \ldots \cdot C_p \cdot \Gamma_{>n}}[c_{1\,C_1}^{\ \Gamma} \cdot \ldots \cdot c_{p\,C_p}^{\ \Gamma} \cdot \uparrow_{\Gamma_{>n}}^{n\Gamma}]) = \sigma^1 \ldots \sigma^p \varphi_p^{n+1}(L(b)_A^{C_1 \cdot \ldots \cdot C_p \cdot \Gamma_{>n}}, L(c_p)_{C_p}^{\Gamma}, \ldots, L(c_1)_{C_1}^{\Gamma})$. Por hipótese de indução, temos que $\Gamma \vdash L(c_i) : C_i$ para todo $1 \leq i \leq p$ e que $C_1 \cdot \ldots \cdot C_p \cdot \Gamma_{>n} \vdash L(b) : A$. Considere a seguinte derivação obtida por sucessivas aplicações da regra

(*sigma*):

$$\dfrac{\dfrac{\dfrac{\dfrac{C_1 \cdot \ldots \cdot C_p \cdot \Gamma \vdash \varphi_p^{n+1}(L(b)_A^{C_1 \cdot \ldots \cdot C_p \cdot \Gamma_{>n}}) : A \qquad \Gamma \vdash L(c_p) : C_p}{C_1 \cdot \ldots \cdot C_{p-1} \cdot \Gamma \vdash \sigma^p(\varphi_p^{n+1}(L(b)), L(c_p)) : A}\,\boxdot}{C_1 \cdot \ldots \cdot C_{p-2} \cdot \Gamma \vdash \sigma^{p-1}\sigma^p(\varphi_p^{n+1}(L(b)), L(c_p), L(c_{p-1})) : A}}{\vdots}}{\Gamma \vdash \sigma^1 \ldots \sigma^{p-1}\sigma^p\varphi_p^{n+1}(L(b), L(c_p), \ldots, L(c_1)) : A}$$

onde $\boxdot$ corresponde a $\Gamma \vdash L(c_{p-1}) : C_{p-1}$.

□

Proposição 4.12 *Seja X uma meta-variável de tipo atômico e,*

$$X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \tag{4.10}$$

uma equação flexível-rígida no $\lambda\sigma$-cálculo. Se uma aplicação da regra **Exp-App**$_{\lambda\sigma}$ *à equação (4.10) gera uma nova equação da forma $X =^?_{\lambda\sigma} (\underline{r}\ H_1 \ldots H_k)$ então uma aplicação de* **Exp-App**$_{\lambda s_e}$ *à equação:*

$$L(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]) =^?_{\lambda s_e} (\underline{m}\ L(b_1) \ldots L(b_q))$$

gera uma equação da forma $X =^?_{\lambda s_e} (\underline{r}\ W_1 \ldots W_k)$.

Prova. Suponha que uma aplicação da regra **Exp-App**$_{\lambda\sigma}$ à equação (4.10) gere uma nova equação da forma $X =^?_{\lambda\sigma} (\underline{r}\ H_1 \ldots H_k)$. Se $m > n$ então a equação $X =^?_{\lambda\sigma} (\underline{m-n+p}\ H_1 \ldots H_q)$ é gerada. Neste caso, temos que a desigualdade $\infty \geq m + (p+1) - (n+1) > p$ é verdadeira e, a equação $X =^?_{\lambda s_e} (\underline{m+p-n}\ W_1 \ldots W_q)$ é gerada por uma aplicação de **Exp-App**$_{\lambda s_e}$.

Se $r \in \{1, \ldots, p\}$ então o tipo alvo de a_r coincide com o tipo de X e, assim pela Proposição 4.11, temos que o tipo alvo de $L(a_r)$ coincide com o tipo de X e, assim uma aplicação de **Exp-App**$_{\lambda s_e}$ gera uma equação da forma $X =^?_{\lambda s_e} (\underline{r}\ W_1 \ldots W_k)$ porque r é o super-índice de um operador σ tal que $(\underline{r}\ W_1 \ldots W_k)$ tem o tipo de X. □

Proposição 4.13 *Seja P (resp. Q) um problema de unificação no λs_e-cálculo (resp. $\lambda\sigma$-cálculo). Então as árvores de derivação de P (resp. Q) e $T(P)$ (resp. $L(Q)$) coincidem, pois ambas sempre podem ser construídas utilizando-se a mesma estratégia.*

Prova. • Suponha que a regra **Dec-**$\lambda_{\lambda s_e}$ possa ser aplicada em P então P possui (pelo menos) uma equação da forma $\lambda_A.a =^?_{\lambda s_e} \lambda_A.b$. Neste caso, $T(P)$ possui (pelo menos) uma equação da forma $\lambda_A.T(a) =^?_{\lambda\sigma} \lambda_A.T(b)$ e, portanto **Dec-**$\lambda_{\lambda\sigma}$ também se aplica.

• Se **Dec-App**$_{\lambda s_e}$ pode ser aplicada a P então este possui (pelo menos) uma equação da forma $(\underline{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\underline{n}\ b_1 \dots b_p)$. Neste caso, $T(P)$ possui (pelo menos) uma equação da forma $(\underline{n}\ T(a_1) \dots T(a_p)) =^?_{\lambda\sigma} (\underline{n}\ T(b_1) \dots T(b_p))$ e, portanto a regra **Dec-App**$_{\lambda\sigma}$ pode ser aplicada.

• Se **App-Fail**$_{\lambda s_e}$ pode ser aplicada a P então este possui (pelo menos) uma equação da forma $(\underline{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\underline{m}\ b_1 \dots b_q)$ com $m \neq n$. Neste caso, $T(P)$ possui (pelo menos) uma equação da forma $(\underline{n}\ T(a_1) \dots T(a_p)) =^?_{\lambda\sigma} (\underline{m}\ T(b_1) \dots T(b_q))$ e, portanto a regra **Dec-Fail**$_{\lambda\sigma}$ pode ser aplicada.

• Se **Exp-**$\lambda_{\lambda s_e}$ pode ser aplicada a P então este possui (pelo menos) uma ocorrência de meta-variável cujo tipo não é atômico, digamos X. Como $T(X) = X$ então $T(P)$ também contém uma meta-variável de tipo funcional e, portanto a regra **Exp-**$\lambda_{\lambda\sigma}$ se aplica.

• Se **Exp-App**$_{\lambda s_e}$ pode ser aplicada a P então o resultado segue da Proposição 4.8.

• Se **Normalise**$_{\lambda s_e}$ pode ser aplicada a P então este passo de normalização foi originado devido a uma aplicação de **Exp-**$\lambda_{\lambda s_e}$ ou **Exp-App**$_{\lambda s_e}$ já que estas são as únicas regras que introduzem novos termos que podem gerar novos rédices. Se o passo de normalização foi originado por uma aplicação de **Exp-**$\lambda_{\lambda s_e}$ então uma abstração foi inserida em um termo que está numa posição funcional e, pelo mesmo motivo uma aplicação de **Normalise**$_{\lambda\sigma}$ pode ser aplicada ao problema $T(P)$. Se o passo de normalização foi originado por uma aplicação de **Exp-App**$_{\lambda s_e}$ então uma meta-variável de tipo atômico foi substituído por um termo cuja cabeça é um índice *à la* de Bruijn que está inserido dentro de um número não nulo de operadores σ e/ou φ. Sendo assim, $T(P)$ possui um termo contendo um índice *à la* de Bruijn que está aplicado a uma substituição explícita e, portanto **Normalise**$_{\lambda\sigma}$ se aplica a $T(P)$.

• Se **Replace**$_{\lambda s_e}$ pode ser aplicada a P então P deve conter (pelo menos) uma equação da forma $X =^?_{\lambda s_e} a$, onde X é uma meta-variável não resolvida. Existem duas possibilidades para uma tal equação: ou ela foi criada por uma aplicação da regra **Exp-App**$_{\lambda s_e}$, ou já existia no problema original. No primeiro caso, uma aplicação da regra **Exp-App**$_{\lambda\sigma}$ à equação correspondente em $T(P)$ gera uma equação da forma $X =^?_{\lambda\sigma} a'$, onde X também não é resolvida em $T(P)$ e, portanto

Replace$_{\lambda\sigma}$ se aplica a $T(P)$. □

Exemplo 4.14 *Considere a árvore de derivação no $\lambda\sigma$-cálculo contruída no Exemplo 2.15. Aplicando a função L à equação (2.8), obtemos a equação:*

$$\lambda_{B\to B}.(\underline{1}((\varphi_0^2 X)\ \underline{3})) =^{?}_{\lambda s_e} \lambda_{B\to B}.(\underline{1}(\underline{2}(\underline{4}\ \underline{3}))).$$

Utilizando a mesma estratégia, construímos a árvore de unificação dada nas figuras 4.1, 4.2, 4.3, 4.4 e 4.5. Note que existe uma correspondência biunívoca entre os vértices destas árvores, e mais ainda, cada um de seus vértices pode ser obtido do vértice correspondente no outro cálculo utilizando-se as funções T e L.

Neste capítulo mostramos que unificação no $\lambda\sigma$-cálculo e no λs_e-cálculo podem gerar árvores de derivação iguais desde que sejam utilizadas a mesma estratégia em ambos os cálculos. Desta forma a relação estrutural entre HOU *à la* Huet e HOU no $\lambda\sigma$-cálculo também é válida para o λs_e-cálculo.

$$\lambda_{B\to B}.(\underline{1}((\varphi_0^2 X)\ \underline{3})) =^?_{\lambda s_e}\ \lambda_{B\to B}.(\underline{1}(\underline{2}(\underline{4}\ \underline{3})))$$

Dec-λ

$$(\underline{1}((\varphi_0^2 X)\ \underline{3})) =^?_{\lambda s_e}\ (\underline{1}(\underline{2}(\underline{4}\ \underline{3})))$$

Dec-App

$$((\varphi_0^2 X)\ \underline{3}) =^?_{\lambda s_e}\ (\underline{2}(\underline{4}\ \underline{3}))$$

Exp-λ

$$X =^?_{\lambda s_e}\ \lambda_A.Y \wedge ((\varphi_0^2 X)\ \underline{3}) =^?_{\lambda s_e}\ (\underline{2}(\underline{4}\ \underline{3}))$$

Replace

$$X =^?_{\lambda s_e}\ \lambda_A.Y \wedge ((\varphi_0^2(\lambda_A.Y))\ \underline{3}) =^?_{\lambda s_e}\ (\underline{2}(\underline{4}\ \underline{3}))$$

Normalise

$$X =^?_{\lambda s_e}\ \lambda_A.Y \wedge ((\varphi_1^2 Y)\sigma^1\underline{3}) =^?_{\lambda s_e}\ (\underline{2}(\underline{4}\ \underline{3}))$$

Exp-App

$$Y =^?_{\lambda s_e}\ (\underline{2}\ H_1) \wedge X =^?_{\lambda s_e}\ \lambda_A.Y \wedge ((\varphi_1^2 Y)\sigma^1\underline{3}) =^?_{\lambda s_e}\ (\underline{2}(\underline{4}\ \underline{3}))$$

Replace

$$Y =^?_{\lambda s_e}\ (\underline{2}\ H_1) \wedge X =^?_{\lambda s_e}\ \lambda_A.(\underline{2}\ H_1) \wedge ((\varphi_1^2(\underline{2}\ H_1))\sigma^1\underline{3}) =^?_{\lambda s_e}\ (\underline{2}(\underline{4}\ \underline{3}))$$

Normalise

$$Y =^?_{\lambda s_e}\ (\underline{2}\ H_1) \wedge X =^?_{\lambda s_e}\ \lambda_A.(\underline{2}\ H_1) \wedge (\underline{2}((\varphi_1^2 H_1)\sigma^1\underline{3})) =^?_{\lambda s_e}\ (\underline{2}(\underline{4}\ \underline{3}))$$

Dec-App

$$Y =^?_{\lambda s_e}\ (\underline{2}\ H_1) \wedge X =^?_{\lambda s_e}\ \lambda_A.(\underline{2}\ H_1) \wedge ((\varphi_1^2 H_1)\sigma^1\underline{3}) =^?_{\lambda s_e}\ (\underline{4}\ \underline{3})$$

Exp-App

T_2 T_1

Exp-App

T_3 T_4

Figura 4.1: Árvore de derivação no λs_e-cálculo.

$$H_1 =^?_{\lambda s_e} \underline{1} \wedge Y =^?_{\lambda s_e} (\underline{2}\ H_1) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ H_1) \wedge ((\varphi_1^2 H_1)\sigma^1 \underline{3}) =^?_{\lambda s_e} (\underline{4}\ \underline{3})$$

Replace

$$H_1 =^?_{\lambda s_e} \underline{1} \wedge Y =^?_{\lambda s_e} (\underline{2}\ \underline{1}) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ \underline{1}) \wedge ((\varphi_1^2 \underline{1})\sigma^1 \underline{3}) =^?_{\lambda s_e} (\underline{4}\ \underline{3})$$

Normalise

$$H_1 =^?_{\lambda s_e} \underline{1} \wedge Y =^?_{\lambda s_e} (\underline{2}\ \underline{1}) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ \underline{1}) \wedge \underline{3} =^?_{\lambda s_e} (\underline{4}\ \underline{3})$$

App-Fail

Falha

Figura 4.2: A árvore T_1 no λs_e-cálculo.

$$H_1 =^?_{\lambda s_e} (\underline{4} H_2) \wedge Y =^?_{\lambda s_e} (\underline{2}\ H_1) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ H_1) \wedge ((\varphi_1^2 H_1)\sigma^1 \underline{3}) =^?_{\lambda s_e} (\underline{4}\ \underline{3})$$

Replace

$$H_1 =^?_{\lambda s_e} (\underline{4}\ H_2) \wedge Y =^?_{\lambda s_e} (\underline{2}\ (\underline{4}\ H_2)) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge ((\varphi_1^2 (\underline{4}\ H_2))\sigma^1 \underline{3}) =^?_{\lambda s_e} (\underline{4}\ \underline{3})$$

Normalise

$$H_1 =^?_{\lambda s_e} (\underline{4}\ H_2) \wedge Y =^?_{\lambda s_e} (\underline{2}\ (\underline{4}\ H_2)) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge (\underline{4}\ (\varphi_1^2 (H_2)\sigma^1 \underline{3})) =^?_{\lambda s_e} (\underline{4}\ \underline{3})$$

Dec-App

$$H_1 =^?_{\lambda s_e} (\underline{4}\ H_2) \wedge Y =^?_{\lambda s_e} (\underline{2}\ (\underline{4}\ H_2)) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge (\varphi_1^2 (H_2)\sigma^1 \underline{3}) =^?_{\lambda s_e} \underline{3}$$

Figura 4.3: A árvore T_2 no λs_e-cálculo.

$$H_2 =^?_{\lambda s_e} \underline{3} \wedge H_1 =^?_{\lambda s_e} (\underline{4}\ H_2) \wedge Y =^?_{\lambda s_e} (\underline{2}\ (\underline{4}\ H_2)) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge (\varphi_1^2 (H_2)\sigma^1 \underline{3}) =^?_{\lambda s_e} \underline{3}$$

Replace

$$H_2 =^?_{\lambda s_e} \underline{3} \wedge H_1 =^?_{\lambda s_e} (\underline{4}\ \underline{3}) \wedge Y =^?_{\lambda s_e} (\underline{2}\ (\underline{4}\ \underline{3})) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ (\underline{4}\ \underline{3})) \wedge (\varphi_1^2 (\underline{3})\sigma^1 \underline{3}) =^?_{\lambda s_e} \underline{3}$$

Normalise

$$H_2 =^?_{\lambda s_e} \underline{3} \wedge H_1 =^?_{\lambda s_e} (\underline{4}\ \underline{3}) \wedge Y =^?_{\lambda s_e} (\underline{2}\ (\underline{4}\ \underline{3})) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ (\underline{4}\ \underline{3}))$$

Sucesso

Figura 4.4: A árvore T_3 no λs_e-cálculo.

$$H_2 =^?_{\lambda s_e} \underline{1} \wedge H_1 =^?_{\lambda s_e} (\underline{4}\ H_2) \wedge Y =^?_{\lambda s_e} (\underline{2}\ (\underline{4}\ H_2)) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ (\underline{4}\ H_2)) \wedge (\varphi_1^2 (H_2)\sigma^1 \underline{3}) =^?_{\lambda s_e} \underline{3}$$

Replace

$$H_2 =^?_{\lambda s_e} \underline{1} \wedge H_1 =^?_{\lambda s_e} (\underline{4}\ \underline{1}) \wedge Y =^?_{\lambda s_e} (\underline{2}\ (\underline{4}\ \underline{1})) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ (\underline{4}\ \underline{1})) \wedge (\varphi_1^2 (\underline{1})\sigma^1 \underline{3}) =^?_{\lambda s_e} \underline{3}$$

Normalise

$$H_2 =^?_{\lambda s_e} \underline{1} \wedge H_1 =^?_{\lambda s_e} (\underline{4}\ \underline{1}) \wedge Y =^?_{\lambda s_e} (\underline{2}\ (\underline{4}\ \underline{1})) \wedge X =^?_{\lambda s_e} \lambda_A.(\underline{2}\ (\underline{4}\ \underline{1}))$$

Sucesso

Figura 4.5: A árvore T_4 no λs_e-cálculo.

Capítulo 5

Aplicação: *Matching* de Segunda Ordem

Matching é uma operação básica muito utilizada em computação. Em particular, *matching* de segunda ordem nos fornece um ambiente adequado para expressar transformação de programas [HL78] e reconhecimento de padrões em dedução automática [dlTC87]. Neste capítulo adaptaremos as técnicas de unificação em substituições explícitas para desenvolver um algoritmo específico para *matching* de segunda ordem no $\lambda\sigma$-cálculo. Uma versão preliminar das idéias aqui desenvolvidas foram publicadas em [dMKAR05a]

5.1 *Matching* de Ordem Superior

Matching de primeira ordem, assim como unificação de primeira ordem, é um problema decidível e unitário, isto é, quando um problema tem solução então esta solução é única (conhecida como "unificador mais geral") e existe um algoritmo para determiná-la [Rob65]. *Matching* de segunda ordem é também um problema decidível [Hue76], mas as soluções não são mais necessariamente únicas e a noção de unificador mais geral não existe mais. *Matching* de terceira e quarta ordens ainda são problemas decidíveis [Dow94, Pad00], e para ordens superiores a decibilidade do problema de *matching* ainda é um problema que continua em aberto (por quase trinta anos) [Hue76]. Em [Loa03], Loader prova a indecidibilidade de β-*matching* de quinta ordem, mas infelizmente a prova não fornece informações de como tratar o caso geral que inclui η-conversão.

No Capítulo 2 apresentamos o procedimento de unificação de Dowek, Hardin

e Kirchner para o $\lambda\sigma$-cálculo simplesmente tipado. No entanto este procedimento não decide *matching* de segunda ordem como veremos através de um exemplo não terminante. Além disto, [Bur89] mostra que *matching* pode ter um comportamento diferente de unificação dependendo da teoria equacional considerada o que torna importante o estudo de *matching* via substituições explícitas. A seguir apresentamos algumas noções importantes para o restante deste capítulo.

Definição 5.1 (Equação de *matching*[Dow01]) *Uma* equação de matching (de ordem superior) *é uma equação da forma* $a \ll^? b$*, onde* a *e* b *são* λ*-termos do mesmo tipo e sob o mesmo contexto e, além disto* b *não contém meta-variáveis. Uma solução da equação* $a \ll^? b$*, chamada de* matcher*, é uma substituição* σ *tal que* $a\sigma =_{\beta\eta} b$*.*

A definição acima corresponde à noção de "filtragem", que por definição assume que os lados esquerdo e direito em uma equação de *matching* são termos sem meta-variáveis comuns. Caso contrário, estas meta-variáveis podem ser renomeadas como é usual na teoria da reescrita. Uma outra noção de *matching* conhecida como "semi-unificação" caracterizada por $(\exists\sigma, a\sigma =_{\beta\eta} b\sigma =_{\beta\eta} b)$, não será tratada aqui.

Definição 5.2 (Problema de Matching) *Um* problema de matching *(de ordem superior) é dado por uma conjunção de um número finito de equações de* matching*, todas bem tipadas sob o mesmo contexto. A ordem de um problema de* matching *é dada pela mais alta ordem de suas meta-variáveis.*

Se M é um problema de *matching*, denotamos por $\mathcal{M}(M)$ o conjunto de todas as soluções de M.

5.2 *Matching* de Segunda Ordem no $\lambda\sigma$-cálculo

Definiremos problemas de *matching* no $\lambda\sigma$-cálculo de forma similar à definição que fizemos para problemas de unificação. Assim, um problema de *matching* no $\lambda\sigma$-cálculo é dado por uma conjunção de um número finito de equações da forma:

$$\bigwedge_i a_i \ll^?_{\lambda\sigma} b_i)$$

onde a_i e b_i são $\lambda\sigma$-termos do mesmo tipo no mesmo contexto e b não contém ocorrências de meta-variáveis. Denotaremos por $\mathcal{M}_{\lambda\sigma}(M)$ o conjunto de todas as soluções do problema de *matching* M na linguagem do $\lambda\sigma$-cálculo.

Sabemos que a linguagem do $\lambda\sigma$-cálculo é uma extensão não trivial da linguagem do λ-cálculo e, portanto decidibilidade de *matching* de segunda ordem no $\lambda\sigma$-cálculo é uma questão que aparece naturalmente. Um passo óbvio que podemos tomar é utilizar o procedimento de unificação do $\lambda\sigma$-cálculo que conhecemos para resolver problemas de *matching*. No entanto, este procedimento pode gerar reduções infinitas para problemas de *matching* de segunda ordem arbitrários. De fato, seja Γ um contexto e, considere o seguinte problema de segunda ordem:

$$X_A^{A\to A\cdot\Gamma}[(\lambda_A.\underline{1}_A^{A\cdot\Gamma})_{A\to A}^{\Gamma}\cdot id_\Gamma^\Gamma]_A^\Gamma =^?_{\lambda\sigma} b_A^\Gamma$$

onde b é um termo sem ocorrências de meta-variáveis bem tipado no contexto Γ. Considere a seguinte derivação:

$X_A^{A\to A\cdot\Gamma}[(\lambda_A.\underline{1}_A^{A\cdot\Gamma})_{A\to A}^{\Gamma}\cdot id_\Gamma^\Gamma]_A^\Gamma =^?_{\lambda\sigma} b_A^\Gamma \to^{\mathbf{Exp-App}}$

$X_A^{A\to A\cdot\Gamma}[(\lambda_A.\underline{1}_A^{A\cdot\Gamma})_{A\to A}^{\Gamma}\cdot id_\Gamma^\Gamma]_A^\Gamma =^?_{\lambda\sigma} b_A^\Gamma \wedge$
$X_A^{A\to A\cdot\Gamma} =^?_{\lambda\sigma} (\underline{1}_{A\to A}^{A\to A\cdot\Gamma}\ Y_A^{A\to A\cdot\Gamma})_A^{A\to A\cdot\Gamma} \to^{\mathbf{Replace}}$

$(\underline{1}_{A\to A}^{A\to A\cdot\Gamma}\ Y_A^{A\to A\cdot\Gamma})_A^{A\to A\cdot\Gamma}[(\lambda_A.\underline{1}_A^{A\cdot\Gamma})_{A\to A}^{\Gamma}\cdot id_\Gamma^\Gamma] =^?_{\lambda\sigma} b_A^\Gamma \wedge$
$X_A^{A\to A\cdot\Gamma} =^?_{\lambda\sigma} (\underline{1}_{A\to A}^{A\to A\cdot\Gamma}\ Y_A^{A\to A\cdot\Gamma})_A^{A\to A\cdot\Gamma} \to^{\mathbf{Normalise}}$

$Y_A^{A\to A\cdot\Gamma}[(\lambda_A.\underline{1}_A^{A\cdot\Gamma})_{A\to A}^{\Gamma}\cdot id_\Gamma^\Gamma] =^?_{\lambda\sigma} b_A^\Gamma \wedge$
$X_A^{A\to A\cdot\Gamma} =^?_{\lambda\sigma} (\underline{1}_{A\to A}^{A\to A\cdot\Gamma}\ Y_A^{A\to A\cdot\Gamma})_A^{A\to A\cdot\Gamma} \to^{\mathbf{Exp-App}} \ldots$

A partir deste ponto podemos repetir a estratégia **Exp-App**, **Replace** e **Normalise** já que o último problema gerado é composto por duas equações da forma flexível-rígida, onde a primeira é igual ao problema original a menos de renomeamento de variáveis.

Portanto o procedimento de Dowek, Hardin e Kirchner não nos permite responder diretamente se *matching* de segunda ordem na linguagem do $\lambda\sigma$-cálculo é um problema decidível ou não. A seguir definiremos uma classe de problemas de unificação de segunda ordem que nos permitirá construir um algoritmo específico para *matching* de segunda ordem no $\lambda\sigma$-cálculo. Esta classe é originada a partir de problemas que são gerados no λ-cálculo simplesmente tipado e portanto nosso algoritmo será capaz de resolver todos os problemas de *matching* de segunda ordem que são relevantes. O lema a seguir é utilizado para construir uma estratégia de normalização que vai ser útil na caracterização desta classe.

Lema 5.3 *Sejam a um termo bem tipado e, $b_1\cdot\ldots\cdot b_r\cdot\uparrow^{r'}$ e $c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'}$ substituições bem tipadas na linguagem do $\lambda\sigma$-cálculo, onde:*

- $r, r', s, s' \geq 0$;
- $b_1, \ldots, b_r$ *são termos de tipo atômico;*
- $\begin{cases} c_1, \ldots, c_{r'} \text{ têm tipo arbitrário e } c_{r'+1}, \ldots, c_s \text{ têm tipo atômico, se } r' < s; \\ c_1, \ldots, c_s \text{ têm tipo arbitrário, se } r' \geq s. \end{cases}$

Se o termo $(a[b_1 \cdot \ldots \cdot b_r \cdot \uparrow^{r'}])[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}]$ *está bem tipado então existe uma estratégia de normalização que resulta no termo* $a[a_1 \cdot \ldots \cdot a_p \uparrow^n]$ *onde* $a_1 \cdot \ldots \cdot a_p \uparrow^n$ *é uma substituição em forma* $\lambda\sigma$*-normal tal que os termos* $a_1, \ldots, a_p$ *têm tipo atômico e* $\begin{cases} p = r \ e \ n = r' - s + s', & \text{se } r' \geq s; \\ p = r + s - r' \ e \ n = s', & \text{se } r' < s. \end{cases}$

Prova. Considere a seguinte estratégia de normalização:
$(a[b_1 \cdot \ldots \cdot b_r \cdot \uparrow^{r'}])[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}] \rightarrow_{Clos}$
$a[(b_1 \cdot \ldots \cdot b_r \cdot \uparrow^{r'}) \circ (c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'})] \rightarrow^*_{Assoc,Map}$
$a[b_1[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}] \cdot \ldots \cdot b_r[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}] \cdot (\uparrow^{r'} \circ (c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}))]$.

Se $r' \geq s$ então podemos aplicar a regra $(ShifCons)$ s vezes para obter o termo:

$$a[b_1[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}] \cdot \ldots \cdot b_r[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}] \cdot \uparrow^{r'-s+s'}]$$

e, como os termos $b_1, \ldots, b_r$ por hipótese têm tipo atômico, temos que a forma $\lambda\sigma$-normal de cada um dos termos:

$$b_1[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}], \ldots, b_r[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}]$$

também tem tipo atômico.

Se $r' < s$ então podemos aplicar a regra $(ShifCons)$ r' vezes para obter o termo:

$$a[b_1[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}] \cdot \ldots \cdot b_r[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}] \cdot c_{r'+1} \cdot \ldots c_s \cdot \uparrow^{s'}]$$

e, como os termos $b_1, \ldots, b_r, c_{r'+1}, \ldots, c_s$ por hipótese têm tipo atômico, temos que a forma $\lambda\sigma$-normal de cada um dos termos:

$$b_1[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}], \ldots, b_r[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}], c_{r'+1}, \ldots, c_s$$

também tem tipo atômico. □

A proposição a seguir caracteriza uma classe de problemas de segunda ordem na linguagem do $\lambda\sigma$-cálculo para a qual o procedimento apresentado na Seção 2.4 sempre pára.

Proposição 5.4 (Caracterização para problemas de segunda ordem) *Seja P um problema de unificação de segunda ordem que está na imagem da função de pré-cozimento. Então todo subtermo atômico e flexível que ocorra em qualquer problema derivado de P (utilizando-se as regras da Tabela 2.1) tem a forma $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$, onde $p, n \geq 0$, X é uma meta-variável atômica e $a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n$ é uma substituição em forma $\lambda\sigma$-normal com $a_1, \ldots, a_p$ atômicos.*

Prova. A prova é por indução sobre o comprimento da derivação que gerou o termo que contém $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ como subtermo.

Como P está na imagem do pré-cozimento então todas as ocorrências de meta-variáveis em P são da forma $X[\uparrow^n]$, onde n denotada o número de abstratores que englobam X e, como $p = 0$, o resultado segue por vacuidade.

Agora suponha que a proposição valha para o problema P' derivado de P através das regras da Tabela 2.1. Seja P'' um problema derivado de P' após uma aplicação da regra:

- **Dec-λ** ou **Dec-App**: Neste caso, o termo $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ já era subtermo de P' já que estas regras não mudam as estruturas das substituições explícitas que ocorrem no termo e, o resultado segue por hipótese de indução.

- **Exp-λ**: Neste caso, o termo $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ é um subtermo da nova equação que foi gerada ou este subtermo já estava em P'. No primeiro caso, a única meta-variável introduzida tem a forma Y, que pode ser vista como $Y[id]$, ou ainda $Y[\uparrow^0]$, de onde obtemos o resultado por vacuidade. No último caso, o resultado segue por hipótese de indução.

- **Exp-App**: Neste caso, o termo $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ é um subtermo ocorrendo em alguma das equações recém introduzidas ou o mesmo já ocorria em P'. No primeiro caso, todas as novas meta-variáveis têm a forma $H(= H[\uparrow^0])$ e o resultado segue por vacuidade. No último caso a proposição segue por hipótese de indução.

- **Normalise**: Neste caso, ou o termo $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ já era subtermo de P' e o resultado segue por hipótese de indução, ou o mesmo foi gerado por aplicações das regras da Tabela 1.1. No último caso, cada ocorrência da meta-variável Y em P que esteja no escopo de k abstratores, é escrita da forma $Y[\uparrow^k]$ de acordo com a definição de pré-cozimento. Como P é um problema de

segunda ordem, os tipos das meta-variáveis que ocorrem em P são da forma $A_1 \to \ldots \to A_r \to A$, onde os tipos $A_1, \ldots, A_r, A$ são atômicos e portanto os possíveis argumentos de $Y[\uparrow^k]$ têm tipo atômico. De fato, durante o passo de normalização sempre podemos extender um termo para a sua forma η-longa e, desta forma sejam $b_1, \ldots, b_r$ os argumentos de $Y[\uparrow^k]$. Após sucessivas aplicações das regras **Exp-λ** e **Replace** o subtermo $(Y[\uparrow^k]\ b_1 \ldots b_r)$ pode ser reduzido para a forma $Z[b'_1 \cdot \ldots \cdot b'_r \cdot \uparrow^k]$ onde Z é uma nova meta-variável introduzida por uma aplicação da regra **Exp-λ** e os termos $b'_1, \ldots, b'_r$ são atômicos. Os k abstratores que englobam a meta-variável Y em P podem iniciar, no máximo, k simulações de β-reduções e, apesar de estes abstratores terem tipo arbitrário (de qualquer ordem) ao se propagar as possíveis substituições geradas por eles até o subtermo $Z[b'_1 \cdot \ldots \cdot b'_r \cdot \uparrow^k]$, obteremos um novo termo da forma:

$$(Z[b'_1 \cdot \ldots \cdot b'_r \cdot \uparrow^k])[c_1 \cdot \ldots \cdot c_k \cdot \uparrow^m] \tag{5.1}$$

onde $c_1, \ldots, c_k$ são termos de tipo arbitrário, cada um introduzido por uma aplicação da regra (*beta*) ou da regra (*Abs*). De acordo com o Lema 5.3, existe uma estratégia de normalização que reduz o termo (5.1) para a forma: $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ onde $a_1, \ldots, a_p$ são termos atômicos.

- **Replace**: Neste caso, o resultado segue por hipótese de indução.

□

Em outras palavras, a Proposição 5.4 nos diz que qualquer subtermo atômico e flexível construído a partir de um problema de unificação de segunda ordem originado no λ-cálculo simplesmente tipado tem a seguinte estrutura:

$$X[\ \underbrace{a_1 \cdot \ldots \cdot a_p}_{\substack{\text{tipos} \\ \text{atômicos}}} \cdot\ \uparrow^n]$$

Esta caracterização nos permite concluir que todas as possíveis "projeções" sobre os termos $a_1, \ldots, a_p$ são de tipo atômico e portanto não precisam inserir novas meta-variáveis durante o processo de unificação. Desta forma podemos otimizar a regra **Exp-App** para esta situação específica. Antes de apresentar o algoritmo, precisamos definir uma notação adequada para *matching* de ordem superior.

5.3 Notação de Unificação por Transformação

Problemas de *matching* são caracterizados pelo fato de que o lado direito das equações não podem ser instanciados. Utilizando diretamente as regras da Tabela 2.1 problemas de *matching* são transformados em problemas de unificação em geral. De fato, aplicações da regra **Exp-λ** introduzem novas equações da forma flexível-flexível cujo lado direito precisa ser instanciado. Como estamos particularmente interessados em problemas de *matching*, apresentaremos uma notação que nos permita manter em classes distintas os problemas de *matching* e problemas de unificação em geral. Esta notação é baseada em [Nip93] e é conhecida como *unificação por transformação*. Assim, se M é uma conjunção de equações de *matching* na linguagem do $\lambda\sigma$-cálculo, sua representação na notação de unificação por transformação é dada pelo par $\langle \sigma, M \rangle$, onde σ é uma substituição de primeira ordem (*grafting*).

Definição 5.5 (Formas Resolvidas) *Uma* forma resolvida *é um par $\langle \sigma, \{\} \rangle$, onde σ é um* grafting *e o segundo elemento do par é uma conjunção vazia.*

A seguir apresentaremos um algoritmo para *matching* de segunda ordem que resolve qualquer problema pertencente à classe caracterizada na Proposição 5.4. Apesar de esta classe ser um subconjunto próprio do conjunto de todos os problemas de *matching* de segunda ordem que podem ser gerados na linguagem do $\lambda\sigma$-cálculo, temos que a mesma é expressiva o suficiente para conter todos os problemas de segunda ordem que são gerados a partir do λ-cálculo simplesmente tipado. De fato, esta classe contém todos os problemas cujos termos estão na imagem da função de pré-cozimento e, assim é capaz de resolver todos os problemas de *matching* de segunda ordem que são importantes na prática, isto é, todos os problemas que podem ser gerados a partir do λ-cálculo simplesmente tipado. Como veremos, este algoritmo consiste em uma adaptação do procedimento geral de Dowek, Hardin e Kirchner.

5.4 Algoritmo para *Matching* de Segunda Ordem

As regras para *matching* de segunda ordem são dadas na Tabela 5.1. As regras **Dec$_m$-λ**, **Dec$_m$-App**, **Dec$_m$-Fail** e **Normalise$_m$** correspondem exatamente às regras **Dec-λ**, **Dec-App**, **Dec-Fail** e **Normalise** da Tabela 2.1 escritas na notação

Dec$_m$-λ	$\dfrac{\langle \sigma, M \wedge \lambda_A.a \ll^?_{\lambda\sigma} \lambda_A.b\rangle}{\langle \sigma, M \wedge a \ll^?_{\lambda\sigma} b\rangle}$
Dec$_m$-App	$\dfrac{\langle \sigma, M \wedge (\underline{n}\ a_1 \ldots a_p) \ll^?_{\lambda\sigma} (\underline{n}\ b_1 \ldots b_p)\rangle}{\langle \sigma, M \wedge a_1 \ll^?_{\lambda\sigma} b_1 \wedge \ldots \wedge a_p \ll^?_{\lambda\sigma} b_p\rangle}$
Dec$_m$-Fail	$\dfrac{\langle \sigma, M \wedge (\underline{n}\ a_1 \ldots a_p) \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)\rangle}{Falha}$, se $m \neq n$.
Exp$_m$-λ	$\dfrac{\langle \sigma, M\rangle}{\exists Y : (A_1 \cdot \ldots \cdot A_k \cdot \Gamma \vdash Y : B), \langle \sigma', \{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}M\rangle}$ se $(\Gamma \vdash X : A_1 \to \ldots \to A_k \to B) \in \mathcal{TV}ar(M)$, $Y \notin \mathcal{TV}ar(M)$, e X não é uma variável resolvida, onde $\sigma' = \{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}\sigma$.
Imit	$\dfrac{\langle \sigma, M \wedge X[a_1 \cdot \ldots \cdot a_r \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)\rangle}{\langle \sigma', \sigma' M \wedge e_1 \wedge \ldots \wedge e_q\rangle}$ se X tem tipo atômico e $m > n$; onde $\sigma' = \{X \mapsto (\underline{m-n+r}\ H_1 \ldots H_q)\}\sigma$, $e_i = H_i[\sigma' a_1 \cdot \ldots \cdot \sigma' a_r \cdot \uparrow^n] \ll^?_{\lambda\sigma} b_i$ $(1 \leq i \leq q)$, $H_1, \ldots, H_q$ são meta-variáveis novas com tipo apropriado e contexto $\Gamma_{H_i} = \Gamma_X (1 \leq i \leq q)$, e $\underline{m-n+r}$ tem, no máximo, ordem 3.
Proj	$\dfrac{\langle \sigma, M \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)\rangle}{\bigvee_{j \in R_p} \langle \{X \mapsto \underline{j}\}\sigma, \{X \mapsto \underline{j}\}M \wedge \{X \mapsto \underline{j}\}a_j \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)\rangle}$ se X tem tipo atômico e não é uma variável resolvida. onde R_p é um subconjunto de $\{1, \ldots, p\}$ tal que j tem o mesmo tipo de X.
Normalise$_m$	$\dfrac{\langle \sigma, M \wedge a \ll^?_{\lambda\sigma} b\rangle}{\langle \sigma', M \wedge a' \ll^?_{\lambda\sigma} b'\rangle}$ se a ou b não está em forma **Eta**-longa. onde a' (resp. b') é a forma **Eta**-longa de a (resp. b), e σ' é obtido de σ por normalização de todos os seus termos. Se a (resp. b) não é uma variável resolvida e a (resp. b) caso contrário.

Tabela 5.1: Regras para *Matching* de Segunda Ordem

de unificação por transformação. A regra **Exp$_m$-λ** é a adaptação para *matching* da regra **Exp-λ** e, a diferença entre elas, além da notação, é que **Exp$_m$-λ** sempre substitui uma meta-variável de tipo funcional por uma abstração cujo corpo é uma meta-variável nova de tipo atômico. Além disto o *grafting* gerado já é automatica-

mente aplicado ao problema de *matching* atual. Sendo assim, um passo de **Exp$_m$-λ** é equivalente a várias aplicações de **Exp-λ** seguidas por uma aplicação de **Replace**. Observe que se a regra usual **Exp-λ** não for sucedida por uma aplicação de **Replace** então podemos aplicar **Exp-λ** indefinidamente. Para evitar isto, Dowek, Hardin e Kirchner definiram a noção de *estratégia honesta*. Nossa definição de **Exp$_m$-λ** evita a necessidade de uma tal estratégia. As regras **Imit** e **Proj** geram *graftings* para a cabeça X do termo flexível de uma equação da forma flexível-rígida, desde que X tenha tipo atômico. **Imit** está designada para realizar passos de imitação, enquanto que **Proj** realiza as projeções. A principal diferença entre estas duas regras é que **Proj** nunca introduz meta-variáveis em suas substituições. A regra **Imit**, por sua vez, exige que a cabeça do termo rígido seja, no máximo de terceira ordem para evitar que alguma das meta-variáveis introduzidas tenha ordem maior do que 2. Ou seja, como a cabeça do termo rígido tem, no máximo, ordem 3 concluímos que seus argumentos (que correspondem às novas meta-variáveis introduzidas) têm, no máximo, ordem 2.

A seguir provaremos que não é possível construir uma derivação infinita com as regras da Tabela 5.1 a partir de problemas de *matching* de segunda ordem cujos termos perteçam à classe caracterizada na Proposição 5.4. Para isto, definiremos uma medida adequada para os $\lambda\sigma$-termos.

Definição 5.6 (Comprimento de um $\lambda\sigma$-termo) *Seja $a \in \Lambda_{\lambda\sigma}(\mathcal{X})$. Indutivamente definimos $|a|$, o comprimento de a, como sendo:*

- *se $a = X$ ou $a = \underline{1}$ então $|a| = 1$*
- *se $a = (b\ c)$ então $|a| = 1 + |b| + |c|$*
- *se $a = \lambda.b$ então $|a| = 1 + |b|$*
- *se $a = b[s]$ então $|a| = |b| + ||s||$, onde o tamanho da substituição s, denotado por $||s||$, é indutivamente definido por:*
  - *se $s =\uparrow$ ou $s = id$ então $||s|| = 0$*
  - *se $s = c.d$ então $||s|| = |c| + ||d||$*
  - *se $s = u \circ v$ então $||s|| = ||u|| + ||v||$*

Definição 5.7 *Seja M um problema de* matching *da seguinte forma:*

$$a_1 \ll^?_{\lambda\sigma} b_1 \wedge \ldots \wedge a_n \ll^?_{\lambda\sigma} b_n$$

Defina $\mu(M) = (\xi, \xi', \xi'')$ *da seguinte forma:*

- $\xi = \sum_{i=1}^{n} |b_i|$.
- $\xi' =$ *o número de meta-variáveis que ocorrem em* M.
- $\xi'' =$ *a soma das ordens do tipo das meta-variáveis que ocorrem em* M.

Agora denote por $<$ *a ordem lexicográfica usual sobre estas triplas.*

Proposição 5.8 (Terminalidade do Algoritmo de *Matching*) *Derivações de problemas de* matching *de segunda ordem que pertencem à classe caracterizada na Proposição 5.4 utilizando-se as regras da Tabela 5.1 são sempre finitas.*

Prova. Seja M_ϵ um problema de *matching* marcado com a notação de árvores de unificação definida na Seção 2.2. Mostraremos que $\mu(M_\alpha) < \mu(M_{\alpha i})$, onde $M_{\alpha i}$ $(i > 0)$ é obtido de M_α depois de uma aplicação de alguma das regras da Tabela 5.1.

- Uma aplicação de $\mathbf{Dec}_m$**-**λ decresce o comprimento dos termos de ambos os lados da equação considerada e, portanto $\mu(M_\alpha) < \mu(M_{\alpha 1})$.
- Uma aplicação de $\mathbf{Dec}_m$**-App** substitui uma equação por um número finito de novas equações formadas por subtermos da equação que foi substituída e, portanto ξ decresce e temos que $\mu(M_\alpha) < \mu(M_{\alpha 1})$.
- Uma aplicação de $\mathbf{Dec}_m$**-Fail** sempre gera um nó de falha e, portanto pára.
- Uma aplicação de $\mathbf{Exp}_m$**-**λ substitui uma meta-variável de tipo funcional por uma abstração cujo corpo é uma meta-variável de tipo atômico e, daí ξ'' decresce, enquanto que as duas primeiras componentes da tripla permanecem inalteradas. Portanto, $\mu(M_\alpha) < \mu(M_{\alpha 1})$.
- Uma aplicação de **Normalise** tem que se concluir após um número finito de passos porque o $\lambda\sigma$-cálculo é fracamente terminante e, após um número finito de aplicações das regras dadas na Tabela 2.1, uma equação da forma $a \ll^?_{\lambda\sigma} b$ é convertida em uma equação da forma $a' \ll^?_{\lambda\sigma} b$, onde a' é a forma **Eta**-longa de a e portanto, ξ permanece inalterado, ξ' e ξ'' não podem aumentar pois nenhuma das regras do $\lambda\sigma$-cálculo introduz novas meta-variáveis. Daí concluímos que $\mu(M_\alpha) \leq \mu(M_{\alpha 1})$, mas como **Normalise** só pode ser aplicada

uma vez temos apenas duas possibilidades: o problema atual já está em forma resolvida ou alguma das outras regras pode ser aplicada ao problema. No primeiro caso, a derivação pára e, no segundo caso μ decresce estritamente após a aplicação de qualquer que seja a outra regra.

- Uma aplicação de **Imit** decresce ξ pois o lado direito de todas as novas equações geradas é formado por subtermos do lado direito da equação:

$$X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)$$

  e portanto $\mu(M_\alpha) < \mu(M_{\alpha 1})$.

- Sejam $M_{\alpha 1}, \ldots, M_{\alpha r}$ $(r > 0)$ os problemas de *matching* gerados após uma aplicação de **Proj** ao problema M_α, que por hipótese contém uma equação da forma:

$$X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)$$

  onde os termos $a_1, \ldots, a_p$ são atômicos (ver Proposição 5.4). Temos para todo $1 \leq i \leq r$, que $\mu(M_{\alpha i}) < \mu(M_\alpha)$ já que o valor de ξ' decresce porque nenhuma meta-variável nova é introduzida no problema $M_{\alpha i}$ e, ξ permanece inalterado. Desta forma, uma aplicação de **Proj** a um problema de *matching* pode gerar uma disjunção (finita) de outros problemas, de forma que para cada um destes novos problemas gerados, a medida μ decresce. Assim concluímos que não é possível gerar derivações infinitas com este conjunto de regras.

□

A seguir provaremos que o *grafting* gerado como solução de problema de *matching* está sempre na imagem da função de pré-cozimento. Como veremos isto é consequência do fato de que tais *graftings* têm a forma $X \mapsto a$, onde a é um termo fechado.

Proposição 5.9 *Qualquer forma resolvida de um problema de* matching *de segunda ordem, obtida por aplicações das regras da Tabela 5.1, está na imagem da função de pré-cozimento.*

Prova. Se $\langle \sigma, \{\} \rangle$ é uma forma resolvida então qualquer elemento de σ é da forma $X \mapsto a$, onde X é uma meta-variável e a é um termo sem ocorrências de meta-variáveis. Como índices *à la* de Bruijn sempre estão na imagem da função de pré-cozimento, concluímos que a também está na imagem da função de pré-cozimento. □

De acordo com a Proposição 5.9, podemos concluir que para transformarmos um **grafting** (que é solução de um problema de *matching* de segunda ordem) em uma substituição correspondente do λ-cálculo simplesmente tipado então basta removermos a codificação dos índices *à la* de Bruijn. Esquematicamente temos a seguinte situação:

$$M \xrightarrow{\textbf{Pré-cozimento}} M_F \xrightarrow{\text{Algoritmo de } \textit{Matching}} M'_F \xrightarrow{\textbf{Pré-cozimento}^{-1}} M'$$

Exemplo 5.10 *Seja M um problema de em matching de segunda ordem e contendo apenas a equação $\Gamma \vdash \lambda_A.(X\ \underline{3}) \ll^? \lambda_A.(\underline{2}(\underline{43})) : A \to B$, onde $\Gamma = A \to B \cdot A \cdot A \to A \cdot nil$, A e B são tipos atômicos e $\Gamma \vdash X : A \to B$. Após aplicação a função de pré-cozimento vamos obter a equação $\lambda_A.(X[\uparrow]\ \underline{3}) \ll^?_{\lambda\sigma} \lambda_A.(\underline{2}(\underline{43}))$. O algoritmo gera a seguinte redução:*

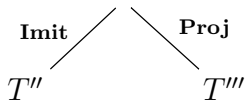
$$\langle\{\}, \{\lambda_A.(X[\uparrow]\ \underline{3}) \ll^?_{\lambda\sigma} \lambda_A.(\underline{2}(\underline{4}\ \underline{3}))\}\rangle$$
$$\Big| \ \mathbf{Dec}_m\text{-}\lambda$$
$$\langle\{\}, \{(X[\uparrow]\ \underline{3}) \ll^?_{\lambda\sigma} (\underline{2}(\underline{4}\ \underline{3}))\}\rangle$$
$$\Big| \ \mathbf{Exp}_m\text{-}\lambda$$
$$\langle\{X \mapsto \lambda_A.Y\}, \{((\lambda_A.Y)[\uparrow]\ \underline{3}) \ll^?_{\lambda\sigma} (\underline{2}(\underline{4}\ \underline{3}))\}\rangle$$
$$\Big| \ \mathbf{Normalise}_m$$
$$\langle\{X \mapsto \lambda_A.Y\}, \{Y[\underline{3}\cdot \uparrow] \ll^?_{\lambda\sigma} (\underline{2}(\underline{4}\ \underline{3}))\}\rangle$$
$$\Big| \ \mathbf{Imit}$$
$$\langle\{Y \mapsto (\underline{2}\ H_1), X \mapsto (\lambda_A.(\underline{2}\ H_1))\}, \{H_1[\underline{3}\cdot \uparrow] \ll^?_{\lambda\sigma} (\underline{4}\ \underline{3})\}\rangle$$
$$\mathbf{Imit} \swarrow \quad \searrow \mathbf{Proj}$$
$$T \qquad\qquad T'$$

onde T' é dada por:

$$\langle\{H_1 \mapsto \underline{1}, Y \mapsto (\underline{2}\ \underline{1}), X \mapsto \lambda_A.(\underline{2}\ \underline{1})\}, \{\underline{1}[\underline{3}\cdot \uparrow] \ll^?_{\lambda\sigma} (\underline{4}\ \underline{3})\}\rangle$$
$$\Big| \ \mathbf{Normalise}_m$$
$$\langle\{H_1 \mapsto \underline{1}, Y \mapsto (\underline{2}\ \underline{1}), X \mapsto \lambda_A.(\underline{2}\ \underline{1})\}, \{\underline{3} \ll^?_{\lambda\sigma} (\underline{4}\ \underline{3})\}\rangle$$
$$\Big| \ \mathbf{Dec}_m\text{-}\mathbf{Fail}$$
$$Falha$$

e T é dada por:

$$\langle\{H_1 \mapsto (\underline{4}\ H_2), Y \mapsto (\underline{2}(\underline{4}\ H_2)), X \mapsto \lambda_A.(\underline{2}(\underline{4}\ H_2))\}, \{H_2[\underline{3}\cdot \uparrow] \ll^?_{\lambda\sigma} \underline{3}\}\rangle$$



onde T'' e T''' são, respectivamente dadas por:

$$\langle\{H_2 \mapsto \underline{3}, H_1 \mapsto (\underline{4}\ \underline{3}), Y \mapsto (\underline{2}(\underline{4}\ \underline{3})), X \mapsto \lambda_A.(\underline{2}(\underline{4}\ \underline{3}))\}, \{\}\rangle$$

$$e$$

$$\langle\{H_2 \mapsto \underline{1}, H_1 \mapsto (\underline{4}\ \underline{1}), Y \mapsto (\underline{2}(\underline{4}\ \underline{1})), X \mapsto \lambda_A.(\underline{2}(\underline{4}\ \underline{1}))\}, \{\}\rangle$$

A seguir apresentamos uma prova de correção e completude para as regras da Tabela 5.1 e para o algoritmo de *matching* de segunda ordem apresentado. Nestas provas codificaremos o problema de *matching* $\langle\sigma, M\rangle$ como sendo o problema de unificação P construído da seguinte maneira:

- $X =^?_{\lambda\sigma} a$ é uma equação de P sempre que $X \mapsto a$ está em σ;
- $X =^?_{\lambda\sigma} a$ é uma equação de P sempre que $X \ll^?_{\lambda\sigma} a$ é uma equação de M.

Proposição 5.11 (Correção do Algoritmo de *Matching*) *As regras dadas na Tabela 5.1 são corretas, isto é, se $\langle\sigma', M'\rangle$ pode ser obtido de $\langle\sigma, M\rangle$ após uma aplicação de uma dessas regras, então $\mathcal{M}_{\lambda\sigma}(\langle\sigma', M'\rangle) \subseteq \mathcal{M}_{\lambda\sigma}(\langle\sigma, M\rangle)$.*

Prova.

1. **Dec$_m$-λ**: Seja γ uma solução do problema $\langle\sigma, M \wedge a \ll^?_{\lambda\sigma} b\rangle$. Como para todo termo a e todo *grafting* γ vale que $\gamma\lambda_A.a =_{\lambda\sigma} \lambda_A.\gamma a$, concluímos que γ é solução de $\langle\sigma, M \wedge \lambda_A.a \ll^?_{\lambda\sigma} \lambda_A.b\rangle$.

2. **Dec$_m$-App**: Seja γ uma solução do problema $\langle\sigma, M \wedge a_1 \ll^?_{\lambda\sigma} b_1\rangle$. Como para qualquer *grafting* γ e termos $\underline{m}, a_1, \ldots, a_r$, vale que $\gamma(\underline{m}\ a_1 \ldots a_r) =_{\lambda\sigma} (\underline{m}\ \gamma a_1 \ldots \gamma a_r)$, concluímos que γ é solução de $\langle\sigma, M \wedge (\underline{m}\ a_1 \ldots a_r) \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_r)\rangle$.

3. **Exp$_m$-λ**: Suponha que γ seja uma solução do problema P' que corresponde à codificação do problema:

$$\langle\{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}\sigma, \{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}M\rangle.$$

Isto significa que as soluções de P' coincidem com as soluções de:

$$\{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}P \wedge X =^{?}_{\lambda\sigma} \lambda_{A_1} \ldots \lambda_{A_k}.Y$$

e daí, $\gamma\{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\} \in \mathcal{M}_{\lambda\sigma}(\langle\sigma, M\rangle)$. Isto nos mostra que:

$$\mathcal{M}_{\lambda\sigma}(\langle\{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}\sigma, \{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}M\rangle) \subseteq \mathcal{M}_{\lambda\sigma}(\langle\sigma, M\rangle).$$

4. **Imit**: Seja γ uma solução de P' que corresponde à codificação do problema:

$$\langle\sigma', \sigma'M \wedge H_1[\sigma'a_1 \cdot \ldots \cdot \sigma'a_r \cdot \uparrow^n] \ll^{?}_{\lambda\sigma} b_1 \wedge \ldots \wedge H_q[\sigma'a_1 \cdot \ldots \cdot \sigma'a_r \cdot \uparrow^n] \ll^{?}_{\lambda\sigma} b_q\rangle$$

dado pela regra **Imit**. Isto significa que γ é solução de:

$$P \wedge H_1[\sigma'a_1 \cdot \ldots \cdot \sigma'a_r \cdot \uparrow^n] =^{?}_{\lambda\sigma} b_1 \wedge \ldots \wedge H_q[\sigma'a_1 \cdot \ldots \cdot \sigma'a_r \cdot \uparrow^n] =^{?}_{\lambda\sigma} b_q.$$

Considerando-se a estrutura da regra **Dec-App** temos a seguinte igualdade:

$$\gamma((\underline{m}\ H_1[\sigma'a_1 \cdot \ldots \cdot \sigma'a_r \cdot \uparrow^n] \ldots H_q[\sigma'a_1 \cdot \ldots \cdot \sigma'a_r \cdot \uparrow^n]) =_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)).$$

Portanto, γ é também solução da seguinte conjunção que está contida na codificação P:

$$\gamma(X[a_1 \cdot \ldots \cdot a_r \cdot \uparrow^n] =^{?}_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \wedge X =^{?}_{\lambda\sigma} (\underline{m-n+r}\ H_1 \ldots H_q)).$$

Como γ também resolve todas as outra equações contidas em P, concluímos que $\gamma \in \mathcal{M}_{\lambda\sigma}(\langle\sigma, M\rangle)$.

5. **Proj**: Seja γ uma solução de P' que corresponde a codificação do problema:

$$\langle\{X \mapsto \underline{j}\}\sigma, \{X \mapsto \underline{j}\}M \wedge \{X \mapsto \underline{j}\}a_j \ll^{?}_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)\rangle$$

dado pela regra **Proj**. Isto significa que:

$$\gamma(\{X \mapsto \underline{j}\}a_j =_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)).$$

Esta última igualdade pode ser reescrita da seguinte forma:

$$\gamma(X[a_1 \cdot \ldots \cdot a_r \cdot \uparrow^n] =_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \wedge X =_{\lambda\sigma} \underline{j})$$

e, esta conjunção está contida na codificação P de $\langle\sigma, M\rangle$. Como γ também resolve todas as outra equações contidas em P, concluímos que $\gamma \in \mathcal{M}_{\lambda\sigma}(\langle\sigma, M\rangle)$.

6. **Normalise**: Se γ é solução de um problema de *matching* cujos termos estão em forma η-longa então, pela terminalidade (fraca) e confluência do $\lambda\sigma$-cálculo, γ também continua sendo solução do problema onde os termos não necessariamente se encontram em forma η-longa visto que as meta-variáveis que ocorrem em ambos os casos são exatamente as mesmas.

□

Proposição 5.12 (Completude das Regras de *Matching*) *O conjunto de regras dado na Tabela 5.1 é completo no sentido que, se $\langle\sigma', M'\rangle$ pode ser deduzido de $\langle\sigma, M\rangle$ por alguma dessas regras, então $\mathcal{M}_{\lambda\sigma}(\langle\sigma, M\rangle) \subseteq \mathcal{M}_{\lambda\sigma}(\langle\sigma', M'\rangle)$.*

Prova.

1. **Dec$_m$-λ**: Seja γ uma solução do problema $\langle\sigma', M' \wedge \lambda_A.a \ll^?_{\lambda\sigma} \lambda_A.b\rangle$. Como para todo termo a e todo *grafting* γ vale que $\gamma\lambda_A.a =_{\lambda\sigma} \lambda_A.\gamma a$, concluímos que γ é solução de $\langle\sigma, M \wedge a \ll^?_{\lambda\sigma} b\rangle$.

2. **Dec$_m$-App**: Seja γ uma solução do problema:

$$\langle\sigma, M \wedge (\underline{m}\ a_1 \ldots a_r) \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_r)\rangle.$$

   Como para qualquer *grafting* γ e termos $\underline{m}, a_1, \ldots, a_r$, vale que

$$\gamma(\underline{m}\ a_1 \ldots a_r) =_{\lambda\sigma} (\underline{m}\ \gamma a_1 \ldots \gamma a_r)$$

   concluímos que γ é solução de:

$$\langle\sigma, M \wedge a_1 \ll^?_{\lambda\sigma} b_1 \wedge \ldots \wedge a_r \ll^?_{\lambda\sigma} b_r\rangle.$$

3. **Exp$_m$-λ**: Seja γ uma solução do problema P, codificação de $\langle\sigma, M\rangle$, onde M contém pelo menos uma ocorrência de meta-variável não resolvida, digamos X, de tipo funcional da forma $A_1 \to \ldots \to A_k \to A$ ($k > 0$). Queremos mostrar que γ é também solução de $\{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}P \wedge X =^?_{\lambda\sigma} \lambda_{A_1} \ldots \lambda_{A_k}.Y$, que corresponde à codificação do problema:

$$\langle\{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}\sigma, \{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}M\rangle$$

   obtido de $\langle\sigma, M\rangle$ após uma aplicação de **Exp$_m$-λ**. De fato, por definição, γ resolve todas as equações de $\{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}P$, exceto aquelas que

contém ocorrências de Y, mas neste caso a informação sobre o termo que deve substituir Y é obtida da equação $\gamma X =^{?}_{\lambda\sigma} \lambda_{A_1} \ldots \lambda_{A_k}.Y$, já que sabemos que γ possui uma substituição para X, sempre que X seja não resolvida.

4. **Imit** e **Proj**: Seja γ uma solução de P, codificação de:

$$\langle \sigma, M \wedge X[a_1 \cdot \ldots \cdot a_r \cdot \uparrow^n] \ll^{?}_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \rangle,$$

onde $r, n, q \geq 0$ e X é uma meta-variável de tipo atômico. Sendo assim, temos que $\gamma X = (\underline{j}\ c_1 \ldots c_s)$ $(s \geq 0)$ e, aplicando γ à codificação P, obtemos a igualdade:

$$(\underline{j}\ c_1 \ldots c_s)[\gamma a_1 \cdot \ldots \cdot \gamma a_r \cdot \uparrow^n] =_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \tag{5.2}$$

Temos dois casos a considerar:

- $j \leq r$ (Projeção): Neste caso, de acordo com a Proposição 5.4 temos que $s = 0$ e, portanto após alguns passos de normalização, a igualdade (5.2) assume a forma $\gamma a_j =_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)$, e assim concluímos que γ é solução de:

$$\langle \{X \mapsto \underline{j}\}\sigma, \{X \mapsto \underline{j}\}M \wedge \{X \mapsto \underline{j}\}a_j \ll^{?}_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \rangle.$$

- $j > r$ (Imitação): Neste caso, a igualdade 5.2 é equivalente a:

$$(\underline{j - r + n}\ c_1[\gamma a_1 \cdot \ldots \cdot \gamma a_r \cdot \uparrow^n] \ldots c_s[\gamma a_1 \cdot \ldots \cdot \gamma a_r \cdot \uparrow^n]) =_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)$$

e, portanto $j - r + n = m$. Daí concluímos que $j = m - n + r$ e $s = q$. Consequentemente temos as seguinte igualdades:
$c_1[\gamma a_1 \cdot \ldots \cdot \gamma a_r \cdot \uparrow^n]) =_{\lambda\sigma} b_1$
$\vdots$
$c_s[\gamma a_1 \cdot \ldots \cdot \gamma a_r \cdot \uparrow^n]) =_{\lambda\sigma} b_s$.
Estas igualdades nos permitem resolver as q equações geradas pela regra **Imit** e, portanto γ é solução de:

$$\langle \sigma', \sigma' M \wedge H_1[\sigma' a_1 \cdot \ldots \cdot \sigma' a_r \cdot \uparrow^n] \ll^{?}_{\lambda\sigma} b_1 \wedge \ldots \wedge H_q[\sigma' a_1 \cdot \ldots \cdot \sigma' a_r \cdot \uparrow^n] \ll^{?}_{\lambda\sigma} b_q \rangle.$$

□

A proposição a seguir nos fornece uma noção de completude para as regras de *matching* no sentido que as aplicações destas regras preservam as soluções dos problemas.

Proposição 5.13 (Completude do Algoritmo de *Matching*) *Seja $\langle \sigma, M\rangle$ um problema de* matching *de segunda ordem em forma não-resolvida e que possui uma solução. Então $\langle \sigma, M\rangle$ pode ser reduzido para uma forma resolvida.*

Prova. Mostraremos que qualquer problema de *matching* de segunda ordem que tenha solução e que não esteja em forma resolvida pode ser reduzido utilizando-se alguma das regras de *matching* dadas na Tabela 5.1. Formas resolvidas devem ser obrigatoriamente obtidas como consequência do Teorema 5.8 que garante a finitude de qualquer derivação que utilize estas regras. A prova é por análise de casos:

- Se $\langle \sigma, M\rangle$ contém algum termo que não está em forma η-longa então a regra **Normalise** pode ser aplicada.
- Se $\langle \sigma, M\rangle$ contém alguma meta-variável de tipo funcional então a regra $\mathbf{Exp}_m$-λ pode ser aplicada.
- Se todas as meta-variáveis que ocorrem em $\langle \sigma, M\rangle$ são atômicas então temos dois casos a considerar:

  1. Existe uma meta-variável que ocorre na cabeça de alguma equação de M: Neste caso, M possui uma equação da forma:

  $$X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \tag{5.3}$$

  onde $n, p, q \geq 0$ e os b_i's são termos fechados, isto é sem ocorrência de meta-variáveis. Por hipótese, existe um *grafting* $X \mapsto (\underline{k}\ c_1 \ldots c_s)$ que é solução da equação (5.3). Se $k \leq p$ então o tipo alvo de algum dos a_i's coincide com o tipo de X e portanto a regra **Proj** pode ser aplicada. Quando $k > p$ então normalizando a equação:

  $$(\underline{k}\ c_1 \ldots c_s)[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)$$

  concluímos que $k - p + n = m$, de onde segue que $m > n$ e portanto a regra **Imit** pode ser aplicada.

  2. Todas as meta-variáveis ocorrem internamente nas equações de $\langle \sigma, M\rangle$: Se um termo que contém tais ocorrências tem tipo funcional então **Dec-λ** se aplica, caso contrário a cabeça do termo é um índice *à la* de Bruijn e **Dec-App** pode ser aplicada.

□

O algoritmo proposto quando aplicado a um problema de *matching* vai gerar uma forma resolvida e em seguida pára. Uma forma resolvida representa uma solução em forma de *grafting*. Por outro lado, o algoritmo também pára quando o problema original não tem solução. Observe que a regra **Dec$_m$-Fail** detecta apenas falhas triviais, isto é, falhas geradas por equações da forma rígida-rígida. O exemplo a seguir apresenta o comportamento do algoritmo para um problema sem solução.

Exemplo 5.14 *Sejam A, B tipos atômicos, $\Gamma = B \cdot nil$ um contexto e $\Gamma \vdash Y : B \to B \to A$. Considere o seguinte problema de* matching *no λ-cálculo simplesmente tipado:*

$$\lambda_A\lambda_B.(Y\ \underline{3}\ \underline{1}) \ll^? \lambda_A\lambda_B.\underline{2}$$

que está bem tipado no contexto Γ. Para tentar resolver este problema utilizando nosso algoritmo, precisamos convertê-lo para a notação de unificação por transformação:

$$\langle\{\}, \{\lambda_A\lambda_B.(Y[\uparrow^2]\ \underline{3}\ \underline{1}) \ll^? \lambda_A\lambda_B.\underline{2}\}\rangle.$$

Após duas aplicações de **Dec$_m$-λ**, *obtemos:*

$$\langle\{\}, \{(Y[\uparrow^2]\ \underline{3}\ \underline{1}) \ll^? \underline{2}\}\rangle.$$

Como Y tem tipo funcional, podemos aplicar a regra **Exp$_m$-λ**:

$$\langle\{Y \mapsto \lambda_B\lambda_B.X\}, \{((\lambda_B\lambda_B.X)[\uparrow^2]\ \underline{3}\ \underline{1}) \ll^? \underline{2}\}\rangle$$

onde X é uma meta-variável nova de tipo A e contexto Γ. Após um passo de **Normalise**, *temos:*

$$\langle\{Y \mapsto \lambda_B\lambda_A.X\}, \{X[\underline{1} \cdot \underline{3} \cdot \uparrow^2] \ll^? \underline{2}\}\rangle.$$

Como os termos $\underline{1}$ e $\underline{3}$ têm tipo B e X tem tipo A, a regra **Proj** *não se aplica. Além disto, o operador $\uparrow^2$ impede que X seja substituído por variáveis que eram ligadas no problema original e, desta forma* **Imit** *também não pode ser aplicada. O algoritmo então pára nesta forma não resolvida indicando que o problema não tem solução.*

Conclusão

Neste trabalho estudamos unificação de ordem superior em cálculos de substituições explícitas. Em particular comparamos o algoritmo de Huet para o λ-cálculo simplesmente tipado [Hue75] com o método desenvolvido por Dowek, Hardin e Kirchner para o $\lambda\sigma$-cálculo [DHK00]. Este último, por sua vez também é comparado com o método de unificação desenvolvido por Ayala-Rincón e Kamareddine para o λs_e-cálculo [ARK01]. Adicionalmente, como aplicação das idéias desenvolvidas para unificação, apresentamos uma adaptação do procedimento de unificação para decidir o problema de *matching* de segunda ordem no $\lambda\sigma$-cálculo. As principais contribuições deste trabalho são:

- Enriquecemos a noção de *árvore de matching* de Huet introduzindo uma nova estrutura chamada *árvore de unificação*. As árvores de unificação refinam a noção de árvores de *matching* porque conseguem explicitar todos os passos do algoritmo de Huet, enquanto que nas árvores de *matching* de Huet os passos de simplificação são implícitos. Além disto, esta estrutura se mostrou essencial para que pudéssemos fornecer uma apresentação precisa do algoritmo de Huet e, também para estabelecermos a correspondência estrutural entre HOU *à la* Huet e via substituições explícitas.

- Adaptamos o algoritmo de Huet para a notação *à la* de Bruijn. Apesar de ser uma translação simples da apresentação tradicional com nomes, esta é importante para simplificar a comparação entre os métodos de unificação em estudo já que esta é a notação utilizada pelos cálculos de substituições explícitas aqui considerados.

- Utilizando as noções de árvores de unificação para o λ-cálculo e árvores de derivação para o $\lambda\sigma$-cálculo, respectivamente, combinadas com a noção de pseudo-cozimento, provamos que o algoritmo de unificação de Huet e o método de unificação do $\lambda\sigma$-cálculo preservam uma importante relação estrutural entre subproblemas: para um dado problema de unificação P no λ-cálculo sim-

plesmente tipado, temos que para cada subproblema de P em sua árvore de unificação $\mathcal{A}(P)$, existe uma contra-parte na árvore de derivação de sua forma pré-cozida P_F. Isto nos permite concluir que o método de unificação no $\lambda\sigma$-cálculo é uma generalização do algoritmo de Huet e que, soluções computadas por este são sempre computadas pelo primeiro.

- Mostramos que unificação no $\lambda\sigma$-cálculo e no λs_e-cálculo podem gerar árvores de derivação iguais desde que seja utilizada a mesma estratégia em ambos os cálculos. Esta comparação é feita utilizando-se duas funções T e L que traduzem termos do λs_e-cálculo para o $\lambda\sigma$-cálculo e vice-versa, respectivamente. Desta forma a relação estrutural entre HOU *à la* Huet e HOU no $\lambda\sigma$-cálculo também é válida para o λs_e-cálculo.

- Adaptamos o procedimento de unificação de [DHK00] para problemas de *matching* de segunda ordem. Para isto, caracterizamos uma classe de problemas de segunda ordem que nos permite construir um algoritmo específico para *matching* de segunda ordem no $\lambda\sigma$-cálculo. Este algoritmo utiliza uma notação apropriada para tratar problemas de *matching* e é baseada na noção de unificação por transformação introduzida em [Nip93].

Acreditamos que esta comparação estrutural é importante para uma melhor compreensão dos métodos de unificação baseados em substituições explícitas além de ajudar na compreensão de questões práticas e implementacionais assim como no papel das substituições explícitas em unificação de ordem superior. Extensões naturais deste trabalho incluem:

- A elaboração de uma versão otimizada do método de unificação do $\lambda\sigma$-cálculo utilizando idéias relacionadas com o conceito de pseudo-cozimento. De fato, o pseudo-cozimento combina a noção usual de pré-cozimento simultaneamente com a aplicação de regras de unificação, permitindo assim com que a simulação de vários passos de unificação sejam realizados de uma só vez.

- Atualmente estamos investigando a possibilidade de expandir as idéias desenvolvidas para *matching* de segunda ordem para ordens maiores. Para o caso específico de *matching* de terceira ordem, estamos adaptando o algoritmo de Dowek [Dow94] a partir de uma noção expandida de árvores de Böhm que tem se mostrado adequada para a linguagem do $\lambda\sigma$-cálculo. Uma versão preliminar deste trabalho pode ser encontrada em [dMKAR05b].

Referências

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

[ARdMK05] M. Ayala-Rincón, F.L.C. de Moura, and F. Kamareddine. Comparing and implementing calculi of explicit substitutions with eta-reduction. *Annals of Pure and Applied Logic*, 134:5–41, 2005.

[ARK01] M. Ayala-Rincón and F. Kamareddine. Unification via the λs_e-Style of Explicit Substitution. *The Logical journal of the Interest Group in Pure and Applied Logics*, 9(4):489–523, 2001.

[ARK03] 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. *Matemática Contemporânea*, 24:1–22, 2003.

[Bar84] H. P. Barendregt. *The Lambda Calculus : Its Syntax and Semantics (revised edition)*. North Holland, 1984.

[Bar92] H. P. Barendregt. λ-calculi with types. *Handbook of Logic in Computer Science*, II, 1992.

[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 *LNCS*, pages 363–368. Springer Verlag, 1995.

[Bur89] H.J. Burckert. Matching - a special case of unification? *journal of Symbolic Computation*, 8:523–536, 1989.

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

[CH85] T. Coquand and G. Huet. Constructions: a higher-order proof system for mechanizing mathematics. *EUROCAL85 in LNCS 203*, 1985.

[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.

[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. *Inf. Comput.*, 157(1-2):183–235, 2000.

[dlTC87] T. B. de la Tour and R. Caferra. Proof analogy in interactive theorem proving: A method to express and use it via second order pattern matching. In *Proceedings of AAAI 87*, pages 95–99. Morgan Kaufmann, 1987.

[dMARK06] F.L.C. de Moura, M. Ayala-Rincón, and F. Kamareddine. SUBSEXPL: A Framework for Simulating and Comparing Explicit Substitutions Calculi. *Journal of Applied and Non-classical Logics*, 16(1-2):119–150, 2006.

[dMKAR05a] F.L.C. de Moura, F. Kamareddine, and M. Ayala-Rincón. Second order matching via explicit substitutions. In F. Baader and A. Voronkov, editors, *11th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'04)*, volume 3452 of *LNAI*, pages 433–448. Springer-Verlag, 2005.

[dMKAR05b] F.L.C. de Moura, F. Kamareddine, and M. Ayala-Rincón. Third-Order Matching via Explicit Substitutions. Departamento de Matemática, Universidade de Brasília, unpublished document available at `http://ayala.mat.unb.br/publications.html`, 2005.

[Dow94] G. Dowek. Third order matching is decidable. *APAL*, 69:135–155, 1994.

[Dow01] G. Dowek. Higher-Order Unification and Matching. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 16, pages 1009–1062. MIT press and Elsevier, 2001.

[Gol81] W. Goldfarb. The Undecidability of the Second-Order Unification Problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[Hin97] J. R. Hindley. *Basic Simple Type Theory*. Number 42 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.

[HL78] G. Huet and B. Lang. Proving and applying program transformations expressed with second order patterns. *Acta Informatica*, 11:31–55, 1978.

[Hue73] G. Huet. The undecidability of unification in third order logic. *Information and Control*, 22(3):257–267, April 1973.

[Hue75] G. Huet. A Unification Algorithm for Typed λ-Calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[Hue76] G. Huet. *Résolution d'équations dans les langages d'ordre 1,2,...,ω*. PhD thesis, University Paris-7, 1976.

[Hue02] G. Huet. Higher Order Unification 30 years Later. In V. A. Carreño, C. A. Muñoz, and S. Tahar, editors, *Theorem Proving in Higher Order Logics - TPHOLs 2002*, volume 2410 of *LNCS*, pages 3–12. Springer, 2002.

[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.

[Loa03] R. Loader. Higher order β matching is undecidable. *Logic journal of the Interest Group in Pure and Applied Logics*, 11(1):51–68, 2003.

[Luc72] C. L. Lucchesi. The undecidability of the unification problem for third order languages. Technical Report CSRR 2060, University of Waterloo, 1972.

[Mau85] M. Mauny. *Compilation des langages fonctionnels dans les combinateurs catégoriques - Application au langage ML.* PhD thesis, Université Paris VII, Paris, France, 1985.

[Nip93] T. Nipkow. Functional unification of higher-order patterns. In *Proc. 8th IEEE Symp. Logic in Computer Science*, pages 64–74, 1993.

[NM88] G. Nadathur and D. Miller. An Overview of λProlog. In K.A. Bowen and R.A. Kowalski, editors, *Proc. 5th Int. Logic Programming Conference*, Cambridge, MA, pages 810–827. MIT Press, 1988.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *lncs*. Springer, 2002.

[Pad00] V. Padovani. Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, 10(3):361–372, 2000.

[PS99] F. Pfenning and C. Schürmann. System description: Twelf - a meta-logical framework for deductive systems. In *16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *LNAI*. Springer-Verlgar, 1999.

[Río93] A. Ríos. *Contributions à l'étude de λ-calculs avec des substitutions explicites*. Thèse de doctorat, Université Paris VII, 1993.

[Rob65] J. A. Robinson. A Machine-oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Ven06] D. L. Ventura. Cálculos de substituições explícitas que preservam a propriedade de redução do sujeito. Master's thesis, Departamento de Matemática, Universidade de Brasília, 2006.