1 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.

1.1 Cálculo λ simplesmente tipado (STλC)

1.1.1 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 1.1 (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.

1.1.2 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*.

1.1.3 Regras de derivação e Cálculo de sequêntes

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

Definição 1.2 (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 1.3.

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 1.4 (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:

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

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. ($\Rightarrow -intro$)

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

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

1.1.4 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 $\lambda_{\to}$. 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*.

1.1.5 Bem-tipagem em $\lambda_{\to}$

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

O objetivo é mostrar que o termo $M \equiv \lambda y : \alpha \to \beta.\lambda z : \alpha.yz$ 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:

$$\frac{?}{\emptyset \vdash \lambda y : \alpha \to \beta.\lambda z : \alpha.yz : \dots} \; ?$$

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

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

Novamente, a única regra possível é a abstração:

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

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:

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

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.

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

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

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

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.

1.1.6 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.

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

Usando a regra da aplicação, tem-se:

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

O lado direto se segue da regra da aplicação:

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

Usando essa subárvore, pode-se ver que yx possui o tipo $yx : \beta$.

O lado esquerdo se segue da abstração:

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

abstraindo novamente:

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

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

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

e

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

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 $\gamma \to \beta$.

1.1.7 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:

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

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

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

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

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

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

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

Usando a regra da abstração:

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

Novamente:

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

1.1.8 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 1.6 ((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 $\Gamma \upharpoonright \Phi$, 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. $\Gamma \upharpoonright \{z, u, x_1\} = x_1 : \rho_1, z : \tau$

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

Lemma 1.1. (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 1.2. (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 $\Gamma' \vdash M : \sigma$.