Some Constructs for Sort Introduction in Logical Specifications

Paulo A. S. VELOSO
e-mail: veloso@inf.puc-rio.br
María Claudia MERÉ
e-mail: mere@inf.puc-rio.br

Abstract

The process of implementing a formal specification on another one is very important in formal software development and it can be described in terms of simple basic logical concepts relating the formal specifications involved.

The **logical approach** to formal specifications regards *specifications* as axiomatic presentations of theories in many-sorted logic, and an *implementation* of a formal specification A on another formal specification C is an interpretation of A into a conservative extension B of C. The idea here is that the "concrete" specification C is extended so as to incorporate concrete versions to which the "abstract" symbols of A can then be mapped. This in turn involves examining how new symbols, together with their properties, can be added to an existing formal specification.

An extension consists of the addition of new symbols and axioms, a simple, but important, special case being the so-called extensions by definitions. The new symbols to be introduced correspond to sorts, functions or predicates. Extensions by the addition of function and predicate symbols are well studied in the literature. In particular, extensions by definitions of function and predicate symbols are characterised by the properties of *conservativeness* together with *eliminability.*

Our purpose here is to analyse **conservative extensions that introduce new sorts**. This is of importance because it occurs often in implementing formal specifications, when new sorts are "constructed" from (or represented in terms of) the concrete ones. For a simple example consider the overused implementation of stacks by arrays with indices. Here one wishes to represent a stack by means of an array, containing the elements stored in the stack, together with an index, indicating the position of the topmost element. Now, the specification of arrays has three sorts: **Arrays, Indices** and **Elements**. One then wishes to extend it by adding a product sort **Arrays x Indices**. But, not every pair $< A, i >$ will represent a stack; only a subsort (consisting of those with $i \leq$ size(A)). Further, pairs with arrays differing only above their indices are to be identified (for they represent the same stack), so one wishes a quotient sort.

We will specify and analyse some well-known sort introduction constructs akin to those found in programming languages, namely cartesian product, discriminated union, subsorting and quotient. This last construct, quotient, even though not prevalent in programming languages, corresponds to the representation of the equality on an abstract sort, which is necessary for a correct implementation.

We will specify each construct by means of a *parameterised formal specification*, in that it is a presentation of a theory in the new, extended language, which is a conservative extension of its parameter: a presentation of a theory in old language which describes its applicability condition.

In the context of implementation of formal specifications, the new sorts introduced should be somehow "connected" to the old ones. This is what our intuition requires and is what happens in the case of the four constructs examined. Also, in each case the extension will be shown to exhibit a generalised eliminability property, which corroborates the intuitive feeling that these amount to extension by definition of a new sort, even though they do not have the property of unique expandability of their models (but, the expansion is unique up to isomorphism).

Key words: Formal specifications, sort constructors, extensions, implementation, abstract data types, parameterised specifications, formal program development.

Some Constructs for Sort Introduction in Logical Specifications

María Claudia MERÉ
e-mail: mere@inf.puc-rio.br
Paulo A. S. VELOSO
e-mail: veloso@inf.puc-rio.br

Resumo

O processo de implementar uma especificação formal em outra é de grande importância no desenvolvimento formal de programas, podendo ser descrito em termos de conceitos lógicos simples que relacionam as especificações formais envolvidas.

Na **proposta lógica** para especificações formais, especificações são apresentações axiomáticas em lógica poli-sortida, e uma implemantação de uma especificação A em outra especificação formal C é uma interpretação de A em uma extensão conservativa B de C. A idéia aqui é que a especificação concreta C seja estendida atê incorporar versões concretas nas quais os símbolos abstratos de A possam ser trasladados. Para isto é precisso examinar como acrescentar novos símbolos, e suas propriedades, a uma especificação formal já existente.

Uma extensão consiste do acréscimo de novos símbolos e axiomas, um caso especial simples, porém importante, sendo as extensões por definições. Os novos símbolos a serem introduzidos correspondem a sortes, funções ou predicados. As extensões por acréscimo de símbolos de função e de predicado estão bem estudadas na literatura. Em particular, as extensões por definição de símbolos de função e de predicado caracterizam-se pelas propriedades de *conservatividade* e de *eliminabilidade*.

Nosso objetivo aqui é analisar **extensões conservativas que introduzem novos sortes**. Isto é importante pois ocorre com freqüência na implementação de especificações formais, quando novos sortes são "construídos" (ou representados) a partir dos concretos. Como um simples exemplo considere a conhecida implementação de pilhas (stack) usando vetores (array) e ponteiros (índice). Neste caso desejamos representar a pilha por meio do vetor, contendo os elementos registrados na pilha, junto com um ponteiro, indicando a posição do elemento do topo. A especificação do vetor contém três sortes: vetor, ponteiro e elementos. Ao especificar a pilha, desejamos estender a especificação do vetor acrescentando um sorte produto **Vetor x Ponteiro**. Mas, não todos os pares da forma $< A, i >$ representarão uma pilha: só um subsorte (contendo aqueles pares $< A, i >$ tais que $i \leq$ comprimento(A)). Além disto, pares com vetores que diferem nos elementos acima da marca do ponteiro devem ser identificados (já que eles representam a mesma pilha). Assim devemos fazer um sorte quociente para identificá-los.

Especificaremos e analisaremos alguns construtores de introdução de sortes, similares aqueles encontrados nas linguagens de programação, produto cartesiano, união discriminada, subsorte e quociente. Este último construtor, quociente, mesmo não aparecendo nas linguagens de programação, corresponde á representação da igualdade num sorte abstrato, o que é necessário para uma implementação correta.

Especificaremos cada construtor por meio de uma *especificação formal parametrizada*, isto é, uma apresentação de uma teoria numa linguagem nova, extendida, que é uma extensão conservativa de seu parámetro: uma apresentação de uma teoria na linguagem antiga que descreve sua condição de aplicabilidade.

No contexto de implementação de especificações formais, os novos sortes introduzidos devem estar "conectados" aos antigos. Isto é o que a nossa intuição requere e é o que acontece nos quatro casos examinados. Também, provaremos que cada extensão possue uma propriedade generalizada de eliminabilidade, o que concorda com o nosso sentimento intuitivo de que correspondem a extensões por definicação de um novo sorte, mesmo que elas não possuam a propriedade de expansão única de seus modelos (mas, a expansão é única a menos de isomorfismos).

Palavras chave: Especificações formais, construtores de sortes, extensões, implementação, tipos abstratos de dados, especificações parametrizadas, desenvolvimento formal de programas.

Contents

1 Introduction

The process of implementation is very important in formal software development and it can be described in terms of simple basic logical concepts relating the formal specifications involved ([TM87, Vel87]).

The result of implementing an abstract specification A in terms of another one, C, is a module:

- representing objects of A in terms of those of C;
- implementing operations and predicates of A by means of procedures using operations and predicates of C.

The representation of the objects of A in terms of those of C is generally carried out by representing the sorts corresponding to the abstract ones by means of sorts constructed from those of C. Programming languages usually provide some sort constructs, like enumeration, subrange, records, variant records and discriminated union.

But, the representation part can be abstracted into a formal specification of how the sorts representing the abstract ones are connected to the given concrete sorts. This is similar to the replacement of actual procedure texts by specifications of their input-output behaviors. In the latter case, we provide (perhaps incomplete) definitions of the operations and predicates of A in terms of those of C.

The abstraction described above leads to the concept of implementation of formal specifications. A formal specification is a presentation of a theory in many-sorted logic. And, an implementation of a formal specification A on another formal specification C amounts to an interpretation of A into a conservative extension B of C ([Vel87]). The idea here is that the "concrete" specification C is extended so as to incorporate concrete versions to which the "abstract" symbols of A can then be mapped, but without disturbing the properties of the old symbols.

This in turn involves examining how new symbols, together with their properties, can be added to an existing formal specification. An extension consists of the addition of new symbols and axioms, a simple, but important, special case being the so-called extensions by definitions. The new symbols to be introduced correspond to sorts, functions or predicates. Extensions by the addition of function and predicate symbols are well studied in the literature ([Sho67]). In particular, extensions by definitions of function and predicate symbols are characterised by the properties of conservativeness together with eliminability.

Our purpose here is to analyse conservative extensions that introduce new sorts. This is of importance because it describes how new sorts are "constructed" from given ones, which occurs in the implementation of formal specifications, when new sorts are represented in terms of the concrete ones.

We wish to specify constructs that introduce new sorts. We examine some well-known sort introduction constructs akin to those found in programming languages, namely, cartesian product (corresponding to records), discriminated union, subsorting (corresponding to subrange) and quotient ([Hoa74]). This last construct, quotient, even though not prevalent in programming languages, corresponds to the representation of the equality on an abstract sort, which is necessary for a correct implementation.

For a simple example consider the overused implementation of stacks by arrays with indices. Here one wishes to represent a stack by means of an array, containing the elements stored in the stack, together with an index, indicating the position of the topmost element.

Now, the specification of arrays has three sorts: **Arrays, Indices** and **Elements**. One then wishes to extend it by adding a product sort **Arrays x Indices**. But, not every pair $< A, i >$ will represent a stack; only a subsort (consisting of those with $i \leq$ size(A)). Further, pairs with arrays differing only above their indices are to be identified (for they represent the same stack), so one wishes a quotient sort.

Each construct is presented as a parameterised specification, i.e. as a conservative extension of a parameter specification. Also, they exhibit a generalised eliminability property, which corroborates the intuitive feeling that these amount to extensions by definition of a new sort.

2 Construction of sorts

Let us first describe each sort construct we shall be examining. We will describe them in the meta-language, by relying on well-known set-theoretical concepts. We will, also, discuss some syntactical and semantical aspects. In the next section we shall try to formalise these descriptions within appropriate first-order languages.

2.1 Description of the constructs

The sort construct **product** introduces a new sort consisting of the records (ordered pairs) of objects of the given sorts together with the corresponding selectors. Thus, we have a new set $\mathbf{C} = \mathbf{S_1} \times \mathbf{S_2}$, together with its two projections $\mathbf{p_k} : \mathbf{C} \to \mathbf{S_k}$, with $\mathbf{p_k}(< \mathbf{a_1}, \mathbf{a_2} >) = \mathbf{a_k}$ where $k = 1, 2$.

The sort construct **subsorting** introduces a new sort consisting of the objects of the given sort satisfying a given property established by the relativisation predicate. Thus, we have a new set $\mathbf{N} = \{\mathbf{a} \in \mathbf{S}; \mathbf{r}(\mathbf{a})\ \textbf{holds}\ \}$. Since the elements of this new set $\mathbf{N}$ need not be represented in the same way as those of $\mathbf{S}$, one also introduces an injective insertion map $\mathbf{j} : \mathbf{N} \to \mathbf{S}$ to carry out the conversion.

The sort construct **discriminated union** introduces a new sort consisting of the objects of each one of the given sorts. Thus, we have a new set $\mathbf{U} = \mathbf{S_1} + \mathbf{S_2}$ (disjoint union). Again, because of possibly distinct representations, one also introduces injective insertion maps $\mathbf{i_k} : \mathbf{S_k} \to \mathbf{U}$ (where $k = 1, 2$) to carry out the conversion.

The sort construct **quotient** introduces a new sort consisting of the equivalence classes of the objects of the given sort under a given equivalence relation over it. Thus, we have a new set $\mathbf{M} = \{[\mathbf{a}/\mathbf{q}]; \mathbf{a} \in \mathbf{S}\}$. Again, it is natural to introduce the (surjective) projection onto the quotient set $\mathbf{p} : \mathbf{S} \to \mathbf{M}$ mapping each element $\mathbf{a}$ of $\mathbf{S}$to its $\mathbf{q}$-class $[\mathbf{a}/\mathbf{q}]$.

2.2 Syntax and semantics of the constructs

In terms of languages, each sort construct passes from an old language L to a new, extended language L' with the newly introduced sort and some conversion function symbols connecting the new sort with old ones. Since we assume each sort S to come equipped with its equality symbol $=_S$, this extension also adds the equality symbol corresponding to the new sort. For instance, the sort construct **product** starts from an old language L with sorts S_1 and S_2, together with their equality symbols, and passes to a new, extended language L' with new sort C and the projections $p_1 : C \to S_1$ and $p_2 : C \to S_2$, together with the equality symbol $=_C$ over new sort.

In terms of semantics, it is quite natural to regard (the semantics of) each sort construct **constr** as a kind of operation **CONSTR** on structures: it expands a structure of the old language L to a structure of the new language L' by adding a set to realise the new sort and conversion functions.

But, such operations may have preconditions for their correct applicability. Consider, for instance, the operation **SUBSRT**, corresponding to the sort construct subsorting. This operation is supposed to introduce a new set $\mathbf{N}$ consisting of the objects of the given set $\mathbf{S}$ satisfying the relativisation predicate $\mathbf{r}$. Since we require, as usual in logic, sorts to be realised by nonempty sets, we can apply this operation only when the extension of the relativisation predicate is nonempty.

Thus, it seems natural to view (the semantics of) each sort construct **constr** as a partial operation **CONSTR** on structures. This corresponds roughly to the idea of parameterised data types. Operation **SUBSRT**, for instance, has as parameter a data type, consisting of a structure of the old language L satisfying the applicability condition: the extension of the relativisation predicate is nonempty.

But, we prefer to follow an alternative, internalised, route, as in [VM94]. We specify each sort construct **constr** by means of a parameterised formal specification $CONSTR$, in that it is a presentation of a theory in the new, extended language L', which is a conservative extension of its parameter: a presentation $PARAM$ of a theory in old language L which describes its applicability condition ([MVS91]). For instance, for sort construct **product**, the old language L will have only sort symbols S_1 and S_2 (together with their equality symbols) and new, extended language L' will have in addition symbols for the new sort C and the projections p_1 and p_2 (together with the equality symbol $=_C$ over new sort). For a sort construct with preconditions, like **subsorting**, the parameter presentation will consist of the applicability condition, expressed in old language L, with symbols for the old sort S (together with its equality) and relativisation predicate r.

Now, if one wishes to apply a sort construct to introduce a new sort into a given specification, one has to identify in the language of the given specification its arguments and to guarantee the applicability condition. In other words, one applies sort construct **constr** to a given specification $SPEC$ by instantiating the parameter presentation $PARAM$ into $SPEC$.

3 Specification of the constructs

In this section we shall provide for each sort construct, informally described in the previous section, a formal specification as a parameterised presentation. For the first two sort constructs, **product** and **discriminated union**, the parameter presentation will consist of its language together with empty set of axioms, since there are no preconditions for applying them. On the other hand, for the last two sort constructs, **subsorting** and **quotient**, the parameter presentation will have a nonempty set of axioms expressing the preconditions for applying them.

The specifications for the first three constructs are axiomatisations of their usual description [Hoa74, Vel87]. While the specification for the last one axiomatises the construction of the quotient with its natural projection ([MV92]).

Later on we shall establish some properties of these specifications.

Cartesian Product

We first describe the parameter presentation $PARAM_P$. Its language L_P has sort symbols S_1 and S_2, together with their (binary) equality symbols $=_{S_1}$ over sort S_1 and $=_{S_2}$ over sort S_2. Its set G_P of axioms is empty, since there are no preconditions for forming the cartesian product of sets. We now describe the extended presentation $PROD$. Its language L'_P amounts to L_P plus the new sort symbol C, together with its (binary) equality symbol $=_C$ over sort C, and two unary operation symbols $p_1 : C \to S_1$ and $p_2 : C \to S_2$. Its set G'_P of axioms consists of two sentences of L'_P stating that the projections are jointly surjective and that the projections are jointly injective.

Then, a specification of the cartesian product of two sorts is:

$PROD = PARAM_P+$

sort C

operations $p_1 : C \to S_1$

$p_2 : C \to S_2$

axioms

- $\forall x_1 : S_1\ \forall x_2 : S_2\ \ \exists y : C \quad p_1(y) =_{S_1} x_1 \wedge p_2(y) =_{S_2} x_2$
- $\forall y, y' : C \quad p_1(y) =_{S_1} p_1(y') \wedge p_2(y) =_{S_2} p'_2(y) \Longrightarrow y =_C y'$

Discriminated Union

We first describe the parameter presentation $PARAM_U$. Its language L_U has sort symbols S_1 and S_2, together with their (binary) equality symbols $=_{S_1}$ over sort S_1 and $=_{S_2}$ over sort S_2. Its set G_U of axioms is empty, since there are no preconditions for forming the disjoint union of sets. We now describe the extended presentation $DISC–UNION$. Its language L'_U amounts to L_U plus the new sort symbol U, together with its (binary) equality symbol $=_U$ over sort U, and two unary operation symbols $i_1 : S_1 \to U$ and $i_2 : S_2 \to U$. Its set G'_U of axioms consists of four sentences of L'_U stating that the images of the insertions cover the new sort, that these images are disjoint and expressing the injectivity of each insertion.

Then, a specification of the (discriminated) union of two sorts is:

$DISC–UNION = PARAM_U+$

sort U

operations $i_1 : S_1 \to U$

$i_2 : S_2 \to U$

axioms

- $\forall y : U \ \ (\exists x_1 : S_1 \quad y =_U i_1(x_1)) \vee (\exists x_2 : S_2 \quad y =_U i_2(x_2))$
- $\forall x_1 : S_1\ \forall x_2 : S_2\ \neg\ (i_1(x_1) =_U i_2(x_2))$
- $\forall x_1, x'_1 : S_1 \ \ i_1(x_1) =_U i_1(x'_1) \Rightarrow x_1 =_{S_1} x'_1$
- $\forall x_2, x'_2 : S_2 \ \ i_2(x_2) =_U i_2(x'_2) \Rightarrow x_2 =_{S_2} x'_2$

Subsort

We first specify the parameter presentation $PARAM_S$. Its language L_S has a sort symbol S, together with its (binary) equality symbol $=_S$ over sort S, and a unary (relativisation) predicate symbol r over sort S. Its set G_S of axioms consists of the following sentence of L_S:

(NVR) $(\exists x : S) r(x)$.

Axiom (NVR) expresses that (the extension of) the relativisation predicate must be nonempty. We now describe the extended presentation $SUBSRT$. Its language L'_S amounts to L_S plus the new sort symbol N, together with its (binary) equality symbol $=_N$ over sort N, and a unary operation symbol $j : N \to S$. Its set G_S of axioms consists of two sentences of L'_S stating that the image of the insertion is (the extension of) the relativisation predicate and expressing the injectivity of the insertion.

Then, a specification of the subsort N of the sort S in according with r is:
$SUBSRT = PARAM_S+$

sort N

operations $j : N \to S$

axioms

- $\forall x : S \quad r(x) \iff \exists y : N \;\; x =_S j(y)$
- $\forall y, y' : N \;\; j(y) =_S j(y') \Longrightarrow y =_N y'$

Quotient sort

We first specify the parameter presentation $PARAM_Q$. Its language L_Q has a sort symbol S, together with its (binary) equality symbol $=_S$ over sort S, and a binary predicate symbol q over sort S. Its set G_Q of axioms consists of the three sentences of L_Q stating that the predicate symbol q must be an equivalence relation, i.e. expressing reflexivity, symmetry and transitivity.

We now describe the extended presentation $QUOT$. Its language L'_Q amounts to L_Q plus the new sort symbol M, together with its (binary) equality symbol $=_M$ over sort M, and a unary operation symbol $p : S \to M$. Its set G'_Q of axioms consists of two sentences of L'_Q expressing the surjectivity of the projection and that the projection of an element of S consists exactly of the elements related to it under q.

Then, a specification of the quotient sort M of the sort S by the relation q is:
$QUOT = PARAM_Q+$

sort M

operations $p : S \to M$

axioms

- $\forall x, x' : S \;\; q(x, x') \iff p(x) =_M p(x')$
- $\forall y : M \;\; \exists x : S \;\; p(x) =_M y$

4 Adequacy of the specifications

In this section we shall present some properties of the parameterised specifications given in the previous section. Some of these properties show that the specifications are adequate, because the new sorts exhibit the desired behavior. We also provide some logical properties to these extensions; in particular, we will prove that they are similar to classical extensions by explicit definition of predicate and function symbols.

4.1 Characterisation and adequacy of the constructs

In this section we establish some model-theoretic properties of the specifications presented in Section 3.

For each one of the four specifications, we examine their models, which leads to their characterisation, showing that the new sort indeed exhibits the desired behavior. These results indicate the adequacy of the specifications in capturing behavior that is to be expected from the intuitive view of these sort introducing constructs. Also, we obtain, up to isomorphism, unique expandability which entails conservativeness.

We shall examine each construct in turn in the following four subsection.

Cartesian product

If $PROD$ and $PARAM_P$ are the specifications involving in the presentation of the product sort, the next result characterises the models of $PROD$. It shows that the new sort may be regarded as consisting of the desired ordered pairs and behaves as (categorical) product ([HS73]) of the given sorts.

Theorem 4.1 (Characterisation of Models) *$\mathcal{A}' \models PROD$ iff, with $\mathcal{A}'/L = \mathcal{A}$,*

1. *$\mathcal{A} \models PARAM_P$, and*
2. *there exists a bijection $h : C^{\mathcal{A}'} \to S_1^{\mathcal{A}} \times S_2^{\mathcal{A}}$ such that $h(c) =< p_1^{\mathcal{A}'}(c), p_2^{\mathcal{A}'}(c) >$.*

From this theorem we obtain the following results:

Corollary 4.1 (Expansiveness) *The extension $PARAM_P \subseteq PROD$ is expansive.*

Corollary 4.2 (Conservativeness) *The extension $PARAM_P \subseteq PROD$ is conservative.*

Discriminated Union

In this subsection, $DISC\text{–}UNION$ and $PARAM_U$ stand for the specifications involving in the presentation of the union sort.

The next result characterises the models of $DISC\text{–}UNION$. It shows that the new sort may be viewed as the disjoint union of the given sorts, as desired, and exhibits the behavior of their (categorical) coproduct ([HS73]).

Theorem 4.2 (Characterisation of Models) *$\mathcal{A}' \models DISC\text{–}UNION$ iff, with $\mathcal{A}'/L = \mathcal{A}$,*

1. *$\mathcal{A} \models PARAM_U$, and*
2. *there exists a bijection $h : S_1^{\mathcal{A}} + S_2^{\mathcal{A}} \to U^{\mathcal{A}'}$ such that $h(a) = i_1^{\mathcal{A}'}(a)$ for each $a \in S_1^{\mathcal{A}}$ and $h(b) = i_2^{\mathcal{A}'}(b)$ for each $b \in S_2^{\mathcal{A}}$.*

Corollary 4.3 (Expansiveness) *The extension $PARAM_U \subseteq DISC\text{–}UNION$ is expansive.*

Corollary 4.4 (Conservativeness) *The extension $PARAM_U \subseteq DISC\text{–}UNION$ is conservative.*

Subsort

In this subsection $SUBSRT$ and $PARAM_S$ refer to the specifications involving in the presentation of the subsort. Notice the precondition $PARAM_S \models \exists x : S\ r(x)$

Theorem 4.3 (Characterisation of Models) *$\mathcal{A}' \models SUBSRT$ iff, with $\mathcal{A}'/L = \mathcal{A}$,*

1. *$\mathcal{A} \models PARAM_S$, and*
2. *there exists a bijection $h : N^{\mathcal{A}'} \to r^{\mathcal{A}}$, where $r^{\mathcal{A}} = \{a \in S^{\mathcal{A}} \ / \ \mathcal{A} \models r(x)[a]\}$, such that $h(c) = j^{\mathcal{A}'}(c)$.*

This result establishes that the new sort may be thought of as consisting of those elements that satisfy the relativisation predicate. But, the elements of the new sort may very well have a different representation. This possibly different representation is one reason for the conversion function $j : N \to S$. This is illustrated by the (unsigned) naturals as a subsort of the (signed) integers.

Corollary 4.5 (Expansiveness) *The extension $PARAM_S \subseteq SUBSRT$ is expansive.*

Corollary 4.6 (Conservativeness) *The extension $PARAM_S \subseteq SUBSRT$ is conservative.*

Quotient

In this subsection $QUOT$ and $PARAM_Q$ are to be understood as the specifications involving in the presentation of the quotient sort. Notice the precondition on $PARAM_Q$: ref l exivity, symmetry and transitivity of q must be theorems of $PARAM_Q$.

The next result characterises the models of $QUOT$ by showing that the new sort can be thought to consist of the desired equivalence classes, or representatives for them.

Theorem 4.4 (Characterisation of Models) *$\mathcal{A}' \models QUOT$ iff, with $\mathcal{A}'/L = \mathcal{A}$,*

1. *$\mathcal{A} \models PARAM_Q$, and*
2. *there exists a bijection $h : S^{\mathcal{A}}/q^{\mathcal{A}} \to Q^{\mathcal{A}'}$ such that $h([a/q]) = p^{\mathcal{A}'}(a)$, where $[a/q]$ is the equivalence class of $a \in S$ under q.*

Corollary 4.7 (Expansiveness) *The extension $PARAM_Q \subseteq QUOT$ is expansive.*

Corollary 4.8 (Conservativeness) *The extension $PARAM_Q \subseteq QUOT$ is conservative.*

4.2 Invariance of the constructs

Another important property is that the models are **invariant under isomorphisms**, that is, for each construct we have that any pair of models of the extension are isomorphic whenever their reducts to language L are so. This property is the analogous to the one in Beth's Definability Theorem ([Sho67]), in the simple case of extensions by definition of predicate or function symbols.

Actually, the invariant property comes from another, more general, property that also holds for each construct: **the functorial character of models**. The functorial property states that for any pair of models of the extension, any homomorphism between them is the (unique) extension of an homomorphism between their reducts to language L.

Moreover, the property of **invariance under elementary equivalence** holds for each construct. This property affirms that for any pair of models of the extension, they are elementary equivalent whenever their reducts to language L are so.

When performing an extension one expects an increase in both deductive and expressive powers. On the other hand, one often does not wish to disturb the smaller theory. When the extension only adds properties of the new symbols, without disturbing the old ones, one has a conservative extension, which does not increase the deductive power for the properties of the old symbols. Similarly, the increase in expressive power may be inessential, in which case our extension has the property of eliminability ([Sho67]).

But, we need a more general notion of eliminability because we have to face the fact that we now have variables over the new sort. This can be overcome by means of a connection, which indicates how to translate variables ranging over the new sort as variables over the old sorts. In this case, any sentence of extended language L' is equivalent to a sentence of the language L under a connection establishing how the variables of the new sort are connect to variables of the old sorts ([MV94]). Formally for each construct, we have proved:

Proposition 4.1 (Eliminability under connection for product) *Consider sets of variables $\overline{y} = \{y^1, \ldots, y^n\}$ of sort C and $\overline{v}$ of sorts in L. Given sets of variables $\overline{x_1} = \{x_1^1, \ldots, x_1^n\}$ with x_1^k of sort S_1 and $\overline{x_2} = \{x_2^1, \ldots, x_2^n\}$ with x_2^k of sort S_2, such that $\overline{v} \cap \overline{x_1} = \emptyset = \overline{v} \cap \overline{x_2}$, form the conjunction $K(\overline{y}, \overline{x_1}, \overline{x_2})$ of $p_1(y^k) = x_1^k \wedge p_2(y^k) = x_2^k$ for each $k = 1, \ldots, n$.*

For every formula $\rho(\overline{v}, \overline{y})$ of L' having as its free variables those in $\overline{v} \cup \overline{y}$, there exists a formula $\varphi(\overline{v}, \overline{x_1}, \overline{x_2})$ of L with free variables in $\overline{v} \cup \overline{x_1} \cup \overline{x_2}$, such that $PROD \vdash K(\overline{y}, \overline{x_1}, \overline{x_2}) \rightarrow (\rho(\overline{v}, \overline{y}) \leftrightarrow \varphi(\overline{v}, \overline{x_1}, \overline{x_2}))$. We call such a formula φ of L the **reflection of** *ρ into L (via K).*

Proposition 4.2 (Eliminability under connection for union) *Consider sets of variables $\overline{y} = \{y^1, \ldots, y^n\}$ of sort U and $\overline{v}$ of sorts in L. Also, consider sets $\overline{x_1} = \{x_1^1, \ldots, x_1^n\}$ of variables of sort S_1 and $\overline{x_2} = \{x_2^1, \ldots, x_2^n\}$ of variables of sort S_2 such that $\overline{v} \cap \overline{x_1} = \emptyset = \overline{v} \cap \overline{x_1}$.*

For each function $\delta : \{1, \ldots, n\} \rightarrow \{1, 2\}$ let $k^\delta = \wedge_{j=1}^n (y_j = i_{\delta(j)}^j (x_{\delta(j)}^j))$ and $\overline{y}^\delta = \{x_{\delta(1)}^1, \ldots, x_{\delta(n)}^n\}$.

*Given a formula $\rho(\overline{v}, \overline{y})$ of L' having as its free variables those of $\overline{v} \cup \overline{y}$, there exist 2^n formulas $\rho^\delta(\overline{v}, \overline{y}^\delta)$ of L with free variables of $\overline{v} \cup \overline{y}^\delta$, such that $DISC\text{–}UNION \vdash k^\delta \rightarrow (\rho(\overline{v}, \overline{y}) \leftrightarrow \rho^\delta(\overline{v}, \overline{y}^\delta))$. We call such a formula ρ^δ of L the δ-***reflection of** ρ into L (via k^δ).*

Proposition 4.3 (Eliminability under connection for subsort) *Consider sets of variables $\overline{y} = \{y^1, \ldots, y^n\}$ of sort N and $\overline{v}$ of sorts in L. Given a set $\overline{x} = \{x^1, \ldots, x^n\}$ of variables of sort S such that $\overline{v} \cap \overline{x} = \emptyset$ form the conjunction $K(\overline{y}, \overline{x})$ of $j(y^k) = x^k$ for each $k = 1, \ldots, n$.*

For every formula $\rho(\overline{v}, \overline{y})$ of L' having as its free variables those in $\overline{v} \cup \overline{y}$, there exists a formula $\varphi(\overline{v}, \overline{x})$ of L with free variables in $\overline{v} \cup \overline{x}$, such that $SUBSRT \vdash K(\overline{y}, \overline{x}) \rightarrow (\rho(\overline{v}, \overline{y}) \leftrightarrow \varphi(\overline{v}, \overline{x}))$. We call such a formula φ of L the **reflection of** *ρ into L (via K).*

Proposition 4.4 (Eliminability under connection for quotient) *Consider sets of variables $\overline{y} = \{y^1, \ldots, y^n\}$ of sort Q and $\overline{v}$ of sorts in L. Given a set $\overline{x} = \{x^1, \ldots, x^n\}$ of variables of sort S such that $\overline{v} \cap \overline{x} = \emptyset$ form the conjunction $K(\overline{y}, \overline{x})$ of $y^k = p(x^k)$ for each $k = 1, \ldots, n$.*

For every formula $\rho(\overline{v}, \overline{y})$ of L' having as its free variables those in $\overline{v} \cup \overline{y}$, there exists a formula $\varphi(\overline{v}, \overline{x})$ of L with free variables in $\overline{v} \cup \overline{x}$, such that $QUOT \vdash K(\overline{y}, \overline{x}) \rightarrow (\rho(\overline{v}, \overline{y}) \leftrightarrow \varphi(\overline{v}, \overline{x}))$. We call such a formula φ of L the **reflection of** *ρ into L (via K).*

The well-known eliminability property when there does not exist variables over new sorts ([Sho67]) is a particular case of the eliminability under connection property. Using the reflection formulae, we also have results about **definability** of relations: a relation is definable in the extension iff its internalisation was definable in the old language. Here internalisation refers to the process of transforming a relation involving the new sort to its view in the parameter language. For the case of product, a relation involving sort C is internalised by replacing C by S_1 and S_2 and using projections. For quotient a relation R involving sort Q is internalised to R_q where a class is replaced by all its representatives. A relation involving $U = S_1 + S_2$ is internalised by its views depending on "where an element of U came from".

5 Conclusions

We have examined four sort introduction constructs: cartesian product, subsort, discriminated union and quotient.

Introduction of new sorts is of interest because it occurs often in implementing formal specifications. In contrast to the introduction of new function or predicate symbols [Sho67], the case of new sorts is not extensively studied.

The four constructs examined correspond to natural constructions of new sets. Three of them have found their way as data structuring constructs in programming languages [Hoa74]. The quotient construct is of importance in developing and verifying correct implementations [TM87, Vel87].

We have analised them as parameterised formal specifications of data types. In logical terms, a parameterised specification amounts to a presentation of a theory in the extended language which is a conservative extension of its parameter (a presentation of the theory of the old language that describes its applicability condition).

In section 3 we have presented formal specifications for these four constructs by axiomatising their usual descriptions as extensions of given specifications. In each case, in addition to the new sort, we also introduce appropriate conversion functions, connecting it to the old sorts.

In section 4 we proved that, in each case, the new sort is unique, but only up to isomorphism. This contrasts with the case of introduction of new function or predicate symbols by definitions, when we have unique expandability of models. Also, these more familiar extensions by definitions exhibit eliminability: any formula of the extended language is equivalent to some formula in the original language. In our case of introduction of new sorts one should not expect full eliminability, because of the new variables ranging over the sort introduced. Nevertheless, a kind of generalised eliminability is still possible, as we have proved in the propositions called eliminability under connection.

The extension by adding a product sort and by adding a discriminated union are similar to classical extensions by definition of predicate, in that they are conservative and eliminable. The extension of the subsort and the quotient are eliminable but have requeriments for conservativeness, and resemble classical extensions by definition of operation.

Versions of conservativeness and eliminability are important also in the algebraic approach, which emphasises equations between ground terms ([EM85]).

Another sort introduction construct found in programming languages is **enumeration**. This construct is different from the others because it does not need a base language to introduce the new sort. The sort construct $ENUMERAT$ introduces a new sort consisting of the objects named by constants, a usual way to describe a finite set. It is not difficult to specify it.

A property shared by the constructs we have examined is that of producing only finite sorts (from finite sorts). For the needs of implementation it is convenient to have also other constructs, as in some programming languages. Some such constructs, like powerset and mapping (similar to array), still preserve finiteness, but others, like sequences, generate (denumerably) infinite sorts from finite sorts. These constructs and the completeness of the set of constructs will be examined in a forthcoming paper.

References

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specifications, 1: Equations and Initial Semantics* . Springer-Berlag, Berlin, 1985.

[Hoa74] C.A.R. Hoare. Notes on data structuring. In O.J. Dahl, E.W. Dijkstra, and C.A.R. Hoare, editors, *Structured Programming*, pages 83–174. Academic Press, 1974.

[HS73] H. Herrlich and G. E. Strecker. *Category Theory.* Ally and Bacon Inc., Boston, 1973.

[Sho67] J.R. Shoenfield. *Mathematical Logic.* Addison-Wesley Publishing Company, 1967.

[MVS91] T.S.E. Maibaum, P.A.S. Veloso and M.R. Sadler. A logical approach to specification and implementation of abstract data types. Imperial College of Science, Technology and Medicine, Depto. of Computing. Res. Rept. Doc 91/47, London.

[MV92] M. C. Meré and P. A. Veloso. On Extensions by Sorts. *Monografias em Ciências da Computação, D.I., PUC-Rio*, 38/92, December 1992.

[MV94] M. C. Meré and P. A. Veloso. The Eliminability property in extensions by sort. To appear in *Monografias em Ciências da Computação, D.I., PUC-Rio.* Abstract in WoLLIC'94, Pernambuco, Brazil, July 1994.

[TM87] W.M. Turski and T.S.E. Maibaum. *The Specification of Computer Programs.* Addison-Wesley Publishing Company, 1987.

[Vel87] P.A.S. Veloso. *Estruturação e Verificação de Programas com Tipos de Dados.* Editora E. Blücher ltda., 1987.

[VM94] P.A. Veloso and M.C. Meré. Construtores Parametrizados de Sortes. *Monografias em Ciências da Computação, D.I., PUC-Rio*, 14/94, July 1994.