2
Conceitos Básicos

Este capítulo aborda os conceitos básicos necessários para a compreensão deste trabalho e foi dividido em duas seções a primeira que traz um resumo do trabalho sobre síntese construtiva de programas a partir do qual se originou essa tese [47], e uma breve explicação sobre a definição do predicado de provabilidade. Na seção seguinte é apresentado um resumo sobre o processo de síntese construtiva de programas proposto, no qual um programa é gerado a partir de um provador de teoremas em lógica de predicados intuicionista poli-sortida para aritmética, que além das usuais regras de inferência em dedução natural, possui as regras de indução finita e igualdade.

2.1
Síntese Construtiva de Programas

A síntese construtiva de programas é um método de especificação formal fundamentada no Isomorfismo Curry-Howard, no qual se afirma a existência de uma função injetora entre uma prova construtiva de um teorema e um programa (função) que retorna, como saída, uma testemunha para as variáveis quantificadas com o existencial [13]. Esta síntese, inicialmente aplicada sobre o paradigma de programação funcional, é conhecida como: prova como programa (*proof-as-program*)[15].

O processo de síntese construtiva de programas, que se baseia no processo paralelo de construir e verificar programas, é dividido em dois passos:

Passo 1: Encontrar uma prova construtiva para a existência de uma função total f que mapeie os elementos de entrada nos elementos de saída do programa;

Passo 2: Extrair a função f, a partir do conteúdo computacional da prova, associando cada regra de inferência aplicada no processo de prova a uma regra de construção do programa.

A prova construtiva para a existência da função f é obtida através da prova do teorema que descreve essa função. Este teorema possui a seguinte forma: $\forall x \exists y P(x,y)$.

Sabendo que a função f é total, o teorema pode ser interpretado da seguinte forma: "Pode-se extrair um programa que satisfaça a relação especificada por P, tal que, para toda entrada do tipo x este programa terminará e retornará y como saída".

Como a função f é construída a partir da prova, o programa que a soluciona (função) será construído como se fosse um efeito da própria prova, o que acarreta na adequação do programa alvo com as especificações.

Neste trabalho, como será visto a partir da próxima seção, é abordado o segundo passo do processo de síntese de programas: a extração do conteúdo computacional (geração de programas) e a garantia da correção do processo. É suposta uma prova em lógica intuicionista que utilize a dedução natural como sistema dedutivo.

2.1.1 Processo de Síntese Construtiva de Programas 1

É apresentado um processo de extração do conteúdo computacional de uma prova onde, dada a prova é iniciado um mapeamento de todas as configurações da memória para cada regra de inferência, seguido da associação de cada regra de inferência com comandos em uma linguagem de programação imperativa (semelhante à linguagem PASCAL [10]). Ao término do processo tem-se um programa que expressa a solução do problema. Para garantir que o programa reflete a solução do problema expressa pela prova, é apresentado o teorema para provar a correção do sistema.

Marcação das configurações de memória

Do ponto de vista computacional, a execução de um programa pode ser compreendida como o resultado da movimentação de bits dentro da memória do computador. Essa movimentação é resultante da aplicação de

operações sobre os dados do programa. Logo, pode-se descrever todo o processo computacional a partir das modificações de memória efetuadas durante a computação, que são definidas através das variáveis do programa e dos valores atribuídos a elas.

De acordo com a leitura operacional dos quantificadores, dado pelo isomorfismo de Curry-Howard (C.H.), as variáveis quantificadas universalmente são associadas com os dados de entrada do programa e são representadas por variáveis livres, uma vez que podem aceitar qualquer valor (do mesmo tipo da variável), que torna a fórmula verdadeira. Os dados de saída do programa, que estão associados às variáveis quantificadas existencialmente, são representados em uma fórmula por variáveis livres e termos que dependem das variáveis de entrada.

Portanto, partindo da interpretação da prova, que tem por base o isomorfismo de C.H. em que cada regra de inferência pode ser interpretada como um passo na construção do programa, é realizado o processo de marcação das configurações de memória que calcula a configuração da memória para cada aplicação da regra de inferência.

O processo de marcação das configurações da memória em uma prova gera dois conjuntos: um para as variáveis de entrada (variáveis livres) e outro para os termos de saída. No início do processo de marcação dos dados de entrada e de saída, que é aplicado no sentido ascendente ($bottom - up$), os conjuntos estão vazios.

Quando o processo atinge as folhas das árvores de prova, pode-se encontrar algumas variáveis e termos que pertençam ao conjunto de variáveis livres ou ao conjunto de termos de saída que não são utilizados como entrada e saída do programa associado à prova. Esses resíduos são inseridos no conjunto de variáveis de entrada e no conjunto de termos de saída das fórmulas, que pertencem ao caminho da prova derivado, a partir da fórmula em que eles foram detectados pela primeira vez (este processo será realizado na direção descendente ($top - down$)). Assim, os resíduos se propagam até a raiz da árvore de prova (conclusão) cujos conjuntos de variáveis de entrada e de termos de saída estarão adequadamente instanciados [1].

[1] Não serão mais vazios.

Como já foi dito, a marcação das regras de configuração de memória está relacionada com as aplicações das regras de inferência. Esta relação é apresentada através das regras de marcação das configurações de memória na seção seguinte.

Regras de marcação das configurações de memória

Na apresentação das regras será utilizada a seguinte notação:

▪ α_T^V , onde α é uma fórmula, V um conjunto de variáveis de entrada e T um conjunto de termos de saída;

▪ $K \cup a$ representa a inserção de "a"em K, onde a é uma variável de entrada ou de termo de saída e K é uma lista de variáveis de entrada ou de termos de saída.

Observação 2.1.1 *As regras de marcação devem ser analisadas na direção ascendente (bottom−up), de acordo com o processo de marcação (Veja tabela 2.1).*

Observação 2.1.2 *No processo de marcação das configurações de memória, se uma prova utiliza a propriedade de congruência da igualdade, as fórmulas resultantes terão os conjuntos de variáveis de entrada e de variáveis de saída modificados da seguinte forma: as variáveis ou termos substituídos serão retirados do conjunto a que pertencem e serão adicionados os novos termos e variáveis (dos quais o termo substituto depende) em seus respectivos conjuntos. Observe o seguinte exemplo:*

$$\dfrac{\dfrac{\dfrac{\forall x(1*x=x)^{\{\}}_{\{1\}}}{1*x=x^{\{x\}}_{\{1\}}}E\forall \quad \dfrac{\forall h(h+0=h)^{\{\}}_{\{0\}}}{x+0=x^{\{x\}}_{\{0\}}}E\forall}{1*x+0=x^{\{x\}}_{\{0,l\}}} \quad \dfrac{[b=l]^{\{\}}_{\{b,l\}} \quad \dfrac{\dfrac{\forall z\forall p((z=p)\to(z-p=0))^{\{\}}_{\{0\}}}{\forall p((b=p)\to(b-p=0))^{\{b\}}_{\{0\}}}E\forall}{(b=l)\to(b-l=0)^{\{b,l\}}_{\{0\}}}E\forall}{b-l=0^{\{b,l\}}_{\{0\}}}E\to}{1*x+(b-l)=x^{\{x,b,l\}}_{\{(b-l),l\}}}Eq$$

Apesar de ter sido apresentada a marcação de entradas e saídas para todas as regras, existem duas delas que apresentam restrições na sua utilização, visto que a extração de seus conteúdos computacionais não é trivial.

Fórmulas iniciais

Axioma: β_T^V

Fórmulas iniciais

Hipótese: β_T^V

$\forall$ Eliminação

$$\frac{\forall x \alpha(x)_T^V}{\alpha(h)_T^{V \cup h}}$$

$\forall$ Introdução

$$\frac{\alpha(h)_T^{V \cup h}}{\forall y \alpha(y)_T^V}$$

$\exists$ Eliminação

$$\frac{\exists x \alpha(x)_T^V \qquad \begin{array}{c}\alpha(h)_{T \cup h}^V \\ \vdots \\ \gamma_{T_1}^{V_1}\end{array}}{\gamma_{T_1}^{V_1}}$$

$\exists$ Introdução

$$\frac{\alpha(h)_{T \cup h}^V}{\exists x \alpha(x)_T^V}$$

$\wedge$ Eliminação

$$\frac{(\alpha \wedge \beta)_T^V}{\alpha_T^V} \qquad \frac{(\alpha \wedge \beta)_T^V}{\beta_T^V}$$

$\wedge$ Introdução

$$\frac{\alpha_{T_1}^{V_1} \qquad \beta_{T_2}^{V_2}}{(\alpha \wedge \beta)_{T_1 \cup T_2}^{V_1 \cup V_2}}$$

$\vee$ Eliminação

$$\frac{(\alpha \vee \rho)_{T_1}^{V_1} \qquad \begin{array}{c}\alpha_{T_1}^{V_1} \\ \vdots \\ \gamma_T^V\end{array} \qquad \begin{array}{c}\rho_{T_1}^{V_1} \\ \vdots \\ \gamma_T^V\end{array}}{\gamma_T^V}$$

$\vee$ Introdução

$$\frac{\alpha_T^V}{(\alpha \vee \rho)_T^V} \qquad \frac{\rho_T^V}{(\alpha \vee \rho)_T^V}$$

$\rightarrow$ Eliminação

$$\frac{\alpha_{T_1}^{V_1} \qquad (\alpha \rightarrow \rho)_T^V}{\rho_T^V}$$

$\rightarrow$ Introdução

$$\frac{\begin{array}{c}\alpha_{T_1}^{V_1} \\ \vdots \\ \rho_T^V\end{array}}{(\alpha \rightarrow \rho)_T^V}$$

Indução

$$\frac{(k \prec l)_{T_1}^{V_1} \qquad \begin{array}{c}\alpha(a_i)_{T_2}^{V_2} \\ \vdots \\ \alpha(w)_T^{V \cup w}\end{array} \qquad (a_i \prec w)_{T_3}^{V_3}}{\forall y \alpha(y)_T^V}$$

onde k reflete o termo associado com o caso base e ω reflete o termo associado com o passo indutivo.

Regra do absurdo intuicionista

$$\frac{\dfrac{\gamma_{T_1}^{V_1} \qquad \neg\gamma_{T_2}^{V_2}}{\bot}}{\alpha_T^V}$$

Tabela 2.1: Regras de marcação das configurações de memória.

Restrições sobre as regras utilizadas na prova

1. - *Regra de introdução da negação*

   Essa regra está diretamente relacionada com a regra do absurdo intuicionista, que de acordo com a extração dos conteúdos computacionais, expressa a alocação de memória vazia, pois toda utilização da regra do $\perp$ está relacionada com a inferência de propriedades que envolvem dados contraditórios. Logo, associada com a premissa da regra de introdução da negação tem-se uma alocação de memória vazia. Como não se tem certeza sobre este fato, foi decidido fazer a restrição de não se utilizar esta regra e estudá-la melhor [2].

2. - *Eliminação do quantificador existencial*

   Na associação dos conteúdos semi-computacionais de uma prova com esta regra, é considerada a existência de um programa associado à premissa maior, que será referenciado através do comando (*exec*) durante o processo de extração dos conteúdos semi-computacionais (esse comando será substituído pela chamada de um procedimento). No caso da hipotése indutiva, o programa relacionado com o quantificador existencial é o mesmo que está sendo construído. Então, sabe-se qual é a chamada do programa que substituirá o comando *exec*. Caso contrário, o programa referido é somente hipótetico, e como não se sabe o formato da chamada, o usuário fica encarregado de fornecer esta informação para o programa[3].

Após o processo de marcação das entradas e das saídas que é utilizado na de verificação da correção do sistema, são apresentados os comandos da linguagem imperativa (semelhante a linguagem PASCAL [10]) relacionados as específicas regras de inferência em dedução natural.

Associação das regras de inferência com comandos da linguagem imperativa

No processo de geração de programas, que é realizado percorrendo a árvore de prova na direção descendente ($top - down$), cada fórmula está relacionada a um programa resultante da associação das regras de inferência

[2] O capítulo 4 apresenta uma argumentação para a eliminação desta restrição.
[3] A argumentação para a eliminação desta restrição está presente no capítulo 3.

com os comandos de uma dada linguagem de programação. Cada associação está baseada no conteúdo (*lógico* ou *semi-computacional*) da fórmula, de forma que o programa refletirá a semântica associada à prova da fórmula.

Uma fórmula possui *conteúdo lógico* quando é derivada a partir de um axioma (tipo de dados) e descreve a natureza dos objetos utilizados pelo programa para resolver o problema proposto, ou seja, descreve as estruturas de dados do programa e o conjunto de operações que podem ser aplicados a elas.

O *conteúdo semi-computacional* de uma fórmula é o conjunto de informações que expressam as relações entre os dados de entrada e de saída do programa – que é a solução do problema especificado pela fórmula – onde para mais de uma entrada pode-se ter uma ou mais saídas.

O *conteúdo computacional* de uma fórmula é uma função, contida nos conteúdos semi-computacionais, que relaciona a entrada com saídas específicas do programa, refletindo a aplicação das regras de inferência utilizadas para a obtenção da fórmula.

Utiliza-se a seguinte notação para descrever as regras de geração de programas:

1. -$\Lambda : \alpha_T^V$ – onde, Λ é um programa que calcula a propriedade descrita em α_T^V ;
2. -$\overrightarrow{v}$ – lista de variáveis contidas em V;
3. -$\overrightarrow{t}$ – lista de variáveis contidas em T;
4. -Λ, Ψ, Γ – programas;
5. - σ – descrição da alocação de memória (conteúdo lógico);
6. - p – nome do programa relacionado com a fórmula.

Observação 2.1.3 *1. - A comunicação entre as iterações do programa é realizada via arquivo;*

2. - Os comandos da linguagem, na qual o programa será construído, possuem a mesma semântica dos comandos equivalentes nas linguagens de programação imperativas usuais;

◇ *Fórmulas iniciais*

Axiomas

Axiomas descrevem a natureza dos objetos utilizados pelo programa para resolver um problema, possuindo assim, somente conteúdos lógicos.
A regra associada é: $\sigma : \beta_T^V$.

Hipóteses

Hipóteses expressam a configuração de memória requisitada pelo programa relacionado com a prova desta fórmula. Pelas restrições da prova, estas hipóteses são indutivas; logo, o programa associado está sendo construído. A execução do programa fornece a configuração de memória requisitada.
A regra associada é: $p : \delta_T^V$

◇ *Eliminação do Quantificador Universal*

Essa regra pode ser interpretada como a alocação de memória para uma variável que será instanciada com um valor de entrada. A premissa desta regra pode apresentar conteúdos lógicos ou um programa associado a hipótese. Nos dois casos, a ação associada a regra adicionará uma nova alocação de memória ao conjunto de variáveis de entrada.
A regra de associação de comandos é: $\dfrac{\Xi : \forall x \alpha(x)_T^V}{\Xi : \alpha(h)_T^{V \cup h}}$, onde Ξ é uma metavariável e pode expressar conteúdos lógicos ou programas.

◇ *Introdução Quantificador Universal*

Essa regra pode ser interpretada como a atribuição de um valor a um espaço de memória reservado a ele. A aplicação desta regra adicionará ao programa o comando $read(...)$, que atribui um dado de entrada ao espaço de memória associado a variável.
A regra de associação de comandos é: $\dfrac{\Lambda : \alpha(h)_T^{V \cup h}}{\Lambda; read(h) : \forall y \alpha(y)_T^V}$

◇ *Eliminação do Quantificador Existencial*

Essa regra pode ser interpretada da seguinte forma: existe um valor que satisfaz a propriedade e que pode ser utilizado para concluir outras propriedades. Como esta regra é aplicada sobre o conjunto de hipóteses, a premissa maior está associada a um programa e uma de suas saídas é o valor referenciado.

A regra de associação de comandos é: $\dfrac{\Psi : \exists x \alpha(x)^V_T \qquad \begin{array}{c} h \leftarrow exec(\Psi, \overrightarrow{v}) : \alpha(h)^V_{T \cup h} \\ \vdots \\ \Lambda : \gamma^{V_1}_{T_1} \end{array}}{\Lambda : \gamma^{V_1}_{T_1}}$

, onde $\overrightarrow{v}$ descreve a lista de variáveis de entrada.

Observação 2.1.4 *Como a regra de eliminação do quantificador existencial é aplicada sobre uma fórmula que representa uma hipótese indutiva, o comando exec será substituído pela chamada do procedimento.*

◇ *Introdução do Quantificador Existencial*

A aplicação desta regra pode ser interpretada da seguinte forma: existe um valor (h) que satisfaz as propriedades especificadas pela premissa. Assim, h será a saída do programa associado a premissa. Essa regra adicionará ao programa relacionado à premissa o comando $write(...)$, que escreve no arquivo de saída o valor gerado pelo programa.

A regra de associação de comandos é: $\dfrac{\Lambda : \alpha(h)^V_{T \cup h}}{\Lambda; write(h) : \exists x \alpha(x)^V_T}$

◇ *Eliminação da Conjunção*

Essa regra pode ser interpretada da seguinte forma: dada uma prova de uma propriedade que é composta por outras duas, pode-se afirmar que ambas foram provadas.

Se o objetivo for apenas o conjunto de comandos relacionados com uma das conclusões derivadas a partir da aplicação desta regra, é necessário filtrar o programa associado à premissa para eliminar os comandos relacionados à outra conclusão.

Neste caso, tem-se a seguinte regra de associação de comandos:
$\dfrac{\Lambda : (\alpha \wedge \beta)^V_T}{\Pi_1(\Lambda) : \alpha^{V_1}_{T_1}} \dfrac{\Lambda : (\alpha \wedge \beta)^V_T}{\Pi_2(\Lambda) : \beta^{V_2}_{T_2}}$, onde Π_1 e Π_2 são filtros que serão aplicados em função das variáveis que pertencem ao conjunto de entrada ($V = V_1 U V_2$).

Neste sistema o programa é gerado a partir de uma prova normal, (premissas são derivadas de hipóteses indutivas ou de axiomas) que garante que filtros não são necessários. Logo, o programa relacionado com a conclusão será o mesmo da premissa, como se pode ver na seguinte regra de associação de comandos: $\dfrac{\Lambda : (\alpha \wedge \beta)^V_T}{\Lambda : \alpha^{V_1}_{T_1}} \dfrac{\Lambda : (\alpha \wedge \beta)^V_T}{\Lambda : \beta^{V_2}_{T_2}}$

◇ *Introdução da Conjunção*

Esta regra pode ser interpretada da seguinte forma: dada uma propriedade que é composta por duas outras, se for possível obter suas respectivas provas, é possível obter a prova resultante de sua composição.

O programa relacionado à conclusão dessa regra é a composição (expressa pelo operador $\otimes$) dos programas associados as premissas.
A regra de associação de comandos é: $\dfrac{\Lambda : \alpha^V_T \qquad \Psi : \beta^V_T}{\Lambda \otimes \Psi : (\alpha \wedge \beta)^V_T}$

◇ *Eliminação da Disjunção*

Pode-se interpretar esta regra como: existem duas formas de provar uma propriedade, cada uma a partir de suposições diferentes. Sabendo que ao menos uma delas é verdadeira, mas não qual, é necessário fazer as duas provas possíveis.

Na aplicação desta regra, o programa associado à conclusão será o comando condicional. Após verificar qual propriedade (descrita pela premissa menor) é satisfeita, um conjunto de comandos será executado para obter a solução relacionada com a conclusão.

A verificação se a propriedade é satisfeita ou não é realizada por um programa que retorna um valor booleano: $Verdadeiro$, se for satisfeita; e $Falso$, caso contrário.

A regra de associação de comandos é: $\dfrac{\sigma : (\alpha \vee \rho)^{V_1}_{T_1} \qquad \begin{array}{cc} \sigma : \alpha^{V_1}_{T_1} & \sigma : \rho^{V_1}_{T_1} \\ \vdots & \vdots \\ \Lambda : \gamma^V_T & \Psi : \gamma^V_T \end{array}}{if\ (\alpha)\ then(\Lambda)\ else\ (if\ (\rho)\ then(\Psi)) : \gamma^V_T}$

◇ *Introdução da Disjunção*

Esta regra pode ser interpretada da seguinte forma: existe uma propriedade que pode ser expressa de dois modos diferentes. Logo, a prova de uma delas é suficiente para garantir que se tem a prova da anterior.

Na aplicação desta regra, o programa relacionado com a conclusão será o programa associado a premissa.

A regra de associação de comandos é: $\dfrac{\Lambda : \alpha_T^V}{\Lambda : (\alpha \vee \rho)_T^V} \dfrac{\Psi : \rho_T^V}{\Psi : (\alpha \vee \rho)_T^V}$

◇ *Eliminação da Implicação*

A regra associada a regra de eliminação da implicação pode ser interpretada como uma prova de uma propriedade (ρ) que depende da suposição de uma outra (α). Porém, obtendo-se uma prova da suposta propriedade (α), pode-se anexá-la a outra prova (ρ).

Pelas restrições da prova, a premissa menor possui sempre conteúdo lógico; logo, o programa gerado será o mesmo da premissa maior:

$$\frac{\Xi : \alpha_T^V \qquad \Lambda : (\alpha \to \rho)_T^V}{\Lambda : \rho_T^V}$$

◇ *Introdução da Implicação*

Esta regra pode ser interpretada da seguinte forma: supondo-se que uma propriedade é verdadeira, ela pode ser utilizada para provar outra propriedade. Isto reflete a possibilidade da existência de um procedimento que pode ser usado ou não pelo programa. No entanto, de acordo com as restrições da prova, a hipótese utilizada no processo de prova possui somente conteúdo lógico.

Assim, tem-se a seguinte regra de associação de comandos:

$$\frac{\begin{array}{c}\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_m, \Xi : [\alpha]_{T_1}^{V_1} \\ \vdots \\ \Lambda : \rho_T^V\end{array}}{\Lambda : (\alpha \to \rho)_T^V}$$

◇ *Regra do absurdo intuicionista*

A aplicação desta regra pode é interpretada como a não existência do objeto. Logo, a conclusão da aplicação desta regra possui somente contéudos lógicos, e a configuração de memória associada é vazia.

A regra de associação de comandos é: $\dfrac{\begin{array}{c}\sigma : [\alpha]^{V'}_{T'} \\ \vdots \\ \bot^{\{\}}_{\{\}}\end{array}}{\sigma : \gamma^{\{\}}_{\{\}}}$, onde γ descreve uma propriedade qualquer derivada do absurdo intuicionista.

◇ *Indução*

Essa regra de inferência procura provar propriedades sobre um determinado domínio. Ela divide o domínio (passo indutivo) até que se chegue a um subconjunto do domínio, para o qual sabe-se que a propriedade é satisfeita (passo base). Este conceito é análogo aos programas recursivos, que serão gerados pela aplicação desta regra. Pode-se notar que esta regra também é uma forma alternativa para a regra de introdução do quantificador universal, que possui associado a ele o comando *read*. Conseqüentemente, o programa gerado terá o comando relacionado a programas recursivos formados pelos comandos condicionais e *read*:

$$\frac{\begin{array}{ccc}[z \prec l]^{V}_{T} & p : \alpha(a_i)^{V_1}_{T_1} & \\ \vdots & \vdots & \vdots \\ \Psi : \alpha(z)^{V \cup z}_{T} & \Lambda : \alpha(k)^{V \cup k}_{T} & a_i \prec k^{V_2}_{T_2}\end{array}}{\begin{array}{l}\text{Procedure Rec}\{ \\ \quad \text{read}(y); \\ \quad \text{if } (y < l) \text{ then } \Psi \\ \quad \text{else}\{ \qquad\qquad\qquad\qquad\qquad\qquad : \forall y \alpha(y)^{V}_{T} \\ \Lambda \leftharpoondown \{\text{exec}([\text{p}], \overrightarrow{\text{v}}) = \text{Rec*}\} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \} \\ \qquad\qquad\qquad\qquad \}\end{array}}$$

Observação 2.1.5 *Na regra acima, o "* "significa o comando relativo a renomeação do programa hipotético (exec([p],* $\overrightarrow{v}$ *)) pela chamada recursiva (Rec) do programa (*Λ*).*

O processo de síntese de programas é composto de duas partes: marcação das configurações de memória das fórmulas da prova e a extração do conteúdo computacional da prova. Com este método, obtém-se um programa que apresenta a mesma semântica da prova e, para garantir isto,

deve-se provar a corretude do sistema que é obtida pela prova do teorema apresentado na seção seguinte.

Prova de corretude do processo de síntese

Para a compreensão do teorema enunciado nesta seção é necessário a apresentação das definições abaixo:

Definição 2.1 - *Considere S um conjunto de fórmulas e H_0 um conjunto de constantes que ocorrem em S. Se nenhuma constante ocorrer em S, então H_0 é formado por uma simples constante, isto é, $H_0 = \{a\}$.*
H_{i+1}, onde $i = 0, 1, 2, ...$, é a união de H_i com o conjunto de todos os termos da forma $f^n(t_1, \ldots, t_n)$ para todas as função n-árias (f) que ocorrem em S, onde t_j com $j = 1, \ldots, n$, são membros do conjunto H_i. Assim, cada H_i é denominado como conjunto de constantes do nível i de S, e H_∞ , tal que: $H_\infty = \bigcup_{i \in N} H_i$ é denominado como ***Universo de Herbrand*** *de S [36].*

Definição 2.2 - *Seja S um conjunto de fórmulas; H o universo de Herbrand de S, e I uma interpretação de S sobre H, se I satisfizer as seguintes restrições:*

1. *I mapeia todas as constantes de S nelas mesmas; e*
2. *Seja f um identificador para uma função n-ária e $h_1, \ldots, h_n$ elementos de H. Em I, f é uma função que mapeia $(h_1, \ldots, h_n)$(um elemento de H_N), em $f(h_1, \ldots, h_n)$, (um elemento de H).*

I será uma ***Interpretação de Herbrand*** *de S [36].*

Definição 2.3 - *Seja S um conjunto de fórmulas, uma interpretação de Herbrand que satisfaz S é denominada como* ***Modelo de Herbrand*** *de S.*

Teorema 2.4 *Seja M um modelo de Herbrand para o conjunto de axiomas (Δ) e um conjunto de hipóteses que não são axiomas ($\beta_1, ..., \beta_n, \delta_1, ..., \delta_m$), σ uma função de atribuição de valores a variáveis. Dada uma prova Π da fórmula da forma $\forall x \exists y \alpha(x, y)$ obtida a partir de Δ e ($\beta_1, ..., \beta_n, \delta_1, ..., \delta_m$), e um programa Λ gerado pela função G(L(Π)), então:*

$$\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_m \underset{M,\sigma}{\models} \Lambda : \forall x \exists y \alpha(x, y)^{\{\ \}}_{\{\ \}}$$

isto é, se M satisfaz o modelo de $\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_m$ com σ então satisfaz Λ.

Prova. A prova do teorema acima é realizada por indução no tamanho da prova, através da comparação da semântica das modificações sintáticas do programa com os conteúdos semi-computacionais das regras de inferência [47]. Ver figura 2.1. □

Nota: Esta prova é similar a que será apresentada na seção 3.4.1.

$$\frac{\Pi}{\forall x \exists y \alpha(x,y)} \quad \overset{\mathrm{L}}{\Rightarrow} \quad \frac{\Pi}{\forall x \exists y \alpha(x,y)^{\{\}}_{\{\}}} \;\Downarrow \mathrm{G}\; \frac{\Pi}{\Lambda : \forall x \exists y \alpha(x,y)^{\{\}}_{\{\}}}$$

Figura 2.1: Esquema do processo de síntese.

Observação 2.1.6 *Este processo de síntese é composto por duas partes: a marcação das configurações de memória (L) e a extração do conteúdo computacional da prova (G).*

2.2 Exemplo

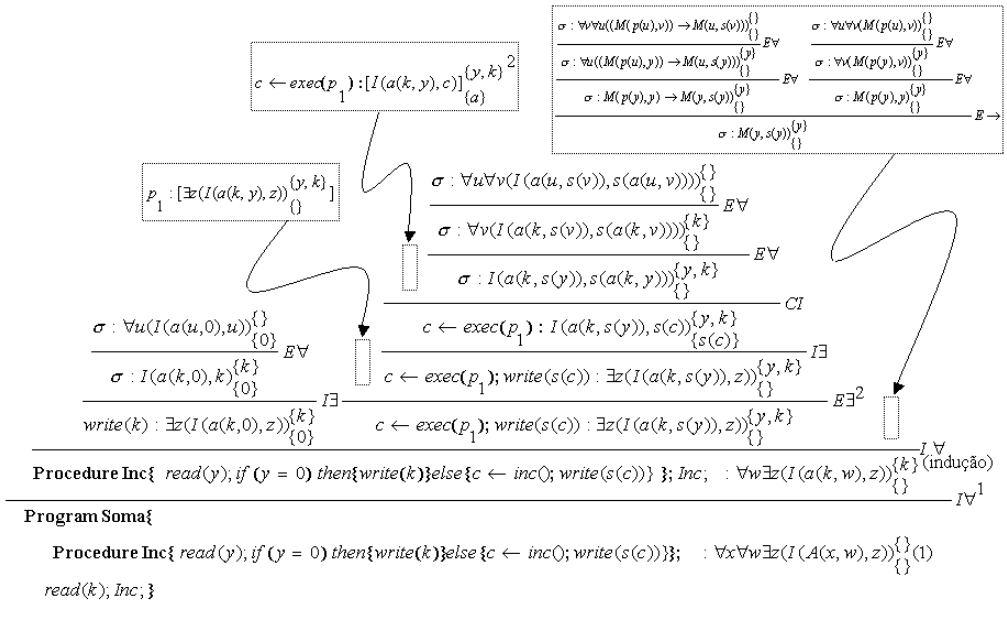

Figura 2.2: Extração do conteúdo computacional para a prova da adição de dois números naturais

2.3
Predicado de Provabilidade

Esta seção apresenta o predicado de provabilidade que é utilizado na seção 4.2.2.

O segundo teorema da incompletude de Gödel afirma que qualquer sistema formal contendo a aritmética de Peano (PA) é consistente se, e somente se, ele não puder provar sua própria consistência.

Esse teorema pode ser generalizado e abstraído de várias maneiras, mas todas as provas são baseadas na noção do predicado de provabilidade (Bew), que desempenha papel fundamental nos estudos modernos da meta-matemática.

Predicado de provabilidade[4]

Conceitos básicos

. $\ulcorner X \urcorner$ é um numeral que representa o número de Gödel para a fórmula X;

. Uma fórmula é atômica se a fórmula tiver uma das seguintes formas $c_1 + c_2 = c_3$, $c_1.c_2 = c_3$, $c_1 = c_2$, ou $c_1 \leq c_2$, onde cada c_1, c_2 ou c_3; ou é uma variável ou é um numeral (sendo que, em uma fórmula, alguns podem ser variáveis e outros numerais)

Pode-se definir a classe de fórmulas Σ_0 indutivamente da seguinte forma:

i - Toda fórmula Σ_0 atômica é Σ_0;

ii - Se F e G são Σ_0, então $\neg F$, $F \vee G$, $F \wedge G$, $F \rightarrow G$ e $F \equiv G$ também são Σ_0,

iii - Para qualquer fórmula Σ_0, qualquer variável v_i e todo c, seja ele numeral ou variável diferente de v_i, a expressão $\forall v_i(v_i \leq c \rightarrow F)$ também é uma fórmula Σ_0.

. Pode-se dizer que uma fórmula α é uma fórmula Σ_1 se ela for da forma $\exists v_{n+1} F(v_1, ..., v_n, v_{n+1})$ onde $F(v_1, ..., v_n, v_{n+1})$ é uma fórmula Σ_o .

[4]O Predicado de provabilidade é apresentado para arimética de Peano, mas cabe ressaltar que este resultado também vale para aritmética de Heyting.

Dados os conceitos acima tem-se que, a fórmula $Bew(v_1)$, onde v_1 é uma variável livre, é chamada de predicado de provabilidade para S, onde S é um sistema formal, se para toda sentença X e Y valem as seguintes condições: 00

P_1 - Se X for provável em S, então $Bew(\ulcorner X \urcorner)$.

P_2 - $Bew(\ulcorner X \to Y \urcorner) \to (Bew(\ulcorner X \urcorner) \to Bew(\ulcorner Y \urcorner))$ é provável em S.

P_3 - $Bew(\ulcorner X \urcorner) \to Bew(\ulcorner Bew(\ulcorner X \urcorner) \urcorner)$ é provável em S.

Suponha que $Bew(v_1)$ seja uma Σ_1- fórmula que expressa[5] o conjunto P do sistema PA. A partir da hipótese da $\varpi-$consistência, $Bew(v_1)$ representa[6] P em PA.

Com base na hipótese de consistência simples, tem-se que $Bew(v_1)$ representa algum *superset* de P, e este fato é suficiente para garantir a implicação que se X é provável em PA então $Bew(\ulcorner X \urcorner)$. Assim, a propriedade P_1 é válida.

Utilizando o mesmo raciocínio para a propriedade P_2, a sentença $Bew(\ulcorner X \to Y \urcorner) \to (Bew(\ulcorner X \urcorner) \to Bew(\ulcorner Y \urcorner))$ é, obviamente, verdadeira (essa verdade é equivalente a proposição que, se $X \to Y$ e X são prováveis em PA, então, por *modus ponens*, Y é provável em PA).

Para a propriedade P_3 a sentença $Bew(\ulcorner X \urcorner) \to Bew(\ulcorner Bew(\ulcorner X \urcorner) \urcorner)$ é verdadeira (essa verdade é equivalente a proposição que, se X é provável, então $Bew(\ulcorner X \urcorner)$, já que $Bew(\ulcorner X \urcorner)$ é verdade se, e somente se, X for provável; $Bew(\ulcorner Bew(\ulcorner X \urcorner) \urcorner)$ é verdade se, e somente se, $Bew(\ulcorner X \urcorner)$ for provável).

Desta forma a noção de verdade de uma sentença se restringe a propriedade P_1.
Tem-se que esta sentença não é apenas verdadeira mas também é provável em PA[44]. Este resultado é obtido a partir de um caso especial do fato que para toda sentença Σ_1- (Y), a sentença $Y \to Bew(Y)$ é provável em PA. O esquema dessa prova pode ser encontrado em [46].

[5]Informalmente, uma fórmula $F(v_1)$, onde v_1 é a única variável livre, expressa um conjunto A se, e somente se, $\forall n(F(\ulcorner n \urcorner)$ é verdade $\leftrightarrow n \in A)$.

[6]Sem maiores detalhes, uma fórmula $F(v_1)$, onde v_1 é a única variável livre, representa um conjunto A se, e somente se, $\forall n \in A, F(\ulcorner n \urcorner)$ for provável.