Síntese construtiva de programas utilizando lógica intuicionista e dedução natural

Geiza Maria Hamazaki da Silva[1]
Edward Hermann Haeusler[2]

Resumo

A síntese construtiva ou prova como programa *(proofs-as-programs)* [BC85] é baseada no isomorfismo *Curry-Howard* [H69] e no fato de que uma prova construtiva para um teorema (em alguma teoria) que descreve um problema trabalhado, pode ser interpretada como a descrição de um processo, isto é um algoritmo [GLT89]. Neste programa, apresentamos um método de síntese construtiva de programas no qual o programa descrito através de uma linguagem imperativa é gerado a partir de uma prova em lógica intuicionista poli-sortida utilizando como sistema dedutivo a dedução natural. Através do conceito de conteúdo semi-computacional de uma fórmula, será demonstrado que o programa gerado é realmente uma representação para a solução do problema especificado através do teorema da teoria que descreve os tipos de dados envolvidos.

Palavras Chaves: Lógica intuicionista, Síntese construtiva.

1. Introdução

Garantir que programas são implementados de forma a cumprir a especificação (correção dos programas), e que estes estão documentados de modo que seja fácil sua manutenção e

[1] PUC/Rio, Departamento de Informática.
[2] PUC/Rio, Departamento de Informática.

adaptação são questões fundamentais em computação. Têm sido propostos vários métodos que almejam solucionar estes problemas ([CIU98], [BSW90], [G79], [CM98] e [BBSSZ98]), e iremos apresentar um método denominado síntese construtiva de programas, no qual partindo de uma especificação obtemos um programa que será correto por construção.

Pressupondo que um programa capaz de solucionar um problema pode ser considerado como uma solução construtiva para o mesmo, têm sido propostos vários métodos para a obtenção de programas a partir da prova da existência da solução do problema em algum formalismo lógico.

A descrição de um problema em lógica de predicados pode ser vista dentro do escopo do GPT (*General Problem Theory*) [V84], já que esta é capaz de descrever dados de entrada, de saída e a relação existente entre os mesmos. No entanto, essas informações não são suficientes para que possamos garantir a existência de um método que solucione o problema.

Se considerarmos que a sentença que descreve um problema é um teorema (em alguma teoria), e se conseguirmos uma prova construtiva para o mesmo, poderemos entendê-la não apenas como uma transcrição sintática, mas sim como uma descrição de um determinado objeto, ou seja, uma descrição de um algoritmo [GLT89].

O isomorfismo *Curry-Howard*, associa regras de inferência em dedução natural utilizadas em uma prova a regras de formação de λ-termo (comandos em uma linguagem de programação), de tal forma que uma prova de δ (uma fórmula) pode ser vista como um λ-termo do tipo δ (um tipo). Deste modo, podemos dizer que uma prova tem interpretação computacional, isto é, pode ser vista como um conjunto de comandos em uma linguagem de programação, ou seja, um programa [H69].

Com base nessas considerações, surgiu a idéia de construir um programa a partir de uma prova, através de um processo conhecido como síntese construtiva de programas ou prova como programa (*proofs-as-programs*) [BC85]. Há

propostas de vários sintetizadores construtivos, nos quais a especificação dos problemas é realizada em lógicas construtivas (por exemplo, a teoria intuicionista dos tipos). Esses sistemas utilizam como sistema dedutivo o cálculo de seqüentes ([CIU98], [BSW90] e [G79]) ou o mecanismo de reescrita ([CM98]), e geram programas nos paradigmas de programação lógica e funcional.

O processo de síntese construtiva de programas inicia-se com a prova, em uma dada teoria, do teorema que descreve o problema, utilizando para isso uma lógica construtiva, seguido da construção de um programa que solucione o problema especificado pelo teorema. O processo de construção do programa a partir de uma prova denomina-se usualmente de "*extração do conteúdo computacional de uma prova*". Como o isomorfismo *Curry-Howard* [H69] é a base deste processo de síntese, iremos propor um sintetizador construtivo no qual o programa é gerado a partir de uma prova em dedução natural, evitando o passo de conversão que é utilizado nos trabalhos apresentados na literatura ([CIU98], [BSW90], [G79] e [CM98]). Neste método, a partir de uma prova em lógica de predicados poli-sortida intuicionista, será construído um programa em linguagem imperativa.

Através do conceito semi-computacional de uma fórmula, constata-se que o programa gerado é realmente uma representação de uma solução do problema especificado pelo teorema da teoria que descreve os tipos de dados envolvidos.

Na seção 2 será apresentado um processo de síntese construtiva de programas, no qual é realizada a marcação das entradas e das saídas do programa a ser gerado, seguido da associação de cada regra de inferência com comandos na linguagem imperativa. Na seção 3, é descrita a prova de que o programa gerado cumpre a especificação, ou seja, a prova de sua corretude. A seção 4 contém um exemplo do nosso mecanismo de síntese construtiva e na seção 5 é apresentada a conclusão do trabalho.

2. O processo de síntese de programas

No nosso processo de síntese, partiremos da existência de um provador de teoremas em lógica de predicados intuicionista poli-sortida com aritmética, o qual além das regras de inferência usuais, possui regras de inferência para a igualdade e indução. O provador construirá uma prova normal em dedução natural, para um determinado teorema, que será a entrada para o sintetizador de programas.

Existem restrições[3] em relação às regras de inferência aplicadas na prova que é fornecida como entrada para o sintetizador: 1 – as provas não poderão conter a regra de introdução da negação; 2 – a regra de eliminação do quantificador existencial só poderá ser aplicada sobre a fórmula que representa a hipótese indutiva, ou seja, as outras hipóteses não poderão conter este conectivo em suas fórmulas. Esta última restrição poderá ser atenuada se admitirmos como soluções programas parametrizados.

O processo de síntese de programas a partir da prova do problema especificado, é divido em duas fases: 1 – a marcação das entradas e das saídas do programa a ser gerado e, 2 – associação de comandos em uma linguagem imperativa a cada regra de inferência.

Marcação das entradas e saídas

Segundo a leitura operacional dos conectivos dada pelo isomorfismo *Curry-Howard*, as entradas do programa em uma prova são associadas às variáveis quantificadas pelo quantificador universal. Estas são representadas na fórmula por variáveis livres, visto que podem assumir qualquer valor (desde que do mesmo tipo da variável) de forma que a fórmula seja sempre verdadeira. Já as saídas do programa em uma prova

[3] Estas restrições serão comentadas detalhadamente mais à frente.

estão relacionadas com o quantificador existencial. Os termos associados aos quantificadores existenciais são representados nas fórmulas que compõem a prova através das variáveis livres e dos termos que estão relacionados com as variáveis de entrada.

Como podemos observar, as entradas e saídas do programa relacionadas a uma fórmula são descritas por suas variáveis livres e termos. Assim, o processo de marcação das entradas e das saídas em uma prova adiciona, partindo da conclusão (teorema provado), os termos e variáveis livres de cada fórmula no seu conjunto de termos de saída e no conjunto de variáveis de entrada (respectivamente).

Este processo marca, inicialmente, as listas de termos e variáveis livres da conclusão como vazias. Em seguida, são utilizadas as regras para a marcação das entradas e das saídas, descritas abaixo, no sentido *bottom-up* (i.e., da conclusão para as premissas). Ao chegarmos às folhas da árvore de prova podem ser encontradas algumas variáveis e termos pertencentes aos conjuntos de variáveis de entrada ou de termos de saída que não serão utilizados como entradas ou saídas no programa associado à prova. Essas variáveis e termos refletem os dados que estão em memória e que não são utilizados, sendo considerados resíduos do processo de marcação de entradas e saídas. Esses resíduos serão inseridos nos conjuntos de variáveis de entrada e de termos de saída das fórmulas pertencentes ao ramo de prova derivado a partir da fórmula no qual eles são detectados pela primeira vez (esse processo será realizado no sentido *top-down*). Assim, esses resíduos serão propagados até a raiz da árvore de prova (conclusão) cuja lista de variáveis de entrada e termos de saída não serão mais vazios.

Regras para a marcação das entradas e saídas

As regras para marcação das entradas e saídas estão relacionadas com as aplicações das regras de inferência, por isso, as fórmulas utilizarão a representação: α^{V}_{T}, onde α é a

fórmula, V o conjunto de variáveis de entrada e T o conjunto de termos de saída.

Na apresentação das regras, a notação K∪a expressa a operação K∪{a}, onde K pode ser o conjunto das variáveis de entrada ou o conjunto dos termos de saída.

A seguir são apresentadas as regras para marcação das entradas e das saídas. Estas devem ser analisadas no sentido da conseqüência para as premissas devido ao sentido do processo de marcação (*bottom-up*).

Hipóteses: axiomas e não axiomas

$$\beta^{V}_{T}$$

Observação se β for um axioma V = { }

Eliminação do quantificador universal

$$\frac{\forall y\,\alpha(y)^{V}_{T}}{\alpha(a)^{V\cup a}_{T}}$$

Introdução do quantificador universal

$$\frac{\alpha(a)^{V\cup a}_{T}}{\forall y\,\alpha(y)^{V}_{T}}$$

Eliminação do quantificador existencial

$$\frac{\exists y\,\alpha(y)^{V}_{T} \qquad \begin{array}{c}\alpha(a)^{V}_{T\cup a}\\ \vdots \\ \delta^{V'}_{T'}\end{array}}{\delta^{V'}_{T'}}$$

Introdução do quantificador existencial

$$\frac{\alpha(b)^{V}_{T\cup b}}{\exists y\,\alpha(y)^{V}_{T}}$$

Eliminação da conjunção

Introdução da conjunção

$$\frac{(\alpha \wedge \beta)^{V}_{T}}{\alpha^{V}_{T}} \qquad \frac{(\alpha \wedge \beta)^{V}_{T}}{\beta^{V}_{T}} \qquad\qquad \frac{\alpha^{V}_{T} \quad \beta^{V}_{T}}{(\alpha \wedge \beta)^{V}_{T}}$$

Eliminação da disjunção

$$\frac{(\alpha \vee \beta)^{V}_{T'} \qquad \begin{matrix}[\alpha]^{V}_{T'} \\ \vdots \\ \gamma^{V}_{T}\end{matrix} \qquad \begin{matrix}[\beta]^{V}_{T'} \\ \vdots \\ \gamma^{V}_{T}\end{matrix}}{\gamma^{V}_{T}}$$

Introdução da disjunção

$$\frac{\alpha^{V}_{T}}{(\alpha \vee \beta)^{V}_{T}} \qquad \frac{\beta^{V}_{T}}{(\alpha \vee \beta)^{V}_{T}}$$

Eliminação da implicação

$$\frac{\alpha^{V'}_{T'} \quad (\alpha \rightarrow \beta)^{V}_{T}}{\beta^{V}_{T}}$$

Introdução da implicação

$$\frac{\begin{matrix}[\alpha]^{V}_{T'} \\ \vdots \\ \beta^{V}_{T}\end{matrix}}{(\alpha \rightarrow \beta)^{V}_{T}}$$

Indução

$$\frac{[K \prec l)^{V'}_{T'} \qquad [\alpha(a_i)]^{V}_{T_1} \qquad \begin{matrix}\vdots \\ \alpha(K)^{V \cup K}_{T} \quad \alpha(w)^{V \cup W}_{T} \quad ai \prec w^{V}_{T_2}\end{matrix}}{\forall y \alpha(y)^{V}}$$

Regra do absurdo intuicionista

$$\frac{\beta^{V}_{T_1} \quad \neg \beta^{V}_{T_2}}{\begin{matrix}\perp^{\{\}} \\ \{\}\end{matrix}} \qquad \frac{\begin{matrix}\perp^{\{\}} \\ \{\}\end{matrix}}{\begin{matrix}\alpha^{\{\}} \\ \{\}\end{matrix}}$$

T

Onde, K – reflete o termo associado ao caso base e, w – reflete o termo associado ao passo indutivo.

Observação: 1. No processo de marcação das entradas e saídas, se uma prova utilizar a propriedade de congruência da igualdade, as fórmulas resultantes da utilização desta regra terão o conjunto de variáveis de entrada e de termos de saída alterados da seguinte forma: as variáveis ou termos substituídos serão retirados do conjunto[4] a que pertencem, e devem ser adicionados os termos e as variáveis (dos quais o termo substituto depende) em seus respectivos conjuntos.

Exemplo: A regra de inferência para a propriedade de congruência da igualdade quando esta é estabelecida para um termo de saída:

$$\frac{x = y \quad P(y)}{P(x)}$$

onde V'[x] retorna as variáveis de entrada relacionadas com o termo x.

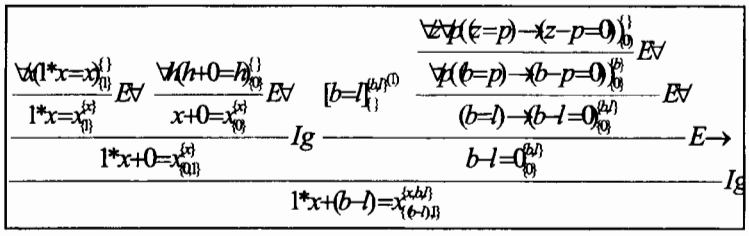


Figura 1: Exemplo da aplicação do processo de marcação de entrada e saída. Parte do exemplo da seção 4

[4] Conjunto das variáveis de entrada e de termos de saída.

Mapeamento das regras de inferência com comandos

No processo de geração de programas, cada fórmula na prova está relacionada a um programa que resulta da associação das regras de inferência – utilizadas no seu processo de prova – com comandos em uma determinada linguagem de programação (regras de geração de programas). Desta forma, um programa refletirá a semântica da prova da fórmula associada. Em função do formato deste artigo, apresentamos os conceitos abaixo de forma abreviada e informal[5].

Uma fórmula possui *conteúdo lógico*, em contrapartida a possuir conteúdo computacional, se esta descreve a natureza dos objetos utilizados pelo programa para solucionar o problema proposto, isto é, descreve as estruturas de dados de um programa e o conjunto de operações que podem ser aplicadas às mesmas.

O *conteúdo semi-computacional* (CSC) de uma fórmula é o conjunto de informações que expressa as relações entre as entradas e saídas de qualquer programa que sirva de solução para o problema especificado na fórmula, onde para mais de uma entrada podemos ter a mesma saída ou mais de uma saída. O conteúdo semi-computacional ($CSC^{\theta}_{M}(\alpha^{V}_{T})$) é descrito por triplas que possuem a seguinte estrutura: $< L, < \vec{b}, \vec{o} >, W >$, onde: L é o arquivo com a lista de valores de entrada, $\vec{b}$ é a lista dos valores de entrada em memória, $\vec{o}$ é a lista dos valores de saída em memória, W é o arquivo com a lista de valores de saída.

A definição do CSC é feita por recursão na complexidade da fórmula. Por exemplo, a definição para fórmulas com o quantificador universal é:

$$CSC^{\theta}_{M}(\forall x\, \alpha(x)^{V}_{T} = \{\forall b_i (\{b_i \frown L, \vec{b}, \vec{o}, w\} / \{L, \{b_i \frown \vec{b}, \vec{o}\}, w\} \in CSC^{\theta}_{M}(\alpha(h)^{V \cup h}_{T}))\}$$

Onde, θ - é um conjunto de fórmulas, $x \frown y$ – expressa a concatenação de x com o y, L é o arquivo com a lista de valores de entrada, $\vec{b}$ é a lista dos valores de entrada em memória, $\vec{o}$ é a

[5] Maior detalhes em [S99]

lista dos valores de saíde em memória, e W é o arquivo com a lista de valores de saída.

A definição do $CSC^{\theta}_{M}(\forall y \alpha(y)^{V}_{T}$ expressa que, para um termo estar presente no arquivo dos valores de entrada, este deverá pertencer à lista dos valores de entrada que estão em memória.

O *conteúdo computacional* de uma fórmula é uma função, contida no conteúdo semi-computacional, que relaciona entradas com saídas específicas do programa. Estas refletem a aplicação das regras de inferência utilizadas para a obtenção da fórmula.

As regras de construção do programa, a partir da prova, são baseadas na seguinte asserção: $\Lambda : \alpha^{V}_{T}$ – onde, Λ é um programa que calcula a propriedade descrita em α^{V}_{T}.

Apresentamos a seguir a descrição das regras de geração de programas, na qual utilizaremos a seguinte notação: Λ, Ψ, T – programas; σ - descrição das alocações de memória; p – nome do programa relacionado à formula.

Observação: Os comandos da linguagem na qual será gerado o programa possuem a mesma semântica dos comandos equivalentes nas linguagens imperativas usuais.

• Hipóteses – axiomas e não axiomas

As hipóteses descrevem a natureza dos objetos utilizados pelo programa para solucionar o problema proposto. Assim, elas possuirão apenas conteúdo lógico (exceto a hipótese indutiva). Suas fórmulas não apresentam quantificadores existenciais; logo, os programas relacionados a estas não geram saídas. Desta forma, podemos dizer que teremos associados a elas programas vazios, juntamente com alocações de memória (σ) descritas pelo conteúdo lógico. Assim, a asserção associada à hipótese β é: $\sigma : \beta^{V}_{T}$

A hipótese indutiva (δ) descreve a configuração de memória que deve ser satisfeita para a execução do programa, sendo que esta configuração é fornecida por um programa

associado à mesma. Teremos a seguinte asserção associada a essa fórmula: $\mathbf{p} : \delta^{V}_{T}$, onde **p** é um símbolo para o programa.

• Eliminação do quantificador universal

Esta regra será aplicada sobre fórmulas pertencentes ao conjunto de hipóteses, sendo que a ação associada a essa regra dependerá da natureza da hipótese.

1. Axiomas e hipóteses não indutivas

$$\frac{\sigma : \forall y\ \alpha(y)^{V}_{T}}{\sigma : \alpha(a)^{V\ \cup a}_{T}}$$

2. Hipótese indutiva

$$\frac{p : \forall y\ \alpha(y)^{V}_{T}}{p : \alpha(a)^{V\ \cup a}_{T}}$$

• Introdução do quantificador universal

$$\frac{\Lambda : \alpha(a)^{V\ \cup a}_{T}}{read(a);\ \Lambda : \forall y\ \alpha(y)^{V}_{T}}$$

• Eliminação do quantificador existencial

Na geração de programas, o valor que satisfaz a propriedade especificada pela premissa maior será utilizado pelo programa associado à conclusão desta regra (T). Logo, o valor procurado é o resultado da execução do programa associado à premissa maior desta regra.

$$\frac{\begin{array}{l} a \leftarrow exec\ (p, \overrightarrow{v}) : [\alpha(a)^{V}_{T \cup a}] \\ p : \exists y \alpha(y)^{V}_{T} \qquad : \\ \qquad\qquad T : \delta^{V'}_{T}, \end{array}}{\qquad\qquad}$$

$$T : \delta^{V'}_{T},$$

O comando *exec*(...) prepara o ambiente para a execução da função, isto é, atribui para as variáveis de entrada do programa os valores de entrada passados por parâmetro. Além disso, realiza a chamada da função e retorna o último termo de saída após a execução de Λ.

• Introdução do quantificador existencial

$$\frac{\Lambda : \alpha\,(b)^{V}_{T} \cup b}{\Lambda;\ write\ (b) : \exists y\ \alpha\,(y)^{V}_{T}}$$

Observação: O comando *write*(...) escreve no arquivo de saída o valor gerado internamente pelo programa.

• Eliminação da conjunção

Se, na geração do programa, desejarmos apenas o conjunto de comandos referentes a uma das conclusões da regra de eliminação da conjunção, devemos passar um filtro no programa associado à premissa desta regra de forma a eliminarmos os comandos que dizem respeito a outra conclusão. A asserção associada à eliminação da conjunção:

$$\frac{\Lambda : \alpha \wedge \beta^{V}_{T}}{\Pi_1(\Lambda) : \alpha^{V}_{T}} \qquad \frac{\Lambda : \alpha \wedge \beta^{V}_{T}}{\Pi_2(\Lambda) : \beta^{V}_{T}}$$

onde Π_1 e Π_2 são filtros que serão aplicados em função das variáveis pertencentes à V que realmente ocorrem em α e em β respectivamente, eliminando do programa Λ os comandos relativos à β e α.

Neste trabalho não há necessidade de aplicarmos filtros no programa, pois como geramos programas a partir de provas normais com as restrições preestabelecidas, temos que as regras de eliminação da conjunção serão aplicadas sobre axiomas, que

possuem somente conteúdo lógico, não havendo assim necessidade de aplicarmos o filtro, ou sobre hipóteses indutivas. Neste último caso, esta hipótese será utilizada na prova do ramo de prova da indução, podendo ser aplicada para provar tanto o lado esquerdo quanto o lado direito da implicação. Em ambos os casos, sempre será necessário a utilização conjunta das propriedades (descritas por β e α) e como a prova é normal, não haverá a eliminação do conectivo para posterior introdução, logo a regra de eliminação da disjunção não será utilizada sobre a hipótese indutiva.

Podemos concluir que o programa associado à conclusão da regra e a sua premissa será o mesmo. Teremos, então, a segunte asserção:

$$\frac{\Lambda : \alpha \wedge \beta^{V}_{T}}{\Lambda : \alpha^{V}_{T}} \qquad \frac{\Lambda : \alpha \wedge \beta^{V}_{T}}{\Lambda : \beta^{V}_{T}}$$

- Introdução da conjunção

$$\frac{\Lambda : \alpha^{V}_{T} \qquad \Psi : \beta^{V}_{T}}{\Lambda \otimes \Psi : (\alpha \wedge \beta)^{V}_{T}}$$

O comando "$\otimes$" realiza a composição de comandos que podem ser executados em qualquer ordem.

- Eliminação da disjunção

$$\begin{array}{ccc} & \sigma : [\alpha]^{V'}_{T'} & \sigma : [\beta]^{V'}_{T'} \\ \sigma : [\alpha \vee \beta]^{V'}_{T'} & \vdots & \vdots \\ & \Lambda : \gamma^{V} & \Psi : \gamma^{V} \end{array}$$

$$\frac{\quad T \qquad\qquad T \quad}{\textit{If } (\alpha) \textit{ then } (\Lambda) \textit{ else (if } (\beta) \textit{ then } (\Psi)) : \gamma^{V}_{T}}$$

• Introdução da disjunção

$$\frac{\Psi : \alpha^{V}_{T}}{\Psi : (\alpha \vee \beta)^{V}_{T}} \qquad \frac{\Lambda : \beta^{V'}_{T'}}{\Lambda : (\alpha \vee \beta)^{V'}_{T}}$$

• Eliminação da implicação

Em um programa, a aplicação desta regra corresponde a trocarmos uma marcação da utilização do procedimento por sua chamada após a sua implementação, não alterando o resto do corpo do programa. A asserção associada à eliminação da implicação é:

$$\frac{\Psi : \alpha^{V}_{T} \qquad \Lambda : (\alpha \rightarrow \beta)^{V'}_{T'}}{[\Lambda \Leftarrow \{exec(p,v) = \Psi\}] : (\alpha \rightarrow \beta)^{V'}_{T'}}$$

onde $[\Lambda \Leftarrow \{exec([p],v) = \Psi\}]$, denota o programa construído (Λ) com as chamadas a um procedimento suposto (p), que será retornado após as suas substituições pela chamada do procedimento após sua implementação (Ψ).

De acordo com as restrições da prova da qual estamos gerando programas, temos que a premissa menor da regra de eliminação da implicação possui sempre conteúdo lógico. Desta forma, o programa a ser gerado pela aplicação da regra é o próprio programa associado com a premissa maior. Teremos, assim, a seguinte asserção: $\sigma : \alpha^{V} \qquad \Lambda : (\alpha \rightarrow \beta)^{V'}$

$$\frac{T \qquad\qquad T'}{\Lambda : \beta^{V'}_{T'}}$$

Observações:

1. A restrição de que a premissa menor possui sempre conteúdo lógico se deve ao fato de que, se esta tivesse um programa associado a ela, a marcação de sua chamada no corpo do programa relacionado com a premissa maior deveria ser alterada. Como estamos gerando programas a partir de provas normais, temos associados à premissa maior apenas o nome do programa e, desta forma, não sabemos em que lugar devemos alterar a marcação procedimento por sua chamada. Desde modo, faz-se necessária a colocação desta restrição.

2. Na eliminação da implicação, consideramos que a premissa menor possui sempre conteúdo lógico. Entretanto, existem casos em que a premissa menor possui um programa associado a ela, que, devido às restrições na estrutura da prova, é derivada a partir da hipótese indutiva, na qual o programa relacionado após a sua execução fornece a configuração de memória para a execução do programa relacionado com a premissa maior. Assim, podemos interpretar o programa associado à premissa menor como conteúdo lógico, satisfazendo a restrição da regra de eliminação da implicação.

• Introdução da implicação

A aplicação da regra da introdução da implicação pode ser assim interpretada: supondo-se que uma propriedade seja verdadeira, esta poderá ser utilizada na prova de outra propriedade. Na geração de programas, esse fato corresponde à possibilidade da existência de um procedimento que poderá, ou não, ser utilizado pelo programa. Caso seja utilizado, teremos uma marcação da sua utilização (*Dec* ...) para alterarmos a chamada quando o procedimento for implementado. A asserção associada à intrdução da implicação é:

$$p : [\alpha]^{V'}_{T'}$$

$$\frac{\begin{array}{c}\vdots\\ \Lambda : \beta^{V'}_{T'}\end{array}}{Dec\, p\, (\Lambda : (\alpha \rightarrow \beta)^{V'}_{T}}$$

Entretanto, de acordo com o modelo de prova da qual estamos gerando programas, a hipótese utilizada no processo de prova posssui apenas conteúdo lógico, indicando quais as estruturas de dados necessárias para a execução do programa. Desta maneira, o programa a ser gerado pela aplicação da regra é o programa associado à sua premissa. Teremos, assim, a seguinte asserção:

$$\frac{\begin{array}{c}\sigma : [\alpha]^{V'}_{T'}\\ \vdots\\ \Lambda : \beta^{V'}_{T}\end{array}}{\Lambda : (\alpha \rightarrow \beta)^{V'}_{T}}$$

• Aplicação da Regra de inferência da indução

Ao analisarmos a estrutura da regra de inferência da indução, podemos perceber que o seu conceito é análogo ao processo efetuado pelos programas recursivos e que esta regra é uma forma alternativa da aplicação da regra de introdução do quantificador universal. Conseqüentemente, o programa gerado deverá conter o comando associado à aplicação desta regra (*read*(...)) e um programa recursivo formado por um comando condicional. As chamadas recursivas serão declaradas nos pontos da prova em que foi utilizada a hipótese indutiva, que estão marcadas com o comando *exec* (...). A asserção[6] associada à regra de inferência da indução é:

$$\begin{array}{ccc}[z \prec l]^{V'}_{T'} & [p : \alpha\,(a_i)^{V^1}_{T_1}] & \\ \vdots & \vdots & \vdots\end{array}$$

[6] Esta asserção é apenas um esquema básico, visto que, para cada sorte a regra terá um formato.

$$\Psi : \alpha(z)^{V \cup Z}_{T} \quad \Lambda : \alpha(k)^{V \cup k}_{T} \quad a_i \prec k^{V_2}_{T_2}$$

Procedure Rec { ***read(y);***
if (y < l) then $\{\Psi\}$
else $: \forall y \alpha [y]^{V}_{T}$

$\{ [\Lambda \Leftarrow \{exec([p], \vec{v} = Rec\}^*]\}$

Observações: 1. Se esta regra for a última a ser aplicada no processo de prova, então em vez de *Procedure* este programa estará declarado como *Program*; 2. Na asserção acima, o * significa o comando relativo a renomeação do programa hipotético pela chamada recursiva.

• Regra do Absurdo intuicionista

As premissas utilizadas para a aplicação da regra do absurdo intuicionista descrevem propriedades contraditórias, o que torna inconsistentes as informações (sobre os objetos utilizados pelo programa) nelas contidas. Por isso, a aplicação desta regra é interpretada como a inexistência do objeto, o que leva a fórmula resultante de sua aplicação a possuir apenas conteúdo lógico e as alocações de memória serem consideradas como vazias (liberação do espaço de memória alocado pelas premissas). A asserção associada à prova da redução ao absurdo é:

$$\frac{\begin{array}{c}\sigma : \beta^{V}_{T} \\ \vdots \\ \perp^{\emptyset}_{\emptyset}\end{array}}{\sigma : \gamma^{V'}_{T'}}$$

Restrições sobre as regras utilizadas na prova

As regras abaixo citadas não foram utilizadas por não serem triviais no seu processo de extração do conteúdo semi-computacional.

Introdução da Negação: Esta regra está diretamente associada à prova por absurdo (intuicionista), que, de acordo com o processo de extração do conteúdo semi-computacional, expressa apenas que as alocações de memória são vazias. Desta maneira, temos alocações de memória vazias associadas à fórmula sobre a qual foi aplicada a regra de introdução da negação. Entretanto, existem contra-exemplos para essa regra de extração do conteúdo semi-computacional, como, por exemplo, a prova que para um determinado domínio formado de dois elementos, existe sempre um elemento diferente do outro. Como esta regra deve ser melhor estudada, restringimos a sua utilização.

Eliminação do quantificador existencial: Ao extrairmos o conteúdo semi-computacional de uma prova com esta regra, consideramos a existência de um programa associado à sua premissa maior. Este programa será referenciado através de uma marcação (*exec*) durante o processo de extração do conteúdo semi-computacional da prova que, em determinado ponto do processo, será alterado pela chamada do programa.

No caso da hipótese indutiva, o programa relacionado com o quantificador existencial é o próprio programa a ser construído. Logo, saberemos qual será a chamada do programa que substituirá a marcação.

Se as hipóteses não indutivas pudessem conter o quantificador existencial, o programa referenciado seria apenas hipotético, e não saberíamos o formato de sua chamada, assim ficaria ao encargo do usuário suprir o programa com essa informação.

Extração do conteúdo semi-computacional (CSC)

O conteúdo *semi-computacional* (CSC) de uma fórmula é o conjunto de informações que descrevem as aplicações das operações sobre os objetos utilizados pelo programa, e é baseado nas marcações das variáveis e termos. Ele será utilizado na verificação se o comando associado à aplicação de

uma regra de inferência realmente reflete o CSC da conclusão desta regra.

Definição 1 – consideremos *S* um conjunto de fórmulas e *Ho* um conjunto de constantes que ocorrem em *S*. Se nenhuma constante ocorre em *S*, então *Ho* é formado por uma simples constante, isto é, $Ho = \{a\}$. H_{i+1} $(i = 0,1,2,...)$ é a união de H_i com o conjunto de todos os termos da forma $f^n(t^1, ..., t^n)$ para todas as funções n-árias (f) que ocorrem em *S*, onde t_j $(j = 1, ..., n)$ 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: $H_\infty = \cup_{i \in |N} Hi$ é donimado como ***Universo de Herbrand de S*** [CL73].

Definição 2 – sendo *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 $h_1, ..., h_n$ elementos de *H*. Em *I*, f é uma função que mapeia $(h_1, ..., h_n)$ (um elemento de H^n), em f $(h_1, ..., h_n)$ (um elemento de H^n).

I será uma **Interpretação de Herbrand** de *S* [CL73].

Definição 3 – Sendo *S* um conjunto de fórmulas, uma interpretação de *Herbrand* que satisfaz *S* é denominada como **Modelo de Herbrand** de *S*.

Nas definições sobre o CSC que serão apresentadas, temos que: θ - é um conjunto de fórmulas; σ - expressa a configuração de memória, atribuições de valores às variáveis, sobre as quais determinada propriedade é válida; Γ - é o conjunto de axiomas que descrevem a teoria, onde a solução do problema (conclusão da prova) é representada; $a^\frown C$ – expressa a concatenação do elemento *a* com o objeto *C*; e $C^\frown a$ – expressa a concatenação do objeto *C* com o objeto *a*.

Considerando-se que M é modelo de *Herbrand* (H), ou seja, uma estrutura sobre o universo de *Herbrand* para a linguagem $L(\Gamma)$[7], tal que:

$$M \underset{H}{|=} \Gamma^{V}_{T}$$

Abaixo apresentamos as definições $CSC^{\theta}_{M}(\alpha^{V}_{T})$, de acordo com a estrutura de α.

(i) Fórmula atômica:

$$CSC^{\theta}_{M}(A^{V}_{T}) = \left\{ \langle nil, \vec{b}, \vec{o} \rangle, nil \rangle \;/\; \begin{array}{l} \forall v_i \in V,\ \forall b_i \in \vec{b}\ e\ \forall \sigma,\ (\sigma(v_i) = b_i) \\ e \\ \forall t_i \in T\ e\ \forall o_i \in \vec{o}, [[t_i]]\sigma = o_i \\ e \\ Se\ M \underset{H,\sigma}{|=} \theta\ então\ M \underset{H,\sigma}{|=} A^{V}_{T} \end{array} \right\}$$

(ii) Quantificador Universal ($\forall$)

$$CSC^{\theta}_{M}(\forall x \alpha(x)^{V}_{T}) = \left\{ \forall b_i \left[\langle b_i \frown L, \vec{b}, \vec{o} \rangle, w \rangle \;/\; \langle L, \langle b_i \frown \vec{b}, \vec{o} \rangle, w \rangle \in CSC^{\theta}_{M}(\alpha(h)^{V}_{T \cup h}) \right] \right\}$$

(iii) Quantificador Existencial ($\exists$)

$$CSC^{\theta}_{M}(\exists x \alpha(x)^{V}_{T}) = \left\{ \langle L, \vec{b}, \vec{o} \rangle, w \frown o_i \rangle \;/\; \langle L, \vec{b}, \vec{o} \frown o_i \rangle, w \rangle \in CSC^{\theta}_{M}(\alpha(h)^{V}_{T \cup h}) \right\}$$

(iv) Conjunção ($\wedge$)

$$CSC^{\theta}_{M}((\alpha \wedge \beta)^{V}_{T}) = \left\{ \langle L_1 \frown L_2, \langle b, o \rangle, w_1 \frown w_2 \rangle \;/\; \begin{array}{l} (K_\alpha = \langle L_1, \vec{b}, \vec{o} \rangle, w_1 \rangle) \in CSC^{\theta}_{M}(\alpha^{V}_{T}) \\ e \\ (K_\beta = \langle L_2, \vec{b}, \vec{o} \rangle, w_2 \rangle) \in CSC^{\theta}_{M}(\beta^{V}_{T}) \end{array} \right\}$$

---

[7] O modelo de Herbrand para as fórmulas marcadas é o mesmo para as não marcadas.

(v) Disjunção ($\vee$)

$$CSC^{\theta}_{M}((\alpha \vee \beta)^{V}_{T}) = CSC^{\theta}_{M}(\alpha^{V}_{T}) \cup CSC^{\theta}_{M}(\beta^{V}_{T})$$

(v) Implicação ($\rightarrow$)

$$CSC^{\theta}_{M}((\alpha \rightarrow \beta)^{V}_{T}) = CSC^{\theta}_{M}(\beta^{V}_{T})$$

(vi) Absurdo intuicionista ($\perp$)

$$CSC^{\theta}_{M}(\perp^{V}_{T}) = \{\}$$

Definição 4 – $U \subseteq CSC^{\theta}_{M}(\alpha^{V}_{T})$, U é completo se e somente se: $\forall k_1 \in U$, tal que $k_1 = \langle L, \langle \vec{b}', \vec{o}' \rangle, w \rangle$ e $\forall k_2 \in U$, tal que $k_2 = \langle L', \langle \vec{b}', \vec{o}' \rangle, w' \rangle$ temos que: $L = L'$, $\vec{o} = \vec{o}'$ e $w = w'$

Definição 5 – Seja: $k = \langle L, \langle \vec{b}, \vec{o} \rangle, w \rangle$, onde $\vec{b}, \vec{o} \in M$ (modelo de Herbrand). Temos que Λ (um programa) calcula k se e somente se:

$$\vdash_{Hoare} \{in = L \wedge (v_1 = b_1, \ldots, v_n = b_n)\} \Lambda \{(t_1 = o_1, \ldots, t_m = b_m) \wedge out = W\}$$

Onde $V = \{v_1, \ldots, v_n\}$ e $T = \{t_1, \ldots, t_m\}$

Definição 6 - $\theta \models\models \Lambda: (\alpha^{V}_{T})$ se e somente se:
$\exists U$ completo $\subseteq CSC^{\theta}_{M}(\alpha^{V}_{T})$, tal que, $\forall u \in U$, o programa Λ calcula u.

3. Prova de corretude do processo de síntese

Utilizando o método proposto, obtemos um programa que tem a mesma semântica da prova, sendo que a garantia de sua correção será apresentada através da prova de que o mesmo cumpre a sua especificação. O processo de síntese de programas é composto de duas partes: a marcação das entradas (variáveis) e saídas (termos) das fórmulas da prova e a extração do

conteúdo computacional da prova, que serão referenciadas, respectivamente, pelas funções *MarcaVarTermos* e *GeraProg*.

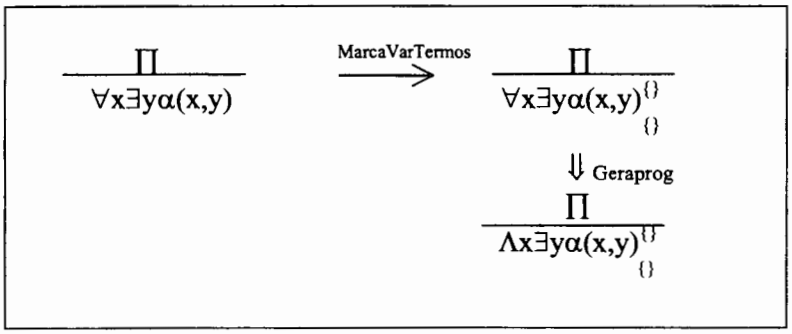


Figura 2 – Esquema de construção do programa

Quando o processo de síntese obtém sucesso, devemos provar que o programa gerado satisfaz a especificação, isto é, provar a sua corretude. Para isso, utilizaremos o seguinte lema, para a prova do teorema que garante a correção do programa gerado:

Lema: Seja M um modelo de *Herbrand* para Δ, sendo este um conjunto de axiomas e Π uma prova da fórmula:

$\overset{\Delta}{\alpha}$: Π e Π' = *GeraProg(MarcaVarTermo*(Π)). Sendo $(\Pi_1 \prec \Pi)$[8]

então:

a) Um subconjunto CSC' do CSC de uma fórmula é dito completo se e somente se todo ρ ∈ CSC' possui os mesmos arquivos com a lista de valores de entrada, as mesmas lista dos valores de saída em memória e os mesmos arquivos com a lista dos valores de saída; e

b) $\theta \models\!= \Lambda : \alpha^{Y}_{T}$ se e somente se existe um CSC' completo contido ou igual ao CSC de uma fórmula, tal que, para todo ρ ∈

[8] $\Pi_1 \prec \Pi$ expressa que Π_1 é uma derivação contida em Π.

CSC', o programa Λ realiza a transformação de entrada e saída expressa por ρ.

Prova: A prova do lema foi realizada por indução no tamanho da prova, através da comparação da semântica da alteração sintática do programa com o conteúdo semi-computacional das regras de inferência.
Parte a) Este é garantido pela correção do cálculo intuicionista.
Parte b)
Caso base: Seja Δ um conjunto de axiomas

1. *Axiomas*

Pela regra de extração do conteúdo semi-computacional relacionado aos axiomas, temos que estes não possuem programas relacionados, pois expressam somente conteúdo lógico. Pelas hipóteses do lema, M é um modelo de *Herbrand* para eles.

$$\models_{M} \sigma : \Delta$$

2. *Hipóteses (não axiomas)*

a) *Hipóteses que possuem coneúdo lógico* – Pela parte "a" do lema temos que:

$$\beta_i \models_{M,\sigma} \sigma : (\beta_i)^V{}_T$$

b) *Hipóteses que possuem conteúdo semi-computacional (hipótese indutiva)*. Pelas hipóteses do lema, essas são corretas por construção, logo $\Delta \models \rho_i : \delta$.

Caso Indutivo[9]:

[9] As regras semânticas dos comandos utilizados são semelhantes as apresentadas em [HW72].

Observação: Nas provas a serem apresentadas, utilizaremos um abuso de notação: δ_i ao invés de $\rho_i : \delta_i$.

- *Introdução do quantificador universal*

Supondo que a regra de introdução do quantificador universal seja a última a ser aplicada em uma derivação D:

$$D = \left\{ \begin{array}{l} \Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k \\ \vdots \\ \Lambda : \alpha(h)^{V \cup h}_{T} \\ \hline \\ read\ (h);\ \Lambda : \forall y \alpha(y)^{V}_{T} \end{array} \right.$$

Pela hipótese indutiva: $\Delta, \beta_1,..., \beta_n, \delta_1, ..., \delta_k \models \Lambda:\alpha(h)^{V \cup h}_{T}$ se e somente se: $\exists\, U$ completo $\subseteq CSC^{\theta}_{M}(\alpha(h)^{V \cup h}_{T})$, tal que $\forall k \in U$ sendo $k = \langle L, h \overset{\frown}{\rightarrow} b, o \rangle, W \rangle$, temos que:

$\vdash_{Hoare}$ $\{in = L \wedge (v = b) \wedge (h = b_i)\}\ \Lambda\ \{(t = o) \wedge out = W\}$ onde $b_i \in b$ e $i = 1 \ldots n$

O resultado da composição do comando *read* (...) com o programa possui a seguinte regra semântica:

(1)

$$\frac{\{in = \langle b_i \rangle L \wedge (\overrightarrow{v=b}) \wedge b_i = b_i\} / \{read(h)\ \{in = \langle b_i \rangle L \wedge (\overrightarrow{v=b}) \wedge h = b_i\}\ \{in = L \wedge (\overrightarrow{v=b}) \wedge (h = b_i)\} \Lambda \{(t = o) \wedge out = W\}}{\{in = \langle b_i \rangle L \wedge (\overrightarrow{v=b}) \wedge b_i = b_i\}\ read(h);\ \Lambda\ \{(\overrightarrow{t=o}) \wedge out = W\}}$$

Dado que:

$$CSC^{\theta}_{M}(\forall x \alpha(x)^{V}_{T}) = \left\{ \langle b_i \frown L, \overrightarrow{b, o}, W \rangle \Big/ \forall b_i \in M\left(\langle L, \langle b_i \overset{\frown}{\rightarrow} b, o \rangle, W \rangle \in CSC^{\theta}_{M}(\alpha(h)^{V}_{T}) \right) \right\}$$

Se $U' \subseteq CSC^{\theta}_{M}(\forall y \alpha(y)^{V}_{T})$, então: $U' = \{\langle b_i \frown L, \overrightarrow{b, o}, W \rangle\} / \{\langle L, \langle b_i \overset{\frown}{\rightarrow} b, o \rangle, W \rangle \in U\}$

Como U' é formado a partir de U, temos que U' é completo.

Pela hipotese indutiva o programa Λ calcula K_2.

Seja $K_1 \in U', K_1 = \{\langle b_i \cap L, \langle \vec{b}, \vec{o} \rangle, W \rangle$, tal que:
$K_2 = (\langle L, \langle b_i \cap \vec{b}, \vec{o} \rangle, W \rangle \in CSC^{\theta}{}_{M}(\alpha(h)^{V}{}_{T}))$

Assim, por (1), temso que (*read(h)*; Λ) calcula K_1.
Sendo K_1 uma tripla arbitrária pertencente a U', podemos concluir que: $\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k \models read(...); \Lambda : \forall y \alpha(y)^{V}{}_{T}$

• *Eliminação do quantificador existencial*

Supondo que a regra de eliminação do ∃ seja a última a ser aplicada em uma derivação D:

$$D = \begin{cases} \dfrac{\begin{array}{cc} \Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k & \alpha \leftarrow exec(\Lambda, \vec{v}) : [\alpha(\alpha)^{V}{}_{T \cup \alpha}] \\ \vdots & \vdots \\ \Lambda : \exists y \alpha(y)^{V}{}_{T} & T : \gamma^{V'}{}_{T'} \end{array}}{T : \gamma^{V'}{}_{T'}} \end{cases}$$

podemos observar que o comando associado a aplicação desta regra é o comando *exec*(...), que prepara o ambiente para a execução da função, isto é, atribui para as variáveis de entrada do programa os valores de entrada passados por parâmetro. Além disso, realiza a chamada da função e retorna o último termo de saída após a execução de Λ.

Este pela hipótese indutiva:
1. $\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k \models \Lambda : \exists y \alpha(y)^{V}{}_{T}$, se e somente se:

$\exists U_1$ completo $\subseteq CSC^{\theta}_{M}(\exists y\alpha(y)^{V}_{T})$, tal que $\forall k \in U$ sendo $k = < L, < \vec{b}, \vec{o} >, w^{\frown}o_i >$, temos que:
$\vdash_{Hoare} \{in=L \wedge (\vec{v} = \vec{b})\} \Lambda \{(\vec{t} = \vec{o}) \wedge out = W\langle o_i\rangle\}$, onde $o_i \in \vec{o}$ e i = 1,..., n

2. $\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k, \models T : \gamma^{V'}_{T}$, se e somente se:
$\exists\ U_2$ completo $\subseteq CSC^{\theta}_{M}(\gamma^{V'}_{T})$, tal que $\forall k \in U_2$ sendo $k = < L, < \vec{b}, \vec{o} >, W >$, temos que:
$\vdash_{Hoare} \{in=L \wedge (\vec{v} = \vec{b})\} \wedge \{(\vec{t} = \vec{o}) \wedge out = W\}$, onde $o_i \in \vec{o}$ e i = 1,..., n

Na hipótese indutiva 2 temos que δ_j está associada ao seguinte programa: $\alpha \leftarrow exec\ (\Lambda, \vec{v})$

O comando *exec*(...) possui a seguinte regra semântica:

$$\frac{\{in=L \wedge (\vec{v} = \vec{o})\} \Lambda \{(\vec{t} = \vec{o}) \wedge out = W\langle o_i\rangle\}}{\{in=L \wedge (\vec{v} = \vec{o})\}\ Z \leftarrow exec(\Lambda, \vec{v})\ \{(Z = o_i) \wedge (\vec{t} = \vec{o}) \wedge out = W\}}$$

Dado que:
$CSC^{\theta}_{M}(\exists y\alpha(y)^{V}_{T}) = \{\langle L, \langle \vec{b}, \vec{o}\rangle, w^{\frown}o_i\rangle \mid \langle L, \langle \vec{b}, o^{\frown}o_i\rangle, w\rangle \in CSC^{\theta}_{M}(\alpha(h)^{V}_{T\cup h})\}$
Pela hipótese indutiva (2):

$\langle L, \langle \vec{b}, \vec{o}\rangle, W\rangle \in U_2$ a partir da suposição que $b_i \leftarrow exec(\Lambda, \vec{v})$, tal que $b_i \in \vec{b}$.

Pela hipótese indutiva (1) o programa Λ calcula o $CSC^{\theta}_{M}(\exists y\alpha(y))$, que é descrito pela tripla: $k = \langle L, \langle b,o\rangle, w^{\frown}o_i\rangle$.

Podemos observar pela semântica do comando *exec*(...), que este retorna o último termo que seria escrito no arquivo de saída, logo, $exec(\Lambda,V) = o^{\Lambda}_{i}$.

Logo, $CSC^{\theta}_{M}(\gamma^{V'}_{T}) = \langle L, \langle \vec{b}, \vec{o}\rangle, W\rangle$, onde $b_i = o^{\Lambda}_{i}$ tal que: $\vec{b}_i \in b$ e $o^{\Lambda}_{i} \in U_1$.

Portanto, sendo k uma tripla arbitrária pertencente a U_1, podemos concluir que:

$$\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k, \underset{M, \sigma}{\models} T : \gamma^{V'}{}_T$$

• *Regra da Indução*

Supondo que a regra da inferência da indução seja a última a ser aplicada em uma derivação D:

$$D = \left\{ \begin{array}{l} \qquad\qquad\qquad\qquad \Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k\ [p : \alpha(a)^{V \cup a}{}_T] \\ \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ \qquad\qquad\qquad\qquad\qquad\qquad p : \exists x \beta(x)^{\vec{V}}{}_T \\ \Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k\ [b \prec l]^{V'}{}_T \qquad q \leftarrow exec(p, \vec{v}) : \beta(q)^{V''}{}_T \\ \qquad\qquad \vdots \qquad\qquad\qquad\qquad \vdots \qquad\qquad\qquad \vdots \\ \Lambda : \alpha(b)^{V \cup b}{}_T \qquad\qquad\qquad \Psi : \alpha(c)^{V \cup c}{}_T \qquad\qquad a \prec c^{V}{}_T \\ \hline \textbf{Procedure Rec } \{ \textbf{ read(x);} \\ \qquad\qquad \textbf{If (x < l) then } \{\Lambda\} \\ \qquad\qquad \textbf{else} \qquad\qquad\qquad\qquad : \forall y \alpha(y)^{V}{}_T \\ \qquad \{[\Psi \Leftarrow \{\textbf{exec(p, } \vec{v}\textbf{) = Rec}\}^*] \\ \qquad\qquad \} \end{array} \right.$$

onde, β é uma subfórmula de $\alpha(a)$, b é o termo relacionado com o caso base, c é o termo relacionado com o passo indutivo, x é a variável de entrada, a qual será igual a b ou a c, e "*" representa a substituição do programa hipotético pela chama da recursiva.

Pela hipótese indutiva:

1. $\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k, y \prec l \models \Lambda : \alpha(b)^{V \cup b}{}_T$, se e somente se:

   $\exists\ U_l$ completo $\subseteq CSC^{\theta \cup (y \prec l)}{}_M(\alpha(b)^{V \cup b}{}_T)$, tal que $(\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k) = \theta$ e $\forall\ k \in U_l$, sendo $k = < L, \vec{b}, o >$, $W >$, temos que: $\vdash \{in = L \wedge (v = b) \wedge b = b_i\} \Lambda \{(t = o) \wedge out = W\}$ onde $b_i \in \vec{b}$ e $i = 1..., n$

2. $\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k, p{:}\alpha(a) \models \Psi{:}\alpha(c)^{V \cup c}{}_T$, se e somente se:

$\exists\ U_2$ completo $\subseteq CSC^{\theta\cup(a)}_{M}(\alpha(c)^{V\cup c}_{T})$, tal que $(\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k) = \theta$ e $\forall\ k \in U_2$ sendo $k = < L, < c^{\frown} b, \vec{o} >, W >$, temos que: $\vdash \{in = L \wedge (\vec{v} = \vec{b}) \wedge c = b_i\} \Psi \{(\vec{t} = \vec{o}) \wedge out = W\}$ onde $b_i \in \vec{b}$ e $i = 1..., n$

Pela derivação D, notamos que as conclusões $\alpha(b)$ e $\alpha(c)$ são provadas a partir de hipóteses exclusivas, por isso iremos associar um comando condicional a essas conclusões.

Desta forma, o comando gerado terá a seguinte regra semântica:

$$\frac{\{in = L \wedge (\vec{v} = \vec{b}) \wedge (b \to l) \wedge b = b_i\} \Lambda\ \{(\vec{t} = \vec{o}_i) \wedge out = W\}\ \{in = L \wedge (\vec{v} = \vec{b}) \wedge \neg(v_x \succ l) \wedge c = b_i\} \Psi \{(\vec{t} = \vec{o}) \wedge out = W\}}{\{in = L \wedge (\vec{v} = \vec{b}) \wedge x = \vec{b}_i\}\ if\ (x \to l)\ then\ \{\Lambda\}\ else\ \{\Psi\}\ \{(\vec{t} = \vec{o}) \wedge out = W\}}$$

onde $x = c$ (quando $x \geq 1$) ou $x = b$ (quando $x < 1$).

Pela hipótese indutiva, da prova de $\alpha(b)$ com a hipótese $(b \prec l)$ extraímos o $CSC^{\theta\cup(y\prec l)}_{M}(\alpha(b)^{V\cup b}_{T})$, e da prova de $\alpha(c)$ com $\alpha(a)$ extraímos o $CSC^{\theta\cup\alpha(a)}_{M}(\alpha(c)^{V\cup c}_{T})$. Como $(b \prec l)$ e $\alpha(a)$ são hipóteses exclusivas, temos que $CSC^{\theta\cup(y\prec l)}_{M}(\alpha(b)^{V\cup b}_{T}) \cup CSC^{\theta\cup\alpha(a)}_{M}(\alpha(c)^{V\cup c}_{T}) = CSC^{\theta}_{M}(\alpha(c)^{V\cup x}_{T})$.

Logo, se $U' \subseteq CSC^{\theta}_{M}(\alpha^{V}_{T})$, então $U' = U_1 \cup U_2$.

Já que U' é formado a partir de U_1 e U_2, temos que U' é completo, pois a lista dos valores de entrada de U_1 (que está relacionado com os elementos menores que l) e de U_2 (que está relacionado com os elementos maiores ou iguais a l) são disjuntas.

Portanto:

(1) $\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k \models_{M,\sigma} If\ (x \prec l)\ then\ \{\Lambda\}\ else\ \{\Psi\}\ \alpha(x)^{V\cup x}_{T}$

O resultado da composição do comando *read* (...) com o comando condicional gerado acima possui a seguinte regra semântica:

(2)

$$\frac{\{in = L \wedge (\vec{v} = \vec{b}) \wedge x = b_i\}}{\{in = (b_i) L \wedge (\vec{v} = \vec{b}) \wedge b_i = b_i\} read(x) \{in = L \wedge (\vec{v} = \vec{b}) \wedge x = b_i\}\ \ if(x \to l)\ then\ \{\Lambda\} else\ \{\Psi\}}$$

$\{(t = o) \wedge out = W\}$

$\{in = \langle b_i\rangle L \wedge (\vec{v} = \vec{b}) \wedge b_i = b_i\}$ read(x); if $(x \prec l)$ then $\{\Lambda\}$else $\{\Psi\}\{(\vec{t} = \vec{o}) \wedge out = W\}$

Dado que:

$CSC^{\theta}_{M}(\forall x\alpha(x)^{V}_{T}) = \{\langle bi^{\frown}L, \vec{b}, \vec{o}\rangle, W\rangle / \forall b_i \in M\langle L, \langle b_i{}^{\frown}\vec{b}, \vec{o}\rangle, W\rangle \in CSC^{\theta}_{M}(\alpha(y)^{V\cup y}{}_{T})\}$

Se $(U_{\forall} \subseteq CSC^{\theta}_{M}(\forall y\alpha(y)^{V}{}_{T})$, então $U_{\forall}$: $\{\langle b_i{}^{\frown}L, \vec{b}, \vec{o}\rangle, W\rangle / \langle L, \langle b_i{}^{\frown}\vec{b}, \vec{o}\rangle, W\rangle \in U'\}$

Como $U_{\forall}$ é formado a partir de U', temos que $U_{\forall}$ é completo.

Seja $K_1 \in U_{\forall} K_1 = \langle b_i{}^{\frown}L, \vec{b}, \vec{o}\rangle, W\rangle$, tal que: $K_2 = \langle L, \langle b_i{}^{\frown}\vec{b}, \vec{o}\rangle, W\rangle \in (U' \subseteq CSC^{\theta}_{M}(\alpha(y)^{V\cup\alpha}{}_{T})$

Por (1), temos que o programa : *If (x <l) then* $\{\Lambda\}$ else $\{\Psi\}$ calcula K_2.

Assim, por (2), temos que *(read(x); If (x < l) then* $\{\Lambda\}$ *else* $\{\Psi\}$ calcula K_1. Sendo K_1 uma tripla arbitária pertencente a $U_{\forall}$, podemos concluir que:

$\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k \models$ *read (x); If (x < l) then {Λ} else {Ψ}:* $(\forall x\alpha(x)^{V}{}_{T})$

No programa acima gerado, será adicionada uma marcação (*Procedure...*) de forma que o comando *read* (...) juntamente com o condicional será o corpo de um procedimento.

Teremos assim o seguinte programa associado à conclusão $(\forall y\alpha(y)^{V}{}_{T})$:

Procedure Rec {
  read (x);
  if (x < l) then {Λ} else {Ψ(Rec)}
}

Pela hipótese indutiva (2) $(\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k p : \alpha(a) \models \Psi : \alpha(c)^{V\cup c}{}_{T})$: $L, \langle c^{\frown}\vec{b}, \vec{o}\rangle, w\rangle \in U_2$ a partir da suposição que p: $\alpha(a)^{V\cup a}{}_{T}$.

Por construção, temos que o programa suposto, associado com a variável de programa ρ, calcula $CSC^{\theta}_{M}(\alpha(a)^{V\cup a}{}_{T})$

Esse programa está associado à hipótese indutiva *(α(a))*, a qual pode possuir um quantificador existencial em sua fórmula. Desta forma, podemos ter no corpo do programa Ψ a marcação: " ... ← *exec*(*p*, ...)". Na construção do programa essa marcação será substituída pela chamada do programa associado a ela, que neste caso (regra da indução), será o próprio programa construído. Assim, teremos uma chamada recursiva.

Dado a semântica da chamada recursiva:

$$\frac{\{in = L \wedge (\vec{v} = \vec{b})\}\, Z \leftarrow exec(\Lambda, \vec{v})\{(Z = o_i) \wedge (\vec{t} = \vec{o}) \wedge out = W\} \;\vdash_{Corpo\,(\Lambda) = \Psi}^{\{in = L \wedge (v = b)\}\Psi\{(t = o) \wedge out = W\}}}{\{in = L \wedge (\vec{v} = \vec{b})\}\, Z \leftarrow \Lambda\, \{(Z = o_i) \wedge (\vec{t} = \vec{o}) \wedge out = W\}}$$

temos que o programa associado à hipótese indutiva é o ponto fixo do programa gerado. Assim, podemos afirmar que o programa gerado é recursivo.

Portanto:

$$\Delta, \beta_1, ..., \beta_n, \delta_1, ..., \delta_k \mid= \begin{cases} \text{Procedure Rec} \\ \quad \text{read } (x); \\ \quad \text{if } (x < l) \text{ then } \{\Lambda\} \quad : \forall y \alpha(y)^{V}{}_{T} \\ \quad \text{else} \\ \quad \{[\Psi \Leftarrow \{\text{exec}(p, \vec{v}) = \text{Rec}\}^*]\} \end{cases}$$

A prova das outras regras de inferência é feita de forma análoga das regras que foram apresentadas.

Teorema: Sejam Π uma prova para uma fórmula da forma $\forall x \exists y \alpha(x,y)$, a partir de um conjunto de axiomas (Δ) e um conjunto de hipóteses que não são axiomas (θ), e Λ o programa resultante da função *GeraProg(MarcaVarTermos*(Π)), então: θ, $\Delta \underset{M,\sigma}{\models} \Lambda : \forall x \exists y \alpha(x,y)^{\{\}}{}_{\{\}}$

Prova: Aplicando o lema anterior teremos a prova deste teorema.

4. Exemplo

Essa seção procura mostrar o nosso mecanismo de síntese

construtiva através de um exemplo no qual é gerado um programa que calcula o resto de uma divisão.

Devido ao tamanho da árvore de prova, esta e seus ramos serão apresentados em blocos. O bloco da prova principal, isto é, que contém a raiz da árvore de prova (teorema a ser provado), terá uma moldura sólida; os que apresentam moldura tracejada representam os ramos que estão conectados a outros através da numeração apresentada ao lado esquerdo na parte superior da moldura.

Exemplo:

Árvore de prova:

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots\\ \alpha(a) & \beta(a)\end{array}}{\alpha(a)\wedge\beta(a)}\,I\wedge}{\forall x(\alpha(x)\wedge\beta(x))}\,I\forall$$

Representação em blocos

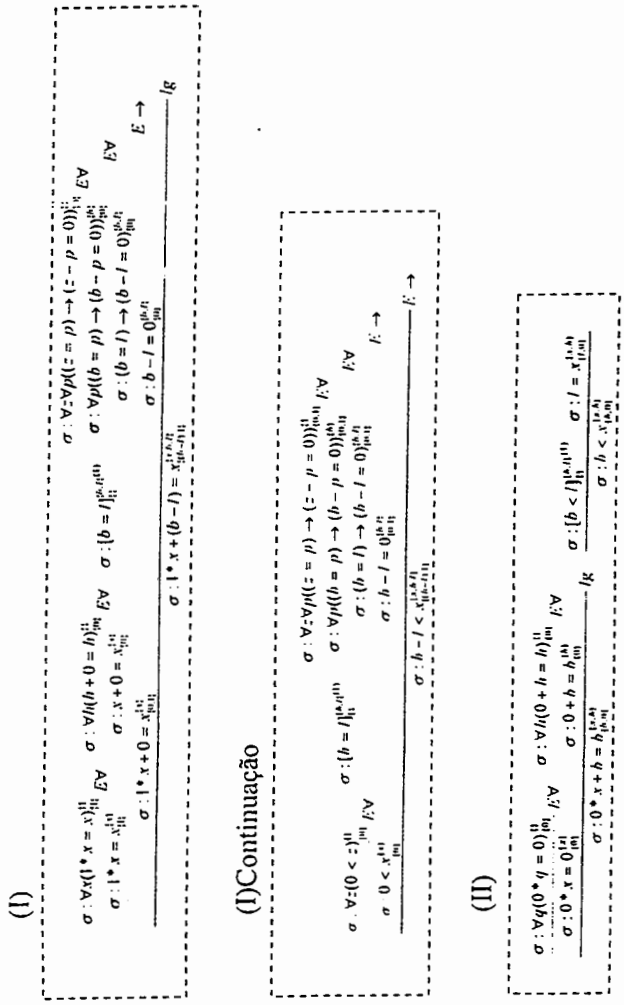
(I) $\begin{array}{cc}\vdots & \vdots\\ \alpha(a) & \beta(a)\end{array}$

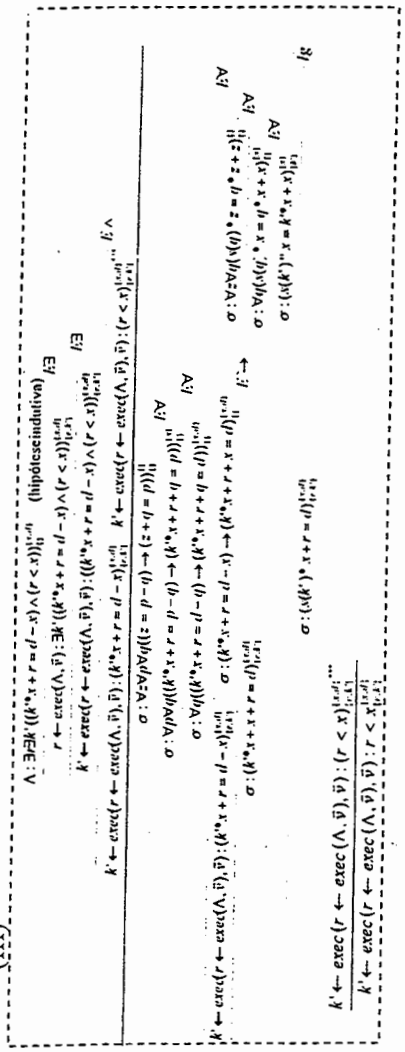
$$\dfrac{\dfrac{(I)}{\alpha(a)\wedge\beta(a)}\,I\wedge}{\forall x(\alpha(x)\wedge\beta(x))}\,I\forall$$

Com o objetivo de facilitar o entendimento da prova, não foi utilizada a notação pré-fixada para as operações de adição, subtração e multiplicação, nem os predicados de igualdade e das relações de menor e de maior. O funcional s(x) expressa a operação de sucessor.

O programa será gerado a partir da prova do teorema: $\forall v \forall u \exists r \exists k((k*v + r = u) \wedge (r < v))$ com a utilização dos axiomas: $\forall x(x*1=x)$, $\forall x(x+0=x)$, $\forall q(0*q=0)$, $\forall z(0<z)$, $\forall z \forall p(z=p) \rightarrow (z-p=0)$, $\forall z \forall p(p>0) \rightarrow (z-p<z))$, $\forall z \forall q((z*s(q)) \rightarrow (z*q+z)$, $\forall z \forall p \forall q((z=p-q) \rightarrow (z+q=p))$, e das hipóteses: $y>0$, $1 = y$.

Exemplo da extração do conteúdo semi-computacional da especificação do cálculo do resto de uma divisão:





(III)

** Para a realização da prova foi inserida a seguinte hipótese: $y=/$, que representa uma restrição de memória, onde $/$ possui o valor de y.

5. Conclusão

Neste trabalho, apresentamos um método automático de síntese de programas onde dado uma prova de uma especificação, em lógica intuicionista de predicados poli-sortida utilizando como sistema dedutivo a dedução natural, obtemos um programa a partir do cálculo do conteúdo semi-computacional e do mapeamento das regras de inferência com comandos da linguagem imperativa. Também apresentamos a prova que o programa gerado cumpre a especificação.

Devido as restrições sintáticas impostas para a prova[10], da qual extraímos o conteúdo semi-computacional, provavelmente houve a perda do poder de expressão da linguagem lógica, o que pode acarretar a perda de completude do processo de síntese.

Dentre as principais contribuições deste trabalho, podemos citar a proposta de um novo sintetizador capaz de gerar programas legíveis em uma linguagem imperativa, cujo mapeamento **possui a prova de corretude**. Apesa de no estágio atual os programas gerados terem um aspecto funcional, podemos observar que não é possível encontrar na literatura nenhuma abordagem similar, já que os trabalhos existentes tratam da geração de programas em linguagens funcionais e lógicas ilegíveis. Ainda podemos citar o fato de que, no processo de síntese construtiva de programas proposto, é fornecido como entrada uma especificação em lógica de predicados. Isto permite expressarmos os problemas de forma mais simples do que os sintetizadores que **utilizam** teoria intuicionista dos tipos (**Nuprl**[CIU98], **Oyster** [BSW90] e **NJL** [G79]) e lógica equacional (**Lemma** [CM98]).

Como projetos futuros, propomos o estudo da possibilidade de extrair conteúdo semi-computacional de provas que utilizam as regras de introdução da negação e que possuam como hipóteses fórmulas que contenham o conectivo do

[10] Vide seção 2.

quantificador existencial; e a geração de programas em que as leituras e saídas de dados sejam feitas via memória principal.

Abstract

The constructive synthesys or proof-as-program [BC85] is based on the Curry-Howard isomophism [H69] and on the fact that a constructive proof, for a theorem that describes a problem, can be seen as a description of a problem. In this work, we present a method of construtive program synthesys where a program (in an imperative language) is generate by a proof in many-sorted intuitionistic logic using as a deductive system the Natural Deduction. Using the concept of semi-computacional content of a formula, we will show that the generated program is a representation of the solution to the problem especified by the theorem, in a theory that describes the data types used.

Keywords: Intuitionistic Logic, Constructives Synthesis.

Referências Bibliográficas

[BBSSZ98] BENL, H., BEGER, U., SCHWICHTENBERG, H., SEISENBERGER, M. And ZUBER, W. Proof theory at work: Program development in the Minlog system, *Automated Deduction,* W. Bibeland P.H. Schimitt, eds., Vol II, Kluwer 1998.

[BC85] BATES, J.L. and CONSTABLE, R.L. – "Proof as Programs". *ACM Transactions on Programming Languagens and Systems*, 7(1): 113 – 136, 1985.

[BSW90] BUNDY, A, SMAIL, A, and WIGGINS, G.A – *"The synthesis of logic programs from inductive proofs"*. In J. Lloyd, editor Computational Logic, pags 135 – 149. Springer-Verlag, 1990. Esprit Basic Research Series. Also available from

Edinburgh as DAI Research.
[CIU98] CALDWELL, J.L., IAN, P., UNDERWOOD, J.G. – "Search algorithms in Type Theory". http://meru.cs.uwyo.edu/~jlc/papers.html.

[CL73] CHAG, C., LEE, R.C – *"Symbolic Logic and Mechanical Theorem Prover"*. Academic Press. 1973.

[CM98] CHARARAIN, J. e MULLER, S. – "Automated synthesis of recursive programs from a ∀∃ logical specification". *Journal of Automated Reasoning* 21: 233-275, 1998.

[G79] GOTO, S. – "Program synthesis form natural deductions proofs". *International Joint Conference on Artificial Intelligence,* 339-341. Tokyo 1979.

[GLT89] GIRARD, J., LAFONT, Y. And TAYLOR, P. – *Proof and Types.* Cambrigde University Press, 1989.

[H69] HOWARD, W.A – "The Formulae-as-Types Notion of Construction". Encontrado em Hindley, J.R., Seldin, J.P., *To H.B. Curry: Essays on combinatory logic, Lambda Calculus and Formalisation.* Academic Press, 1980.

[HW72] HOARE, C.A R and WHIRTH, N. – "An axiomatic Definition of the Programming Language PASCAL". – December, 1972. *Acta Informatica* 2, 335-355, Springer-Verlag 1973.

[V84] VELOSO, P.A.S. – "Outlines of a mathematical theory of general problems". *Philosophia Naturalis*, 21(2/4): 234-362, 1984.

[S99] SILVA, G.M.H. – "Um Estudo em Síntese Construtiva de Programas utilizando Lógica Intuicionista". Dissertação de Mestrado – Departamento de Informática – PUC – Rio, 1999.