Tipos, Categorias e Lógicas

Notas em Teoria dos Tipos, Teoria das Categorias e Lógica.

edição 0.0

Autores: Mateus Galdino

2024

Contents

1 Prefácio

A Teoria dos Tipos é uma área nova crescente de ampla interseção com outras áreas também novas ou áreas já consolidadas. Essa interseção dá frutos práticos interessantes para as áreas envolvidas. Por exemplo: no campo da computação, a maioria das linguagens de programação possui tipagem e, sem entrar no mérito das diferentes abordagens de tipagem para cada linguagem, é possível descrever a tipagem delas utilizando uma teoria dos tipos adequada. Já no campo da matemática, é possível perceber que as teorias dos tipos descritas aqui possuem modelos conhecidos na teoria das categorias que permitem formulações de objetos matemáticos já conhecidos (como Grupos, Espaços Topológicos, etc). No meio desses dois exemplos, existe a tentativa de aproximar a computação dos fundamentos da matemática a partir de assistentes de prova.

Essas notas foram escritas por dois fins. O primeiro é expor em lingua portuguesa a vasta gama de conceitos explorados na teoria dos tipos, deixando essa área da matemática e da computação o mais acessível possível para iniciantes vindos de diversos ambientes. Em lingua inglesa, já existem várias fontes possíveis para adentrar essa teoria, que serão referenciadas a partir dessas notas, mas em português as poucas fontes que existem estão em dissertações acadêmicas pouco preocupadas com a difusão das ideias para fora de seus nichos. O segundo fim é, em certa medida, conseguir, através dessa exposição, que mais e mais pessoas tenham interesse pelo assunto e começem a pesquisar, visto que nos centros e depertamentos brasileiros, sejan de matemática ou de computação, essa área recebe pouca a nenhuma atenção, já que os professores especializados nesses assuntos já não estão comprometidos a ensinar os alunos de graduação essa área. Dessa forma, essas notas também se colocam como um desafio: ensinar o máximo de teoria dos tipos possível para alunos de graduação.

Essas notas então podem ser utilizadas sem dúvida por professores que queiram se aventurar no ensino da teoria dos tipos.

A primeira parte tenta desenvolver as diversas teorias de tipos denominadas de λ-cubo. Essa primeira parte usa como base (o primeiro subcapítulo de cada capítulo) o livro *Type Theory and Formal Proof* de Nederpelt e Geuvers, mas adentra tópicos mais profundos em cada teoria.

Já a segunda parte desenvolve outras construções paralelas ao λ-cubo, derivadas do λ-calculo não-tipado, como o $\lambda\mu$-cálculo e o κ-cálculo. Cada cálculo é retirado de artigos diferentes e compilados no mesmo lugar.

A parte três desenvolve a teoria das categorias necessária para a semântica de cada uma das teorias dos tipos desenvolvidas nas partes I e II, desenvolvendo o conceito de categorias até a teoria dos Topos e construções paralelas. Essa parte é bastante influenciada pelo livro *Introduction to Higher Order Categorical Logic* de Lambek e Scott.

A parte IV entra nas diversas teorias homotópicas de tipos, desde sua precursora, a *Teoria dos Tipos de Martin-Löf*, e a original do livro *Homotopy Type Theory* até construções mais recentes. A maioria dessas teorias está espalhada em diversos artigos, então o trabalho aqui se torna compilá-las em um único lugar de forma a criar um fio condutor entre elas.

A parte V dessenvolve a semântica categorial das HoTT utilizando conceitos da teoria das ∞-categorias, teoria das homotopias (em suas versões simpliciais e cúbicas) e conceitos já trabalhados na parte III.

A parte VI é a parte final e serve como apêndice para colocar definições voltadas para a lógica e a teoria da prova, com a exposição do cálculo de sequêntes, da dedução natural e de outras áreas correlatas. Essa parte trás inspiração no livro *Logic and Structure* do Dirk van Dalen e *An Introduction to Proof Theory* de Galvan et al.

Links importantes:

- Caso o leitor encontre algum erro ou problema nas notas, por favor avisar em `https://github.com/MateusGaldinoLG/notasTT/issues`.
- Caso o leitor queira contribuir no geral com adição ou escrita de temas: `https://github.com/MateusGaldinoLG/notasTT`
- Para verificar o progresso da escrita do livro: `https://github.com/MateusGaldinoLG/notasTT/blob/main/README.md`

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 $x \in \Lambda$
- $M \in \Lambda \rightarrow (\lambda x.M) \in \Lambda$
- $M, N \in \Lambda \rightarrow MN \in \Lambda$

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 $0 \leq i < n$:

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

Ou Seja:

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

Lemma 1.2.

1. $\twoheadrightarrow_\beta$ é uma extensão de $\rightarrow_\beta$, ou seja: se $M \rightarrow_\beta N$, 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 $M \equiv M_0 \rightarrow_\beta M_1 \equiv N$, que é a mesma coisa que $M \rightarrow_\beta N$
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 $M_i \rightarrow_\beta M_{i+1}$ ou $M_{i+1} \rightarrow_\beta M_i$

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 $(\lambda x.(\lambda y.yx)z)v \to_\beta zv$ (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 $\Delta\Delta \to_\beta \Delta\Delta\Delta \to_\beta \Delta\Delta\Delta\Delta \to_\beta \ldots$. 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 $N_i \to_\beta N_{i+1}$, 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 $N_i \to_\beta N_{i+1}$, 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:

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

. 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 $M \equiv M_0 \rightleftarrows_\beta M_1 \dots M_{k-1} \rightleftarrows_\beta M_k \equiv N$. 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.

   □.

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, $N_1 \equiv N_2$.

   □.

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 $\lambda +$ **Ext**.

Teorema 1.3 (Teorema de Curry)**.** As teorias $\lambda_{\beta\eta}$ e $\lambda +$ ***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, $M = N$. □

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

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:

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

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

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

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. $true := \lambda x y.x$
2. $false := \lambda x y.y$
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:

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

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

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

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

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

1.2 Modelos

1.3 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 intrínseca / Tipagem ontológica*) Nesse estilo de tipagem, só termos que possuem tipagem que satisfaz a construção de tipos interna à teoria são aceitos. Cada termo possui um tipo único.

2. (*Tipagem à la Curry / Tipagem implícita / Tipagem extrínseca / Tipagem semântica*) Nesse estilo de tipagem, os termos são os mesmos do cálculo λ não tipado e 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.

Exemplos

1. (Tipagem intrínseca): Seja x do tipo $\alpha \to \alpha$ e y do tipo $(\alpha \to \alpha) \to \beta$, então yx possui o tipo β. Se z possuit tipo β e u possuir tipo γ, então $\lambda zu.z$ tem tipo $\beta \to \gamma \to \beta$ e a aplicação $(\lambda zu.z)(yx)$ é permitida pois o tipo β de yx equivale ao tipo β que $\lambda zu.z$ recebe.

2. (Tipagem extrínseca): Nessa tipagem, começa-se com o termo $M \equiv (\lambda zu.z)(yx)$ e tenta-se adivinhar qual seu tipo e o tipo de suas variáveis. É possível notar que $(\lambda zu.z)(yx)$ é uma aplicação, então $(\lambda zu.z)$ precisa ter um tipo $A \to B$, yx precisa ter um tipo A e M terá um tipo B. Mas se $\lambda zu.z$ possui o tipo $A \to B$, então $\lambda u.z$ possui o tipo B e, como o termo é uma abstração, B precisa ser um tipo funcional, ou seja $B \equiv C \to D$. Logo $u : C$ e $z : D$. Já no caso de $yx : A$, y precisa ter um tipo funcional para ser aplicado a x, logo sendo $x : E$, $y : E \to F$. Logo temos que $x : E, y : E \to A, z : A, u : C$. Só é necessário então substituir A, C, E com tipos variáveis como α, β, γ: $x : \alpha, y : \alpha \to \beta, z : \beta, u : \gamma$.

No caso do exemplo 2, é possível escrever $x : \alpha, y : \alpha \to \beta, z : \beta, u : \gamma \vdash (\lambda z : \beta.\lambda u : \gamma.z)(yx) : \gamma \to \beta$. A lista à esquerda da $\vdash$ (lê-se catraca) é chamada de *contexto*.

2.1.3 Regras de derivação e Cálculo de sequêntes

É necessário, na tipagem intrínseca, definir a coleção de todos os λ-termos tipados:

Definição 2.2 (λ-termos pré-tipados)**.** A coleção $\Lambda_{\mathbb{T}}$ de λ-termos pré-tipados é definida pela BNF:
$\Lambda_{\mathbb{T}} = V | (\Lambda_{\mathbb{T}} \Lambda_{\mathbb{T}}) | (\lambda V : \mathbb{T}.\Lambda_{\mathbb{T}})$

Para expressar as tipagens dos λ-termos, é necessário desenvolver um conjunto de definições que ainda não foram mostradas:

Definição 2.3.

1. Uma *sentença* é $M : \sigma$, onde $M \in \Lambda_{\mathbb{T}}$ e $\sigma \in \mathbb{T}$. Nessa sentença, M é chamado de *sujeito* e σ de *tipo*

2. Uma *declaração* é uma sentença com uma *variável* como sujeito

3. Um *Contexto* é uma lista, possivelmente nula, de declarações com diferentes sujeitos

4. Um *Juizo* possui a forma $\Gamma \vdash M : \sigma$, onde Γ é o contexto e $M : \sigma$ é uma sentença.

Para estudar a tipagem, será utilizado um sistema de derivações trazido da lógica chamado de *Cálculo de sequêntes*. O cálculo de sequêntes dá a possibilidade de gerar juizos de forma formal utilizando árvores de derivação no estilo:

$$\frac{\text{premissa 1} \qquad \text{premissa 2} \qquad \ldots \qquad \text{premissa } n}{\text{Conclusão}}$$

Acima da linha horizontal estão as premissas, que são cada uma um juizo, e abaixo da linha horizontal está a conclusão, que é em si um juizo também. A linha marca uma regra de derivação específica da teoria que se está trabalhando.

Definição 2.4 (Regras de derivação para o STλC)**.**

- (*var*) $\Gamma \vdash x : \sigma$, dado que $x : \sigma \in \Gamma$.
- (*appl*)

$$\frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\ \text{appl}$$

- (*abst*)

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma . M : \sigma \to \tau}\ \text{abst}$$

A regra *(var)* não possui premissas e possui como conclusão o fato que dado um contexto Γ, se existe uma declaração em Γ, essa declaração é derivável através de Γ. Essa primeira regra é tratada como axioma em (Hindley, 1997), pois, assim como todo axioma, ela é derivável sem precisar de premissas. Na construção da árvore de dedução, essa regra está no topo como uma "raiz".

A regra *(appl)* é equivalente no cálculo ao que foi feito antes. Essa regra também é chamada na literatura de $\to -elim$ ou $\to E$

A regra *(abs)* é equivalente no cálculo à abstração e pode ser chamada na literatura de $\to -intro$ ou $\to I$.

Exemplo:

$$\dfrac{\dfrac{\dfrac{(1)\ y : \alpha \to \beta, z : \alpha \vdash y : \alpha \to \beta \qquad (2)\ y : \alpha \to \beta, z : \alpha \vdash z : \beta}{(3)\ y : \alpha \to \beta, z : \alpha \vdash yz : \beta}\ \text{appl}}{(4)\ y : \alpha \to \beta \vdash \lambda z : \alpha . yz : \alpha \to \beta}\ \text{abs}}{(5)\ \emptyset \vdash \lambda y : \alpha \to \beta . \lambda z : \alpha . yz : (\alpha \to \beta) \to \alpha \to \beta}\ \text{abs}$$

Dada a derivação já montada, sua leitura pode ser feita de baixo para cima, feito levando em conta as premissas mais fundamentais até a conclusão final, de forma a adicionar informação aos juízos a cada passo, ou de cima para baixo, feito para entender qual caminho leva até o objetivo final.

1. Os passos (1) e (2) usam a regra (*var*)
2. O passo (3) usa a regra (*app*) usando (1) e (2) como premissas
3. O passo (4) usa a regra ((abs)) com (3) como premissa

4. O passo (5) usa a regra ((abs)) com (4) como premissa

As regras de derivação podem ser entendidas em outros contextos:
Matemática: Seja $A \to B$ o conjunto de todas as funções de A para B, então as regras se tornam:

1. (*aplicação funcional*)

$$\frac{\text{se } f \text{ é um membro de } A \to B \qquad \text{e se } c \in A}{\text{então } f(c) \in B}$$

2. (*abstração funcional*)

$$\frac{\text{Se para } x \in A \text{ segue-se que } f(x) \in B}{\text{então } f \text{ é membro de } A \to B}$$

Lógica: Seja $A \Rightarrow B$ "A implica em B", então pode-se ler $A \to B$ como $A \Rightarrow B$. As regras se tornam:

1. ($\Rightarrow -elim$)

$$\frac{A \to B \qquad A}{B}$$

2. ($\Rightarrow -intro$)

$$\begin{array}{c} A \\ \vdots \\ \hline B \end{array}$$

A regra de eliminação é denominada de *Modus Ponens*. Ambas as regras como estão escritas aí são parte das regras definidas na *Dedução Natural*, um cálculo análogo ao cálculo de sequêntes (Toda árvore definida na dedução natural possui um equivalente no cálculo de sequêntes). Esse estilo de dedução natural é chamado de *Dedução natural no estilo de Gentzen*, para diferenciá-lo da *Dedução natural no estilo de Fitch* que é escrito como:

Definição 2.5 ($\lambda_\to$-termos legais)**.** Um termo M pré-tipado em $\lambda_\to$ é chamado *legal* se existe um contexto Γ e um tipo ρ tal que $\Gamma \vdash M : \rho$.

2.1.4 Problemas resolvidos no STLC

No geral, existem três tipos de problemas relacionados a julgamentos na teoria dos tipos:

1. *Bem-tipagem* (*Well-typedness*) ou *Tipabilidade*: esse problema surge da questão

$$? \vdash \text{termo} :?$$

Ou seja, saber se um termo é legal e, se não é, mostrar onde sua contrução falha.
(1a) *Atribuição de tipos*, que surge da questão:

$$\text{contexto} \vdash \text{termo} :?$$

. Ou seja, dado um contexto e um termo, derive seu tipo.

2. *Checagem de tipos*, que surge da questão

$$\text{contexto} \vdash^{?} \text{termo} : \text{tipo}$$

. Ou seja, se é realmente verdadeiro que o termo possui o tipo no determinado contexto.

3. *Encontrar o termo*, que surge da questão:

$$\text{contexto} \vdash ? : \text{tipo}$$

. Um tipo particular desse problema é quando o contexto é vazio, ou seja

$$\emptyset \vdash ? : \text{tipo}$$

.

Todos esses problemas são *decidíveis* em $\lambda_{\rightarrow}$. Ou seja, para cada um deles existe um *algoritmo* (um conjunto de passos) que produz a resposta. Em outros sistemas, encontar um termo se torna *indecidível*.

2.1.5 Bem-tipagem em $\lambda_{\rightarrow}$

Para exemplificar os passos necessários para resolver a bem-tipagem em $\lambda_{\rightarrow}$, será utilizado o exemplo descrito em 1.1.3, dessa vez passo a passo.

O objetivo é mostrar que o termo $M \equiv \lambda y : \alpha \rightarrow \beta.\lambda z : \alpha.yz$ é um termo legal. Logo, precisamos encontrar um contexto Γ e um tipo ρ tal que $\Gamma \vdash M : \rho$.

Primeiro, como não existem variáveis livres em M, o contexto inicial pode ser considerado vazio: $\Gamma = \emptyset$.

Inicialmente, o primeiro passo é descobrir qual a premissa, ou premissas, que gera o termo e a regra de dedução:

$$\frac{?}{\emptyset \vdash \lambda y : \alpha \rightarrow \beta.\lambda z : \alpha.yz : \ldots} ?$$

Como a primeira parte do termo é um λy, a única regra possível inicialmente é a abstração:

$$\dfrac{\dfrac{\dfrac{?}{y : \alpha \to \beta \vdash \lambda z : \alpha . yz : \ldots}\,?}{\emptyset \vdash \lambda y : \alpha \to \beta . \lambda z : \alpha . yz : \ldots}\,\text{abs}}$$

Novamente, a única regra possível é a abstração:

$$\dfrac{\dfrac{\dfrac{\dfrac{?}{y : \alpha \to \beta, z : \alpha \vdash yz : \ldots}\,?}{y : \alpha \to \beta \vdash \lambda z : \alpha . yz : \ldots}\,\text{abs}}{\emptyset \vdash \lambda y : \alpha \to \beta . \lambda z : \alpha . yz : \ldots}\,\text{abs}}$$

Sobrou do lado direito da catraca o termo yz que, vendo o contexto, é a aplicação de outros dois termos, logo a única regra possível é a aplicação:

$$\dfrac{\dfrac{\dfrac{\dfrac{y : \alpha \to \beta, z : \alpha \vdash y : \alpha \to \beta \qquad y : \alpha \to \beta, z : \alpha \vdash z : \beta}{y : \alpha \to \beta, z : \alpha \vdash yz : \ldots}\,\text{appl}}{y : \alpha \to \beta \vdash \lambda z : \alpha . yz : \ldots}\,\text{abs}}{\emptyset \vdash \lambda y : \alpha \to \beta . \lambda z : \alpha . yz : \ldots}\,\text{abs}}$$

Como as premissas mais superiores são geradas de (var), não há mais nenhum passo de premissas e a tipagem pode ser realizada de cima para baixo.

$$\dfrac{\dfrac{\dfrac{\dfrac{y : \alpha \to \beta, z : \alpha \vdash y : \alpha \to \beta \qquad y : \alpha \to \beta, z : \alpha \vdash z : \beta}{y : \alpha \to \beta, z : \alpha \vdash yz : \beta}\,\text{appl}}{y : \alpha \to \beta \vdash \lambda z : \alpha . yz : \ldots}\,\text{abs}}{\emptyset \vdash \lambda y : \alpha \to \beta . \lambda z : \alpha . yz : \ldots}\,\text{abs}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{y : \alpha \to \beta, z : \alpha \vdash y : \alpha \to \beta \qquad y : \alpha \to \beta, z : \alpha \vdash z : \beta}{y : \alpha \to \beta, z : \alpha \vdash yz : \beta}\,\text{appl}}{y : \alpha \to \beta \vdash \lambda z : \alpha . yz : \alpha \to \beta}\,\text{abs}}{\emptyset \vdash \lambda y : \alpha \to \beta . \lambda z : \alpha . yz : \ldots}\,\text{abs}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{y : \alpha \to \beta, z : \alpha \vdash y : \alpha \to \beta \qquad y : \alpha \to \beta, z : \alpha \vdash z : \beta}{y : \alpha \to \beta, z : \alpha \vdash yz : \beta}\,\text{appl}}{y : \alpha \to \beta \vdash \lambda z : \alpha . yz : \alpha \to \beta}\,\text{abs}}{\emptyset \vdash \lambda y : \alpha \to \beta . \lambda z : \alpha . yz : (\alpha \to \beta) \to \alpha \to \beta}\,\text{abs}}$$

Se existisse algum problema no caso de encontrar variáveis com tipagem incongruente nas últimas premissas ou não ter mais nenhum passo, então o termo não seria bem-tipado.

2.1.6 Checagem de tipos em $\lambda_{\rightarrow}$

Seja o juizo

$$x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash (\lambda z : \beta.\lambda u : \gamma.z)(yx) : \gamma \rightarrow \beta$$

é necessário construir uma árvore de inferências que demonstre que $\gamma \rightarrow \beta$ é o tipo correto do termo do lado direito.

$$\frac{?}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash^{?} (\lambda z : \beta.\lambda u : \gamma.z)(yx) : \gamma \rightarrow \beta}\ ?$$

Usando a regra da aplicação, tem-se:

$$\frac{\dfrac{?}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash \lambda z : \beta.\lambda u : \gamma.z : \ ?}\ ? \qquad \dfrac{?}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash yx : \ ?}\ ?}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash^{?} (\lambda z : \beta.\lambda u : \gamma.z)(yx) : \gamma \rightarrow \beta}\ \text{appl}$$

O lado direto se segue da regra da aplicação:

$$\frac{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash x : \alpha \rightarrow \alpha \qquad x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash y : (\alpha \rightarrow \alpha) \rightarrow \beta}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash yx : \ ?}\ \text{appl}$$

Usando essa subárvore, pode-se ver que yx possui o tipo $yx : \beta$.
O lado esquerdo se segue da abstração:

$$\frac{\dfrac{?}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta, z : \beta \vdash \lambda u : \gamma.z : \ ?}\ ?}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash \lambda z : \beta.\lambda u : \gamma.z : \ ?}\ \text{abst}$$

abstraindo novamente:

$$\frac{\dfrac{\dfrac{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta, z : \beta, u : \gamma \vdash z : \beta}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta, z : \beta \vdash \lambda u : \gamma.z : \ ?}\ \text{abst}}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash \lambda z : \beta.\lambda u : \gamma.z : \ ?}\ \text{abst}$$

Agora, é possível "descer" novamente "coletando" os tipos que foram deixados para trás:

$$\frac{\dfrac{\dfrac{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta, z : \beta, u : \gamma \vdash z : \beta}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta, z : \beta \vdash \lambda u : \gamma.z : \gamma \rightarrow \beta}\ \text{abst}}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash \lambda z : \beta.\lambda u : \gamma.z : \ ?}\ \text{abst}$$

e

$$\frac{\dfrac{\dfrac{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta, z : \beta, u : \gamma \vdash z : \beta}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta, z : \beta \vdash \lambda u : \gamma.z : \gamma \rightarrow \beta}\ \text{abst}}{x : \alpha \rightarrow \alpha, y : (\alpha \rightarrow \alpha) \rightarrow \beta \vdash \lambda z : \beta.\lambda u : \gamma.z : \beta \rightarrow \gamma \rightarrow \beta}\ \text{abst}$$

Seja $\Gamma \equiv x : \alpha \to \alpha, y : (\alpha \to \alpha) \to \beta$, a árvore completa fica:

$$\dfrac{\dfrac{\dfrac{\Gamma, z : \beta, u : \gamma \vdash z : \beta}{\Gamma, z : \beta \vdash \lambda u : \gamma . z : \gamma \to \beta}\,\text{abst}}{\Gamma \vdash \lambda z : \beta . \lambda u : \gamma . z : \beta \to \gamma \to \beta}\,\text{abst} \qquad \dfrac{\Gamma \vdash x : \alpha \to \alpha \qquad \Gamma \vdash y : (\alpha \to \alpha) \to \beta}{\Gamma \vdash yx : \beta}\,\text{appl}}{\Gamma \vdash (\lambda z : \beta . \lambda u : \gamma . z)(yx) : \gamma \to \beta}\,\text{appl}$$

Dessa forma, é possível perceber que sim, a aplicação de $\lambda z : \beta . \lambda u : \gamma . z : \beta \to \gamma \to \beta$ com $yx : \beta$ possui o tipo $\gamma \to \beta$.

2.1.7 Encontrar termos em $\lambda_{\to}$

Seja o tipo $A \to B \to A$. A pergunta que fica é: é possível encontrar um termo para esse tipo? Essa pergunta é, vista do ponto da lógica, a mesma coisa que "é possível computar uma prova para essa proposição?" (Isso será visto mais adiante). Isso é a mesma coisa que: $?\ : A \to B \to A$. Pelas regras de inferência:

$$\dfrac{\dfrac{}{?}\,?}{?\ \vdash\ ?\ : A \to B \to A}\,?$$

Supondo um termo $x : A$, pode-se escrever a árvore como:

$$\dfrac{\dfrac{\dfrac{}{?}\,?}{x : A \vdash\ ?\ : B \to A}\,?}{x : A \vdash\ ?\ : A \to B \to A}\,\text{abst}$$

E supondo um outro termo $y : B$, pode-se escrever como:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{?}\,?}{x : A, y : B \vdash\ ?\ : A}\,?}{x : A, y : B \vdash\ ?\ : B \to A}\,\text{abst}}{x : A, y : B \vdash\ ?\ : A \to B \to A}\,\text{abst}$$

Como já existe um termo de tipo A, pode-se substituir o termo desconhecido por x:

$$\dfrac{\dfrac{\dfrac{x : A, y : B \vdash x : A}{x : A, y : B \vdash\ ?\ : B \to A}\,\text{abst}}{x : A, y : B \vdash\ ?\ : A \to B \to A}\,\text{abst}$$

Usando a regra da abstração:

$$\dfrac{\dfrac{\dfrac{x : A, y : B \vdash x : A}{x : A, y : B \vdash \lambda y . x : B \to A}\,\text{abst}}{x : A, y : B \vdash\ ?\ : A \to B \to A}\,\text{abst}$$

Novamente:

$$\dfrac{\dfrac{\dfrac{x : A, y : B \vdash x : A}{x : A, y : B \vdash \lambda y . x : B \to A}\,\text{abst}}{x : A, y : B \vdash \lambda x y . x : A \to B \to A}\,\text{abst}$$

2.1.8 Propriedades gerais do STλC

Ficaram faltando nas definições anteriores a explicação de algumas propriedades gerais da sintaxe do STλC.

Algumas propriedades sobre os contextos:

Definição 2.6 ((Domínio, subcontexto, permutação, projeção))**.**

1. Se $\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n$, então o *domínio* de Γ ou $dom(\Gamma)$ é a lista $(x_1, \ldots, x_n)$.
2. Um contexto Γ' é um *subcontexto* do contexto Γ, ou $\Gamma' \subseteq \Gamma$ se todas as declarações que ocorrem em Γ' também ocorrem em Γ na mesma ordem.
3. Um contexto Γ' é uma *permutação* do contexto Γ, ou $\Gamma' \subseteq \Gamma$ se todas as declarações que ocorrem em Γ' também ocorrem em Γ e vice-versa
4. Se Γ é um contexto e Φ o conjunto de variáveis, então a *projeção* de Γ em Φ, ou $\Gamma \restriction \Phi$, é o subcontexto Γ' de Γ com $dom(\Gamma') = dom(\Gamma) \cap \Phi$

Em uma lista, a ordem dos elementos importa.

Exemplo: Seja $\Gamma \equiv y : \sigma, x_1 : \rho_1, x_2 : \rho_2, z : \tau, x_3 : \rho_3$, então:

1. $dom(\emptyset) = ()$, onde $\emptyset$ é chamado de lista vazia;
2. $dom(\Gamma) = (y, x_1, x_2, z, x_3)$
3. $\emptyset \subseteq (x_1 : \rho_1, z : \tau) \subseteq \Gamma$
4. $\Gamma \restriction \{z, u, x_1\} = x_1 : \rho_1, z : \tau$

Uma propriedade importante de $\lambda_\rightarrow$ é a seguinte:

Lemma 2.1. (Lemma das variáveis livres)
Se $\Gamma \vdash L : \sigma$, então $FV(L) \subseteq dom(\Gamma)$.

Como consequência desse lemma, seja x uma variável livre que ocorre em L, então x possui um tipo, o qual é declarado no contexto Γ. Em um juizo, não é possível ocorrer confusão sobre o tipo de qualquer variável, pois todas as variáveis ligadas possuem seu tipo, antes da ligação λ.

Para provar esse lemma, é necessário usar uma técnica de prova chamada de *indução estrutural*. Essa indução ocorre da seguinte forma:

Seja $\mathcal{P}$ a propriedade geral que se quer provar para uma expressão arbitrária $\mathcal{E}$, procede-se da seguinte forma:

- Assumindo que $\mathcal{P}$ é verdadeira para toda expressão $\mathcal{E}'$ usada no construto $\mathcal{E}$ (*Hipótese Indutiva*),
- e provando que $\mathcal{P}$ também é verdadeira para $\mathcal{E}$.

Prova do Lemma: Seja $\mathcal{J} \equiv \Gamma \vdash L : \sigma$, e suponha que $\mathcal{J}$ é a conclusão final de uma derivação e assuma que o conteudo do Lemma vale para as premissas usadas para inferir a conclusão.

Pela definição das regras de inferência, existem três possibilidades de regra para conclusão: (var), $(appl)$ e $(abst)$. Provando por casos:

1. Se $\mathcal{J}$ é a conclusão da regra (var)
   Então $\mathcal{J}$ possui a forma $\Gamma \vdash x : \sigma$ se seguindo de $x : \sigma \in \Gamma$. O L do lemma é o x e precisamos provar que $FV(x) \subseteq dom(\Gamma)$. Mas isso é consequência direta de $x : \sigma \in \Gamma$.

2. Se $\mathcal{J}$ é a conclusão da regra $(appl)$
   Então $\mathcal{J}$ deve ter a forma $\Gamma \vdash MN : \tau$ e precisa-se provar que $FV(MN) \in dom(\Gamma)$. Por indução, a regra já é válida para as premissas de $(appl)$, que são $\Gamma \vdash M : \sigma \to \tau$ e $\Gamma \vdash N : \sigma$.
   Assim, pode-se assumir que $FV(M) \subseteq dom(\Gamma)$ e $FV(N) \subseteq dom(\Gamma)$. Como $FV(MN) = FV(M) \cup FV(N)$, então $FV(MN) \subseteq dom(\Gamma)$.

3. Se $\mathcal{J}$ é a conclusão da regra $(abst)$
   Então $\mathcal{J}$ deve ter a forma $\Gamma \vdash \lambda x : \sigma.M : \sigma \to \tau$ e precisa-se provar que $FV(\lambda x : \sigma.M) \in dom(\Gamma)$. Por indução, a regra já é válida para a premissa de $(abst)$, que é $\Gamma, x : \sigma \vdash M : \tau$.
   Assim, pode-se assumir que $FV(M) \subseteq dom(\Gamma) \cup \{x\}$. Como $FV(\lambda x : \sigma.M) = FV(M) \backslash \{x\}$, então $FV(M) \backslash \{x\} \subseteq dom(\Gamma)$.

Outras propriedades também podem ser provadas no mesmo estilo de indução:

Lemma 2.2. (Afinamento, Condensação, Permutação)

1. (*Afinamento*) Sejam Γ' e Γ'' contextos tais que $\Gamma' \subseteq \Gamma''$. Se $\Gamma' \vdash M : \sigma$, então $\Gamma'' \vdash M : \sigma$

2. (*Condensação*) Se $\Gamma \vdash M : \sigma$, então também $\Gamma \restriction FV(M) \vdash M : \sigma$

3. (*Permutação*) Se $\Gamma \vdash M : \sigma$ e Γ' é uma permutação de Γ, então Γ' também é um contexto e $\Gamma' \vdash M : \sigma$.

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