

## 4

### Estudo sobre o conteúdo computacional das regras do $\perp$

Pelo teorema (3.4), para toda prova de uma fórmula da forma  $\forall x_1 \dots \forall x_n \exists y \alpha(x_1, \dots, x_n, y)$  pode-se extrair um programa que é a solução para o problema enunciado pela fórmula. Neste teorema, não foi considerada a regra de introdução da negação, pois apesar dela ser uma aplicação direta da regra do  $\perp$  intuicionista, não se sabia se esta regra possuía conteúdo computacional ou não. A questão da necessidade ou não da regra do absurdo ter conteúdo computacional é discutida neste capítulo (seção 4.1), no contexto da aritmética de Heyting, que além do conjunto de regras de inferência utilizado no sistema sobre o qual o teorema (3.4) foi formalizado é inserida a regra  $\omega$  computacional. Foi obtida uma resposta parcial para esta questão, que afirma que para qualquer fórmula da forma acima referenciada, é possível obter uma prova, a partir da qual um programa é gerado, na qual a regra do  $\perp$  sempre apresenta conteúdo lógico. Após esse resultado, foi provado que toda prova<sup>1</sup> realizada com a regra  $\omega$  pode ser obtida utilizando apenas a regra de inferência da indução finitária, de forma que ambas não possuem a regra do  $\perp$  com conteúdo computacional (seção 4.2).

Na seção seguinte será apresentada a primeira parte deste resultado, onde é realizado um estudo sobre a possibilidade de sempre se obter, para qualquer teorema da forma  $\forall x_1 \dots \forall x_n \exists y \alpha(x_1, \dots, x_n, y)$ , provas onde a regra do  $\perp$  apresenta sempre conteúdo lógico. Isto é realizado a partir de conceitos de computabilidade [49] (funções recursivas) e a utilização da regra  $\omega$ .

---

<sup>1</sup>No contexto da aritmética de Heyting com a propriedade de reflexividade

## 4.1

### Conteúdo computacional da regra do $\perp$

Pelo teorema (3.4), para toda prova de uma fórmula da forma  $\forall x_1 \dots \forall x_n \exists y \alpha(x_1, \dots, x_n, y)$  pode-se extrair um programa que é uma solução para o problema enunciado pela fórmula. Esse teorema possui a restrição sobre a regra de introdução da negação. De acordo com o isomorfismo *Curry-Howard* o  $\perp$  só possui conteúdo lógico; mas será que a introdução da negação possui também somente conteúdo lógico? Se tivesse conteúdo computacional qual seria?

Nesta seção, será apresentado um resultado parcial para estas questões, que é expresso pelo seguinte teorema:

**Teorema 4.1** *Seja  $f$  uma função recursiva (total) de  $\mathcal{N}^n$  em  $\mathcal{N}$ . Considere DNHA como a representação do sistema de HA da aritmética intuicionista e que usa como sistema dedutivo a Dedução Natural. Existe uma prova  $\Pi$  da fórmula  $\forall x_1 \dots \forall x_n \exists y \alpha(x_1, \dots, x_n, y)$  em DNHA, tal que, nenhuma aplicação da regra do  $\perp$  tem conteúdo computacional associado, e  $\alpha(x_1, \dots, x_n, y)$  representa  $f$ .*

Sabendo que todo programa é uma função recursiva, que as fórmulas que descrevem o problema são da forma  $\forall x_1 \dots \forall x_n \exists y \alpha(x_1, \dots, x_n, y)$  e que pelo isomorfismo Curry-Howard [13] as entradas e saídas do programa gerado estão relacionadas, respectivamente, com o quantificador universal e existencial, conclui-se que o programa gerado sempre pára e gera a saída correspondente à dada entrada ( não se pode esquecer que isso ocorre se a entrada respeitar o tipo de dados especificado como entrada do problema). Logo, o programa gerado está relacionado com as funções recursivas totais.

Com base nesse fato, na prova do teorema 4.1 é realizada uma análise no sentido inverso ao de síntese de programas: será relacionada cada função recursiva (programa) à prova da fórmula que a descreve.

A prova é realizada por indução na definição de funções recursivas totais e utiliza os conceitos de ser representável numericamente (ver [34] ou [41]) bem como a interpretação dada pelo isomorfismo Curry-Howard (ver [26] e [13]). Neste trabalho será utilizada a definição de função primitiva recursiva descrita em [41].



### 4.1.1

#### Prova do Teorema - Funções Primitivas Recursivas

Segue a apresentação da primeira parte da prova do teorema 4.1

##### *Funções Básicas*

**Sucessor** A função  $suc : \mathcal{N} \rightarrow \mathcal{N}$  é representada pelo axioma  $\forall x \exists y (suc(x) = y)$  em  $HA$ , e não existe razão para se ter um sistema computacional que implemente esta função utilizando a negação com conteúdo computacional. Neste trabalho, é assumido que existe uma máquina que implementa este sistema baseado nos axiomas de  $HA$ .

**Zero** Pode-se observar que a prova abaixo, da fórmula que representa a função Zero:  $Zero(x) = 0$ , não possui as regras do  $\perp$ .

$$\frac{\frac{0 = 0}{\exists y (y = 0)} \exists_I}{\forall x \exists y (y = 0)} \forall_I$$

**Projeção** A fórmula que representa a função de projeção,  $\prod(x_1, \dots, x_n) = x_i$  ( $i = 1, n$ ), possui a seguinte prova que não contém a regra de inferência do  $\perp$ .

$$\frac{\frac{\frac{\forall x_i (x_i = x_i)}{x_i = x_i} \forall_E}{\exists y (y = x_i)} \exists_I}{\forall x_1 \dots \forall x_n \exists y (y = x_i)} \forall_I$$

##### *Operações Básicas*

**Composição** Este caso é consequência direta da definição 4.2, da hipótese indutiva sobre  $\Pi_\chi$  e  $\Pi_{f_i}$  ( $i = 1, \dots, n$ ), dos fatos 4.1.2 e 4.1.3.

**Recursão Primitiva** Para simplificar apresenta-se apenas o caso em que função( $\varphi$ ) é aplicada a somente uma variável ( $\varphi(0) = q$ ,  $\varphi(suc(y)) = \chi(y, \varphi(y))$ ). Para o caso geral, basta replicar a construção e o raciocínio da função de composição, que aparece no item acima, considerando a composição geral. Tem-se que a prova  $\Pi_\chi$ , que possui como conclusão a fórmula  $\forall x_1 \forall x_2 \exists y \Theta_\chi(x_1, x_2, y)$ , pela hipótese indutiva representa  $\chi$  e possui apenas regras do  $\perp$  com conteúdos lógicos. A prova abaixo representa  $\varphi$  e possui regras do  $\perp$



vários problemas aritméticos, por exemplo, a implementação da função de Ackerman. Como este tipo de comando está relacionado com a função de minimização, será apresentado, na seção seguinte, a prova de que a fórmula que representa a função de minimização não possui a utilização da regra do  $\perp$  com conteúdo computacional.

#### 4.1.2 Minimização

Estendendo a discussão sobre o conteúdo computacional da regra do  $\perp$  - quando esta é utilizada nas provas que representam funções recursivas - será abordada a função de minimização. Considerando a prova para as funções primitivas recursivas presente na seção 4.1, é possível estendê-la para funções recursivas (totais) adicionando à argumentação a prova para a função de minimização.

Nesta prova será utilizado o sistema  $IHA_\omega$ , isto é, a aritmética intuicionista de Heyting ( $IHA$ ) com a regra  $\omega$  ao invés da regra da indução finita. Uma das razões para se utilizar a regra  $\omega$  é saber que  $IHA_\omega$  possui normalização, o que não ocorre em  $HA$  com a regra de indução [41]. Só é possível atribuir conteúdo computacional a provas normalizadas, dado que as saídas do programa (testemunhas fornecidas pelas regras de introdução existencial) são obtidas através do processo de normalização.

É necessário restringir a regra  $\omega$  para adicionar conteúdo computacional, podendo ser citado [30] como primeiro trabalho que trata a regra  $\omega$  com restrição. Não se pode deixar de mencionar [37]; e outros trabalhos que também utilizam a regra  $\omega$  computacionalmente como: [16] e [18].

#### Regra $\omega$ computacional

Será utilizada uma versão da regra  $\omega$  similar a encontrada em [18]. Ao invés de definir a regra  $\omega$  em sua forma original, será definida a regra  $\omega$  para o caso em que sua aplicação (instância) é computável.

**Definição 4.3** *Seja*

$$\frac{\begin{array}{ccccc} \Pi_0 & \Pi_1 & & \Pi_n & \\ \alpha(0) & \alpha(1) & \dots & \alpha(n) & \dots \end{array}}{\forall x \alpha(x)} \text{ regraw}$$

uma instância da regra  $\omega$ .

Pode-se afirmar que uma instância desta regra é computável se, e somente se, existe uma função (total) computável  $f$  tal que retorna  $\Pi_n$  para todo  $n$ .

Uma instância computável da regra  $\omega$  pode ser codificada de forma finita, e uma possível codificação é  $f : \omega$ , i.e.,  $f$  é uma função que computa as premissas que são utilizadas como código da regra. No processo de prova será utilizada uma codificação para a função  $f$ , logo, o procedimento de normalização no subsistema  $HA_\omega^{comp}$  ( $HA$  com a regra  $\omega$ -computacional) é garantido pelas aplicações das funções e composições.

**Conteúdo computacional da regra  $\omega$  computacional**

Se existe uma prova normal para a fórmula  $\forall x \alpha(x)$  e se essa prova terminar com uma aplicação da regra  $\omega$  computacional, então essa prova carrega em si o código da função  $f$  que gera cada premissa para a aplicação desta regra. Sabendo que o conteúdo computacional do quantificador universal é a atribuição de um valor a uma determinada posição de memória (ver 2.1.1 e 3.3) o comando <sup>4</sup> que reflete o conteúdo computacional da regra  $\omega$  computacional é:  $read(x); y := exec(exec(f, \rightarrow x), x); write(y); end$ .

**Prova da minimização**

Seja  $\chi : \mathcal{N} \times \mathcal{N} \rightarrow \mathcal{N}$  e sua minimização:  $\varphi : \mathcal{N} \rightarrow \mathcal{N}$ ,  $\varphi(x) = \mu y (\chi(x, y) = 1)$ . Pela hipótese indutiva, a prova de  $\Pi_\chi$  está relacionada com a fórmula  $\forall x_1 \forall x_2 \exists y \Theta_\chi$ . Sabendo que, pela hipótese do teorema  $\varphi$  é uma função total, que por definição:  $\Theta_\varphi(a, b) \equiv \Theta_\chi(a, b, 1) \wedge \forall z (z < b \rightarrow \neg \Theta(a, z, 1))$  e considerando a função  $\Upsilon_\chi$ , tal que para cada  $a \in \mathcal{N}$ , produz uma prova em  $HA_\omega^{comp}$ , tem-se que existe um  $b \in \mathcal{N}$  tal que  $\varphi(a) = b$ .

<sup>4</sup>Para facilitar o entendimento foi associado a aplicação da regra de introdução do quantificador universal o comando *read*.

Segue abaixo a prova de  $\Upsilon_\chi(a)^5$

$$\begin{array}{c}
 \left( \frac{\frac{\Pi_\chi}{\forall x_1 \forall x_2 \exists y \Theta_\chi(x_1, x_2, y)}}{\exists y \Theta_\chi(a, b, y)} \right) \frac{\left\{ \frac{\left( \frac{\frac{\Pi_\chi}{\forall x_1 \forall x_2 \exists y \Theta_\chi(x_1, x_2, y)}}{\exists y \Theta_\chi(a, z, y)} \right) [\Theta_\chi(a, z, 1)]}{\frac{c_z = 1}{\frac{\perp}{\neg \Theta_\chi(a, z, 1)}}} \frac{\Theta_\chi(a, z, c_z)}{Inj} \neg c_z = 1 \right\}}{\forall z (z < b \rightarrow \neg \Theta_\chi(a, z, 1))} \frac{\Theta_\chi(a, b, 1)}{z < b} \\
 \hline
 \frac{\Theta_\chi(a, b, 1)}{\Theta_{\varphi(a, b)}} Def
 \end{array}$$

A prova da fórmula  $\forall x \exists y \Theta_\varphi(x, y)$  que representa a minimização é apresentada a seguir:

$$\frac{\frac{\Upsilon_\chi(0)}{\Theta_\varphi(0, b_0)} \quad \frac{\Upsilon_\chi(1)}{\Theta_\varphi(1, b_1)} \quad \dots \quad \frac{\Upsilon_\chi(n)}{\Theta_\varphi(n, b_n)}}{\exists y \Theta_\varphi(0, y) \quad \exists y \Theta_\varphi(1, y) \quad \dots \quad \exists y \Theta_\varphi(n, y)} \omega - rule \\
 \hline
 \forall x \exists y \Theta_\varphi(x, y)$$

e o conteúdo computacional associado a esta prova é:

$\Upsilon_\chi(a)$  representa a função  $f$  que computa as premissas que são utilizadas como código da regra  $\omega$  computacional.

```

read(a);
b := 0;
while y <> 1 do
  < a, k, z > =  $\chi(a, b, y)$   b e y são variáveis auxiliares que recebem os valores de
  b := k
  y := z
od
write(b);
b e y respectivamente.

```

#### Observação 4.1.4

Pode-se observar que, na prova relacionada com a função de minimização, existe apenas uma utilização da regra do  $\perp$  e que ela, explicitamente, só possui conteúdo lógico. Em um programa, ele funciona como um guarda do sistema que verifica se a condição de parada do *loop* é verdadeira ou falsa. Logo, para todo teorema  $\phi$ , em  $HA_\omega$ , existe uma prova em  $DNHA$  que não possui a utilização da regra do  $\perp$  com conteúdo computacional.

Nesta seção é apresentado um resultado que possibilita a construção de um processo de síntese de programas, no qual um programa imperativo é gerado a partir de uma prova<sup>6</sup> realizada por um provador de teoremas<sup>7</sup> que, além das regras de inferência usuais, possui a implementação para a regra da indução, para a regra da igualdade e para a regra  $\omega$  computacional. Além da prova de correção, foi obtida a completude deste processo de síntese de programas.

Considerando que nem todos os provadores de teorema possuem uma implementação para a regra  $\omega$  computacional, é interessante saber se é possível expressar as provas realizadas utilizando regra  $\omega$  computacional, na qual as regras do  $\perp$  não possuem conteúdo computacional, utilizando indução finita de forma que estas também não tenham a regra do  $\perp$  com conteúdo computacional. Este resultado é apresentado a seguir.

## 4.2

### Relação entre a Regra $\omega$ computacional e a Indução Finita

A partir do teorema de normalização para  $HA_\omega$  [40], foi apresentado na seção anterior como extrair o conteúdo computacional de provas, que não possuem regras do  $\perp$  com conteúdo computacional e nas quais as regras  $\omega$  são restritas a sua versão computacional. Entretanto, pode-se querer trabalhar com

<sup>6</sup>De um teorema da forma  $\forall x_1 \dots \forall x_n \exists y \alpha(x_1, \dots, x_n, y)$  e que não possui a regra do  $\perp$  com conteúdo computacional.

<sup>7</sup>Para lógica intuicionista que utiliza como sistema dedutivo a dedução natural[31].

provas baseada em induções finitas ao invés da utilização da regra  $\omega$  computacional.

Nesta seção, é apresentada a prova de que todas as provas - no contexto da aritmética de Heyting com a propriedade de reflexividade - realizadas em um sistema que possui regra  $\omega$  computacional podem ser efetuadas em sistemas que possuem somente a indução finita, sendo que ambas não utilizam da regra do  $\perp$  com conteúdo computacional

#### 4.2.1

#### Conteúdo Computacional: Regra $\omega$ computacional X Indução Simples

Abaixo segue uma definição e o teorema que expressa o resultado a ser apresentado nesta seção:

**Definição 4.4** Uma prova  $\Pi$  em  $HA_{\omega}^{comp}$  possui propriedade de reflexividade se, e somente se, para toda premissa  $\alpha$  de toda aplicação da regra  $\omega$  computacional em  $\Pi$ :  $\vdash_{HA} Bew(\ulcorner \alpha^b \urcorner) \rightarrow \alpha^b$ , onde  $\alpha^1 = \alpha$  e  $\alpha^b = Bew(\ulcorner \alpha^{b-1} \urcorner)$ .

**Teorema 4.5** Seja  $\Pi$  uma prova da fórmula  $\alpha$  na  $HA_{\omega}^{comp}$ , tal que:

- ◇  $\Pi$  tenha a propriedade de reflexividade e,
  - ◇  $\Pi$  não contenha regras do  $\perp$  com conteúdo computacional
- então, existe uma prova  $\Pi'$  de  $\alpha$  em  $HA$ , onde as regras do  $\perp$  também não possuem conteúdo computacional.

#### Conceitos básicos

Aqui serão introduzidos alguns teoremas e definições que são necessários para a compreensão da prova do teorema 4.5.

**Definição 4.6** Considera-se a definição usual do predicado de provabilidade presente em [46](ver capítulo 2):

$\forall y (Bew(y) \longleftrightarrow \exists x Pr(x, y))$ , onde  $Pr(x, y)$  é um predicado primitivo recursivo que é válido quando  $x$  for uma prova de  $y$ .

**Teorema 4.7** Teorema de Löb:

$\vdash_{HA} Bew(\ulcorner \alpha \urcorner) \rightarrow \alpha$  se, e somente se,  $\vdash_{HA} \alpha$ , onde  $\ulcorner \alpha \urcorner$  é o número de Gödel para a fórmula  $\alpha$ . [44]

**Definição 4.8**  $\vdash^k \alpha$  se  $\exists \Pi$  tal que:  $|\Pi| = k$ , onde  $k$  é o tamanho de  $\Pi$ , calculado pelo número de linhas da prova e  $\Pi$  é uma prova de  $\alpha$ .

**Observação 4.2.1** *Afirma-se que  $\frac{k}{HA} \vdash Bew(\ulcorner \alpha \urcorner) \rightarrow \alpha$  se, e somente se,  $\frac{k}{HA} \vdash \alpha$ , pois de acordo com a prova do teorema 4.7 apresentada em [[46] p.57], pode-se calcular o tamanho da prova, que é constante.*

**Teorema 4.9** *Parik*<sup>8</sup>:

$\frac{}{PA} \vdash \forall x \alpha(x)$  se, e somente se, existe um  $k$  tal que para todo  $n$   $\frac{k}{PA} \vdash \alpha(n)$  [11]

**Observação 4.2.2** *Observe que o teorema 4.9 é válido para HA.*

## 4.2.2

### Prova do Teorema 4.5

Esta prova tem como objetivo demonstrar que qualquer prova em  $HA_{\omega}$ , que não utiliza a regra do  $\perp$  com conteúdo computacional, pode ser transformada em uma prova em  $HA$  com uma indução finita, que também não utiliza a regra do  $\perp$  com conteúdo computacional.

*Prova.* Considere a seguinte aplicação da regra  $\omega$  computacional:

$$\frac{\begin{array}{ccccccc} \Pi_0 & & \Pi_1 & & & & \Pi_n \\ \alpha(0) & & \alpha(1) & & \dots & & \alpha(n) & & \dots \end{array}}{\forall x \alpha(x)} = \Pi$$

onde  $f(i) = \Pi_i$ ,  $i \in N$  e  $f$  é uma função recursiva.

Nenhuma regra do  $\perp$  em  $\Pi_i$  possui conteúdo computacional e não há nenhuma outra aplicação da regra  $\omega_c$  (Regra  $\omega$  computacional) em qualquer outro  $\Pi_i$ .

Como  $f$  é uma função recursiva(total), pelo teorema da representação[41](ou prova de correção para  $f$ ) tem-se que:

$$\frac{}{HA} \vdash \forall n Pr(f(n), \ulcorner \alpha(n) \urcorner).$$

$$\text{Assim, } \frac{}{HA} \vdash \exists p (Pr(p, \ulcorner \forall n Pr(f(n), \ulcorner \alpha(n) \urcorner) \urcorner)).$$

Sabendo que  $|p|$  representa o tamanho da prova:

$$\forall n \exists p_n \frac{|p|+1}{HA} \vdash Pr(p_n, \ulcorner Pr(f(n), \ulcorner \alpha(n) \urcorner) \urcorner).$$

Dado que  $k = p + 1$  e aplicando a definição 4.6:  $\exists k \forall n \frac{k}{HA} \vdash Bew(\ulcorner Bew(\ulcorner \alpha(n) \urcorner) \urcorner)$

---

<sup>8</sup>ver prova em Apêndice A.

Sabendo que  $\Pi$  tem propriedade reflexiva (definição 4.4), e utilizando o teorema 4.7:

$$\exists k \forall n \stackrel{k}{\vdash}_{HA} \alpha(n) \quad (1)$$

Utilizando uma direção do teorema [11]: Se existe um  $k$  tal que,  $\forall n \stackrel{k}{\vdash}_{HA} \alpha(n)$  então  $\stackrel{k}{\vdash}_{HA} \forall x \alpha(x)$  e aplicando este resultado sobre (1):

$$\stackrel{k}{\vdash}_{HA} \forall n \alpha(n)$$

Pode-se observar na prova do teorema [11], presente no apêndice, que em nenhum momento da prova - sobre a existência de uma relação entre a regra  $\omega$  computacional (na prova  $\Pi$ ) e a indução simples (na prova  $\Pi'$ ) - houve a utilização da negação, não havendo assim a inserção de novas regras do  $\perp$ .

Logo, pode-se concluir que a prova resultante,  $\Pi'$  de  $\alpha$  em  $HA$ , não possui a regra do  $\perp$  com conteúdo computacional.

□

Com estes resultados, tem-se a prova da correção e da completude do processo de síntese de programas, que gera um programa imperativo a partir de uma prova de um teorema<sup>9</sup> que representa uma função total realizada por um provador de teoremas<sup>10</sup> que, além das regras de inferência usuais, possui a implementação das regras da indução, da igualdade e regra  $\omega$ .

Também foi observado que é possível construir um processo de síntese de programas em um sistema semelhante ao anteriormente citado - no contexto da aritmética de Heyting com a propriedade de reflexividade e com a diferenciação da não implementação da regra  $\omega$ . No entanto, essa possibilidade depende se a prova com indução finita for normal, pois, no caso contrário, é preciso determinar mecanismos para a extração de conteúdos computacionais de provas não normais. Esses tópicos serão abordados em trabalhos futuros.

<sup>9</sup>Da forma  $\forall x_1 \dots \forall x_n \exists y \alpha(x_1, \dots, x_n, y)$ .

<sup>10</sup>Para lógica intuicionista que utiliza como sistema dedutiva a dedução natural.