Sobre o Teorema da Modularização: Importância e uma Prova por Quociente

Haydée Werneck Poubel*
Paulo Augusto Silva Veloso
Departamento de Informática
PUC - RJ
Rua Marquês de São Vicente, 225
22453-900 Gávea - Rio de Janeiro

Resumo

Especificações são tratadas como apresentação de teorias, ou seja, teorias em linguagens poli-sortidas de primeira ordem definidas por um conjunto de axiomas.

Discutimos o papel fundamental desempenhado pelo Teorema da Modularização no processo de implementação pelo paradigma do passo canônico e na passagem de parâmetros nas especificações parametrizadas. Apresentamos uma demonstração do Teorema da Modularização que introduz uma nova forma de se construir teorias: a teoria quociente induzida por uma interpretação.

Abstract

Specifications are treated as theory presentation, i.e., theories in many-sorted first-order languages presented by a set of axioms.

We discuss the fundamental role played by the Modularization Theorem in the process of implementation by means of the canonical step paradigm and in parameter instantiation of parameterized specifications. We also present a proof of the Modularization Theorem that introduces a new concept: the quocient theory induced by an interpretation.

1 Introdução

O Teorema da Modularização (TM) desempenha papel fundamental no processo de refinamentos sucessivos, sendo uma ferramenta básica para a composição de passos canônicos de implementação, bem como na instanciação de parâmetros em especificações parametrizadas ([6], [15]). Este resultado também é importante na aplicação de métodos de resolução de problemas e na formulação de modelos precisos para o processo de desenvolvimento de programas ([5], [11], [16], [17]).

Neste trabalho discute-se e ilustra-se a importância desse resultado e apresenta-se uma demonstração, baseada em uma técnica nova, quociente de teorias, a qual parece ser de interesse independente.

*Em licença do Departamento de Matemática da UnB

Sabemos que as motivações para a proposta lógica de se lidar com especificações formais vêm principalmente de duas fontes: a proximidade conceitual para a verificação de programas ([7]) e a construção de programas ([6] e [15]).

Conforme discutido e ilustrado na seção 2, o TM permite que o desenvolvimento formal de especificações e de programas seja feita pela composição de passos de refinamento. Além disso, esse resultado indica de maneira natural e direta, porém precisa, como se processa a instanciação de parâmetros [14].

É importante ressaltar que essas composições e instanciações são feitas automaticamente, sem a necessidade de nenhum esforço extra por parte do programador. Daí decorre a importância de demonstrações do TM: versões construtivas indicam algoritmos para a sua aplicação. Mais ainda, outras demonstrações aumentam o elenco de alternativas, possibilitando a escolha em cada caso. Trata-se, portanto, de um caso claro onde a importância de uma demonstração transcende sua finalidade, matemática, de estabelecer o resultado, pois indica caminhos para implementações , algumas mais eficientes que as outras. Uma indicação disto é o fato de que esse resultado fornece a base lógica para um dos sistemas, desenvolvido no Kestrel Institute, mais sofisticados e bem sucedidos de desenvolvimento de programas por transformações ([10]).

A importância (de alguma versão) da Propriedade da Modularização, para preservar a estrutura modular, tem sido notada por vários pesquisadores ([2], [3], [14], [6]). Sua nova demonstração, apresentada aqui, por se basear no conceito de quociente de teoria, tem a virtude de tratar o caso geral, poli-sortido, da mesma maneira simples e natural do caso mono-sortido, conforme apontado em [9].

2 Importância do Teorema da Modularização

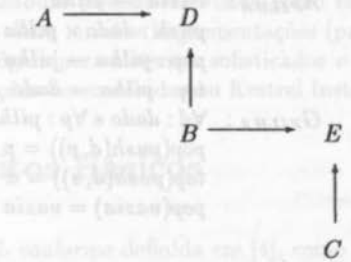
Consideremos inicialmente o problema da implementação de uma especificação abstrata A em uma especificação mais concreta C, conforme descrito em [6]. Queremos obter um módulo que represente os objetos de A em termos daqueles de C, e operações e predicados de A por procedimentos usando operações e predicados de C.

Em termos de especificações formais, ou seja, teorias apresentadas por um conjunto de axiomas, em geral precisamos estender a especificação concreta C adicionando símbolos que correspondam àqueles de A bem como alguns outros símbolos auxiliares. Obtemos assim uma nova especificação D que não deve acrescentar qualquer propriedade aos símbolos de C, já que C estava pré-definida. Tal propriedade de uma extensão D a caracteriza como extensão conservativa, [9]. Além disso, queremos fazer uma correspondência dos símbolos de A para os de D, $A \xrightarrow{i} D$, e essa correspondência deve preservar as propriedades de A, ou seja, todas as consequências dos axiomas de A têm que ser consequência dos axiomas de D. Tal propriedade caracteriza i como uma interpretação de teorias, [9].

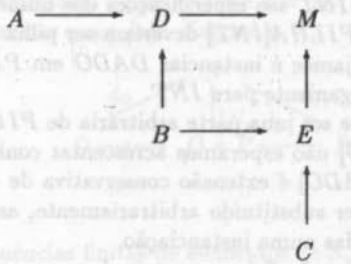
Assim formalizamos o conceito de implementação de A em C como uma interpretação de A em uma extensão conservativa, D, de C. Graficamente temos:

$$\begin{array}{ccc} A & \xrightarrow{\ i\ } & D \\ & & \uparrow e \\ & & C \end{array}$$

Tomando o processo de implementação feito por sucessivas aplicações do "passo canônico de implementação" ([6]) mostrado acima, é necessário compor tais passos. Dadas as implementações



gostaríamos de obter uma especificação M que fosse uma extensão conservativa de E na qual pudéssemos interpretar D. Portanto, queremos obter



O TM garante que tal especificação M, construída como se deve esperar, ou seja, juntando-se as especificações obtidas nos dois passos envolvidos, é realmente um extensão conservativa de E.

Agora, consideremos um dos mais antigos objetivos dos trabalhos de pesquisas em especificações formais: blocos padronizados a partir dos quais são construídas especificações maiores e que possam ser reaproveitadas em diferentes situações. Em outras palavras, podemos dizer que a especificação está estruturada em uma parte "parâmetro" e uma parte "contexto", ou ainda um pouco mais formalmente, que temos uma especificação parametrizada ([2]).

Suponha dada uma especificação X (parâmetro) e uma especificação S (parâmetro + contexto) que estende X. Desde que X está "embutido" em S e pode eventualmente ser substituido por uma outra especificação Y, podemos ver S como uma especificação parametrizada por X.

Tome por exemplo a especificação

$$DADO = S_{DADO} : dado$$

e a extensão

$$PILHA[DADO] = DADO + PILHA \text{ onde}$$

$$\begin{aligned} PILHA = \\ S_{PILHA}: & \ pilha \\ A_{PILHA}: & \ vazia :\rightarrow pilha \\ & \ push : dado \times pilha \rightarrow pilha \\ & \ pop : pilha \rightarrow pilha \\ & \ top : pilha \rightarrow dado \\ G_{PILHA}: & \ \forall d : dado \ e \ \forall p : pilha \\ & \ pop(push(d,p)) = p \\ & \ top(push(d,s)) = d \\ & \ pop(vazia) = vazia \end{aligned}$$

Embora o par $(DADO, PILHA[DADO])$ seja uma especificação parametrizada vemos que $PILHA$ não é uma especificação .

Agora, gostaríamos de instanciar $DADO$ por vários parâmetros e ainda obter uma especificação . Então, se NAT e INT são especificações dos números naturais e inteiros, respectivamente, $PILHA[NAT]$ e $PILHA[INT]$ deveriam ser pilha de naturais e pilha de inteiros, respectivamente. O que desejamos é instanciar $DADO$ em $PILHA[DADO]$ por NAT para obter $PILHA[NAT]$, e analogamente para INT.

Note que $DADO$ não pode ser uma parte arbitrária de $PILHA[DADO]$. Ao estendermos $DADO$ para $PILHA[DADO]$ não esperamos acrescentar conhecimentos novos a respeito de $DADO$, ou seja, $PILHA[DADO]$ é extensão conservativa de $DADO$. Isso não significa que o parâmetro $DADO$ possa ser substituído arbitrariamente, as propriedades da especificação $DADO$ têm que ser preservadas numa instanciação.

Formalmente, uma especificação S é *parametrizada* por uma sub-especificação X (chamada *parâmetro*) se e só se S é uma extensão conservativa de X, e uma *instanciação de parâmetro* é uma interpretação $X \xrightarrow{p} Y$. Graficamente temos:

$$\begin{array}{ccc} X & \rightarrowtail & S \\ {\scriptstyle p}\downarrow & & \\ Y & & \end{array}$$

Novamente, pelo TM podemos construir uma especificação M que seja uma extensão conservativa de Y e na qual interpretamos S, ou seja, obtemos o diagrama comutativo

$$\begin{array}{ccc} X & \rightarrowtail & S \\ {\scriptstyle p}\downarrow & & \downarrow \\ Y & \longrightarrow & M \end{array}$$

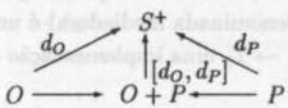
Observamos, que conforme nos sugeriu o exemplo, $PILHA[NAT]$ e $PILHA[INT]$ são instanciações de $PILHA[DADO]$, que são extensões conservativas de NAT e INT, respectivamente.

Uma outra área, relacionada, de aplicação dessas idéias e técnicas é a de formalização de conceitos gerais de problema (como um tipo abstrato de dados) e de métodos de resolução de problemas (tais como redução, divisão e conquista, programação dinâmica, etc.) [13], [16], [17], [11], [12]. Pois, estes métodos podem ser vistos como especificações parametrizadas, e sua aplicação a um problema dado envolve implementações (parametrizadas) e instanciações. Estas idéias formam a base lógica para sistemas sofisticados e bem sucedidos de refinamento de programas por transformações desenvolvidos no Kestrel Institute [10].

3 Alguns Conceitos Básicos

Entendemos uma *linguagem* L conforme definida em [4], como uma linguagem poli-sortida de primeira ordem com seus conjuntos de símbolos lógicos e extra-lógicos. Há várias maneiras de precisar esta idéia. Aqui usaremos a seguinte.

Denotaremos $L =< S, A, d >$, onde S é o conjunto de sortes, $A = O + P$ é o co-produto[1] dos conjuntos de símbolos de operações e predicados e $d = [d_O, d_P]$ é a única função declaração dos símbolos de A que faz comutar o seguinte diagrama:

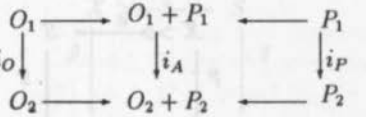


onde S^+ é o conjunto das sequências finitas de elementos de S.

Uma linguagem L' é uma *sublinguagem* de L quando L pode ser obtida de L' por adição de alguns símbolos extra-lógicos. Notação: $L' \subseteq L$.

Uma *teoria* T consiste de todas as consequências de um conjunto de sentenças sobre uma linguagem L_T. Uma *apresentação de uma teoria* T consiste de uma linguagem L_T e um conjunto de axiomas G_T que são sentenças de L_T. Denotaremos $T =< L_T, G_T >$. Uma *especificação* é a apresentação de uma teoria.

Uma *tradução* i de uma linguagem $L_1 =< S_1, A_1, d_1 >$ em uma linguagem $L_2 =< S_2, A_2, d_2 >$ é um par de funções (i_S, i_A) onde $A_1 \xrightarrow{i_A} A_2$ é dada por i_A, o morfismo soma do co-produto a seguir



e $S_1 \xrightarrow{i_S} S_2$ são tais que o seguinte diagrama comuta:

[1] Na categoria dos conjuntos o "co-produto" (único a menos de isomorfismos) de dois conjuntos O e P, denotado por $O + P$, é dado pela "união disjunta" $O \times \{1\} \cup P \times \{2\}$ (ver [1]).

$$\begin{array}{ccc} A_1 & \xrightarrow{i_A} & A_2 \\ {\scriptstyle d_1}\downarrow & & \downarrow{\scriptstyle d_2} \\ S_1^+ & \xrightarrow[i_S^+]{} & S_2^+ \end{array}$$

com i_S^+ o homomorfismo natural que estende i_S a sequências.

Dadas duas especificações $T_1 =< L_1, G_1 >$ é e $T_2 =< L_2, G_2 >$, entendemos uma *interpretação* $T_1 \xrightarrow{i} T_2$ como uma interpretação entre teoria como definida em [4], onde i é uma tradução da linguagem L_1 na linguagem L_2 tal que para cada sentença $\alpha \in L_1$, se $G_1 \vdash \alpha$ então $G_2 \vdash i(\alpha)$ ($\vdash$ é a relação de dedução na lógica considerada).

Dizemos que uma especificação $T_2 =< L_2, G_2 >$ é uma *extensão* de uma especificação $T_1 =< L_1, G_1 >$ se $L_1 \supset L_2$ e se $G_1 \vdash \alpha \in L_1$ então $G_2 \vdash \alpha$. Denotaremos uma extensão por $T_1 >\!\!\longrightarrow T_2$. Uma extensão T_2 de uma especificação T_1 é *conservativa* quando para todas as sentenças $\alpha \in L_1$, se $G_2 \vdash \alpha$ então $G_1 \vdash \alpha$. Denotaremos extensão conservativa por $T_1 >>\!\!\longrightarrow T_2$.

Entendemos uma *implementação* de uma especificação T_1 em uma especificação T_2 como uma tripla (T_3, i, e) onde T_3 (denominada mediadora) é uma especificação tal que $T_1 \xrightarrow{i} T_3$ e $T_2 >>\!\xrightarrow{e}\!\!\!\!\!\!\longrightarrow T_3$. Denotaremos $T_1 \rightsquigarrow T_2$ uma implementação de T_1 em T_2.

4 O Teorema da Modularização

Em geral, dado um diagrama

$$\begin{array}{ccc} X & >\!\!\!\!\!\!\longrightarrow^{e} & S \\ {\scriptstyle p}\downarrow & & \\ Y & & \end{array}$$

onde e e p são interpretações, ele pode ser completado para um diagrama comutativo

$$\begin{array}{ccc} X & >\!\!\!\!\!\!\longrightarrow^{e} & S \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ Y & \xrightarrow[h]{} & T \end{array}$$

que também é um pushout ([2]).[2]

[2]O conceito geral e formal de um "diagrama pushout" pode ser encontrado em [1]. A relevância desse conceito para nossos propósitos, segundo [2], é devida à precisão com que o objeto T é construído pela "combinação" dos objetos S e Y identificando as partes desses, dadas por X, p e e. O pushout fornece o objeto "minimal" T, e os morfismos g e h fornecem as transformações de S e Y, respectivamente, em T.

Aqui em particular, a interpretação e é uma extensão conservativa. Nesse caso, usando a demonstração dada em [2] temos que g é uma extensão de p pela identidade, ou seja, $g = (g_S, g_A)$ é definida por

$$g_S = \begin{cases} p_S & \text{em } S_X \\ 1_S & \text{em } S_S - S_X \end{cases}$$

e

$$g_A = \begin{cases} p_A & \text{em } A_X \\ 1_A & \text{em } A_S - A_X \end{cases}$$

h é uma extensão e $T =< L_Y + L_S, G_Y \cup g(G_S) >$, onde $+$ denota um pushout de

$$\begin{array}{ccc} L_X & \xrightarrow{e} & L_S \\ {\scriptstyle p}\downarrow & & \\ L_Y & & \end{array}$$

O Teorema da Modularização que demonstraremos a seguir, nos garante que h é também conservativa se e o for. Esse resultado é obtido por meio de uma técnica nova, quociente de teorias, através do conhecido *Lema da Interpolação de Craig* na versão formulada como segue:

Lema da Interpolação de Craig : Sejam as linguagens de primeira ordem L_1 e L_2, e as teorias T_1 e T_2 definidas, respectivamente, em cada uma dessas linguagens. Dada uma sentença $\alpha_2 \in L_2$ tal que $T_1 \cup T_2 \vdash \alpha_2$, existe um conjunto de sentenças $\Delta \subseteq L_1 \cap L_2$ tal que

$$T_1 \vdash \delta \text{ para cada } \delta \in \Delta$$
$$\Delta \cup T_2 \vdash \alpha_2$$

Essa formulação do Lema de Craig aparece em [8], e é denominada "splitting interpolation".

Teorema 4.1 *(Modularização): Dado o diagrama pushout*

$$\begin{array}{ccc} X & \overset{e}{\twoheadrightarrow} & S \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ Y & \xrightarrow[h]{} & T \end{array}$$

a interpretação h é uma extensão conservativa.

Demonstração: Sabemos que $G_T = G_Y \cup g(G_S)$.

Suponha $\alpha_Y \in L_Y$ tal que $G_Y \cup g(G_S) \vdash \alpha_Y$.

Pelo *Lema da Interpolação de Craig* ([8]) existe um conjunto Δ de sentenças com $\sum(\Delta) \subseteq \sum(g(G_S)) \cap \sum(G_Y)$, onde $\sum(A)$ representa o conjunto de todos os símbolos que ocorrem nas sentenças de A, tal que

$$g(G_S) \vdash \Delta \text{ (i.e. } g(G_S) \vdash \delta \text{ para cada } \delta \in \Delta),$$
$$\Delta \cup G_Y \vdash \alpha_Y$$

Desde que $\sum(G_Y) \subseteq L_Y$ e todos os símbolos de L_Y que estão em $g(G_S)$ são traduções de símbolos de L_X por p, temos $\Delta = p(I)$ para algum $I \subseteq L_X$. Então $g(G_S) \vdash p(I)$ e $p(I) \cup G_Y \vdash \alpha_Y$

Gostaríamos de mostrar que $G_Y \vdash p(I)$ e portanto teremos $G_Y \vdash \alpha_Y$ que é o resultado desejado, ou seja, h é uma extensão conservativa.

Suponhamos inicialmente que a interpretação $g = (g_A, g_S)$ seja injetiva, isto é, g_A e g_S sejam injetivas e portanto $p = (p_A, p_S)$ é injetiva.

Desde que $T =< L_Y + L_S, G_Y \cup g(G_S) >$, $G_S \xrightarrow{g} g(G_S)$ é sobre.

Assim, para cada $\gamma_S \in G_S$ temos $g(\gamma_S) \in g(G_S)$.

Logo, existe $g^{-1} : g(S) \longrightarrow S$ que é uma interpretação, pois $\alpha \in g(G_S)$ se e só se $g^{-1}(\alpha) \in G_S$.

De $g(G_S) \vdash p(I) = g(I), I \subseteq L_X$ temos $G_S \vdash I, I \subseteq L_X$.

Desde que ϵ é conservativa, $G_X \vdash I$ e como p é interpretação, segue $G_Y \vdash p(I)$.

Tratemos agora do caso geral em que $g = (g_A, g_S)$ não precisa ser injetiva e portanto $p = (p_A, p_S)$ também não.

Vamos mostrar que a partir do diagrama

$$\begin{array}{ccc} X & \overset{\epsilon}{\twoheadrightarrow} & S \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle g} \\ Y & \xrightarrow[h]{} & T \end{array}$$

podemos construir um diagrama comutativo de interpretações

$$\begin{array}{ccc} \dot{X} & \overset{\epsilon'}{\twoheadrightarrow} & \dot{S} \\ {\scriptstyle \dot{p}}\downarrow & & \downarrow{\scriptstyle \dot{g}} \\ Y & \xrightarrow[h]{} & T \end{array}$$

onde $\dot{g}$ é injetiva, e portanto concluir que h é extensão conservativa.

Defina os seguintes conjuntos de símbolos e sortes:

$\dot{A}_X = A_X / R_{A_X}$ onde $R_{A_X} = \{(a,b) \in A_X \times A_X \mid p_A(a) = p_A(b)\}$

$\dot{S}_X = S_X / R_{S_X}$ onde $R_{S_X} = \{(r,q) \in S_X \times S_X \mid p_S(r) = p_S(q)\}$

Defina $\dot{A}_X \xrightarrow{\dot{d}_X} (\dot{S}_X)^+$ por $\dot{d}_X([a]) = [s_1] \cdots [s_k]$ onde $d(a) = s_1 \cdots s_k$

Temos portanto uma linguagem $\dot{L}_X =< \dot{S}_X, \dot{A}_X, \dot{d}_X >$

Considere a tradução de linguagens $t = (t_A, t_S)$ dada por:

$$\begin{array}{cc} A_X \xrightarrow{t_A} \dot{A}_X & S_X \xrightarrow{t_S} \dot{S}_X \\ a \longmapsto [a] & r \longmapsto [r] \end{array}$$

Defina $\dot{G}_X = t(G_X)$. Portanto, $\dot{G}_X$ é um conjunto de sentenças de $\dot{L}_X$.

Lema 4.1 *Sejam φ, β fórmulas de L_X.*
$p(\varphi) = p(\beta) \Leftrightarrow t(\varphi) = t(\beta)$

Seja a especificação $\dot{X} =< \dot{L}_X, \dot{G}_X >$. Por construção $X \xrightarrow{t} \dot{X}$ é uma interpretação, e toda sentença $\dot{\alpha} \in \dot{L}_X$ é da forma $\dot{\alpha} = t(\beta)$ para alguma sentença $\beta \in L_X$.

Seja o conjunto de sentenças

$K_t = \{(\alpha \leftrightarrow \delta) \in L_X \mid t(\alpha) = t(\delta)\}$

Por indução no comprimento da dedução pode-se mostrar o

Lema 4.2 *Para cada* $\beta \in L_X$

$\dot{G}_X \vdash t(\beta) \Leftrightarrow G_X \cup K_t \vdash \beta$

Do mesmo modo defina $S \xrightarrow{t'} \dot{S}$ com $\dot{S} =< \dot{L}_S, \dot{G}_S >$.

Pelas construções mostra-se o seguinte:

Lema 4.3 *Para cada* $\beta \in L_X$, $t(\beta) = t'(\beta)$

e novamente por indução no comprimento da dedução temos:

Lema 4.4 *Para cada* $\beta \in L_X$

$\dot{G}_S \vdash t(\beta) \Leftrightarrow G_S \cup K_t \vdash \beta$

Segue imediato do Lema 4.3 que $\dot{X} \xrightarrow{\epsilon'} \dot{S}$ é uma extensão, ou seja, se $\alpha \in L_X$ e $\alpha \in G_X \subseteq G_S$ então temos que $t(\alpha) \in \dot{G}_X$ e $t'(\alpha) = t(\alpha) \in \dot{G}_S$

Do Lema 4.4 segue que $\dot{X} >\!\!>^{\epsilon'}\!\!\longrightarrow \dot{S}$:

De fato, seja $t(\beta) \in \dot{L}_X$ $(\beta \in L_X)$ tal que $\dot{G}_S \vdash t(\beta)$

$$\Rightarrow G_S \cup K_t \vdash \beta$$
$$\Rightarrow G_S \vdash (K_t \rightarrow \beta)^3$$

Desde que K_t só tem sentenças de L_X e $X >\!\!>^{\epsilon}\!\!\longrightarrow S$

$$\Rightarrow G_X \vdash (K_t \rightarrow \beta)$$
$$\Rightarrow G_X \cup K_t \vdash \beta$$
$$\Rightarrow \dot{G}_X \vdash t(\beta) \text{ pelo Lema 4.2.}$$

Defina agora $\dot{L}_X \xrightarrow{\dot{p}} L_Y$ e $\dot{L}_S \xrightarrow{\dot{g}} L_Y \cup L_S$ por:

$\dot{p} = (\dot{p}_A, \dot{p}_S)$ *onde* $\dot{A}_X \xrightarrow{\dot{p}_A} A_Y$ e $\dot{S}_X \xrightarrow{\dot{p}_S} S_Y$

$[a] \longmapsto p_A(a)$ $[r] \longmapsto p_S(r)$

$\dot{g} = (\dot{g}_A, \dot{g}_S)$ *onde* $\dot{A}_S \xrightarrow{\dot{g}_A} A_Y \cup A_S$ e $\dot{S}_S \xrightarrow{\dot{g}_S} S_Y \cup S_S$

$[a'] \longmapsto g_A(a')$ $[r'] \longmapsto g_S(r')$

Pela construção, é fácil ver que $\dot{p}$ e $\dot{g}$ são traduções de linguagens.

Por definição, para cada fórmula $\alpha \in L_X$ e $\gamma \in L_S$ temos $\dot{p}(t(\alpha)) = p(\alpha)$ e $\dot{g}(t'(\gamma)) = g(\gamma)$.

Assim, se $\alpha \in G_X$, $t(\alpha) \in \dot{G}_X$ e portanto $G_Y \vdash p(\alpha) \Longrightarrow G_Y \vdash \dot{p}(t(\alpha)) \Rightarrow \dot{p}$ é uma interpretação.

Analogamente obtemos que $\dot{g}$ é interpretação.

É fácil verificar que o diagrama

[3] Por $K_t \rightarrow \beta$ entenda-se $k \rightarrow \beta$ onde k é uma conjunção finita de sentenças de K_t

$$\begin{array}{ccc} \tilde{X} & \xrightarrow{e'} & \tilde{S} \\ {\scriptstyle \tilde{p}}\downarrow & & \downarrow{\scriptstyle \tilde{g}} \\ Y & \xrightarrow[h]{} & T \end{array}$$

comuta, já que os diagramas abaixo comutam.

$$\begin{array}{ccc} \tilde{A}_X & \xrightarrow{e'_A} & \tilde{A}_S \\ {\scriptstyle \tilde{p}_A}\downarrow & & \downarrow{\scriptstyle \tilde{g}_A} \\ A_Y & \xrightarrow[h_A]{} & A_Y \cup g(A_S) \end{array} \qquad \begin{array}{ccc} \tilde{S}_X & \xrightarrow{e'_S} & \tilde{S}_S \\ {\scriptstyle \tilde{p}_S}\downarrow & & \downarrow{\scriptstyle \tilde{g}_S} \\ S_Y & \xrightarrow[h_S]{} & S_Y \cup g(S_S) \end{array}$$

Por definição, $\tilde{g} = (\tilde{g}_A, \tilde{g}_S)$ é tal que para cada $[a], [b] \in \tilde{A}_S$ com $\tilde{g}_A([a]) = \tilde{g}_A([b]) \Rightarrow g_A(a) = g_A(b) \Leftrightarrow [a] = [b]$. Similarmente para os sortes.

Portanto $\tilde{g}$ é injetiva, como queríamos. □

5 Outras Aplicações do Teorema da Modularização

Indicamos a seguir alguns conceitos e resultados, úteis para o processo de desenvolvimento de programas, que são consequências do TM.

Pelo TM sabemos que implementações podem ser compostas. Com ele, podemos também mostrar a associatividade dessa composição. Realmente, dadas implementações $A \overset{f}{\rightsquigarrow} B$, $B \overset{g}{\rightsquigarrow} C$ e $C \overset{h}{\rightsquigarrow} D$, pelo TM e propriedades de pushout, temos $(f \circ g) \circ h = f \circ (g \circ h)$, a menos de isomorfismos.

É natural ter uma implementação envolvendo especificações parametrizadas onde a implementação dos parâmetros fica claramente destacada. Um exemplo ilustrando isto é uma implementação de CONJ[DADO] em SEQ[DADO], onde DADO não é implementado, continuando como parâmetro. Um conceito relacionado e mais simples, é o de interpretação parametrizada.

Dadas as especificações parametrizadas $X >\!\!\overset{e}{\longrightarrow} S$, $Y >\!\!\overset{f}{\longrightarrow} T$ dizemos que uma interpretação $S \xrightarrow{h} T$ é uma ***interpretação parametrizada*** quando existir uma instanciação de parâmetro $X \xrightarrow{p} Y$ tal que o diagrama

$$\begin{array}{ccc} X & \xrightarrow{e} & S \\ {\scriptstyle p}\downarrow & & \downarrow{\scriptstyle h} \\ Y & \xrightarrow[f]{} & T \end{array}$$

comuta.

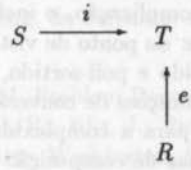
Note que, h determina p pois e e f são extensões.

Segue imediato que a composição de interpretações parametrizadas é uma interpretação parametrizada.

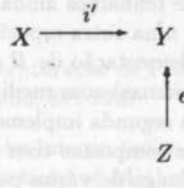
No caso em que o diagrama acima for um pushout temos uma instanciação de parâmetro. No caso em que $Y = X$ e p é a identidade dizemos que temos uma *interpretação de corpo.*

É fácil ver que podemos decompor uma interpretação parametrizada em uma instanciação de parâmetro seguida de uma interpretação de corpo.

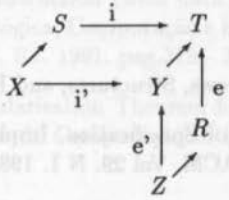
O conceito de implementação parametrizada é uma ligeira generalização do de interpretação parametrizada. Sejam as especificações parametrizadas $X >>\xrightarrow{f} S, Y >>\xrightarrow{g} T$ e $Z >>\xrightarrow{h} R$. O diagrama



define uma *implementação parametrizada* se existir uma implementação



tal que o diagrama a seguir comuta



A composição de implementações parametrizadas é uma consequência do Teorema da Modularização e da propriedade universal do pushout.

Dada uma implementação parametrizada, como acima, dizemos que temos uma *implementação parametrizada instanciada* quando o diagrama horizontal é um pushout (de instanciação) e i é a identidade e *implementação parametrizada do corpo* quando i' e e' são identidades.

Uma implementação parametrizada, como no diagrama acima, pode ser trivialmente decomposta em uma implementação parametrizada instanciada seguida de uma implementação parametrizada do corpo.

6 Conclusões

Na seção 3 apresenta-se uma nova demonstração do TM. A idéia central é identificar símbolos que são igualmente traduzidos ao nível de linguagens e que, portanto, vão tornar naturalmente

equivalentes, a nível de teorias, as sentenças onde ocorrem esses símbolos. Esta é a idéia básica de quociente de teorias, um conceito novo que parece ser de interesse independente. Aqui ele é usado como uma técnica para contornar o problema da falta de fidelidade das interpretações. Note-se que não se pode simplesmente supor que as interpretações são fiéis, pois isto barraria as aplicações pretendidas, uma vez que tanto implementações quanto instanciações de parâmetros envolvem em geral interpretações não fiéis, correspondendo à "adição de detalhes".

As demonstrações anteriores do TM ([18], [19]), apesar de simples no caso mono-sortido, envolvem a adição de novos sortes e funções de conversão em sua extensão ao caso geral, poli-sortido. Isto causa uma certa complicação, e ineficiência, para a automação. A nova demonstração apresenta duas virtudes: do ponto de vista conceitual, trata da mesma maneira simples e natural os casos mono-sortido e poli-sortido, do ponto de vista de aplicação, não envolve a adição de novos sortes com funções de conversão.

A importância deste último fato para a complexidade da implementação final pode ser aquilatada examinando-se os diagramas de composição de implementações na seção 2. Consideremos uma primeira implementação de A em B com mediadora D e uma segunda de B em C com mediadora E. O TM nos fornece uma implementação composta de A em C com mediadora M. Agora, imaginemos que tenhamos ainda um interpretação de C, por exemplo vinda de uma implementação de C em uma outra especificação. A construção da especificação quociente nos permite obter uma implementação de B em C' (mais econômica do que C, no sentido de menos símbolos e menos axiomas) com mediadora E' (mais econômica do que E), que portanto substitui com vantagem a segunda implementação de B em C. Continuando com este processo, obtemos implementações compostas com mediadoras bastante mais econômicas do que antes. O efeito acumulado, ao longo de vários passos, dessas economias em cada passo pode vir a ser bastante substancial.

Referências

[1] Arbib, M. A.; Manes, E. G. Arrows, Structures, and Functors. Academic Press. 1975.

[2] Ehrich, H. -D. On the Theory of Specification, Implementation and Parametrization of Abstract Data Types. Journal ACM. Vol 29. N 1. 1982.

[3] Ehrig, H. and Mahr, B. Fundamentals of Algebraic Specification 1 Equations and Initial Semantics. Springer-Verlag Berlin Heidelberg. 1985.

[4] Enderton, H. B. A Mathematical Introduction to Logic. Academic Press, INC. 1972.

[5] Haeberer, A. M.; Veloso, P. A. S.; Maibaum, T. S. E. Towards Formal Coherent Meta-Models for the Software Development Process. Mon. Em Ciência da Comp. Dep. de Informática. PUC - RJ. N 18. 1992.

[6] Maibaum, T. S. E.; Turski, W. M. The Specification of Computer Programas. Addison-Wesley, Workingham. 1987.

[7] Manna, Z. The Mathematical Theory of Computation. McGraw-Hill, New York. 1974.

[8] Rodenburg, P. H. and Glabbek, R. J. An interpolation Theorem in Equational Logic. Report CS-R8838, Center for Mathematics and Computer Science PO Box 4079, 1009 AB Amsterdam, The Netherlands.

[9] Shoenfield, J. R. Mathematical Logic. Addison-Wesley, Reading. 1967.

[10] Smith, D. R. Constructing Specification Morphisms. Kestrel Institute, Tech. Rept. KES.U.92.1. Palo Alto. April 1992.

[11] Toscani, L. V. e Veloso, P. A. S. A Programação Dinâmica: um caso particular de divisão e conquista. Revista de Informática Teórica e Aplicada. Vol 1. N 2. Maio 1990. p.53-67.

[12] Toscani, L. V. e Veloso, P. A. S. Uma metodologia para cálculo de complexidade de algoritmos. IV Simp. Bras. de Eng. de Software. Águas de São Pedro, SP. 1990. p. 183-192.

[13] Veloso, P. A. S.; Veloso, S. R. M. Problem Decomposition and Reduction: applicability, soundness, completeness. Trappl, R.; Klir, J.; Pichler, F. (eds) Progress in Cybernetics and Systems Research. Hemisphere, Washington, DC. 1981. p. 199-203.

[14] Veloso, P. A. S.; Maibaum, T. S. E.; Sadler, M. R. Programme Development and Theory Manipulation. Proc. International Workshop on Software Specification and Design. London. Ago 1985. p. 228-232.

[15] Veloso, P. A. S. Verificação e Estruturação de Programas com Tipos Abstratos de Dados. Editora Edgard Blücher, São Paulo, SP. 1987.

[16] Veloso, P. A. S. Problem Solving by Interpretation of Theories. Carnielli, W.A.: Alcântara, L.P. (eds). Methods and Applications of Mathematical Logic. American Mathematical Society, Providence. 1988. p. 241-250. Vol 69.

[17] Veloso, P. A. S. Program Construction (with data abstractions) as transformations on theories. Alcoforado, P. (ed). Lógica, Computação e Epistemologia: ensaios em homenagem ao Prof. Jorge Barbosa. ILTC, RJ. 1991. pag.319 - 371.

[18] Veloso, P. A. S. On the Modularisation Theorem for Logical Specifications: its role and proof. Monografias em Ciência da Computação. n 17/92. Dep. Informática-PUC. Rio de Janeiro.

[19] Veloso, P. A. S. On Interpretations of Logical Specifications and the Modularization Theorem. Kestrel Research Inst. Res. Rept. Fev 1993.