Notas em Teoria dos Tipos

Mateus Galdino

Recife, 2023

Contents

1 Prefácio

Esse conjunto de notas reunidas aqui estão relacionadas à Teoria dos Tipos, suas subáreas (Por exemplo, a Teoria dos Tipos Homotópica) e suas áreas irmãs (Teoria das Categorias, etc). Para isso, vou me basear nos seguintes textos:

Para algumas sitações que eu queria explicitar, usarei as seguintes abreviações:
[HoTT] = Homotopy Type Theory: Univalent foundations of mathematics. Voevodsky et al.
[TTFP] = Type Theory and Formal Proof. Geuvers e Nederpelt
[TPL] = Types and Programming Languages. Benjamin Pierce
[IHoTT] = Introduction to Homotopy Type Theory. Egbert Rijke

Part I
O Cubo Lambda

1 Cálculo Lambda não-tipado ($\lambda_{\beta\eta}$)

A teoria dos tipos possui como história de origem algumas tentativas falhas. O conceito de tipos pode ser mapeado para dois matemáticos importantes que fizeram usos bem diferentes dele: Bertrand Russel (e Walfred North Whitehead) na Principia Mathematica e Alonzo Church no seu Cálculo λ simplesmente tipado (STλC).

A teoria dos tipos que é usada hoje, provém do segundo autor e de outros autores que vêm dessa tradição. Por isso, o início dessas notas se propõe a começar do básico, definindo o que é o Cálculo λ não tipado e quais questões levaram Church a desenvolver a teoria dos tipos em cima dele.

Aqui, será traduzido "λ-calculus" como "Cálculo λ", decisão que perde a estética do hífen, mas que mantém a unidade com outras traduções de "X calculus" no corpo matemático brasileiro, como o "Cálculo Diferencial e Integral", o "Cálculo de sequentes", o "Cálculo de variações", etc.

1.1 O Cálculo

1.1.1 Definições

O cálculo lambda serve como uma abstração em cima do conceito de função. Uma função é uma estrutura que pega um *input* e retorna um *output*, por exemplo a função $f(x) = x^2$ pega um input x e retorna seu valor ao quadrado x^2. No cálculo lambda, essa função pode ser denotada por $\lambda x.x^2$, onde $\lambda x.$ simboliza que essa função espera receber como entrada x. Quando se quer saber qual valor a função retorna para uma entrada específica, são usados números no lugar das variáveis, como por exemplo $f(3) = 3^2 = 9$. No cálculo lambda, isso é feito na forma de $(\lambda x.x^2)(3)$.

Esses dois principrios de construção são definidos como:

- **Abstração**: Seja M uma expressão e x uma variável, podemos construir uma nova expressão $\lambda x.M$. Essa expressão é chamada de Abstração de x sobre M

- **Aplicação**: Sejam M e N duas es expressões, podemos construir uma expressão MN. Essa expressão é chamada de Aplicação de M em N.

Dadas essas operações, é preciso também de uma definição que dê conta do processo de encontrar o resultado após a aplicação em uma função. Esse processo é chamado de β-redução. Ela faz uso da substituição e usa como notação os colchetes.

Definição 1.1 (β-redução)**.** A β-redução é o processo de resscrita de uma expressão da forma $(\lambda x.M)N$ em outra expressão $M[x := N]$, ou seja, a expressão M na qual todo x foi substituido por N.

1.1.2 Sintáxe do Cálculo Lambda

É interessante definir a sintaxe do cálculo lambda de forma mais formal. Para isso, são utilizados métodos que podem ser familiares para aqueles que já trabalharam com lógica proposicional, lógica de primeira ordem ou teoria de modelos.

Primeiro, precisamos definir a linguagem do Cálculo λ.

Definição 1.2. (i) Os *termos lambda* são palavras em cima do seguinte alfabeto:

- variáveis: $v_0, v_1, \ldots$
- abstrator: λ
- parentesis: $($, $)$

(ii) O conjunto de λ-termos Λ é definido de forma indutiva da seguinte forma:

- Se x é uma variável, então $x \in \Lambda$
- $M \in \Lambda \to (\lambda x.M) \in \Lambda$
- $M, N \in \Lambda \to MN \in \Lambda$

Na teoria dos tipos e no cálculo lambda, é utilizada uma forma concisa de definir esses termos chamada de Formalismo de Backus-Naur ou Forma Normal de Backus (BNF, em inglês). Nessa forma, a definição anterior é reduzida à:

$$\Lambda = V|(\Lambda\Lambda)|(\lambda V\Lambda)$$

Onde V é o conjunto de variáveis $V = \{x, y, z, \ldots\}$

Para expressar igualdade entre dois termos de Λ utilizamos o simbolo $\equiv$.

Algumas definições indutivas podem ser formadas a partir da definição dos λ-termos.

Definição 1.3 (Multiconjunto de subtermos)**.**

1. (Base) $Sub(x) = \{x\}$, para todo $x \in V$
2. (Aplicação) $Sub((MN)) = Sub(M) \cup Sub(N) \cup \{(MN)\}$
3. (Abstração) $Sub((\lambda x.M)) = Sub(M) \cup \{(\lambda x.M)\}$

Observações:
(i) Um subtermo pode ocorrer múltiplas vezes, por isso é escolhido chamar de multiconjunto (ii) A abstração de vários termos ao mesmo tempo pode ser escrita como $\lambda x.(\lambda y.x)$ ou como $\lambda xy.x$.

Lemma 1.1 (Propriedades de Sub)**.**

- (Reflexividade) Para todo λ-termo M, temos que $M \in Sub(M)$
- (Transitividade)Se $L \in Sub(M)$ e $M \in Sub(N)$, então $L \in Sub(N)$.

Definição 1.4 (Subtermo próprio)**.** L é um subtermo próprio de M se L é subtermo de M e $L \not\equiv M$

Exemplos:

1. Seja o termo $\lambda x.\lambda y.xy$, vamos calcular seus subtermos:

$$\begin{aligned} Sub(\lambda x.\lambda y.xy) &= \{\lambda x.\lambda y.xy\} \cup Sub(\lambda y.xy) \\ &= \{\lambda x.\lambda y.xy\} \cup \{\lambda y.xy\} \cup Sub(xy) \\ &= \{\lambda x.\lambda y.xy\} \cup \{\lambda y.xy\} \cup Sub(x) \cup Sub(y) \\ &= \{\lambda x.\lambda y.xy, \lambda y.xy, x, y\} \end{aligned}$$

2. Seja o termo $(y(\lambda x.(xyz)))$, vamos calcular os seus subtermos:

$$\begin{aligned} Sub(y(\lambda x.(xyz))) &= Sub(y) \cup Sub((\lambda x.(xyz))) \\ &= \{y\} \cup \{(\lambda x.(xyz))\} \cup Sub((xyz)) \\ &= \{y\} \cup \{(\lambda x.(xyz))\} \cup Sub(x) \cup Sub(y) \cup Sub(z) \\ &= \{y\} \cup \{(\lambda x.(xyz))\} \cup \{x\} \cup \{y\} \cup \{z\} = \{y, (\lambda x.(xyz)), x, y, z\} \end{aligned}$$

Outro conjunto importante para a sintaxe do cálculo lambda é o de variáveis livres. Uma variável é dita *ligante* se está do lado do λ. Em um termo $\lambda x.M$, x é uma variável ligante e toda aparição de x em M é chamada de *ligada*. Se existir uma variável em M que não é ligante, então dizemos que ela é *livre*. Por exemplo, em $\lambda x.xy$, o primeiro x é ligante, o segundo x é ligado e y é livre.

O conjunto de todas as variáveis livres em um termo é denotado por FV e definido da seguinte forma:

Definição 1.5 (Multiconjunto de variáveis livres)**.**

1. (Base) $FV(x) = \{x\}$, para todo $x \in V$
2. (Aplicação) $FV((MN)) = FV(M) \cup FV(N) \cup \{(MN)\}$
3. (Abstração) $FV((\lambda x.M)) = FV(M) \setminus \{x\}$

Exemplos:

1. Seja o termo $\lambda x.\lambda y.xyz$, vamos calcular seus subtermos:

$$\begin{aligned} FV(\lambda x.\lambda y.xyz) &= FV(\lambda y.xyz) \setminus \{x\} \\ &= FV(xyz) \setminus \{y\} \setminus \{x\} \\ &= FV(x) \cup FV(y) \cup FV(z) \setminus \{y\} \setminus \{x\} \\ &= \{x, y, z\} \setminus \{y\} \setminus \{x\} \\ &= \{z\} \end{aligned}$$

Vamos definir os termos fechados da seguinte forma:

Definição 1.6. O λ-termo M é dito *fechado* se $FV(M) = \emptyset$. Um λ-termo fechado também é chamado de *combinador*. O conjunto de todos os λ-termos fechados é chamado de Λ^0.

Os combinadores são muito utilizados na *Lógica Combinatória*, mas vamos explorá-los mais a frente.

1.1.3 Conversão

No cálculo Lambda, é possível renomear variáveis ligantes/ligadas, pois a mudança dos nomes dessas variáveis não muda a sua interpretação. Por exemplo, $\lambda x.x^2$ e $\lambda u.u^2$ podem ser utilizadas de forma igual, mesmo que com nomes diferentes. A Renomeação será definida da seguinte forma:

Definição 1.7. Seja $M^{x\to y}$ o resultado da troca de todas as livre-ocorrencias de x em M por y. A relação de renomeação é expressa pelo símbolo $=_\alpha$ e é definida como: $\lambda x.M =_\alpha \lambda y.M^{x\to y}$, dado que $y \notin FV(M)$ e y não seja ligante em M.

Podemos extender essa definição para a definição do renomeamento, chamado de α-conversão.

Definição 1.8 (α-conversão)**.**

1. (Renomeamento) $\lambda x.M =_\alpha \lambda y.M^{x\to y}$
2. (Compatibilidade) Sejam M, N, L termos. Se $M =_\alpha N$, então $ML =_\alpha NL$, $LM =_\alpha LN$
3. (Regra ξ) Para um z qualquer, $\lambda z.M = \lambda z.N$
4. (Reflexividade) $M =_\alpha M$
5. (Simetria) Se $M =_\alpha N$, então $N =_\alpha M$
6. (Transitividade) Se $L =_\alpha M$ e $M =_\alpha N$, então $L =_\alpha N$

A partir dos pontos (3), (4) e (5) dessa definição, é possível dizer que a α-conversão é uma relação de equivalência, chamada de α-equivalência.

Exemplos:

1. $(\lambda x.x(\lambda z.xy)) =_\alpha (\lambda u.u(\lambda z.uy))$
2. $(\lambda x.xy) \neq_\alpha (\lambda y.yy)$

1.1.4 Substituição

Podemos definir agora a substituição de um termo por outro da seguinte forma:

Definição 1.9 (Substituição)**.**

1. $x[x := N] \equiv N$
2. $y[y := x] \equiv y$, se $x \not\equiv y$
3. $(PQ)[x := N] \equiv (P[x := N])(Q[x := N])$
4. $(\lambda y.P)[x := N] \equiv (\lambda z.P^{y \to z})[x := N]$ se $(\lambda z.P^{y \to z})$ é α-equivalente a $(\lambda y.P)$ e $z \notin FV(N)$

A notação $[x := N]$ é uma meta-notação, pois não está definida na sintaxe do cálculo lambda. Na literatura também é possível ver a notação $[N/x]$ para definir a substituição.

1.1.5 Beta redução

Voltando à aplicação, agora com a substituição em mente, podemos dizer que a aplicação de um termo N em $\lambda x.M$, na forma de $(\lambda x.M)N$ é a mesma coisa que $M[x := N]$. Nesse caso, essa única substituição entre termos pode ser descrita na seguinte definição:

Definição 1.10 (β-redução para único passo)**.**

1. (Base) $(\lambda x.M)N \to_\beta M[x := N]$
2. (Compatibilidade) Se $M \to_\beta N$, então $ML \to_\beta NL$, $LM \to_\beta LN$ e $\lambda x.M \to_\beta \lambda x.N$

O termo $(\lambda x.M)N$ é chamado de *redex*, vindo do ingles "reducible expression" (expressão reduzível), e o subtermo $M[x := N]$ é chamado de *contractum* do redex.

Exemplos:

1. $(\lambda x.x(xy))N \to_\beta N(Ny)$
2. $(\lambda x.xx)(\lambda x.xx) \to_\beta (\lambda x.xx)(\lambda x.xx)$
3. $(\lambda x.(\lambda y.yx)z)v \to_\beta (\lambda y.yv)z \to_\beta zv$

Os exemplos 2 e 3 são importantes por duas razões:

- Com o exemplo 3 é possível ver que é possível concatenar várias reduções seguindas, vamos colocar uma definição mais geral a diante que lide com isso.

- Com o exemplo 2 é possível ver que existem termos que, quando beta-reduzidos, retornam eles mesmos. Isso faz com que cálculo lámbda não tipado tenha propriedades interessantes, pois muitas vezes a simplificação não termina. Ou seja, é possível haver cadeias de beta redução que não possuem termo mais simples.

Definição 1.11 (β-redução para zero ou mais passos)**.** $M \twoheadrightarrow_\beta N$ (lê-se: M beta reduz para N em vários passos) se existe um $n \geq 0$ e existem termos M_0 até M_n tais que $M_0 \equiv M$, $M_n \equiv N$ e para todo i tal que $0 \leq i < n$:

$$M_i \to_\beta M_{i+1}$$

Ou Seja:

$$M \equiv M_0 \to_\beta M_1 \to_\beta \cdots \to_\beta M_{n-1} \to_\beta M_n \equiv N$$

Lemma 1.2.

1. $\twoheadrightarrow_\beta$ é uma extensão de $\to_\beta$, ou seja: se $M \to_\beta N$, então $M \twoheadrightarrow_\beta N$
2. $\twoheadrightarrow_\beta$ é reflexivo e transitivo, ou seja:
   - (reflexividade) Para todo M, $M \twoheadrightarrow_\beta M$
   - (transitividade) Para todo L, M, e N. Se $L \twoheadrightarrow_\beta M$ e $M \twoheadrightarrow_\beta N$, então $L \twoheadrightarrow_\beta N$

Prova

1. Na definição 1.11, seja $n = 1$, então $M \equiv M_0 \to_\beta M_1 \equiv N$, que é a mesma coisa que $M \to_\beta N$
2. Se $n = 0$, $M \equiv M_0 \equiv N$
3. A transitividade também segue da definição

Uma extensão dessa β-redução geral é a β- conversão, definida como:

Definição 1.12 (β-conversão)**.** $M =_\beta N$ (lê-se: M e N são β-convertíveis) se existe um $n \geq 0$ e existem termos M_0 até M_n tais que $M_0 \equiv M$, $M_n \equiv N$ e para todo i tal que $0 \leq i < n$: Ou $M_i \to_\beta M_{i+1}$ ou $M_{i+1} \to_\beta M_i$

Lemma 1.3.

1. $=_\beta$ é uma extensão de $\twoheadrightarrow_\beta$ em ambas as direções, ou seja: se $M \twoheadrightarrow_\beta N$ ou $N \twoheadrightarrow_\beta M$, então $M =_\beta N$
2. $=_\beta$ é uma relação de equivalência, ou seja, possui reflexividade, simetria e transitividade
   - (reflexividade) Para todo M, $M =_\beta M$
   - (Simetria) Para todo M e N, se $M =_\beta N$, então $N =_\beta M$
   - (transitividade) Para todo L, M, e N. Se $L =_\beta M$ e $M =_\beta N$, então $L =_\beta N$

1.1.6 Forma Normal

Podemos definir a hora de parar de reduzir, para isso vamos introduzir o conceito de forma Normal

Definição 1.13 (Forma normal β ou β-normalização)**.**

1. M está na forma normal β se M não possui nenhum redex
2. M possui uma formal normal β, ou é β-normalizável, se existe um N na forma normal β tal que $M =_\beta N$. N é chamado de a *forma normal* β de M.

Lemma 1.4. Se M está em sua forma normal β, então $M \twoheadrightarrow_\beta N$ implica em $M \equiv N$

Exemplos:

1. $(\lambda x.(\lambda y.yx)z)v$ tem como formal normal β zv, pois $(\lambda x.(\lambda y.yx)z)v \rightarrow_\beta zv$ (como visto nos exemplos anteriores) e zv está na forma normal β
2. Vamos definir um termo $\Omega := (\lambda x.xx)(\lambda x.xx)$, Ω não está na forma normal β, pois pode ser β-reduzido, mas não possui também forma normal β, pois ele sempre é β-reduzido para ele mesmo.
3. Seja $\Delta := (\lambda x.xxx)$, então $\Delta\Delta \rightarrow_\beta \Delta\Delta\Delta \rightarrow_\beta \Delta\Delta\Delta\Delta \rightarrow_\beta \ldots$. Logo $\Delta\Delta$ não possui forma normal.

Definição 1.14 (Caminho de Redução)**.**
Um caminho de redução finito de M é uma sequência finita de termos $N_0, N_1, \ldots, N_n$ tais que $N_0 \equiv M$ e $N_i \rightarrow_\beta N_{i+1}$, para todo $0 \leq i < n$.
Um caminho de redução infinito de M é uma sequência infinita de termos $N_0, N_1, \ldots$ tais que $N_0 \equiv M$ e $N_i \rightarrow_\beta N_{i+1}$, para todo $i \in \mathbb{N}$

Considerando esses dois tipos de caminhos de redução, vamos definir dois tipos de normalização

Definição 1.15 (Normalização Fraca e Forte)**.**

1. M é *fracamente normalizável* se existe um N na forma normal β tal que $M \twoheadrightarrow_\beta N$
2. M é *fortemente normalizável* se não existem caminhos de requção infinitos começando de M.

Todo termo M que é fortemente normalizável é fracamente normalizável.
Os termos Ω e Δ não são nem fortemente normalizáveis, nem fracamente normalizáveis.
É possível relacionar a normalização fraca com a forma normal β usando a intuição que, se M reduz para ambos N_1 e N_2, então existe um termo N_3 que exista no caminho de redução de ambos N_1 e N_2.

Teorema 1.1 (Teorema de Church-Rosser ou Teorema da Confluência)**.**
Suponha que para um λ-termo M, tanto $M \twoheadrightarrow_\beta N_1$ e $M \twoheadrightarrow_\beta N_2$. Então existe um λ-termo N_3 tal que $N_1 \twoheadrightarrow_\beta N_3$ e $N_2 \twoheadrightarrow_\beta N_3$

A prova desse teorema pode ser encontrada no Livro de Barendregt.

Uma consequência importante desse teorema é que o resultado do calculo feito em cima do termo não depende da ordem que esses cálculos são feitos. A escolha dos redexes não interfere no resultado final.

Corolário 1.1.
Suponha que $M =_\beta N$. Então existe um termo L tal que $M \twoheadrightarrow_\beta L$ e $N \twoheadrightarrow_\beta L$.

prova. Como $M =_\beta N$, então, pela definição, existe um $n \in \mathbb{N}$ tal que:

$$M \equiv M_0 \rightleftarrows_\beta M_1 \ldots M_{n-1} \rightleftarrows_\beta M_n \equiv N$$

. Onde $M_i \rightleftarrows_\beta M_{i+1}$ significa que ou $M_i \rightarrow_\beta M_{i+1}$ ou $M_{i+1} \rightarrow_\beta M_i$. Vamos provar por indução em n:

1. Quando $n = 0$: $M \equiv N$. Então sendo $L \equiv M$, $M \twoheadrightarrow_\beta L$ e $N \twoheadrightarrow_\beta L$ (por zero passsos)

2. Quando $n = k > 0$, entao existe M_{k-1}. Logo temos que $M \equiv M_0 \rightleftarrows_\beta M_1 \ldots M_{k-1} \rightleftarrows_\beta M_k \equiv N$. Por indução, existe um L' tal que $M_0 \twoheadrightarrow_\beta L'$ e $M_{k-1} \twoheadrightarrow_\beta L'$. Vamos dividir $M_{k-1} \rightleftarrows_\beta M_k$ em dois casos

   (a) Se $M_{k-1} \rightarrow_\beta M_k$, então como $M_{k-1} \rightarrow_\beta M_k$ e $M_{k-1} \twoheadrightarrow_\beta L'$, então, pelo Teorema de Church-Rosser, existe um L tal que $L' \twoheadrightarrow_\beta L$ e $M_k \twoheadrightarrow_\beta L$. Logo encontramos L.

   (b) Se $M_k \rightarrow_\beta M_{k-1}$, então como $M_0 \twoheadrightarrow_\beta L'$ e $M_k \twoheadrightarrow_\beta L'$, L' é o próprio L.

   $\square$.

Lemma 1.5.

1. Se M possui forma normal β N, então $M \twoheadrightarrow_\beta N$.

2. Um λ-termo tem no máximo uma forma normal β

Prova

1. Seja $M =_\beta N$, com N como formal normal β. Então, pelo corolário anterior, existe um L tal que $M \twoheadrightarrow_\beta L$ e $N \twoheadrightarrow_\beta L$. Como N é a forma normal, N não é mais redutível e $N \equiv L$. Então $M \twoheadrightarrow_\beta L \equiv N$, logo $M \twoheadrightarrow_\beta N$.

2. Suponha que M possui duas formas normais β N_1 e N_2. Então por (1), $M \twoheadrightarrow_\beta N_1$ e $M \twoheadrightarrow_\beta N_2$. Pelo teorema de Church-Rosser, existe um L tal que $N_1 \twoheadrightarrow_\beta L$ e $N_2 \twoheadrightarrow_\beta L$. Mas como N_1 e N_2 estão na forma normal, $L \equiv N_1$ e $L \equiv N_2$. Então pela transitividade da equivalência, $N_1 \equiv N_2$.

   $\square$.

1.1.7 Teorema do ponto fixo

No Cálculo λ, todo λ-termo L possui um *ponto fixo*, ou seja, existe um λ-termo M tal que $LM =_\beta M$. O termo Ponto Fixo vêm da análise funcional: seja f uma função, então f possui um ponto fixo a se $f(a) = a$.

Teorema 1.2. Para todo $L \in \Lambda$, existe um $M \in \Lambda$ tal que $LM =_\beta M$.

prova: Seja L um λ-termo e defina $M := (\lambda x.L(xx))(\lambda x.L(xx))$. M é um redex, logo:

$$\begin{aligned} M &\equiv (\lambda x.L(xx))(\lambda x.L(xx)) \\ &\to_\beta L((\lambda x.L(xx))(\lambda x.L(xx))) \\ &\equiv LM \end{aligned}$$

Logo $LM =_\beta M$. $\square$

Pela prova anterior, podemos perceber que M pode ser generalizado para todo λ-termo. Esse M será denominado de *Combinador de ponto fixo* e escrito na forma:

$$Y \equiv \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$$

1.1.8 Eta redução

A junção da definição 0.8 com as definições de β-redução gera uma teoria que será chamada aqui de λ_β. Para essa teoria, faltam alguns detalhes que podem, ou não, ser introduzidos a depender do que se precisa.

A η-redução é a segunda redução possível dentro do cálculo λ. Através dela é possível remover uma abstração que não faz nada para o termo interior. Sua definição é:

Definição 1.16 (η-redução)**.**

1. $(\lambda x.Mx) \to_\eta M$, onde $x \notin FV(M)$.

A junção da teoria λ com a η-redução será chamada aqui de $\lambda_{\beta\eta}$.

Uma outra adição possível à teoria λ é chamada de extencionalidade e definida da seguinte forma:

Definição 1.17 (extencionalidade **Ext**)**.** Dados os termos M e N, se $Mx = Nx$ para todo λ-termo x, com $x \notin FV(MN)$, então $M = N$.

Ext introduz no cálculo λ a noção presente na teoria dos conjuntos de igualdade entre funções. Na teoria dos conjuntos, duas funções $f : A \to B$ e $g : A \to B$ são iguais se, para todo $x \in A$, $f(x) = g(x)$.

A união da teoria λ com **Ext** é chamada de $\lambda + \textbf{Ext}$.

Teorema 1.3 (Teorema de Curry)**.** As teorias $\lambda_{\beta\eta}$ e $\lambda + \boldsymbol{Ext}$ são equivalentes.

Prova: Primeiro, é necessário mostrar que η é derivável de $\lambda + \boldsymbol{Ext}$. Seja a igualdade $(\lambda x.Mx)x = Mx$, por **Ext**, $\lambda x.Mx = M$.

Segundo, é necessário mostrar que dado $Mx = Nx$, é possível derivar $M = N$ em $\lambda_{\beta\eta}$. Para isso, seja $Mx = Nx$, realizando ξ-redução, tem-se que $\lambda x.Mx = \lambda x.Nx$. Fazendo η-redução dos dois lados, $M = N$. $\square$

Existe uma outra formulação da extencionalidade dentro do cálculo λ chamado de regra ω. É necessário um equivalente à **Ext** para restrições do cálculo λ que só possuem termos fechados, para isso, é desenvolvida a regra ω:

Definição 1.18 (Regra ω)**.** Dados os termos M e N, se $MQ = NQ$ para todo termo fechado Q, então $M = N$.

Da regra ω é possível deduzir **Ext**, mas não o oposto. A prova dessa dedução não será mostrada.

Posteriormente, será feito uma discussão de teorias dos tipos que aceitam **Ext** como um axioma no estilo de $\lambda + \boldsymbol{Ext}$ e outras que conseguem derivar a extencionalidade através de outras propriedades, como $\lambda_{\beta\eta}$.

1.1.9 Codificações dentro do Cálculo λ

O primeiro exemplo de transformação de funções em λ-termos, $f(x) = x^2$ para $\lambda x.x^2$, pode parecer correto, mas supõe mais que foi definido até então. Pois partindo somente da sintaxe e das transformações vistas nas seções anteriores, não foi definido coisas básicas como o que significa a exponenciação ou o número 2. Se o cálculo *lambda* é colocado como um possível substituto para a teoria das funções baseada na teoria dos conjuntos, então ele deve ser capaz de definir todas essas coisas de forma interna. Por isso, foram desenvolvidas as *codificações*, das quais a primeira e mais conhecida é a *Codificação de Church* (Church Encoding).

Primeiro, é necessário definir os números naturais e, para isso, é preciso de combinadores que traduzam os axiomas de Peano para os números naturais. Ou seja, precisamos definir o número 0 e a função sucessor $suc(x) = x + 1$. Para isso, diferente das outras definições indutivas vistas anteriormente, primeiro serão definidos os números e depois as operações.

Definição 1.19 (Numerais de Church)**.**

1. $zero := \lambda fx.x$
2. $um := \lambda fx.fx$
3. $dois := \lambda fx.f(fx)$
   $\ldots$
4. $n := \lambda fx.f^n x$

Onde $f^n x$ é $f(f(f \ldots x))$ n vezes.

As operações são descritas na forma:

Definição 1.20 (Operações aritméticas)**.**

1. $sum := \lambda m.\lambda n.\lambda fx.mf(nfx)$
2. $mult := \lambda m.\lambda n.\lambda fx.m(nf)x$
3. $suc := \lambda m.\lambda fx.f(mfx)$

Nessas definições os primeiros m e n são os números m e n, como por exemplo $m+n$, $m \times n$, $m+1$, etc.

Exemplos:

1. Prova que $sum\ one\ one \twoheadrightarrow_\beta two$ na codificação:

$$\begin{aligned} sum\ one\ one &\equiv (\lambda m.\lambda n.\lambda fx.mf(nfx))\ one\ one \\ &\twoheadrightarrow_\beta (\lambda fx.onef(onefx)) \\ &\twoheadrightarrow_\beta (\lambda fx.(\lambda gx.gx)f((\lambda gx.gx)fx)) \\ &\twoheadrightarrow_\beta (\lambda fx.(\lambda x.fx)(fx)) \\ &\twoheadrightarrow_\beta (\lambda fx.f(fx)) \\ &\equiv two \end{aligned}$$

2. Prova que $mult\ two\ two \twoheadrightarrow_\beta four$ na codificação:

$$\begin{aligned} mult\ two\ two &\equiv (\lambda m.\lambda n.\lambda fx.m(nf)x)\ two\ two \\ &\twoheadrightarrow_\beta (\lambda fx.\ two(\ two\ f)x) \\ &\twoheadrightarrow_\beta (\lambda fx.(\lambda gy.g(gy))(\ two\ f)x) \end{aligned}$$

Uma vez definida a multiplicação e a soma, é possível definir outras operações como o fatorial e a exponenciação. Isso fica como exercício para o leitor.

Tendo definido operações relacionadas aos números naturais, pode-se perguntar se é possível construir algo lógico dentro do cálculo λ não-tipado. Para isso, é necessário definir a noção de ”verdadeiro” e ”falso”, na forma:

Definição 1.21 (Booleanos)**.**

1. $true := \lambda xy.x$
2. $false := \lambda xy.y$
3. $not := \lambda z.z\ false\ true$
4. *'if x then u else v'* $:= \lambda x.xuv$

Exemplos:

1. Prova que $not(not\ p) \equiv p$ na codificação:

$$\begin{aligned} not(not\ p) &\equiv not((\lambda z.z\ false\ true\ )p) \\ &\twoheadrightarrow_\beta not(p\ false\ true\ ) \\ &\twoheadrightarrow_\beta not(p(\lambda xy.y)(\lambda xy.x)) \\ &\twoheadrightarrow_\beta (\lambda z.z\ false\ true\ )(p(\lambda xy.y)(\lambda xy.x)) \\ &\twoheadrightarrow_\beta (p(\lambda xy.y)(\lambda xy.x))\ false\ true \end{aligned}$$

Se $p \twoheadrightarrow_\beta \mathit{true}$,

$$\begin{aligned}
not(not\ \mathit{true}\ ) &\twoheadrightarrow_\beta ((\lambda xy.x)(\lambda xy.y)(\lambda xy.x))\ \mathit{false}\ \mathit{true} \\
&\twoheadrightarrow_\beta ((\lambda xy.y))\ \mathit{false}\ \mathit{true} \\
&\twoheadrightarrow_\beta \mathit{true}
\end{aligned}$$

Se $p \twoheadrightarrow_\beta \mathit{false}$,

$$\begin{aligned}
not(not\ \mathit{false}\ ) &\twoheadrightarrow_\beta ((\lambda xy.y)(\lambda xy.y)(\lambda xy.x))\ \mathit{false}\ \mathit{true} \\
&\twoheadrightarrow_\beta ((\lambda xy.x))\ \mathit{false}\ \mathit{true} \\
&\twoheadrightarrow_\beta \mathit{false}
\end{aligned}$$

1.2 Modelos

1.3 Codificação em Coq

1.4 A História do cálculo λ

...

2 Teoria dos Tipos Simples

O cálculo λ não-tipado possui alguns entraves ao tentar traduzir as funções matemáticas para seus termos. Um desses entraves é o fato que as funções matemáticas são mapeamentos entre dois conjuntos. Ou seja, essas funções possuem em sua definição os valores que vão esperar e os possíveis valores que vão retornar. A função soma $+ : \mathbb{N} \to \mathbb{N}$ não pode aceitar os valores $true$ ou $false$. Porém, nas codificações do cálculo λ descrito até então (Sem contar com os modelos), isso é possível. Por exemplo, é possível perceber que $false$ e 0 são definidos pelo mesmo termo $\lambda xy.y$ (a definição de 0 é α-equivalente a essa), o que pode gerar confusão em sua aplicação.

Outro problema do Cálculo λ não-tipado é o fato de poder existir recursões infinitas através de termos como Ω e Δ. A tipagem dos termos faz com que esse tipo de fenômeno não ocorra. O que retira a Turing-completude, mas facilita outras coisas.

Para fazer essa descrição ser mais detalhada e evitar esse tipo de erro, Church introduziu tipos.

2.1 Cálculo λ simplesmente tipado (STλC)

2.1.1 Tipos simples

Uma forma simples de começar a tipagem dos λ-termos é considerando uma coleção de variáveis de tipos e uma forma de produzir mais tipos através dessa coleção, chamado de *tipo funcional*

Seja $\mathbb{V}$ a coleção infinita de variáveis de tipos $\mathbb{V} = \{\alpha, \beta, \gamma, \dots\}$, então:

Definição 2.1 (A coleção de todos os tipos simples)**.** A coleção dos tipos simples $\mathbb{T}$ é definida por:

1. (Variável de tipos) Se $\alpha \in \mathbb{V}$, então $\alpha \in \mathbb{T}$

2. (Tipo funcional) Se $\sigma, \tau \in \mathbb{T}$, então $(\sigma \to \tau) \in \mathbb{T}$.

Na BNF, $\mathbb{T} = \mathbb{V}|\mathbb{T} \to \mathbb{T}$

Os parenteses no tipo funcional são associativos à direita, ou seja o tipo $\alpha_1 \to \alpha_2 \to \alpha_3 \to \alpha_4$ é $(\alpha_1 \to (\alpha_2 \to (\alpha_3 \to \alpha_4)))$

Tipos simples arbitrários serão escritos com letras gregas minúsculas (Com excessão do λ) como $\sigma, \tau, \dots$, mas também podem ser escrito como letras latinas maiúsculas $A, B, \dots$ na literatura.

As variáveis de tipos são representações abstratas de tipos básicos como os números naturais $\mathbb{N}$ ou a coleção de todas as listas $\mathbb{L}$. Esses tipos serão explorados mais à frente. Já os tipos funcionais representam funções na matemática como por exemplo $\mathbb{N} \to \mathbb{N}$, o conjunto de funções que leva dos naturais para os naturais, ou $(\mathbb{N} \to \mathbb{Z}) \to \mathbb{Z} \to \mathbb{N}$, o conjunto de funções que recebem como entrada uma função que leva dos naturais aos inteiros e um inteiro e retorna um natural.

A sentença "O termo M possui tipo σ" é escrita na forma $M : \sigma$. Todo termo possui um tipo único, logo se x é um termo e $x : \sigma$ e $x : \tau$, então $\sigma \equiv \tau$.

Como os tipos foram introduzidos para lidar com o cálculo λ, eles devem ter regras para lidar com as operações de aplicação e abstração.

1. (*Aplicação*): No cálculo λ, sejam M e N termos, podemos fazer uma aplicação entre eles no estilo MN. Para entender como entram os tipos, é possível recordar de onde surge a intuição para a aplicação. Seja $f : \mathbb{N} \to \mathbb{N}$ a função $f(x) = x^2$, então, a aplicação de 3 em f é $f(3) = 3^2$. Nesse exemplo, omite-se o fato que para aplicar 3 a f, 3 tem que estar no domínio de f, ou seja, $3 \in \mathbb{N}$. No caso do cálculo λ, para aplicar N em M, M deve ter um tipo funcional, na forma $M : \sigma \to \tau$, e N deve ter como tipo o primeiro tipo que aparece em M, ou seja $N : \sigma$.

2. (*Abstração*): No cálculo λ, seja M um termo, podemos escrever um termo $\lambda x.M$. A abstração "constroi" a função. Para a tipagem, seja $M : \tau$ e $x : \sigma$, então $\lambda x : \sigma.M : \sigma \to \tau$. É possível omitir o tipo da variável, escrevendo no estilo: $\lambda x.M : \sigma \to \tau$.

Alguns exemplos:

1. Seja x do tipo σ, a função identidade é escrita na forma $\lambda x.x : \sigma \to \sigma$.

2. O combinador $\mathbf{B} \equiv \lambda xyz.x(yz)$ é tipado na forma $\mathbf{B} : (\sigma \to \tau) \to (\rho \to \sigma) \to \rho \to \tau$.

3. O combinador $\Delta \equiv \lambda x.xxx$ não possui tipagem. Isso ocorre pois, na aplicação xx, x precisa ter como tipo $\sigma \to \tau$ e σ, mas como x só pode ter um tipo, então $\sigma \to \tau \equiv \sigma$. O que não é possível em $\mathbb{T}$. Logo Δ (e Ω por motivos similares), não faz parte da teoria dos tipos simples.

O último exemplo mostra que o teorema do ponto fixo não ocorre para todos os termos na teoria dos tipos simples e que não existe recursão infinita, fazendo com que a teoria dos tipos simples deixe de ser turing-completa.

2.1.2 Abordagens para a tipagem

Existem duas formas de tipar um λ-termo:

1. (*Tipagem à la Church / Tipagem explícita / Tipagem extrínseca / Tipagem ontológica*) Pode-se prescrever um tipo único à cada variável quando ela for introduzida. Nesse estilo de tipagem, só termos que podem ser bem formados são aceitos.

2. (*Tipagem à la Curry / Tipagem implícita / Tipagem intrínseca / Tipagem semântica*) Pode-se não definir o tipo do termo na sua introdução, mas deixá-lo aberto. Os tipos são buscados para o termo, por tentativa e erro.

Part II

Construções paralelas ao cubo

Part III

Semântica Categorial das teorias do cubo lambda

Part IV

Teorias Homotópicas de Tipos

Part V

Semântica Categorial das teoria homotópicas de tipos

Part VI

Lógica