1 O Sistema F

No Cálculo-Lambda Simplesmente Tipado, é possível definir a função identidade, a função que pega um valor como input e retorna o próprio valor como outpu, para cada tipo definido no cálculo:

- Para os números naturais, $\lambda x : \mathbb{N}.x$
- Para os booleanos, $\lambda x : bool.x$
- Para o tipo das funções dos naturais nos booleanos, $\lambda x : (\mathbb{N} \to bool).x$
- ...

Mas dessa forma, quanto mais tipos a teoria suportar, mais formais diferentes são possíveis de serem criadas. Isso faz com que existam vários termos análogos sem qualquer possibilidade de relação entre eles. O máximo que se pode dizer é fazer uma quantificação além de $\lambda_\to$ e construir um tipo arbitrário α com uma função $f \equiv \lambda x : \alpha.x$ que seria a função identidade arbitrária.

Porém, dado um termo $M : \mathbb{N}$, não é possível escrever fM pois $\alpha \not\equiv \mathbb{N}$. Para fazer isso, é necessário que a função receba também o tipo específico que ela precisa ter para receber o termo M, fazendo um segundo processo de abstração em cima da função da seguinte forma:

$$\lambda\alpha : *.\lambda x : \alpha.x$$

Nesse caso, α se torna uma variável de tipo e $\star$ o tipo de todos os tipos. Esse termo é chamado de *polimórfico*, pois pode possuir diversas formas diferentes a depender do tipo escolhido:

- $(\lambda\alpha : *.\lambda x : \alpha.x)\mathbb{N} \to_\beta \lambda x : \mathbb{N}.x$

Para fazer essa extensão, é necessário adicionar regras de inferência e regras de tipagem que lidem com essa abstração de segunda ordem.

A tipagem para a função identidade $\lambda\alpha : *.\lambda x : \alpha.x$ é o tipo $\Pi\alpha : *.\alpha \to \alpha$, onde Π é o operador que tem como função ligar os tipos, chamado de Tipo Π ou Tipo Produto

Exemplos:

- A função de iteração D que recebe uma função $f : \alpha \to \alpha$ e retorna a aplicação dela duas vezes em cima de um termo $x : \alpha$ pode ser escrita da seguinte forma:

  $$D \equiv \lambda\alpha : *.\lambda f : \alpha \to \alpha.\lambda x : \alpha.f(fx)$$

  Nesse caso, D é a mesma coisa que $f \circ f$. Para os números naturais:

  $$D\mathbb{N} \equiv \lambda f : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.f(fx)$$

  e sendo f a função sucessor s que mapeia $n : \mathbb{N}$ em $n + 1 : \mathbb{N}$, então:

  $$D\mathbb{N}s \to_\beta \lambda x : \mathbb{N}.s(sx)$$

  O tipo de D é: $D : \Pi\alpha : *.(\alpha \to \alpha) \to \alpha \to \alpha$

- A composição de duas funções é a aplicação de uma função em outra. É possível definir o operador de composição $\circ$ da seguinte forma:

$$\circ \equiv \lambda\alpha : *.\lambda\beta : *\lambda\gamma : *.\lambda f : \alpha \to \beta.\lambda g : \beta \to \gamma.\lambda x : \alpha.g(fx)$$

A sua tipagem é: $\circ : \Pi\alpha : *.\Pi\beta : *.\Pi\gamma : *.(\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma$

1.1 O Cálculo Lambda com tipagem de Segunda Ordem

1.1.1 Regras de Inferência

Uma vez inseridas as regras de abstração e aplicação de segunda ordem, é necessário extender as regras de inferência em relação ao STλC

Definição 1.1 (Regra de Inferência para a Abstração)**.**

$$\frac{\Gamma, \alpha : * \vdash M : A}{\Gamma \vdash \lambda\alpha : *.M : \Pi\alpha : *.A}\ abst_2$$

Essa regra define basicamente que, sendo M um termo de tipo A em um contexto onde α possui tipo $*$, então a abstração $\alpha : *.M$ possui o tipo $\Pi\alpha : *.A$. Essa regra da abstração difere da primeira por permitir a definição de α no contexto.

Definição 1.2 (Regra de Inferência para a Aplicação)**.**

$$\frac{\Gamma \vdash M : \Pi\alpha : *.A \qquad \Gamma \vdash B : *}{\Gamma \vdash MB : A[\alpha := B]}\ appl_2$$

1.1.2 O Sistema $\lambda 2$

A sintaxe de $\lambda 2$ segue de forma análoga a λ_σ, sendo descrita pela seguinte BNF:

$$\mathbb{T}2 = \mathbb{V}|(\mathbb{T}2 \to \mathbb{T}2)|(\Pi\mathbb{V} : *.\mathbb{T}2)$$

onde $\mathbb{V}$ é a coleção dos tipos variáveis, denominados de $\alpha, \beta, \gamma, \ldots$.

Para os termos pré-tipados:

Definição 1.3. A coleção dos λ-termos pré-tipados de segunda ordem, ou $\lambda 2$-termos, é definido na seguinte BNF:

$$\Lambda_{\mathbb{T}2} = V|(\Lambda_{\mathbb{T}2}\Lambda_{\mathbb{T}2})|(\Lambda_{\mathbb{T}2}\mathbb{T}2)|(\lambda V : \mathbb{T}2.\Lambda_{\mathbb{T}2})|(\lambda\mathbb{V} : *.\Lambda_{\mathbb{T}2})$$

Onde V é a coleção das variáveis de termos $(x, y, z, \ldots)$. Como existem ambos $\mathbb{V}$ e V, então a BNF possui duas formas de aplicação, uma de primeira ordem $(\lambda V : \mathbb{T}2.\Lambda_{\mathbb{T}2})$ para variáveis de termo e outro de segunda ordem $(\lambda\mathbb{V} : *.\Lambda_{\mathbb{T}2})$ para variáveis de tipo.

Da mesma forma, também existe a aplicação de primeira ordem $(\Lambda_{\mathbb{T}2}\Lambda_{\mathbb{T}2})$ e de segunda ordem $(\Lambda_{\mathbb{T}2}\mathbb{T}2)$.

As regras de parenteses em aplicação e abstração segue as regras vistas anteriormente para o STλC e para o $\lambda_{\beta\eta}$:

- Parenteses mais externos podem ser omitidos
- Aplicação é associativa à esquerda
- Aplicação e $\rightarrow$ precedem ambas abstrações λ e Π
- Abstrações λ e Π sucessivas com o mesmo tipo podem ser combinadas de forma associativa à direita
- Tipos funcionais são escritos de forma associativa à direita

Exemplo: $(\Pi\alpha : *.(\Pi\beta : *.(\alpha \rightarrow (\beta \rightarrow \alpha))))$ pode ser escrito como $\Pi\alpha, \beta : *.\alpha \rightarrow \beta \rightarrow \alpha$.

A definição para declarações e sentenças pode ser estendida da seguinte forma:

Definição 1.4 (Declarações, sentenças)**.**

- Uma *sentença* possui a forma $M : \sigma$ onde $M \in \Lambda_{\mathbb{T}2}$ e $\sigma \in \mathbb{T}2$ ou da forma $\sigma : *$, onde $\sigma \in \mathbb{T}2$
- Uma *declaração* é uma sentença com uma variável de termo ou uma variável de tipo como sujeito

Para $\lambda 2$ como é possível que uma variável de termo faça uso de uma variável de tipo, é necessário que a ordem da aparição dessas variáveis siga uma regra, para que uma variável não seja usada antes de ser declarada. O contexto pode ser descrito como um *domínio* da seguinte forma:

Definição 1.5 (Contexto de $\lambda 2$)**.**

1. $\emptyset$ é um contexto válido de $\lambda 2$
   $dom(\emptyset) = ()$, a lista vazia
2. Se Γ for um contexto de $\lambda 2$, $\alpha \in \mathbb{V}$ e $\alpha \notin dom(\Gamma)$, então $\Gamma, \alpha : *$ é um contexto de $\lambda 2$
   $dom(\Gamma, \alpha : *) = (dom(\Gamma), \alpha)$, ou seja $dom(\Gamma)$ concatenado com α
3. Se Γ for um contexto de $\lambda 2$, se $\rho \in \mathbb{T}2$ tal que $\alpha \in dom(\Gamma)$ para toda variável de tipo livre α existente em ρ e se $x \notin dom(\Gamma)$, então $\Gamma, x : \rho$ é um contexto de $\lambda 2$
   $dom(\Gamma, x : \rho) = (dom(\Gamma), x)$

Exemplos

- $\emptyset$ é um contexto de $\lambda 2$ por (1)
- $\alpha : *$ é um contexto de $\lambda 2$ por (2)
- $\alpha : *, x : \alpha \rightarrow \alpha$ é um contexto de $\lambda 2$ por (3)
- logo $\alpha : *, x : \alpha \rightarrow \alpha, \beta : *$ é um contexto de $\lambda 2$ por (2)

- e $\alpha : *, x : \alpha \to \alpha, \beta : *, y : (\alpha \to \alpha) \to \beta$ é um contexto de $\lambda 2$ por (3), sendo $dom(\Gamma) = (\alpha, x, \beta, y)$

A regra *var* pode ser reconstruida para lidar com os tipos de $\lambda 2$:

Definição 1.6. (Regra var em $\lambda 2$) (*var*) $\Gamma \vdash x : \sigma$ se Γ for um contexto de $\lambda 2$ e $x : \sigma \in \Gamma$

O problema é que, usando as regras até então, não é possível chegar ao juizo $\Gamma \vdash B : *$. Por isso, será introduzida uma nova regra:

Definição 1.7. (Regra de formação) (*form*) $\Gamma \vdash B : *$ se Γ é um contexto de $\lambda 2$, $B \in \mathbb{T}2$ e todas as variáveis de tipo livres em B sejam declaradas em Γ

Regras de $\lambda 2$:

- (*var*) $\Gamma \vdash x : \sigma$ se Γ for um contexto de $\lambda 2$ e $x : \sigma \in \Gamma$
- (*appl*)

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

- (*abst*)

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

- (*form*) $\Gamma \vdash B : *$ se Γ é um contexto de $\lambda 2$, $B \in \mathbb{T}2$ e todas as variáveis de tipo livres em B sejam declaradas em Γ
- (*appl*$_2$)

$$\frac{\Gamma \vdash M : \Pi\alpha : * . A \qquad \Gamma \vdash B : *}{\Gamma \vdash MB : A[\alpha := B]}\ appl_2$$

- (*abst*$_2$)

$$\frac{\Gamma, \alpha : * \vdash M : A}{\Gamma \vdash \lambda\alpha : * . M : \Pi\alpha : * . A}\ abst_2$$

Definição 1.8. ($\lambda 2$-termos legais) Um termo M em $\Lambda_{\mathbb{T}2}$ é chamado de *legal* se existe um contexto de $\lambda 2$ Γ e um tipo ρ em $\mathbb{T}2$ tal que $\Gamma \vdash M : \rho$

1.1.3 Exemplos de Derivação

Seja a seguinte árvore de inferência incompleta:

$$\dfrac{?}{\emptyset \vdash \lambda\alpha : *.\lambda f : \alpha \to \alpha.\lambda x : \alpha.f(fx) : \Pi\alpha : *.(\alpha \to \alpha) \to \alpha \to \alpha}\ ?$$

Primeiro, é necessário utilizar a regra ($abst_2$):

$$\dfrac{\dfrac{?}{\alpha : * \vdash \lambda f : \alpha \to \alpha.\lambda x : \alpha.f(fx) : (\alpha \to \alpha) \to \alpha \to \alpha}\ ?}{\emptyset \vdash \lambda\alpha : *.\lambda f : \alpha \to \alpha.\lambda x : \alpha.f(fx) : \Pi\alpha : *.(\alpha \to \alpha) \to \alpha \to \alpha}\ abst_2$$

Após isso as regras que precisam ser utilizadas já são conhecidas a partir do STλC:

primeiro dois absts seguidos para f e x:

$$\dfrac{\dfrac{\dfrac{\dfrac{?}{\alpha : *, f : \alpha \to \alpha, x : \alpha \vdash f(fx) : \alpha}\ ?}{\alpha : *, f : \alpha \to \alpha \vdash \lambda x : \alpha.f(fx) : \alpha \to \alpha}\ abst}{\alpha : * \vdash \lambda f : \alpha \to \alpha.\lambda x : \alpha.f(fx) : (\alpha \to \alpha) \to \alpha \to \alpha}\ abst}{\emptyset \vdash \lambda\alpha : *.\lambda f : \alpha \to \alpha.\lambda x : \alpha.f(fx) : \Pi\alpha : *.(\alpha \to \alpha) \to \alpha \to \alpha}\ abst_2$$

O resto da Derivação fica como exercício para o leitor

1.1.4 Propriedades de $\lambda 2$

A definição de α-conversão deve ser acomodada para lidar com tipos Π:

Definição 1.9 (α-conversão ou α-equivalência)**.**

1. (Renomeando variáveis de termo)
   $\lambda x : \sigma.M =_\alpha \lambda y : \sigma.M^{x \to y}$ se $y \notin FV(M)$ e y não ocorre como ligante em M
2. (Renomeando variáveis de tipo)
   $\lambda\alpha : *.M =_\alpha \lambda\beta : *.M[\alpha := \beta]$ se β não ocorre em M
   $\Pi\alpha : *.M =_\alpha \Pi\beta : *.M[\alpha := \beta]$ se β não ocorre em M
3. O resto das definições se segue da definição 1.8

Também é possível extender a regra de β-redução:

Definição 1.10. (β-redução de passo único)

1. (Base, de primeira ordem) $(\lambda : \sigma.M)N \to_\beta M[x := N]$
2. (Base, de segunda ordem) $(\lambda\alpha : *.M)T \to_\beta M[\alpha := T]$
3. (Compatibilidade) da mesma forma que definição 1.10

Os lemmas definidos no capítulo 2 também podem ser utilizados aqui:

Lema 1.1. Os seguintes lemas e teoremas também são válidos para $\lambda 2$:

- Lema das variáveis livres
- Lema do afinamento
- Lema da condensação
- Lema da geração
- Lema do subtermo
- Unicidade dos tipos
- Lema da substituição
- Teorema de Church-Rosser
- Redução do sujeito
- Teorema da normalização forte

Lema 1.2 (Lema da permutação)**.** Se $\Gamma \vdash M : \sigma$ e Γ' é uma permutação de Γ e um contexto de $\lambda 2$ válido, então Γ' também é um contexto e $\Gamma' \vdash M : \sigma$.

1.2 O Sistema $\mathcal{F}$ de girard