1 Teoria dos Tipos Dependente

1.1 Teoria dos Tipos dependentes

Na Teoria dos Tipos simples $\lambda_{\rightarrow}$, cada termo depende de outro. Para cada extensão, foram adicionadas novas dependências:

- $\lambda 2$: termos dependem de tipos
- $\lambda \underline{\omega}$: tipos dependem de tipos

Fica faltando então uma teoria dos tipos que abarque tipos que dependem de termos.

- λP: tipos dependem de termos

É essa teoria que será analisáda nesse capítulo. Tipos que pendendem de termos possuem o seguinte formato:

$$\lambda x : A.M$$

onde M é um tipo e x é uma variável de termo (Logo A é um tipo também). A abstração $\lambda x : A.M$ *depende* do termo x.

Exemplos de Motivação:

(1) Na programação, podemos definir uma lista a partir de seu tamanho, por exemplo: $[1, 2]$: List2. Logo $\lambda n : \mathbb{N}.Listn$ também é um tipo, também chamado de *construtor de tipo*, *família de tipos* ou *tipo indexado* (indexado pelo termo $n : \mathbb{N}$) que depende do termo n

(2) Seja $S_n = \{0, n, 2n, 3n, \dots\}$ o conjunto de todos os multiplos não negativos de n. Então $\lambda n : \mathbb{N}.S_n$ mapeia:

  - $0 \mapsto \{0\}$
  - $1 \mapsto \mathbb{N}$ (O conjunto de todos os números naturais)
  - $2 \mapsto \{0, 2, 4, \dots\}$ (O conjunto de todos os números pares)

O tipo de S_n e de List$n$ é $\mathbb{N} \rightarrow *$.
Um exemplo importante é o seguinte:

(3) Seja P_n uma *proposição* para cada $n : \mathbb{N}$. A partir da interpretação de *Proposições-como-Tipos*, $\lambda n : \mathbb{N}.P_n$ é um tipo que mapeia n para sua proposição P_n correspondente, chamado de *função com valor de proposição*. Na lógica, esse tipo de construção é chamado de *Predicado*. Por exemplo, seja a interpretação de P_n como "n é um número primo". Na lógica, esse predicado pode ser verdadeiro ou falso a depender do valor de n

1.1.1 Regras de Inferência de λP

As regras de inferência de λP são as seguintes:

$$(sort)\ \emptyset \vdash * : \square$$

$$(var)\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \text{ se } x \notin \Gamma$$

$$(weak)\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \text{ se } x \notin \Gamma$$

$$(form)\ \frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.B : s}$$

$$(appl)\ \frac{\Gamma \vdash M : \Pi x : A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

$$(abst)\ \frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.B}{\Gamma \vdash \lambda x : A.M : \Pi x : A.B}$$

$$(conv)\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \text{ se } B =_\beta B'$$

As regras $(sort)$, (var) e $(weak)$ em λP são identicas às de $\lambda\underline{\omega}$. Porém, as regras que diferem de $\lambda\underline{\omega}$ são porque:

(i) Com o uso de tipos Π, nas regras de $(form)$, $(appl)$ e $(abst)$ os tipos $\to$ não aparecem como em $A \to B$ e no lugar são colocados tipos como $\Pi x : A.B$.

(ii) No tipo $\Pi x : A.B$, x é necessariamente um *termo*, logo $A : *$. O que difere de $\lambda\underline{\omega}$

Em λP, a abstração só é escrita como $A \to B$ no lugar de $\Pi x : A.B$ se existe a certeza que x não ocorre livre em B.

Na regra $(form)$, existem dois casos possíveis:

1. Se $s = *$, então $A : *$, $B : *$ e $\Pi x : A.B : *$
2. Se $s = \square$, então $A : *$, $B : \square$ e $\Pi x : A.B : \square$

1.1.2 Exemplo de derivação em λP

Primeiro é interessante derivar o tipo $A \to * : \square$:

$$(form)\ \frac{(var)\ \dfrac{\emptyset \vdash * : \square}{\emptyset \vdash A : *} \qquad (weak)\ \dfrac{\emptyset \vdash * : \square \qquad (var)\ \dfrac{(weak)\ \dfrac{\emptyset \vdash * : \square \qquad \emptyset \vdash * : \square}{A : * \vdash * : \square}}{\emptyset \vdash A : *}}{x : A \vdash * : \square}}{\emptyset \vdash A \to * : \square}$$

Uma vez construido esse tipo, é possível utilizar a regra (var) para gerar um habitante desse tipo:

$$(var)\ \frac{\emptyset \vdash A \to * : \square}{P : A \to * \vdash P : A \to *}$$

Seja $x : A$ um termo, é possível construir a aplicação de P com x:

$$(appl)\ \dfrac{P : A \to * \vdash P : A \to * \qquad (var)\ \dfrac{\emptyset \vdash A : *}{x : A \vdash x : A}}{P : A \to *, x : A \vdash Px : *}$$

Podemos gerar o seguinte tipo:

$$(weak)\ \dfrac{P : A \to *, x : A \vdash Px : * \qquad P : A \to *, x : A \vdash Px : *}{P : A \to *, x : A, y : Px \vdash Px : *}$$

e

$$(form)\ \dfrac{P : A \to *, x : A \vdash Px : * \qquad P : A \to *, x : A, y : Px \vdash Px : *}{P : A \to *, x : A \vdash Px \to Px : *}$$

Logo:

$$(form)\ \dfrac{(weak)\ \dfrac{\emptyset \vdash A : * \qquad \emptyset \vdash A \to * : \square}{P : A \to * \vdash A : *} \qquad P : A \to *, x : A \vdash Px \to Px : *}{P : A \to * \vdash \Pi x : A.Px \to Px : *}$$

Para gerar os termos:

$$(abst)\ \dfrac{(abst)\ \dfrac{(var)\ \dfrac{P : A \to *, x : A \vdash Px : *}{P : A \to *, x : A, y : Px \vdash y : Px} \qquad P : A \to *, x : A \vdash Px \to Px : *}{P : A \to *, x : A \vdash \lambda y : Px.y : Px \to Px} \qquad P : A \to * \vdash \Pi x : A.Px \to}{P : A \to * \vdash \lambda x : A\lambda y : Px.y : \Pi x : A.Px \to Px}$$

Fica para o leitor integrar essas díversas árvores em uma única.

1.1.3 Lógica de Predicados mínima em λP

Em λP é possível codificar uma forma de lógica simples chamada de *lógica de predicados mínima*. Essa lógica só possui a implicação e o quantificador universal em sua estrutura. As suas entidades básicas são *proposições*, *conjuntos* e *predicados sobre conjuntos*.

A interpretação de Proposições-como-Tipos (PAT) é feita da seguinte forma:

- Se o termo b habita o tipo B (ou seja, $b : B$) e sendo B interpretada como uma proposição, então b é a *prova* de B, chamado de *objeto de prova*.

- Se um tipo B não possui habitante, então não existe prova de B e B deve ser falso

Em λP, para definir que b habita B temos que realizar um juizo no estilo $\Gamma \vdash b : B$ a partir das regras de inferência descritas anteriormente.

Um conjunto S pode ser codificado como um tipo, então $S : *$. *Elementos* de um conjunto são termos. Então se a é um elemento de S, $a : S$. Se S for o conjunto vazio, S não vai possuir termos.

Exemplos: Se $\mathbb{N} : *$, $3 : \mathbb{N}$

Proposições também podem ser definidas como tipos. Então sendo A uma proposição, $A : *$. Um termo $p : A$ é uma prova de A.

Como visto anteriormente, um predicado P é uma função de um *conjunto* S para o *conjunto de todas as proposições*, então: $P : S \to *$. Logo seja P um predicado arbitrário em S, ou seja $P : S \to *$, então para cada $a : S$ tem-se $Pa : *$. Todo Pa é uma proposição, que é um tipo em λP, logo existem duas possibilidades:

1. Se Pa for *habitado*, ou seja existe $t : Pa$, então o predicado é válido para a

2. Caso Pa não seja habitado, o predicado não se segue para a

Anteriormente, foi identificada a implicação $A \Rightarrow B$ com o tipo $A \to B$ da seguinte forma:

$A \Rightarrow B$ é verdadeiro

Se A é verdadeiro, então B é verdadeiro

Se A é habitado, então B é habitado

Existe uma função mapeando habitantes de A em habitantes de B

Existe uma função $f : A \to B$

$A \to B$ é habitado

A partir das regras de λP é possível obter as regras de eliminação e introdução da implicação:

1. $\Rightarrow$-elim $\dfrac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$

2. $\Rightarrow$-intro $\dfrac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash A \to B : s}{\Gamma \vdash \lambda x : A.M : A \to B}$

Já a quantificação universal, $\forall_{x \in S} P(x)$, de um prediado P dependente de um x elemento de S vai ter sua equivalência encontrada da seguinte forma:

$\forall_{x \in S} P(x)$ é verdadeiro

Para cada x pertencente a S, a proposição $P(x)$ é verdadeira

para cada X em S, o tipo Px é habitado

Existe uma função mapeando cada x em S para um habitante de Px

Existe uma função f tal que $f : \Pi x : S.Px$

$\Pi x : S.Px$ é habitado

Logo, a forma de codificação de $\forall_{x \in S} P(x)$ é o tipo $\Pi x : S.Px$.

As regras de eliminação e introdução do $\forall$ no λP são as seguintes:

1. $\forall$-elim $\dfrac{\Gamma \vdash p : \forall_{x \in S} P(x) \qquad \Gamma \vdash n : S}{\Gamma \vdash pn : P(x)[x := n]}$

2. $\forall$-intro $\dfrac{\Gamma, x : S \vdash M : P(x) \qquad \Gamma \vdash \forall_{x \in S} P(x) : *}{\Gamma \vdash \lambda x : S.P(x) : \forall_{x \in S} P(x)}$

Essas regras correspondem, na dedução natural no estilo de Gentzen às seguintes:

1. $\forall\text{I}\,\dfrac{P(n)}{\forall_{x\in S}P(x)}$

2. $\forall\text{E}\,\dfrac{\forall_{x\in S}P(x)}{P(n)}$

1.1.4 Exemplo de derivação na lógica de predicados mínima

Seja S um conjunto de Q um predicado sobre S, então a seguinte proposição é provável usando a lógica de predicados mínima:

$$\forall_{x\in S}\forall_{y\in S}(Q(x,y)) \Rightarrow \forall_{u\in S}Q(u,u)$$

Na dedução natural, isso se torna:

$$\to\text{I}\,\dfrac{\forall\text{I}\,\dfrac{\forall\text{E}\,\dfrac{\forall\text{E}\,\dfrac{\forall_{x\in S}\forall_{y\in S}(Q(x,y))^1}{\forall_{y\in S}(Q(z,y))}}{Q(u,u)}}{\forall_{u\in S}Q(u,u)}}{\forall_{x\in S}\forall_{y\in S}(Q(x,y)) \Rightarrow \forall_{u\in S}Q(u,u)}$$

Usando as regras de inferência introduzidas anteriormente:
Primeiro, é necessário traduzir essa proposição para um tipo:

$$\Pi x : S.\Pi y : S.Qxy \to \Pi u : S.Quu$$

Então o problema se torna:

$$?\,\dfrac{?}{\emptyset \vdash ? : \Pi x : S.\Pi y : S.Qxy \to \Pi u : S.Quu}$$

Usando as regras, é possível ver que o tipo $\Pi x : S.\Pi y : S.Qxy$ precisa ser definido por um termo único z na abstração:

$$\to\text{-intro}\,\dfrac{?\,\dfrac{?}{z : (\Pi x : S.\Pi y : S.Qxy) \vdash ? : \Pi u : S.Quu} \qquad \emptyset \vdash \Pi x : S.\Pi y : S.Qxy \to \Pi u : S.Quu : *}{\emptyset \vdash \lambda z : (\Pi x : S.\Pi y : S.Qxy).? : \Pi x : S.\Pi y : S.Qxy \to \Pi u : S.Quu}$$

Também por abstração, $\Pi u : S$ também se torna um termo próprio:

$$\to\text{-intro}\,\dfrac{\to\text{-intro}\,\dfrac{z : (\Pi x : S.\Pi y : S.Qxy), u : S \vdash ? : Quu \qquad z : (\Pi x : S.\Pi y : S.Qxy) \vdash S : *}{z : (\Pi x : S.\Pi y : S.Qxy) \vdash \lambda u : S.? : \Pi u : S.Quu}}{\emptyset \vdash \lambda z : (\Pi x : S.\Pi y : S.Qxy).\lambda u : S.? : \Pi x : S.\Pi y : S.Qxy \to \Pi u : S.Quu}$$

A partir daqui é usado a regra da aplicação para o $\forall$:

$$
\dfrac{\dfrac{\dfrac{\dfrac{z : (\Pi x : S.\Pi y : S.Qxy), u : S \vdash z : (\Pi x : S.\Pi y : S.Qxy)}{z : (\Pi x : S.\Pi y : S.Qxy), u : S \vdash zu : \Pi y : S.Quy}\ \forall\text{-elim}}{z : (\Pi x : S.\Pi y : S.Qxy), u : S \vdash zuu : Quu}\ \forall\text{-elim}}{z : (\Pi x : S.\Pi y : S.Qxy) \vdash \lambda u : S.zuu : \Pi u : S.Quu}\ \to\text{-intro}}{\emptyset \vdash \lambda z : (\Pi x : S.\Pi y : S.Qxy).\lambda u : S.zuu : \Pi x : S.\Pi y : S.Qxy \to \Pi u : S.Quu}\ \to\text{-intro}
$$

O lado direto de cada passo é deixado para o leitor fazer por si só (Dica: usar uma folha A4 no modo paisagem).

Logo, o termo que prova que a proposição é verdadeira é o $\lambda z : (\Pi x : S.\Pi y : S.Qxy).\lambda u : S.zuu$. O interessante de descobrir o termo é que, somente a partir do termo, é possível reconstruir toda a prova novamente.