1 Teoria dos Tipos de Martin-Löf

Nesse capítulo, o interesse é dar uma revisão geral da teoria dos tipos dependentes e apresentar a Teoria dos tipos de Martin-Löf, uma teoria dos tipos dependentes com tipos adicionais.

1.1 Apontamentos iniciais

Como visto no capítulo 5, a teoria dos tipos dependentes é uma extensão da teoria dos tipos simples que permite a construção de tipos que dependam de termos. Por exemplo, seja $n : \mathbf{N}$ um termo do tipo dos naturais, é possível construir o tipo Vect n dos vetores de tamanho n.

A teoria dos tipos dependentes de Martin-Löf foi chamada por ele também de Teoria dos Tipos Intuicionista, pois é um tipo de teoria dos tipos que se preocupa com a formalização da matemática intuicionista presente em autores como Bishop. Na apresentação da Teoria dos tipos de Martin-Lof (MLTT), serão feitas algumas mudanças de sintaxe presentes em textos mais recentes como (**??**), por exemplo a apresentação de termos como $a : A$ ao invés de $a \in A$

1.2 Os tipos da MLTT

A exposição dos tipos presentes na teoria dos tipos de Martin-Löf é bastante correlata à exposição dos tipos presentes na teoria dos tipos simples feita na segunda parte do capítulo 3. Como na teoria dos tipos simples, as regras de cada tipo da Teoria dos Tipos de Martin-Lof são divididas em quatro partes:

- Uma *regra de formação* que diz como formar o tipo
- Uma *regra de introdução* que diz como introduzir novos termos do tipo
- Uma *regra de eliminação* que diz como usar e remover termos do tipo
- Uma *regra de computação* que diz como as regras de introdução e de eliminação se relacionam

1.2.1 O tipo de função dependente

Sejam A um tipo e x um elemento desse tipo, $x : A$, um tipo B que depende de x é denotado por $B(x)$, esse tipo será habitado por um termo $b(x)$. Nesse caso, b pode ser visto como uma função que leva um termo $x : A$ a um termo $b(x) : B(x)$, chamada de função dependente. O tipo de todas as funções dependentes que levam um termo $x : A$ qualquer a um termo do tipo $B(x)$ é denotado por $\Pi_{x:A} B(x)$.

Essa apresentação pode ser formalizada pela seguinte regra de formação:

$$\frac{\Gamma, x : A \vdash B(x)\ \text{Type}}{\Gamma \vdash \Pi_{x:A} B(x)\ \text{Type}}\ \Pi\text{-form}$$

Um elemento $f : \Pi_{x:A} B(x)$ é uma função que leva termos $x : A$ a termos $b(x) : B(x)$, essa função pode ser representada por um λ-termo a partir da seguinte regra de introdução:

$$\frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash \lambda x.b(x) : \Pi_{x:A} B(x)} \Pi\text{-intro}$$

Para eliminar um termo $f : \Pi_{x:A} B(x)$ é preciso somente computar o resultado da aplicação $f(x)$ para um $x : A$

$$\frac{\Gamma \vdash f : \Pi_{x:A} B(x)}{\Gamma, x : A \vdash f(x) : B(x)} \Pi\text{-intro}$$

As regras de computação para o tipo de função dependente são as regras de β e η-conversão:

$$\frac{\Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash (\lambda y.b(y))(x) = b(x) : B(x)} \beta$$

$$\frac{\Gamma \vdash f : \Pi_{x:A} B(x)}{\Gamma \vdash (\lambda x.f(x)) = f : \Pi_{x:A} B(x)} \eta$$

Uma vez definido o tipo de funções dependentes é possível definir o tipo de funções ordinárias como:

$$\frac{\Gamma \vdash A \text{ Type} \qquad \Gamma \vdash B \text{ Type}}{\Gamma \vdash A \to B = \Pi_{x:A} B(x) \text{ Type}}$$

As regras para o tipo $A \to B$ são iguais as regras do tipo $\Pi_{x:A} B(x)$.

Uma função importante é a função identidade $id_A : A \to A$, definida como $\lambda x.x : A \to A$.

Para funções dependentes que dependem de mais de um termo, é possível usar a notação

$$\Pi_{x:A} \Pi_{y:B} C(x, y)$$

mas também

$$\Pi_{x:A, y:B} C(x, y)$$

Definição 1.1 ((**??**))**.** Sejam A, B e C três tipos em um contexto Γ, existe uma operação de *composição*

$$\text{comp} : (B \to C) \to ((A \to B) \to (A \to C))$$

escrita como $g \circ f$ para $\text{comp}(g, f)$

O termo comp pode ser construido como:

$$\text{comp} := \lambda g.\lambda f.\lambda x.g(f(x))$$

Lema 1.1 ((**??**))**.** A composição de funções é associativa, ou seja, é possível derivar

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash g : B \to C \qquad \Gamma \vdash h : C \to D}{\Gamma \vdash (h \circ g) \circ f = h \circ (g \circ f) : A \to D}$$

Prova: a ideia principal da prova é que tanto $(h \circ g) \circ f$ quanto $h \circ (g \circ f)$ são evaluados para $h(g(f(x)))$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash f : A \to B}{\Gamma, x : A \vdash f(x) : B} \quad \dfrac{\dfrac{\Gamma \vdash g : B \to C}{\Gamma, y : B \vdash g(y) : C}}{\Gamma, x : A, y : B \vdash g(y) : C}}{\Gamma, x : A \vdash g(f(x)) : C} \quad \dfrac{\dfrac{\Gamma \vdash h : C \to D}{\Gamma, z : C \vdash h(z) : D}}{\Gamma, x : A, z : C \vdash h(z) : D}}{\Gamma, x : A \vdash h(g(f(x))) : A \to D}}{\Gamma, x : A \vdash h(g(f(x))) = h(g(f(x))) : A \to D}}{\Gamma, x : A \vdash ((h \circ g)) \circ f(x) = h \circ (g \circ f(x)) : A \to D}}{\Gamma, x : A \vdash ((h \circ g) \circ f)(x) = (h \circ (g \circ f))(x) : A \to D}}{\Gamma, x : A \vdash (h \circ g) \circ f = h \circ (g \circ f) : A \to D}}{\Gamma \vdash (h \circ g) \circ f = h \circ (g \circ f) : A \to D}$$

Lema 1.2 ((**??**))**.** A composição de funções satisfaz regras de unidade a esquerda e a direita, ou seja é possível derivar:

$$\dfrac{\Gamma \vdash f : A \to B}{\Gamma \vdash id_B \circ f = f : A \to B}$$

e

$$\dfrac{\Gamma \vdash f : A \to B}{\Gamma \vdash f \circ id_A = f : A \to B}$$

Prova: Para a primeira regra:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash A \text{ Type}}{\Gamma, x : A \vdash f(x) : B} \quad \dfrac{\Gamma \vdash A \text{ Type} \quad \dfrac{\Gamma \vdash B \text{ Type}}{\Gamma, y : B \vdash id(y) = y : B}}{\Gamma, x : A, y : B \vdash id(y) = y : B}}{\Gamma, x : A \vdash id(f(x)) = (f(x)) : B}}{\Gamma \vdash \lambda x.id(f(x)) = \lambda x.f(x) : A \to B} \quad \dfrac{\Gamma \vdash f : A \to B}{\Gamma \vdash \lambda x; f(x) = f : A \to B}}{\Gamma \vdash id \circ f = f : A \to B}$$

a segunda regra fica como exercício para o leitor

1.2.2 O tipo dos números naturais

O tipo dos números naturais $\mathbb{N}$ é o primeiro tipo indutivo que será apresentado aqui nessa seção. Ele é, como os outros tipos indutivos, um tipo **postulado**, ou seja, sua regra de formação é construida a partir do contexto vazio da seguinte forma:

$$\dfrac{}{\vdash \mathbb{N} \text{ Type}}\ \mathbb{N}\text{-form}$$

A construção dos elementos do tipo dos inteiros segue a construção dos axiomas de Peano: é postulado um elemento inicial que, na literatura da teoria dos tipos é sempre o 0, e uma função sucessor que permite construir novos elementos

$$\dfrac{}{\vdash 0_{\mathbb{N}} : \mathbb{N}} \qquad \dfrac{}{\vdash succ_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}$$

A eliminação do tipo $\mathbb{N}$ representa o princípio da indução dos números naturais, onde o tipo resultante é o predicado $\forall_{n \in \mathbb{N}} P(n)$. Essa regra é:

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \text{type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \Pi_{n:\mathbb{N}} P(n) \to P(\text{succ}_{\mathbb{N}}(n))}{\Gamma \vdash ind_{\mathbb{N}}(p_0, p_S) : \Pi_{n:\mathbb{N}} P(n)}\ \mathbb{N}\text{-ind}$$

Da esquerda para a direita é possível ver que, sendo p_0 a prova que $P(0_{\mathbb{N}})$ é válido e p_S a prova que dado um termo de tipo $P(n)$ é sempre possível construir um termo $P(\text{succ}_{\mathbb{N}}(n))$, as premissas seguem o axioma de peano para a indução nos números naturais. O termo $ind_{\mathbb{N}}(p_0, p_S)$ pode ser visto também como uma função que recebe os termos p_0 e p_S como entrada, sendo representada de forma geral na seguinte regra equivalente:

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \text{type}}{\Gamma \vdash ind_{\mathbb{N}} : P(0_{\mathbb{N}}) \to ((\Pi_{n:\mathbb{N}} P(n) \to P(\text{succ}_{\mathbb{N}}(n))) \to \Pi_{n:\mathbb{N}} P(n))}$$

Ou seja, para cada família de tipos P sobre $\mathbb{N}$ existe uma *função* $ind_{\mathbb{N}}$ que pega dois argumentos, o primeiro sendo o caso base e o segundo o passo indutivo, e retorna uma seção de P

A regra de computação para $\mathbb{N}$ postula que a função dependente $nd_{\mathbb{N}}(p_0, p_S) : \Pi_{n:\mathbb{N}} P(n)$ se comporta como experado quando aplicada a $0_{\mathbb{N}}$ ou ao sucessor.

Para a aplicação com o $0_{\mathbb{N}}$:

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \text{type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \Pi_{n:\mathbb{N}} P(n) \to P(\text{succ}_{\mathbb{N}}(n))}{\Gamma \vdash ind_{\mathbb{N}}(p_0, p_S, 0_{\mathbb{N}}) = p_0 : P(0_{\mathbb{N}})}$$

Para o sucessor:

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n)\ \text{type} \qquad \Gamma \vdash p_0 : P(0_{\mathbb{N}}) \qquad \Gamma \vdash p_S : \Pi_{n:\mathbb{N}} P(n) \to P(\text{succ}_{\mathbb{N}}(n))}{\Gamma \vdash ind_{\mathbb{N}}(p_0, p_S, \text{succ}_{\mathbb{N}}) = p_0 : P(\text{succ}_{\mathbb{N}})}$$

Exemplo:

Definição 1.2 (Adição, (**??**))**.** Definindo uma função

$$add_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$$

que satisfaça a especificação:

$$add_{\mathbb{N}}(m, 0_{\mathbb{N}}) = m$$

$$add_{\mathbb{N}}(m, \text{succ}_{\mathbb{N}}(n)) = \text{succ}_{\mathbb{N}}(add_{\mathbb{N}}(m, n))$$

$add_{\mathbb{N}}(m, n)$ será denotado por $m + n$

Para sua construção, é necessário usar a regra de eliminação para $P = \mathbb{N}$. Logo, as premissas se tornam:

$$m : \mathbb{N} \vdash \mathbb{N}$$

$$m : \mathbb{N} \vdash add - zero_{\mathbb{N}}(m) : \mathbb{N}$$

$$m : \mathbb{N} \vdash add - \text{succ}_{\mathbb{N}}(m) : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$$

e sua conclusão é:

$$m : \mathbb{N} \vdash add_{\mathbb{N}}(m) := ind_{\mathbb{N}}(add - zero_{\mathbb{N}}(m), add - \text{succ}_{\mathbb{N}}(m)) : \mathbb{N} \to \mathbb{N}$$

O termo $add - zero_{\mathbb{N}}(m)$ é o próprio m, já para a adição do sucessor, é necessário saber qual o comportamento da soma para o sucesso, que é a seguinte:

$$add_{\mathbb{N}}(m, succ_{\mathbb{N}}(n)) = succ_{\mathbb{N}}(add_{\mathbb{N}}(m, n))$$

Logo, $add - succ_{\mathbb{N}}(m)$ é o mesmo que o sucessor da adição:

$$add - succ_{\mathbb{N}}(m) := \lambda n.succ_{\mathbb{N}}$$

A árvore de derivação desse termo é o seeguinte:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash \mathbb{N} \to \mathbb{N} \qquad \vdash succ_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}{n : \mathbb{N} \vdash succ_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}}{m : \mathbb{N}, n : \mathbb{N} \vdash succ_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}}{m : \mathbb{N} \vdash \lambda n.succ_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}}{m : \mathbb{N} \vdash add - succ_{\mathbb{N}}(m) := \lambda n.succ_{\mathbb{N}} : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}$$

A regra de inferência final se torna:

$$\dfrac{\dfrac{\dfrac{\vdots}{m : \mathbb{N} \vdash \mathbb{N}\ \text{Type}} \quad \dfrac{\vdots}{m : \mathbb{N} \vdash add - zero_{\mathbb{N}}(m) := m : \mathbb{N}} \quad \dfrac{\vdots}{m : \mathbb{N} \vdash add - succ_{\mathbb{N}}(m) : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})}}{m : \mathbb{N} \vdash ind_{\mathbb{N}}(add - zero_{\mathbb{N}}(m), add - succ_{\mathbb{N}}(m)) : \mathbb{N} \to \mathbb{N}}}{m : \mathbb{N} \vdash add_{\mathbb{N}}(m) := ind_{\mathbb{N}}(add - zero_{\mathbb{N}}(m), add - succ_{\mathbb{N}}(m)) : \mathbb{N} \to \mathbb{N}}$$

Outra forma de definir a adição é usando *casamento de padrões*, na seguinte forma:

$$add_{\mathbb{N}}(m, 0_{\mathbb{N}}) = m$$

$$add_{\mathbb{N}}(m, succ_{\mathbb{N}}(n)) = succ_{\mathbb{N}}(add_{\mathbb{N}}(m, n))$$

essa especificação é suficiente para descrever o comportamento de $add_{\mathbb{N}}(m)$ a partir dos construtores de $\mathbb{N}$.

Para uma função qualquer f, é possível definir seu comportamento em $\mathbb{N}$ através de

$$f(0_{\mathbb{N}}) = p_0$$

$$f(succ_{\mathbb{N}}(n)) = p_S(n, f(n))$$

Quando f é definida dessa forma, é dito que ela é definida por **casamento de padrões** sobre n

1.2.3 Outros tipos indutivos

O tipo dos números naturais é o caso mais emblemático dos tipos indutivos. Esses tipos seguem regras de introdução similares a $\mathbb{N}$. Sua estrutura é dividida em :

- Construtores que definem a estrutura do tipo através de seus elementos base

- Um princípio indutivo, através das regras de eliminação, que especifica o que é necessário para construir uma família de tipos arbitrária sobre o tipo.
- Regras de computação que definem como os termos gerados indutivamente se relacionam com os construtores.

O Tipo Unitário

Um exemplo básico de um tipo indutivo é o *tipo unitário*, que só possui um construtor.

Definição 1.3 ((**??**))**.** O **tipo unitário** é definido como o tipo $\mathbf{1}$ equipado com um termo

$$\star : \mathbf{1}$$

satisfazendo o princípio indutivo que para qualquer família de tipos $P(x)$ indexada por $x : \mathbf{1}$, existe uma função

$$ind_{\mathbf{1}} : P(\star) \to \Pi_{(x:\mathbf{1})} P(x)$$

para qual a regra de computação

$$ind_{\mathbf{1}}(p, \star) = p$$

é válida

Caso P não dependa de x o princípio da indução se torna

$$ind_{\mathbf{1}} : P \to (\mathbf{1} \to P)$$

Ou seja, nesse caso, para cada $x : P$ existe uma função $pt_x := ind_{\mathbf{1}}(x) : \mathbf{1} \to A$

O Tipo Vazio

O tipo vazio é um tipo indutivo com nenhum construtor e, por não ter nenhum construtor, não possui regras de computação.

Definição 1.4 ((**??**))**.** O **tipo vazio** é o tipo $\varnothing$ que satisfaz o principio de indução que para qualquer família de tipos $P(x)$ indexada por $x : \varnothing$ existe um termo

$$ind_{\varnothing} : \Pi_{x:\varnothing} P(x)$$

Quando P não depende de x, se segue o seguinte termo na indução:

$$ex - falso := ind_{\varnothing} : \varnothing \to P$$

Essa função pode ser usada para mostrar qualquer conclusão a partir de uma contradição. Usando a interpretação de proposições como tipos, é possível construir operadores lógicos a partir desse tipo vazio como a negação:

Definição 1.5 ((**??**))**.** Para qualquer tipo A, a **negação** de A é definida como:

$$\neg A := A \to \varnothing$$

Um tipo A é dito **vazio** se vem equipado com um elemento do tipo $neg A$, ou seja:

$$is - empty(A) := A \to \varnothing$$

Exemplo ((**??**)): para quais quer dois tipos P e Q existe uma função

$$(P \to Q) \to (\neg Q \to \neg P)$$

Prova: Pela regra da abstração, é possível assumir a função $f : P \to Q$. Com isso, é necessário definir uma função $(\neg Q \to \neg P)$. Mas novamente pela regra da abstração é possível assumir $\tilde{q} : Q \to \varnothing$. Com isso sobra para provar $\neg P$ que é o mesmo que $P \to \varnothing$, então é possível abstrair $p : P$ ficando como objetivo construir o termo $\varnothing$.

Dessa forma, usando a aplicação, $fp : Q$ e $\tilde{q}(f(p)) : \varnothing$. Logo, o termo final é:

$$\lambda f.\lambda \tilde{q}.\lambda p.\tilde{q}(f(p)) : (P \to Q) \to (\neg Q \to \neg P)$$

□

O tipo coproduto

Definição 1.6 ((**??**)). Sejam A e B tipos. O **tipo coproduto** $A+B$ é o tipo equipado com

$$\mathtt{inl} : A \to A + B$$

$$\mathtt{inr} : B \to A + B$$

satisfazendo o princípio de indução que para qualquer família de tipos $P(x)$ indexada por $x : A + B$ existe o termo

$$\mathtt{ind}_+ : (\Pi_{(x:A)} P(\mathtt{inl}(x))) \to ((\Pi_{(y:B)} P(\mathtt{inr}(y))) \to \Pi_{(z:A+B)} P(z))$$

Para o qual seguem as regras de computação

$$\mathtt{ind}_+(f, g, \mathtt{inl}(x)) = f(x)$$

$$\mathtt{ind}_+(f, g, \mathtt{inr}(y)) = g(y)$$

Notação: por vezes é escrito $[f, g]$ ao invés de $\mathtt{ind}_+(f, g)$

O coproduto de dois tipos pode ser chamado também de **soma disjunta**.

É possível derivar, sem as dependências, a função

$$\mathtt{ind}_+ : (A \to X) \to ((B \to X) \to (A + B \to X))$$

É interessante notar que, usando a interpretação de proposições como tipos, essa regra é similar à regra da eliminação do *ou* na dedução natural:

$$(P \to Q) \to ((R \to Q) \to (P \vee R \to Q))$$

Outra coisa a notar é que é possível construir a função

$$f + g : A + B \to A' + B'$$

para toda função $f : A \to A'$ e $g : B \to B'$ definida como:

$$(f + g)(\mathtt{inl}(x)) := \mathtt{inl}(f(x))$$

$$(f + g)(\mathtt{inr}(x)) := \mathtt{inr}(g(y))$$

Proposição 1.1 ((**??**)). Sejam dois tipos A e B e suponha que B é vazio, então existe uma função

$$(A + B) \to A$$

Essa proposição pode ser reescrita como: existe uma função

$$is - empty(B) \to ((A + B) \to A)$$

Prova: Pelo princípio indutivo do coproduto $A + B$ é possível ver que para provar $(A + B) \to A$ é necessário construir duas funções $f : A \to A$ e $g : B \to A$. Nesse caso, f é simplesmente a função identidade $id_A : A \to A$. Dizer que B é vazio é o mesmo que construir a função $\tilde{b} : B \to \varnothing$ e também é possível definir $ex - false :\to A$ e g se torna $g := ex - false \circ \tilde{b}$ □

O Tipo dos Inteiros

O conjunto dos inteiros pode ser visto, de forma inicial, como o conjunto dos números naturais unido com o conjunto dos números negativos. Essa visão pode ser codificada através da seguinte definição:

Definição 1.7 ((??)). O tipo dos **Inteiros** é definido como o tipo $\mathbb{Z} := \mathbb{N} + (\mathbf{1} + \mathbb{N})$. O tipo dos inteiros vem equipado com as funções de inclusão dos inteiros positivos e negativos:

$$in - pos := inr \circ inr : \mathbb{N} \to \mathbb{Z}$$

$$in - neg := inl : \mathbb{N} \to \mathbb{Z}$$

e com as constantes

$$-1_{\mathbb{Z}} := in - neg(0)$$

$$0_{\mathbb{Z}} := inr(inl(\star))$$

$$1_{\mathbb{Z}} := in - pos(0)$$

O princípio de indução para os inteiros afirma que, para qualquer família de tipos P sobre $\mathbb{Z}$, é possível definir uma função dependente $f : \Pi_{k:\mathbb{Z}} P(k)$ recursivamente como:

$$f(-1_{\mathbb{Z}}) := p_{-1}$$

$$f(in - neg(succ_{\mathbb{N}}(n))) := p_{-S}(n, f(in - neg(n)))$$

$$f(0_{\mathbb{Z}}) := p_0$$

$$f(1_{\mathbb{Z}}) := p_1$$

$$f(in - pos(succ_{\mathbb{N}}(n))) := p_S(n, f(in - pos(n)))$$

onde os tipos de p_{-1}, p_{-S}, p_0, p_1 e p_S são

$$p_{-1} : P(-1_{\mathbb{Z}})$$

$$p_{-S} : \Pi_{n:\mathbb{N}} P(in - neg(n)) \to P(in - neg(succ_{\mathbb{N}}(n)))$$

$$p_0 : P(0_{\mathbb{Z}})$$

$$p_1 : P(1_{\mathbb{Z}})$$

$$p_S : \Pi_{n:\mathbb{N}} P(in - pos(n)) \to P(in - pos(succ_{\mathbb{N}}(n)))$$

Definição 1.8 ((**??**))**.** A **função sucessora** dos inteiros $\mathrm{succ}_{\mathbb{Z}} : \mathbb{Z} \to \mathbb{Z}$ pode ser definida tomando:

$$\mathrm{succ}_{\mathbb{Z}}(-1_{\mathbb{Z}}) := 0_{\mathbb{Z}}$$

$$\mathrm{succ}_{\mathbb{Z}}(in - neg(\mathrm{succ}_{\mathbb{N}}(n))) := in - neg(n)$$

$$\mathrm{succ}_{\mathbb{Z}}(0_{\mathbb{Z}}) := 1_{\mathbb{Z}}$$

$$\mathrm{succ}_{\mathbb{Z}}(1_{\mathbb{Z}}) := in - pos(1_{\mathbb{N}})$$

$$\mathrm{succ}_{\mathbb{Z}}(in - pos(\mathrm{succ}_{\mathbb{N}}(n))) := in - pos(\mathrm{succ}_{\mathbb{N}}(\mathrm{succ}_{\mathbb{N}}(n)))$$

O Tipo par dependente

Seja B uma família de tipos sobre A, é interessante considerar pares (a, b) de termos, onde $a : A$ e $b : B(a)$. Como nesse par b depende de a, ele é chamado de **par dependente**. O tipo dos pares dependentes é um tipo indutivo

Definição 1.9 ((**??**))**.** Seja B uma família de tipos sobre A. O **tipo par dependente**, também chamado de **Tipo** Σ, é definido como o tipo indutivo $\Sigma_{(x:A)}B(x)$ equipado com uma **função pareadora**

$$pair : \Pi_{x:A}(B(x) \to \Sigma_{(y:A)}B(y))$$

O princípio de indução para $\Sigma_{(x:A)}B(x)$ afirma que para qualquer família de tipos $P(p)$ indexada por $p : \Sigma_{(x:A)}B(x)$, existe uma função

$$ind_{\Sigma} : \Pi_{x:A}\Pi_{y:B(x)}P(pair(x, y)) \to (\Pi_{(z:\Sigma_{(x:A)}B(x))}P(z))$$

satisfazendo a regra de computação

$$ind_{\Sigma}(g, pair(x, y)) = g(x, y)$$

O princípio de indução de tipos Σ pode ser usado para definir as funções de projeção:

Definição 1.10 ((**??**))**.** Seja A um tipo e B uma família de tipos sobre A

1. O **mapa da primeira projeção**

$$pr_1(\Sigma_{(x:A)}B(x)) \to A$$

é definido por indução como

$$pr_1(x, y) := x$$

2. O **mapa da segunda projeção**

$$pr_2 : \Pi_{(p:Sigma_{(x:A)}B(x))}B(pr_1(p))$$

definido por indução como

$$pr_2(x, y) := y$$

O princípio da indução de Σ é uma forma de *Uncurrying*, pois ele pega uma função que aceita duas entradas

$$\lambda x.\lambda y.f(x,y) : \Pi_{x:A}\Pi_{y:B(x)}P(pair(x,y))$$

e retorna uma função que aceita um par de entradas

$$f : (\Pi_{(z:\Sigma_{(x:A)}B(x))}P(z))$$

A operação oposta também é possível de construir, chamada de *Currying*:

$$ev - pair : (\Pi_{(z:\Sigma_{(x:A)}B(x))}P(z)) \to (\Pi_{x:A}\Pi_{y:B(x)}P(pair(x,y)))$$

Um caso especial do tipo Σ ocorre quando B é uma família constante sobre A. Nesse caso, o tipo $\Sigma_{x:A}B$ é o tipo dos pares *ordinários* (x,y) onde $x : A$ e $y : B$, também chamado de **tipo produto**:

Definição 1.11 ((**??**))**.** Sejam dois tipos A e B. É possível definir o **tipo de produtos (cartesianos)** $A \times B$ de A e B como:

$$A \times B := \Sigma_{x:A}B$$

O tipo produto também satisfaz o principio de indução dos tipos Σ, mas sem a dependência em B, da seguinte forma:

$$ind_{\times} : \Pi_{x:A}\Pi_{y:B}P(x,y) \to (\Pi_{z:A\times B}P(z))$$

Os seus mapas projetivos também são definidos de forma similar aos mapas do tipo Σ. Usando a interpretação de proposições como tipos, $A \times B$ é a *conjunção*.