4
Estudo sobre o contéudo computacional das regras do $\perp$

Pelo teorema (3.4), para toda prova de uma fórmula da forma $\forall x_1 \ldots \forall x_n \exists y \alpha(x_1, \ldots, x_n, y)$ 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 possuia 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 arimética 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 ω 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[1] realizada com a regra ω 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 $\forall x_1 \ldots \forall x_n \exists y \alpha(x_1, \ldots, x_n, y)$, 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 ω.

[1] 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 $\forall x_1 \ldots \forall x_n \exists y \alpha(x_1, \ldots, x_n, y)$ 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 Π da fórmula $\forall x_1 \ldots \forall x_n \exists y \alpha(x_1, \ldots, x_n, y)$ em $DNHA$, tal que, nenhuma aplicação da regra do $\perp$ tem conteúdo computacional associado, e $\alpha(x_1, \ldots, x_n, y)$ representa f.*

Sabendo que todo programa é uma função recursiva, que as fórmulas que descrevem o problema são da forma $\forall x_1 \ldots \forall x_n \exists y \alpha(x_1, \ldots, x_n, y)$ 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].

A prova é dividida em dois passos: o primeiro será provar o teorema por indução sobre funções primitivas recursivas f, enquanto no segundo passo este resultado será extendido para funções recursivas através da prova do teorema para a função de minimização.

A seguir serão apresentadas algumas definições úteis na prova do teorema 4.1.

Definição 4.2 *Dado o predicado Theta e as funções* $\chi : \mathcal{N}^m \to \mathcal{N}$ *e* $f_i : \mathcal{N}^n \to \mathcal{N}$ *(*$i = 1, ..., m$*), tal que,* Π_χ *é uma prova para a fórmula* $\forall x_1 \ldots \forall x_m \exists y \Theta_\chi(x_1, \ldots, x_m, y)$*, e para cada* $i \in [1, m]$*,* Π_{f_i} *prova* $\forall x_1 \ldots \forall x_n \exists y \Theta_{f_i}(x_1, \ldots, x_n, y)$ *em alguma teoria* T*.* $\bigcirc_n^m$ *é uma operação que gera a prova* Π_χ *a partir das provas* Π_{f_i}*.*

Observação 4.1.1 *Será utilizado a abreviação* $\forall \vec{z}$ *para simbolizar* $\forall z_1, \ldots, \forall z_k$ *(para qualquer* k*) e* $\vec{a}$ *para* $a_1, \ldots, a_m$ *(para qualquer* m*).*

$$\dfrac{\dfrac{\Pi_\chi}{\dfrac{\forall \vec{x} \exists y \Theta_\chi(\vec{x}, y)}{\exists y \Theta_\chi(b_1, \ldots, b_m, y)} \forall_E} \quad \dfrac{\dfrac{\dfrac{[\Theta_{f_1}(\vec{a}, b_1)]^1 \ldots [\Theta_{f_m}(\vec{a}, b_m)]^m}{\vdots \wedge_I}}{\bigwedge_{i=1,m} \Theta_{f_i}(\vec{a}, b_i)} \quad [\Theta_\chi(b_1, \ldots, b_m, c)]^{m+1}}{\bigwedge_{i=1,m} \Theta_{f_i}(\vec{a}, b_i) \wedge \Theta_\chi(b_1, \ldots, b_m, c)} \wedge_I}{\exists y(\exists y_1 \ldots \exists y_m \bigwedge_{i=1,m} \Theta_{f_i}(\vec{a}, y_i) \wedge \Theta_\chi(y_1, \ldots, y_m, y))} \exists_I$$

$$\dfrac{\exists y(\exists y_1 \ldots \exists y_m \bigwedge_{i=1,m} \Theta_{f_i}(\vec{a}, y_i) \wedge \Theta_\chi(y_1, \ldots, y_m, y))}{\vdots \exists_E^{2,\ldots,m}} \exists_E^{m+1}$$

$$\exists y(\exists y_1 \ldots \exists y_m \bigwedge_{i=1,m} \Theta_{f_i}(\vec{a}, y_i) \wedge \Theta_\chi(y_1, \ldots, y_m, y))$$

$$\dfrac{\left(\dfrac{\Pi_{f_i}}{\dfrac{\forall \vec{x} \exists y \Theta_{f_i}(\vec{x}, y)}{\exists y \Theta_{f_i}(\vec{a}, y)} \forall_E}\right)_{i=1,m}}{\exists y(\exists y_1 \ldots \exists y_m \bigwedge_{i=1,m} \Theta_{f_i}(\vec{a}, y_i) \wedge \Theta_\chi(y_1, \ldots, y_m, y))} \exists_E^1$$

$$\dfrac{\exists y(\exists y_1 \ldots \exists y_m \bigwedge_{i=1,m} \Theta_{f_i}(\vec{a}, y_i) \wedge \Theta_\chi(y_1, \ldots, y_m, y))}{\forall \vec{x} \exists y(\exists y_1 \ldots \exists y_m \bigwedge_{i=1,m} \Theta_{f_i}(\vec{x}, y_i) \wedge \Theta_\chi(y_1, \ldots, y_m, y))} \forall_I$$

Fato 4.1.2 *Na definição acima, se* Θ_χ *e* Θ_{f_i} *representam* χ *e os* f_i*'s respectivamente em DNHA, pelo significado de* Π_χ *e* Π_{f_i}*,* $\bigcirc_n^m(\Pi_\chi, \Pi_{f_1}, \ldots, \Pi_{f_m})$ *é uma prova que garante que* $\lambda x_1 ... \lambda x_n \chi(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$ *é representável em DNHA.*

Fato 4.1.3 *Se as provas* Π_χ *e* Π_{f_i} *possuem somente utilização lógica das regras do* $\perp$*, então* $\bigcirc_n^m(\Pi_\chi, \Pi_{f_1}, \ldots, \Pi_{f_m})$ *também só possui a utilização lógica desta regra.*

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 $\perp$.

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

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

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

Operações Básicas

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

Recursão Primitiva Para simplicar apresenta-se apenas o caso em que função(φ) 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 Π_χ, 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 χ e possui apenas regras do $\perp$ com conteúdos lógicos. A prova abaixo representa φ e possui regras do $\perp$

com conteúdos lógicos.
As fórmulas que são utilizadas para representar φ (Θ_φ) são dadas pela seguinte definição recursiva (Def^2) :
$(CasoBase:)\Theta_\varphi(0,y) \equiv y = q$ e
$(CasoIndutivo)\Theta_\varphi(suc(x),y) \equiv \exists z(\Theta_\varphi(x,z) \wedge \Theta_\chi(suc(x),z,y))$.

$$\dfrac{\dfrac{\forall x(x=x)}{q=q}\forall_E}{\dfrac{\exists y(y=q)}{\forall x\exists y\Theta_\varphi(x,y)}\exists_I}\quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\Theta_\varphi(x,b)]^2\ [\Theta_\chi(suc(x),b,c)]^1}{\Theta_\varphi(x,b)\wedge\Theta_\chi(suc(x),b,c)}\wedge I}{\exists z(\Theta_\varphi(x,z)\wedge\Theta_\chi(suc(x),z,c))}Def}{\dfrac{\Theta_\varphi(suc(x),c)}{\exists y\Theta_\varphi(suc(x),y)}\exists_I}\exists_E(1)}{\exists y\Theta_\varphi(suc(x),y)}\exists_E(2)}{\exists y\Theta_\varphi(suc(x),y)}Ind(3)$$

Desta forma, foi obtida a prova da representação das funções primitivas recursivas em $DNHA$ (similar as encontradas em [34] e [41]). Seguindo o processo de síntese de programas estabelecido nos capítulos 2 e 3, o conteúdo computacional da prova acima é um programa iterativo gerado por um esquema recursivo ([26] e [13]). Assim, a partir de raciocínios indutivos sobre o conteúdo computacional (programa), dado que uma prova de $P(0) \equiv \exists y = q$ não possui a utilização da regra do $\perp$ e pela hipótese que o conteúdo computacional da prova hipotética de $\exists y\Theta_\varphi(x,y)$ possui apenas a regra do $\perp$ com conteúdo lógico, pode-se concluir que a prova acima possui apenas regra do $\perp$ com conteúdo lógico.

□

Assim, foi concluído o primeiro passo da prova do teorema 4.1, onde foi demostrado que não há a necessidade da regra do $\perp$ com conteúdo computacional para as provas de propriedades associadas as funções primitivas recursivas.

As linguagens de programação imperativas possuem comandos relacionados com *loops* "indefinidos"[3] que são utilizados em implementações de

[2]Esta definição é apenas uma forma de compactar e descompactar Θ_φ.
[3]Isto é, sem um número determinado de passos.

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_ω, isto é, a aritmética intuicionista de Heyting (IHA) com a regra ω ao invés da regra da indução finita. Uma das razões para se utilizar a regra ω é saber que IHA_ω 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 existêncial) são obtidas através do processo de normalização.

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

Regra ω computacional

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

Definição 4.3 *Seja*

$$\frac{\begin{array}{cccccc}\Pi_0 & \Pi_1 & & \Pi_n & \\ \alpha(0) & \alpha(1) & \ldots & \alpha(n) & \ldots\end{array}}{\forall x \alpha(x)}\, regra\omega$$

uma instância da regra ω.
Pode-se afirmar que uma instância desta regra é computável se, e somente se, existe uma função(total) computável f *tal que retorna* Π_n *para todo* n.

Uma instância computável da regra ω 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^{comp}_{ω}(HA com a regra ω-computacional) é garantido pelas aplicações das funções e composições.

Conteúdo computacional da regra ω computacional

Se existe uma prova normal para a fórmula $\forall x \alpha(x)$ e se essa prova terminar com uma aplicação da regra ω 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 [4] que reflete o conteúdo computacional da regra ω 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 Π_χ está relacionada com a fórmula $\forall x_1 \forall x_2 \exists y \Theta_\chi$. Sabendo que, pela hipótese do teorema φ 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 Υ_χ, tal que para cada $a \in \mathcal{N}$, produz uma prova em HA^{comp}_{ω}, tem-se que existe um $b \in \mathcal{N}$ tal que $\varphi(a) = b$.

[4] 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]

$$\dfrac{\left(\begin{array}{c}\Pi_\chi\\ \dfrac{\forall x_1 \forall x_2 \exists y \Theta_\chi(x_1,x_2,y)}{\exists y \Theta_\chi(a,b,y)}\end{array}\right)_{\Theta_\chi(a,b,1)} \qquad \left\{\dfrac{\dfrac{\left(\begin{array}{c}\Pi_\chi\\ \dfrac{\forall x_1 \forall x_2 \exists y \Theta_\chi(x_1,x_2,y)}{\exists y \Theta_\chi(a,z,y)}\end{array}\right)_{\Theta_\chi(a,z,c_z)} \quad [\Theta_\chi(a,z,1)]}{\dfrac{c_z = 1 \qquad \neg c_z = 1}{\dfrac{\bot}{\neg\Theta_\chi(a,z,1)}}\, Inj}}{\forall z(z < b \rightarrow \neg\Theta_\chi(a,z,1))}\, z{\leq}b\right\}}{\Theta_{\varphi(a,b)}}\, Def$$

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

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

e o conteúdo computacional associado a esta prova é:

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

read(a);
$b := 0$;
while $y <> 1$ do
$< a, k, z > = \chi(a, b, y)$
$b := k$
$y := z$
od
write(b);

$\dot{b}$ e $\dot{y}$ são variáveis auxiliares que recebem os valores de 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 ϕ, em HA_ω, 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[6] realizada por um provador de teoremas [7] que, além das regras de inferência usuais, possui a implementação para a regra da indução, para a regra da igualdade e para a regra ω 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 ω computacional, é interessante saber se é possível expressar as provas realizadas utilizando regra ω 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 ω computacional e a Indução Finita

A partir do teorema de normalização para HA_ω [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 ω são restritas a sua versão computacional. Entretanto, pode-se querer trabalhar com

[6] De um teorema da forma $\forall x_1 \ldots \forall x_n \exists y \alpha(x_1, \ldots, x_n, y)$ e que não possui a regra do $\perp$ com conteúdo computacional.

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

provas baseada em induções finitas ao invés da utilização da regra ω 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 ω 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 ω 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* Π *em* HA_{ω}^{comp} *possui propriedade de reflexividade se, e somente se, para toda premissa* α *de toda aplicação da regra* ω *computacional em* Π*:* $\underset{HA}{\vdash} Bew(\ulcorner\alpha^b\urcorner) \rightarrow \alpha^b$*, onde* $\alpha^1 = \alpha$ *e* $\alpha^b = Bew(\ulcorner\alpha^{b-1}\urcorner)$*.*

Teorema 4.5 *Seja* Π *uma prova da fórmula* α *na* HA_{ω}^{comp}*, tal que:*
$\diamond$ Π *tenha a propriedade de reflexividade e,*
$\diamond$ Π *não contenha regras do* $\perp$ *com conteúdo computacional*
então, existe uma prova Π' *de* α *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:*
$\underset{HA}{\vdash} Bew(\ulcorner\alpha\urcorner) \rightarrow \alpha$ *se, e somente se,* $\underset{HA}{\vdash} \alpha$*, onde* $\ulcorner\alpha\urcorner$ *é o número de Gödel para a fórmula* α*. [44]*

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

Observação 4.2.1 *Afirma-se que* $\underset{HA}{\overset{k}{\vdash}} Bew(\ulcorner\alpha\urcorner) \rightarrow \alpha$ *se, e somente se,* $\underset{HA}{\overset{k}{\vdash}} \alpha$, *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* [8]*:*
$\underset{PA}{\vdash} \forall x\alpha(x)$ *se, e somente se, existe um* k *tal que para todo* n $\underset{PA}{\overset{k}{\vdash}} \alpha(n)$ *[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_{ϖ}, 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 ω computacional:

$$\dfrac{\begin{array}{cccccc}\Pi_0 & \Pi_1 & & \Pi_n & \\ \alpha(0) & \alpha(1) & \ldots & \alpha(n) & \ldots\end{array}}{\forall x\alpha(x)} = \Pi$$

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

Nenhuma regra do $\perp$ em Π_i possui conteúdo computacional e não há nenhuma outra aplicação da regra ω_c(Regra ω computacional) em qualquer outro Π_i.

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

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

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

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

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

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

[8] ver prova em Apêndice A.

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

$\exists k \forall n \underset{HA}{\overset{k}{\vdash}} \alpha(n)$ (1)

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

$\underset{HA}{\vdash} \forall n \alpha(n)$

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 ω computacional (na prova Π) e a indução simples (na prova Π') - houve a utilização da negação, não havendo assim a inserção de novas regras do $\perp$.
Logo, pode-se concluir que a prova resultante, Π' de α 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[9] que representa uma função total realizada por um provador de teoremas[10] que, além das regras de inferência usuais, possui a implementação das regras da indução, da igualdade e regra ω.

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 ω. 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.

[9] Da forma $\forall x_1 \ldots \forall x_n \exists y \alpha(x_1, \ldots, x_n, y)$.
[10] Para lógica intuicionista que utiliza como sistema dedutiva a dedução natural.