PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 39/92

Tipos de Dados Parametrizados: Especificação, Interpretação e Implementação

Haydée Werneck Poubel
Paulo A. S. Veloso

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, Nº 39/92
Editor: Carlos J. P. Lucena Dezembro, 1992

Tipos de Dados Parametrizados: Especificação, Interpretação e Implementação*

Haydée Werneck Poubel
Paulo A. S. Veloso

* Este trabalho foi patrocinado pela Secretaria de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente, 225 — Gávea
22453-900 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br
techrep@inf.puc-rio.br (for publications only)

Tipos de Dados Parametrizados: Especificação, Interpretação e Implementação

Haydée Werneck Poubel*
Paulo A. S. Veloso
PUCRioInf-MCC 39/92

Resumo

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

Apresentamos uma demonstração do Teorema da Modularização, teorema este que desempenha um papel fundamental no processo de desenvolvimento de programas. Nessa demonstração, introduzimos a noção de teoria quociente induzida por um interpretação.

As categorias de especificações parametrizadas e implementações parametrizadas são definidas.

Palavras Chaves : Especificações Parametrizadas, Interpretações Parametrizadas, Categorias, Implementações Parametrizadas

Abstract

Specifications are considered as theory presentations, that is, theories in many-sorted first-order logic defined by a set of axioms.

We present a proof of the Modularisation Theorem, a theorem that plays a fundamental role in the process of program development. In this proof, we introduce the notion of quotient theory induced by an interpretation.

The categories of parameterised specification and parameterised interpretation are defined.

Key Words : Parameterised Specifications,Parameterised Interpretation, Categories, Parameterised Implementations

*Em licença do Departamento de Matemática da UnB

1 Introdução

Especificações Parametrizadas constituem um caso particular, mas importante, no estudo das especificações porque são construídas a partir de uma especificação distinguida denominada parâmetro.

Quando falamos, por exemplo, do tipo abstrato de dados "pilha" podemos pensar em "pilha de naturais", "pilha de inteiros", "pilha de caracteres", ..., etc. É claro que podemos dar as especificações de cada um desses tipos de dados separadamente, mas também podemos ter uma única especificação para "pilha" que tenha um parâmetro, na qual as variações acima são obtidas por uma mudança de parâmetro.

Nesse trabalho discutimos alguns dos aspectos das especificações parametrizadas através de conceitos da Lógica de Primeira Ordem e da Teoria de Categorias. As seções estão separadas da seguinte forma:

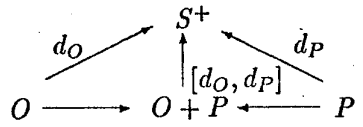
- na seção 2 apresentamos alguns conceitos e introduzimos a notação a ser usada em todo o texto;
- na seção 3 apresentamos uma demonstração do Teorema da Modularização que é equivalente à dada em [4]. A importância desse teorema é porque somente a partir dele é possível tornar o processo de desenvolvimento de programas por meio de implementações, conforme definida no texto, um processo de refinamento por passos;
- na seção 4 definimos o conceito de interpretações parametrizadas, que são morfismos entre especificações parametrizadas, e portanto determinam uma interpretação entre os respectivos parâmetros;
- na seção 5 mostramos a associatividade das implementações e definimos o conceito de implementações parametrizadas, que por sua vez determinam uma implementação entre os respectivos parâmetros.

É importante destacar que nesse trabalho, especificações são consideradas como teorias, e que o processo de desenvolvimento de programas consiste de manipulações de teorias [4].

2 Conceitos Básicos

Entendemos uma *linguagem* L conforme definida em [2], como uma linguagem poli-sortida de primeira ordem com seus conjuntos de símbolos lógicos e extra-

lógicos. Denotaremos $L =< S, A, d >$, onde S é o conjunto de sortes, $A = O + P$ é um co-produto 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:



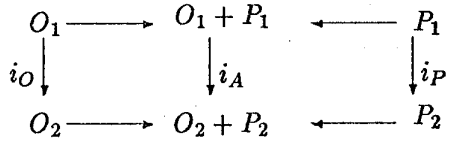
onde S^+ é o conjunto das sequências finitas de elementos de S.

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

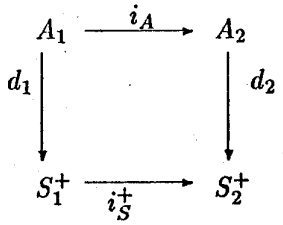
Uma *teoria* T consiste de todas as consequências de um conjunto de sentenças sobre uma linguagem L_T. Uma *apresentação de uma teoria* T consiste de uma linguagem L_T e um conjunto de axiomas G_T que são sentenças de L_T. Denotaremos $T =< L_T, G_T >$.

Uma *especificação* é uma 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 mediador do co-produto a seguir



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



com i_S^+ o homomorfismo natural que estende i_S.

Dadas duas especificações $T_1 =< L_1, G_1 >$ e $T_2 =< L_2, G_2 >$, entendemos uma *interpretação* $T_1 \xrightarrow{i} T_2$ como uma interpretação entre teoria como definida em [2], 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 $T_1 \rightarrowtail T_2$.

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 $T_1 \twoheadrightarrow T_2$.

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 $T_2 \overset{e}{\twoheadrightarrow} T_3$. Denotaremos $T_1 \rightsquigarrow T_2$ uma implementação de T_1 em T_2.

Desde que a composição de interpretações é associativa, consideraremos como em [1], a categoria SPEC cujos objetos são especificações e os morfismos são interpretações.

Seja dada uma especificação X e uma especificação S tal que $X \twoheadrightarrow S$. Desde que X está "embutido" em S e pode eventualmente ser substituido por outra especificação Y tal que $Y \twoheadrightarrow S$, chamaremos X de *parâmetro* e S uma *especificação parametrizada* por X.

É importante notar que em geral, a parte estendida da especificação X em S não é um objeto de SPEC.

Exemplo 1: Seja a especificação

$DADO$ =

S_{DADO} : *dado*

e a extensão conservativa

$PILHADADO = DADO + PILHA$ onde

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

Embora o par $(DADO, PILHADADO)$ seja uma especificação parametrizada vemos que $PILHA$ não é um objeto de SPEC.

Em SPEC temos particulares morfismos que são extensões conservativas. É fácil mostrar que composição de extensões conservativas é uma extensão conservativa. Portanto, especificações e extensões conservativas formam uma subcategoria (não plena) de SPEC.

3 O Teorema da Modularização

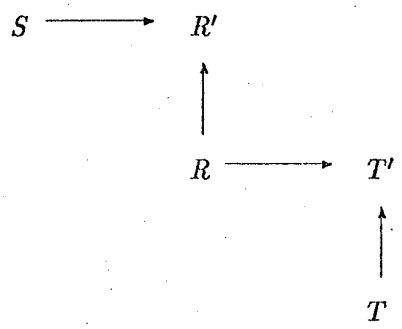
Por definição, uma especificação parametrizada pode ser vista como um par de especificações (X,S) tal que $X \rightarrowtail\!\!\!\!\!\rightarrow S$. Portanto a especificação X está incluída na especificação S.

Isto sugere que podemos pensar em interpretar a especificação dada para o parâmetro X em alguma especificação Y, procedimento usualmente denominado passagem de parâmetro.

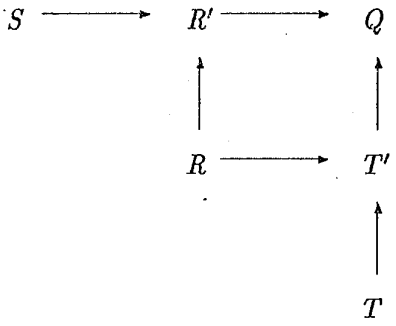
Vamos mostrar que esse processo nos leva a obter uma especificação T tal que $Y \rightarrowtail\!\!\!\!\!\rightarrow T$.

Esse resultado também é fundamental no processo de desenvolvimento de programas desde que ele permite a composição de implementações, determinando assim os passos de refinamento desse processo de desenvolvimento.

Mais precisamente, dadas as implementações



obtemos uma especificação Q que é uma interpretação de R' e é parametrizada por T' tal que

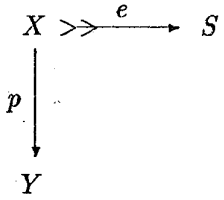


Portanto obtemos uma implementação composta de S em T.

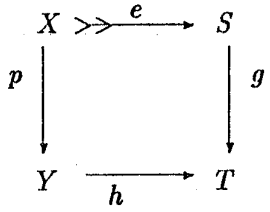
Dada uma especificação parametrizada $X >>\longrightarrow S$ defina:

Def 3.1 *Denominamos instanciação de parâmetro a uma interpretação* $X \xrightarrow{p} Y$.

Desde que SPEC tem pushouts ([1]), o diagrama



pode ser completado pelo pushout, ou seja, o diagrama



é um pushout.

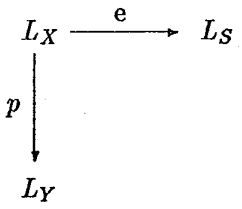
Esse resultado é geral, isto é, vale para interpretações quaisquer. Aqui em particular, a interpretação e é uma extensão conservativa. Nesse caso, usando a demonstração dada em [1] temos que g é uma extensão de p pela identidade, ou seja, $g = (g_S, g_A)$ é definida por

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

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



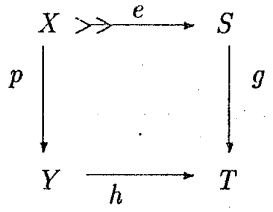
O Teorema da Modularização que demonstraremos a seguir, nos garante que h é também conservativa. Esse resultado é obtido através do conhecido *Lema da Interpolaçào de Craig* na versão formulada como segue:

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 $\alpha_2 \in L_2$ e $T_1 \cup T_2 \vdash \alpha_2$, 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 [3], e é denominada "splitting interpolation".

Teorema 3.1 *(Modularização): Dado o diagrama pushout*



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* ([3]) 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

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

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,

$$\sum(\Delta) \subseteq \sum(g(G_S)) \cap L_Y \Rightarrow \sum(\Delta) \subseteq \sum(p(J))$$

para algum $J \subseteq L_X$.

Seja $\Delta = p(I)$ para algum $I \subseteq L_X$. Então $g(G_S) \vdash p(I)$ e $p(I) \cup G_Y \vdash \alpha_Y$

Gostariamos 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 $T =< L_Y \cup (L_S - L_X), G_Y \cup g(G_S) >$, $G_S \xrightarrow{g} g(G_S)$ é também 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 e é conservativa, $G_X \vdash I \Rightarrow p(G_X) \vdash p(I)$ porque p é interpretação.

Como $G_Y \vdash p(G_X)$ 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

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

podemos construir um diagrama comutativo de interpretações

$$\begin{array}{ccc} \tilde{X} & \overset{e'}{\twoheadrightarrow} & \tilde{S} \\ {\scriptstyle \tilde{p}}\downarrow & & \downarrow{\scriptstyle \tilde{g}} \\ Y & \underset{h}{\longrightarrow} & T \end{array}$$

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

Defina os seguintes conjuntos de símbolos e sortes:

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

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

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

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

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

$A_X \xrightarrow{t_A} \tilde{A}_X \qquad S_X \xrightarrow{t_S} \tilde{S}_X$

$a \longmapsto [a] \qquad r \longmapsto [r]$

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

Lema 3.1 *Sejam φ, β fórmulas de L_X.*
$p(\varphi) = p(\beta) \Leftrightarrow t(\varphi) = t(\beta)$

Seja a especificação $\tilde{X} =< \tilde{L}_X, \tilde{G}_X >$. Por definição $X \xrightarrow{t} \tilde{X}$ é uma interpretação, e toda sentença $\tilde{\alpha} \in \tilde{L}_X$ é da forma $\tilde{\alpha} = t(\beta)$ 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 3.2 *Para cada $\beta \in L_X$*
$\tilde{G}_X \vdash t(\beta) \Leftrightarrow G_X \cup K_t \vdash \beta$

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

Por indução na estrutura das fórmulas mostra-se o seguinte:

Lema 3.3 *Para cada $\beta \in L_X$, $t(\beta) = t'(\beta)$*

e novamente por indução no comprimento da dedução temos:

Lema 3.4 *Para cada $\beta \in L_X$*
$\tilde{G}_S \vdash t(\beta) \Leftrightarrow G_S \cup K_t \vdash \beta$

Segue imediato do Lema 3.3 que $\tilde{X} \xrightarrow{e'} \tilde{S}$ é uma extensão, ou seja, se $\alpha \in L_X$ e $\alpha \in G_X \subseteq G_S$ então temos que $t(\alpha) \in \tilde{G}_X$ e $t'(\alpha) = t(\alpha) \in \tilde{G}_S$

Do Lema 3.4 segue que $\tilde{X} \overset{e'}{>\!\!>\!\!\longrightarrow} \tilde{S}$:
De fato, seja $t(\beta) \in \tilde{L}_X$ $(\beta \in L_X)$ tal que $\tilde{G}_S \vdash t(\beta)$

$$\Rightarrow G_S \cup K_t \vdash \beta$$
$$\Rightarrow G_S \vdash (K_t \rightarrow \beta)^1$$

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

$$\Rightarrow G_X \vdash (K_t \rightarrow \beta)$$
$$\Rightarrow G_X \cup K_t \vdash \beta$$
$$\Rightarrow \tilde{G}_X \vdash t(\beta) \text{ pelo Lema 3.2.}$$

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

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

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

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

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

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

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

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

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

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

É fácil verificar que o diagrama

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

comuta, já que os diagramas abaixo comutam.

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

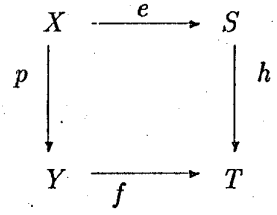
Por definição, $\tilde{g} = (\tilde{g}_A, \tilde{g}_S)$ é tal que para cada $[a], [b] \in \tilde{A}_S$ com $\tilde{g}_A([a]) = \tilde{g}_A([b]) \Rightarrow g_A(a) = g_A(b) \Leftrightarrow [a] = [b]$.

Similarmente para os sortes. Portanto $\tilde{g}$ é injetiva, como queríamos. □

4 Interpretações Parametrizadas

Considere a seguinte

Def 4.1 *Dadas as especificações parametrizadas* $X \overset{e}{>\!\!\!\rightarrow} S$, $Y \overset{f}{>\!\!\!\rightarrow} T$ *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*



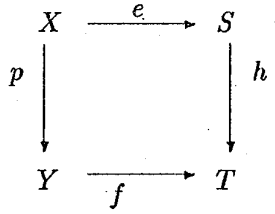
comuta.

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

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

Isso sugere que podemos considerar uma categoria INPAR cujos *objetos* são especificações parametrizadas e cujos *morfismos* são interpretações parametrizadas.

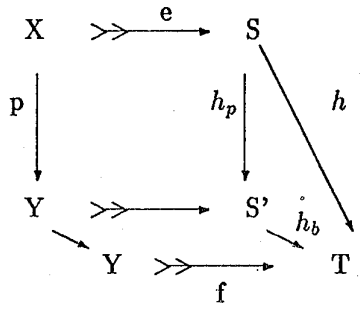
No caso em que o diagrama



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.*

Desde que a categoria SPEC tem pushouts podemos decompor uma interpretação parametrizada por uma instanciação de parâmetro seguida de uma interpretação de corpo. Isto é facilmente reconhecido no seguinte diagrama pushout:

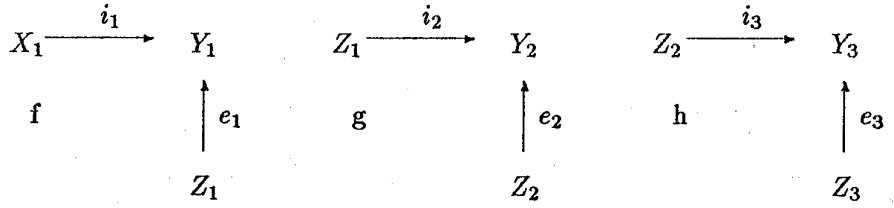


5 Implementações

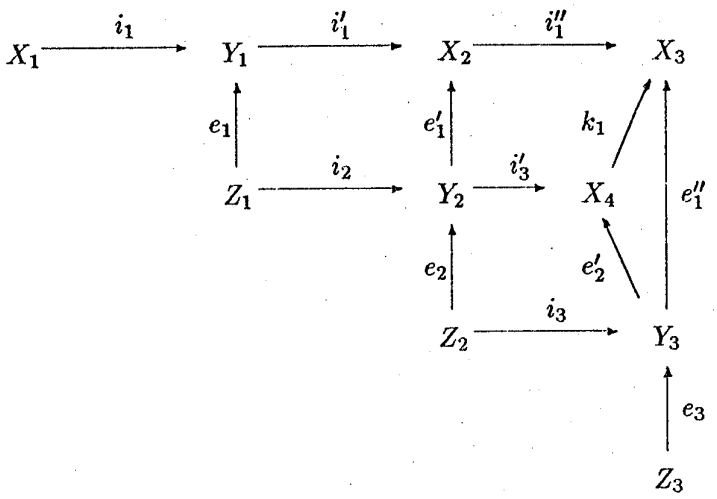
Pelo Teorema da Modularização sabemos que implementações podem ser compostas. Com isso podemos mostrar a associatividade dessa composição.

Teorema 5.1 *A composição de implementações é associativa.*

Demonstração: Sejam as implementações

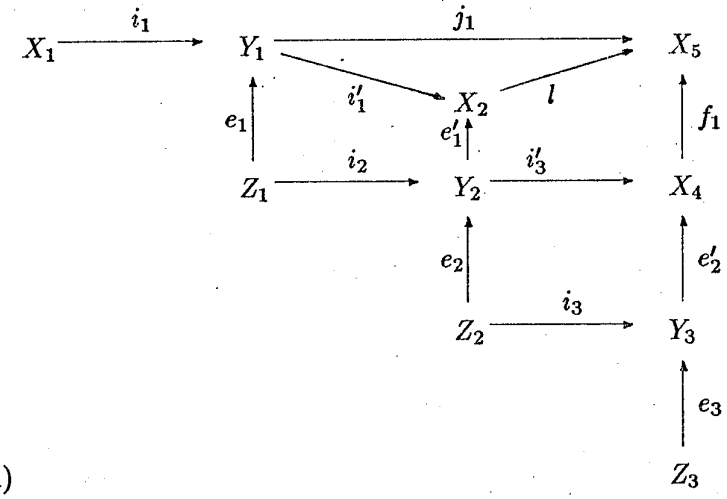


Pelo Teorema da Modularização obtemos os seguintes diagramas pushout:



(i)

que representa $(f \circ g) \circ h$



(ii)

que representa $f \circ (g \circ h)$

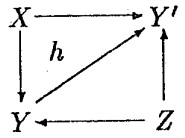
Em (i), (i'_3, e'_2, X_4) é pushout de (i_3, e_2, Z_2) e portanto $\exists!\ k_1 : X_4 \longrightarrow X_3$ tal que $e'_2 \circ k_1 = e"_1$, então $(i"_1, k_1, X_3)$ é pushout de (i'_3, e'_1, Y_2).

Em (ii) (i'_1, e'_1, X_2) é pushout de (i_2, e_1, Z_1) e portanto $\exists!\ l : X_2 \xrightarrow{e} X_5$ tal que $i'_1 \circ l = j_1$, então (e, f_1, X_5) é pushout de (i'_3, e'_1, Y_2).

Logo $(f \circ g) \circ h = f \circ (g \circ h)$ a menos de isomorfismos □

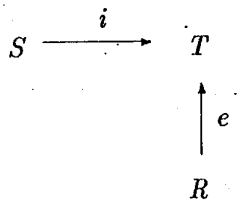
Desse modo podemos definir uma categoria IMPL cujos objetos são especificações e cujos morfismos são implementações. Em IMPL dizemos que duas

implementações $X \overset{f}{\rightsquigarrow} Z$ e $X' \overset{f'}{\rightsquigarrow} Z'$ são equivalentes sob isomorfismos se e somente se $\exists$ um isomorfismo $Y \xrightarrow{h} Y'$ tal que

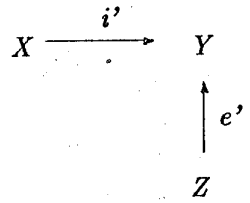


Portanto em IMPL não distinguimos entre implementações com especificações mediadora isomorfas.

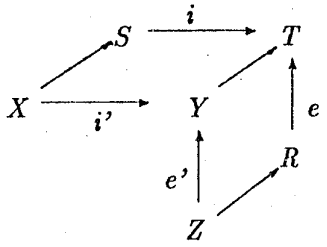
Def 5.1 *Sejam as especificações parametrizadas* $X >\!\!\!\rightarrow^{f} S$, $Y >\!\!\!\rightarrow^{g} T$ *e* $Z >\!\!\!\rightarrow^{h} R$. *O diagrama*



define uma implementação parametrizada se existir uma implementação

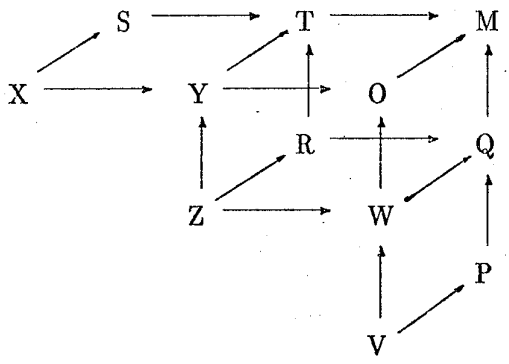


tal que o diagrama a seguir comuta

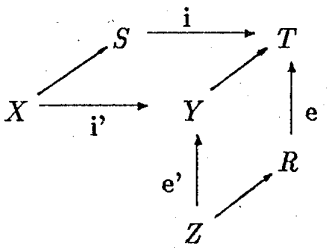


Podemos definir uma categoria IMPAR cujos *objetos* são especificações parametrizadas e cujos *morfismos* são interpretações parametrizadas.

A composição de implementações parametrizadas é uma consequência do Teorema da Modularização e da propriedade universal do pushout, conforme mostrada no diagrama abaixo:

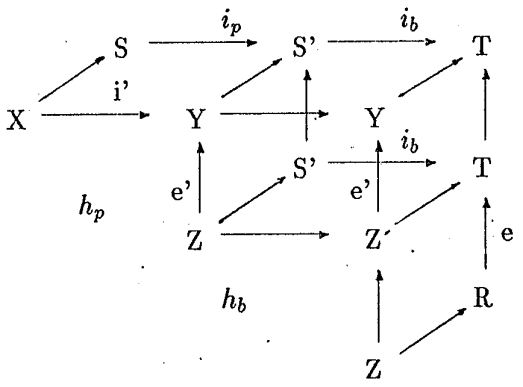


Dada uma implementação parametrizada



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, conforme apresentada no diagrama acima, pode ser trivialmente decomposta em uma implementação parametrizada instanciada h_p seguida de uma implementação parametrizada do corpo h_b. O diagrama abaixo mostra uma tal decomposição:



onde $i_p \circ i_b = i$ é a decomposição da interpretação parametrizada i pelo pushout, vista na seção anterior.

Referências

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

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

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

[4] P. A. S. Veloso. 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.