Label algebras: a systematic use of terms

Gilles Bernot[1] Pascale Le Gall[2]

Abstract:

We give the main definitions and results of a new framework for algebraic specifications: the framework of *label algebras*. The main idea underlying our approach is that the semantics of algebraic specifications can be deeply improved when the satisfaction relation is defined via assignments with range in *terms* instead of values. Surprisingly, there are several cases where even if two terms have the same value, it is possible that one of them is a suitable instance of a variable in an formula while the other one is not. It is for instance the case for algebraic specifications with exception handling or with observability features. We show that our approach is a useful tool for solving this problem.

Keywords: algebraic specifications, exception handling, initial semantics, observability, subsorting, structured specifications.

1 Introduction

This paper is an overview of a new framework for algebraic specifications: the framework of *label algebras*. Many applications of abstract data types require specialized semantics (e.g. observability issues, exception handling, partial functions, etc). For such applications, it has been shown that the simple pioneer semantics proposed by the ADJ group [GTW78] cannot be applied directly, they often lead to inconsistencies, or unreadable specifications. The reason why simple semantics are not suitable is often that the scope of an axiom is too wide. When variables are universally quantified over a sort, several instances of an axiom can lead to inconsistencies. For example, for exception handling, the scope of certain axioms must be restricted to non exceptional assignments only; for observability purposes, only observable consequences of an axiom must be taken into account (usually via observable contexts); for partial functions, only assignments belonging to some definition domains must be considered; etc. Several powerful general frameworks, such as order sorted algebras [Gog78] or unified algebras [Mos89] can be used to restrict the scope of the axioms, allowing a more elaborated definition of the acceptable assignments. These frameworks usually consider assignments which substitute values for variables. Our claim in this paper is that, for some cases, assignments which range in terms are more suitable. Surprisingly, it turns out that even if two terms have the same value, it is possible that one of them is a suitable instance of a variable while the other one is not. For instance, in [BB91], it is shown that observability can sometimes require to characterize a set of observable terms while sets of observable values would not be powerful enough. In the same way, [BBC86] pointed out that exception handling with bounded data structures requires to characterize a set of "Ok-terms" and that considering only sets of "Ok-values" leads to inconsistencies (see Section 2). Label algebras will allow us to "type" terms instead of values: twos terms which have the same value can be labelled with distinct *labels*, while a classical "type" (even in order sorted algebras) must be shared by all the terms having the same value. Thus, label algebras can be considering as a tool to avoid inconsistencies where the "types" of terms (i.e. *labels*) play a role

[1]LIENS, CNRS URA 1327, 45 rue d'Ulm, 75230 PARIS Cedex 05 FRANCE. (bernot@dmi.ens.fr /frulm63.bitnet)

[2]LRI, CNRS URA 410, Bat. 490, Université Paris-Sud, 91405 Orsay, FRANCE. (legall@lri.lri.fr /frlri61.bitnet)

similar to sorts and subsorts in [Gog78]. As described in Section 5.2, we have already used label algebras to define exception algebras (an algebraic framework for exception handling) but the application domain of label algebras seems to be much more general than exception handling.

In Section 2, from an example with exception handling, we will point out the importance of terms in the field of first order algebraic specifications and we will show the necessity of "labelling" terms in order to easily specify exceptions. In Section 3, we will define the framework of label algebras. The main results (e.g. initiality results) will be established in Section 4. In Section 5, we will first sketch out how far this framework can be applied to several classical subjects of abstract data types, such as partial functions, observability features, etc; and then, we will rapidly explain how a new formalism of exception algebras is defined, related to the one of label algebras. Recapitulation and perspectives can be found in Section 6.

We assume that the reader is familiar with algebraic specifications [GTW78][EM85][GB84] and with the elementary definitions of category theory [McL71].

2 The importance of the terms within algebraic specifications

We have claimed that several approaches of algebraic specifications require to take care of terms. Let us mention two applications whose semantics closely depends on terms.

First, let us consider algebraic specifications with observability issues. A crucial aspect of observational specifications is that "what is observable" must be carefully specified. It is often very difficult to prove that two values are observationally equal (while it is sufficient to exhibit two observations which distinguish them to prove that they are distinct). In [Hen89], R. Hennicker uses a predicate *Obs* to characterize the observable values. This powerful framework leads to legible specifications and it provides some theorem proving methods. Nevertheless, it has been shown in [BB91] that there are some specifications which are inconsistent when observability is carried by values and that these inconsistencies can be avoided when observability is expressed which respect to a subset ΣObs of the signature Σ. The use of this subset of operations leads consequently to consider a subset of terms instead of values (see also Section 5.1).

Now, we will go in more details with an example containing exception handling features. Since the works of [GTW78], many algebraic approaches have been proposed to treat many exception handling features such as declaration of exceptions, recovery, etc. For instance, [Gog78] [FGJM85] [GM89] [GDLE84] [Gog87] give solutions to the algebraic treatment of "intrinsic errors" (such as $pred(0)$ or $pop(empty)$), with implicit error propagation and possible recoveries, but they are not able to treat the other kind of errors, especially bounded data structures (see [Bre91] for the crucial importance of bounded data structures). The most difficult point is to simultaneously handle bounded data structures and certain recoveries of exceptional values:

Example 1 In order to specify bounded natural numbers it is indeed not too difficult to specify that all the values belonging to $[0 \dots maxint]$ are Ok-values [BBC86]; let us assume that this is done. We also have to specify that the operation *succ* raises an exception when applied to $maxint$, e.g. $TooLarge$; let us assume that this is done too. When specifying the operation *pred*, we must have the following axiom:

$$pred(succ(x)) = x \tag{1}$$

which is a "normal property" and, as such, should be understood with certain implicit preconditions such as "*if* x *and* $succ(x)$ *are Ok-values*" for example. Let us assume now that we want to recover all $TooLarge$ values on $maxint$. We will then necessarily have $succ(maxint) = maxint$.

Since these two values are equal, we will have to choose: either both of them are erroneous values, or both of them are Ok-values. The first case is not acceptable because it does not cope with our intuition of "recovery." (Moreover, when considering the value $m = maxint - 1$

we clearly need that $pred(maxint) = m$, as a particular case of our "normal property" about *pred*. Thus $succ(m) = maxint$ must be considered as a normal value.) Unfortunately, since $succ(maxint)$ is then a normal value, $x = maxint$ is an acceptable assignment for (1) and we get the following inconsistency: $m = pred(maxint) = pred(succ(maxint)) = maxint$ which propagates in the same way, and all values are equal to 0.

Let us point out that subsorting [Gog78] cannot be used to specify such bounded data structures with recoveries. The axiom (1) necessarily gives rise to a similar paradox because sorts are attached to values. Two terms having the same value must share the same subsorts; consequently $maxint$ and $succ(maxint)$ cannot be distinguished.

Indeed, it is precisely the difference between "exception handling" and "error handling." The term $succ(maxint)$ is not erroneous but it is exceptional while the term $maxint$ is not exceptional; the semantics must take this fact into account. This leads to the following idea: the term $maxint$ is an acceptable assignment for the variable x in the equation (1) while $succ(maxint)$ is not, even though $maxint$ and $succ(maxint)$ have the same value. Thus, exception handling requires taking care of terms inside the algebras and good functional semantics for exception handling should allow such distinctions. This idea has been formalized in [BBC86], where "Ok-terms" are declared: the term $succ^{maxint}(0)$ is "labelled" by *Ok* while the term $succ^{maxint+1}(0)$ is not;[1] and the acceptable assignments of a normal property (called "Ok-axiom") are implicitly restricted to Ok-terms only. This solves the inconsistencies due to the recovery $succ(maxint) = maxint$ with the axiom (1).

These considerations have been our main motivation to develop the framework of *label algebras*.

Usually, algebras are (heterogeneous) sets of values [GTW78][EM85]. Let us remember that a signature is usually a couple $< S, \Sigma >$ where S is a finite set of sorts (or type names) and Σ is a finite set of operation names with arity in S; the objects (algebras) of the category $Alg(\Sigma)$ are heterogeneous sets, A, partitioned as $A = \{A_s\}_{s \in S}$, and with, for each operation name "$f : s_1 \dots s_n \rightarrow s$" in Σ $(0 \leq n)$, a total function $f_A : A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$; the morphisms of $Alg(\Sigma)$ (Σ-morphisms) being obviously the sort preserving, operation preserving applications.

As a consequence of our remarks, labelled terms must also be considered as "first class citizen objects." Given an algebra A, the satisfaction of a normal property must be defined using terms (the usual definition only involves values). A simple idea could be to consider both A and T_Σ (the ground term algebra over Σ). Unfortunately, finitely generated algebras (i.e. such that the initial Σ-morphism from T_Σ to A is surjective) are not powerful enough to cope with enrichment, parametrization or abstract implementation. How is one to deal with both terms and non reachable values ? This question is solved by the free Σ-term algebra $T_\Sigma(A)$. Let us remember its definition.

Notation 1 *Given a heterogeneous "set of variables" $V = \{V_s\}_{s \in S}$, the* free Σ-term algebra with variables in V *is the least Σ-algebra $T_\Sigma(V)$ (with respect to the preorder induced by the Σ-morphisms) such that $V \subseteq T_\Sigma(V)$.*
Since V is not necessarily enumerable, we can consider $T_\Sigma(A)$ for every algebra A. An element of $T_\Sigma(A)$ is a Σ-term such that each leaf contains either a constant of Σ, or a value of A.

For example, if $A = \mathbf{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ is the algebra of all integers over the signature *{zero,succ_,pred_}*, then $succ(succ(zero))$, $succ(succ(0))$, $succ(1)$, etc. are *distinct* elements of $T_\Sigma(\mathbf{Z})$, even though they have the same value when evaluated in $\mathbf{Z}$.

The main technical point underlying our framework is to systematically use $T_\Sigma(A)$ directly inside the *label algebras*. This allows us to have a very precise definition of the satisfaction relation, using assignments with range in $T_\Sigma(A)$ instead of A. Intuitively, a term reflects the

[1] $succ^i(0)$ is an abbreviation for $succ(succ(\dots(0)\,))$ where *succ* appears i times.

"history" of a value; it is a "sequence of calculi" which results in a value. Of course, several histories can provide the same value. This is the reason why labelling is more powerful than typing: it allows us to "diagnose" the history in order to apply a specific treatment or not. Nevertheless, we must be able to relate each term to its final value. The canonical evaluation morphism $eval_A : T_\Sigma(A) \longrightarrow A$, deduced from the Σ-algebra structure of A, relates each term to its final value. Of course, *in the end*, the satisfaction of an equality must be checked on values; thus, $eval_A$ is a crucial tool for defining the satisfaction relation on equational atoms. However, the considered assignments can be precisely restricted to certain kinds of terms/histories *before* checking equalities on values, and this is the reason why the inconsistencies mentioned before can be solved via label algebras.

Notation 2 *We note $\overline{A} = T_\Sigma(A)$ and for every Σ-morphism $\mu : A \to B$, $\overline{\mu} : \overline{A} \to \overline{B}$ denotes the canonical Σ-morphism which extends μ to the corresponding free algebras.*

3 Label algebras

3.1 Basic definitions

Definition 1 *A* label signature *is a triple $\Sigma L = < S, \Sigma, L >$ where $< S, \Sigma >$ is an usual signature and L is a finite set of* labels.

*A ΣL-*algebra *is a couple $\mathcal{A} = (A, \{l_A\}_{l \in L})$ where A is a Σ-algebra, and $\{l_A\}_{l \in L}$ is a L-indexed family such that, for each l in L, l_A is a subset of $\overline{A}$.*

There are no conditions about the subsets l_A: they can intersect several sorts, they are not necessarily disjoint and their union ($\bigcup_{l \in L} l_A$) does not necessarily cover $\overline{A}$.

Definition 2 *Let $\mathcal{A} = (A, \{l_A\}_{l \in L})$ and $\mathcal{B} = (B, \{l_B\}_{l \in L})$ be two ΣL-algebras. A ΣL-morphism $h : \mathcal{A} \to \mathcal{B}$ is a Σ-morphism from A to B such that $\forall l \in L$, $\overline{h}(l_A) \subseteq l_B$.*

When there is no ambiguity about the signature under consideration, ΣL-algebras and ΣL-morphisms will be called *label algebras* and *label morphisms*, or even algebras and morphisms.

Definitions 3 *The category of all ΣL-algebras and ΣL-morphisms is denoted by $Alg_{Lbl}(\Sigma L)$.*

$\mathcal{T}_{\Sigma L}$ is the ΣL-algebra such that the underlying Σ-algebra of $\mathcal{T}_{\Sigma L}$ is the ground terms algebra T_Σ and for each l in L, $l_{T_{\Sigma L}}$ is empty.

$\mathcal{T}riv$ is the ΣL-algebra such that the underlying Σ-algebra of $\mathcal{T}riv$ is the trivial algebra $Triv$ which contains only one element in $Triv_s$ for each s in S and for each l in L, $l_{\mathcal{T}riv} = \overline{Triv}$.

The ΣL-algebra $\mathcal{T}_{\Sigma L}$ (resp. $\mathcal{T}riv$) is clearly initial (resp. terminal) in $Alg_{Lbl}(\Sigma L)$. As usual, a ΣL-algebra $\mathcal{A}$ is called *finitely generated* iff the initial ΣL-morphism from $\mathcal{T}_{\Sigma L}$ to $\mathcal{A}$ is an epimorphism. Thus, $\mathcal{A}$ is finitely generated iff the underlying morphism from T_Σ to A is surjective.

Definitions 4 *The full subcategory of $Alg_{Lbl}(\Sigma L)$ containing all the finitely generated algebras is denoted by $Gen_{Lbl}(\Sigma L)$. Moreover, the signature ΣL is* sensible *iff $\mathcal{T}riv$ belongs to $Gen_{Lbl}(\Sigma L)$.*

The category $Gen_{Lbl}(\Sigma L)$ has the same initial object as $Alg_{Lbl}(\Sigma L)$, and if ΣL is sensible (i.e. if there exists at least one ground term of each sort) then it has the same terminal object too.

Not surprisingly, a "label specification" will be defined by a (label) signature and a set of well formed formulae (axioms):

Definition 5 *Given a label signature ΣL, a ΣL-*axiom *is a well formed formula built on:*

- atoms: *atoms are either equalities* $(u = v)$ *such that* u *and* v *are* Σ*-terms with variables,* u *and* v *belonging to the same sort, or labelling atoms* $(w \in l)$ *such that* w *is a* Σ*-term with variables and* l *is a label belonging to* L*,*

- connectives *belonging to* $\{\neg, \wedge, \vee, \Rightarrow\}$.

(Every variable is implicitly universally quantified.)[2]

A ΣL*-axiom is called* positive conditional *if and only if it is of the form* $a_1 \wedge \ldots \wedge a_n \Rightarrow a$ *where the* a_i *and* a *are positive atoms (if* $n = 0$ *then the axiom is reduced to* a*).*

The predicate " ϵ " should be read "*is labelled by*".

Definition 6 *A* label specification *is a couple* $SP =< \Sigma L, Ax >$ *where* ΣL *is a label signature and* Ax *is a set of* ΣL*-axioms.* SP *is called* positive conditional *iff all its axioms are positive conditional.*

The *satisfaction relation* is indeed the crucial definition of this section. It is of first importance to remark that we consider assignments with range in $\overline{A} = T_\Sigma(A)$ (terms) instead of A (values):

Definition 7 *Let* $\mathcal{A} = (A, \{l_A\}_{l \in L})$ *be a* ΣL*-algebra.*

- $\mathcal{A}$ *satisfies* $(u = v)$*, where* u *and* v *are two terms of the same sort in* $\overline{A}$*, means that, in* A*,* $eval_A(u) = eval_A(v)$ *[the symbol "=" being the set-theoretic equality in the carrier of* A*].*

- $\mathcal{A}$ *satisfies* $(w \in l)$*, where* $w \in \overline{A}$ *and* $l \in L$*, means that* $w \in l_A$ *[the symbol "*$\in$*" being the set-theoretic membership].*

- *Given a* ΣL*-axiom* φ*,* $\mathcal{A}$ satisfies φ*, denoted by* $\mathcal{A} \models \varphi$*, means that for all assignments* $\sigma : V \to \overline{A}$ *(*V *covering all the variables of* φ*),* $\mathcal{A}$ *satisfies* $\sigma(\varphi)$ *according to the "ground atomic satisfaction" defined above and the truth tables of the connectives.*

$\mathcal{A}$ *satisfies a label specification* SP *iff* $\mathcal{A}$ *satisfies all its axioms.* $Alg_{Lbl}(SP)$ *is the full subcategory of* $Alg_{Lbl}(\Sigma L)$ *containing all the algebras satisfying* SP*. A similar notation holds for* Gen_{Lbl}*.*

Notice that $Alg_{Lbl}(SP)$ or $Gen_{Lbl}(SP)$ can be empty categories (for example when SP contains φ and $\neg\varphi$). Providing that the axioms of SP never contain the connective "$\neg$", $Alg_{Lbl}(SP)$ has the same terminal object as $Alg_{Lbl}(\Sigma L)$: $\mathcal{T}riv$. However, as usual, initiality results can be easily obtained only for positive conditional specifications [WB80]. These results are provided in Section 4.

3.2 The partial evaluation constraint

Certain applications of label algebras require that if a term $t \in \overline{A}$ is labelled by l then every *partial evaluation* of t is still labelled by l. For instance, let us return to observability. The predicate Obs used by [Hen89] to characterize the observable values can be reflected by a label: $t \in Ok$ will now mean that the term t is observable. Let us consider the algebra A=$\mathbf{Z}$ and let us assume that A is observed via some boolean terms of $\overline{A}$. If the term $[pred(pred(0)) \leq succ(succ(0))]$ is labelled by Obs, then we clearly would like $[pred(-1) \leq succ(1)]$ to also be observable (i.e. labelled by Obs), as well as $[-2 \leq succ(1)]$, $[-2 \leq 2]$, or $true$ itself. Similarly, when exception handling is involved, if the term $((3 + 4) - 5)$ is an Ok-term (i.e. labelled by Ok), then we probably would like for $(7 - 5)$, or 2, to be also labelled by Ok. More generally, the terms labelled by Ok are sequences of calculi which contain only "normal treatments." This means that an exceptional treatment cannot appear at any stage of the evaluation of such terms. Thus, any partial evaluation of any term labelled by Ok can also be labelled by Ok. Intuitively, since a term of $\overline{A}$ reflects the history

[2] Allowing existential quantifiers is not difficult at all, but the management of the assignments for the satisfaction relation becomes rather tedious...

of a value, if this history does not raise exceptional cases then it can be entirely or partially forgotten (via partial evaluations); but it cannot be forgotten if the term is exceptional because exceptional treatments are specified with respect to this history (often via pattern matching). Thus, this partial evaluation constraint cannot be required for labels which reflect exception names. It would disallow some recovery cases. (For more details, see Section 5.2.) Summing up, some labels of a label signature should follow the so called partial evaluation constraint while some other ones should not.

Definition 8 *A* constrained label signature *is a triple* $\widehat{\Sigma L} = < S, \Sigma, \widehat{L} >$ *where* $< S, \Sigma >$ *is a usual signature and* $\widehat{L}$ *is a couple* (L, C) *such that* L *and* C *are disjoint sets of labels. The labels of* L *are called "unconstrained;" the ones of* C *are called "constrained." We shall note* $\widetilde{L} = L \cup C$ *the set of all labels, and* $\Sigma\widetilde{L} = < S, \Sigma, \widetilde{L} >$ *the corresponding (unconstrained) label signature.*

A constrained label signature can be seen as a label signature (with respect to $\widetilde{L}$) such that a subset of constrained labels (C) is distinguished.

Definitions 9 *Let* A *be a* Σ*-algebra and let* t *be a term in* $\overline{A} = T_\Sigma(A)$.

Let u *be (an occurrence of) a subterm of* t *and let* v *be any term in* $\overline{A}$ *of the same sort as* u. *The term* $t[u \leftarrow v]$ *is the term of* $\overline{A}$ *obtained by replacing (the considered occurrence of)* u *by* v *in* t.

When v *is the value* $eval_A(u)$ *of* A *(remember that* A *is included in* $\overline{A} = T_\Sigma(A)$*), the term* $t[u \leftarrow eval_A(u)]$ *is a* partial evaluation *of* t. *More generally, a term* t' *is a partial evaluation of* t *if it can be obtained via a finite sequence of such partial evaluations.*

For example, $(7 - 5)$ is a partial evaluation of $(3 + 4) - 5$ while $6 - 4$ is not.

Definition 10 *Given a constrained label signature* $\widehat{\Sigma L}$. *A* $\widehat{\Sigma L}$*-algebra is a label algebra over the signature* $\Sigma\widetilde{L}$ *which satisfies the following* partial evaluation constraint*: for every label* l *in* C *and for every term* t *in* $\overline{A}$, *if* t *belongs to* l_A *then all partial evaluations of* t *still belong to* l_A.

$Alg_{Lbl}(\widehat{\Sigma L})$ *is the full subcategory of* $Alg_{Lbl}(\Sigma\widetilde{L})$ *which contains all the* $\widehat{\Sigma L}$*-algebras. A similar notation holds for* Gen_{Lbl}.

Intuitively, the stability of labelling by constrained labels with respect to partial evaluation means that, as soon as a term has been labelled, one can forget its "old history" without modifying its constrained labels. Equivalently, since a term represents a sequence of calculi, it means that constrained labelling does not depend on the particular computational strategy. The only point which matters is that there exists at least one strategy which yields the constrained label.

Remarks 1 $\mathcal{T}_{\Sigma\widetilde{L}}$ and $\mathcal{T}riv$ are constrained label algebras, whatever C is.

If t' is a partial evaluation of t then $eval_A(t') = eval_A(t)$ (the converse property is false). Moreover, for every constrained label algebra $\mathcal{A}$, as the (constant) term $eval_A(t)$ is a partial evaluation of t, constrained labels are compatible with $eval_A$ in the sense that $\forall l \in C, eval_A(l_A) \subseteq l_A$.

The partial evaluation constraint cannot be specified using label axioms.

Definitions 11 *Given a constrained signature* $\widehat{\Sigma L}$, *a* $\widehat{\Sigma L}$-axiom *is simply a* $\Sigma\widetilde{L}$*-axiom. A* constrained label specification *is a pair* $\widehat{SP} = < \widehat{\Sigma L}, Ax >$ *where* $\widehat{\Sigma L}$ *is a constrained label signature and* Ax *is a set of* $\widehat{\Sigma L}$*-axioms.*

The category $Alg_{Lbl}(\widehat{SP})$ *is the full subcategory of* $Alg_{Lbl}(\widehat{\Sigma L})$ *containing all the algebras satisfying* Ax *(according to the satisfaction relation defined in Definition 7). An object of* $Alg_{Lbl}(\widehat{SP})$ *is called a* $\widehat{SP}$*-algebra. (Similar definitions hold for* $Gen_{Lbl}(\widehat{SP})$*.)*

A signature name (or specification name, etc.) surrounded by a hat means now "constrained."

4 Fundamental results

This section deals with initiality results. We show that the classical results of [GTW78] can be extended to the framework of label algebras. They are given for constrained label specifications. Of course, they remain valid for unconstrained label specifications, since a unconstrained specification SP is a constrained specification $\widehat{SP}$ such that $C = \emptyset$. For lake of space, the results are not proved or their proofs are only sketched. Complete proofs can be found in [BL91].

Theorem 1 *Let $\widehat{SP}$ be a positive conditional $\widehat{\Sigma\widetilde{L}}$-specification. Let $\mathcal{X}$ be a $\Sigma\widetilde{L}$-algebra. Let R be a binary relation over X compatible with the sorts of the signature (i.e. R is a subset of $\bigcup_{s\in S} X_s \times X_s$). There is a least $\widehat{SP}$-algebra $\mathcal{Y}$ such that there exists a label morphism $h_Y : \mathcal{X} \to \mathcal{Y}$ and such that $(\mathcal{Y}, h_Y)$ is compatible with R (i.e. $\forall x, y \in X,\ x\ R\ y \implies h_Y(x) = h_Y(y)$).*

Sketch of proof: Let F be the family of all couples $(\mathcal{Z}, h_Z : \mathcal{X} \to \mathcal{Z})$ compatible with R. Let us consider the $\Sigma\widetilde{L}$-algebra $\mathcal{Y} = (Y, \{l_Y\}_{l\in\widetilde{L}})$ as the quotient algebra of $\mathcal{X}$ (h_Y is the quotient $\Sigma\widetilde{L}$-morphism) defined by:

- $\forall x, y \in X,\ (\ h_Y(x) = h_Y(y) \ \Leftrightarrow\ (\forall (\mathcal{Z}, h_Z) \in F,\ h_Z(x) = h_Z(y))\ )$
- $\forall l \in \widetilde{L},\ \forall x \in \overline{X},\ (\ \overline{h_Y}(x) \in l_Y \ \Leftrightarrow\ (\forall (\mathcal{Z}, h_Z) \in F,\ \overline{h_Z}(x) \in l_Z)\ )$

For every $(\mathcal{Z}, h_Z)$ in F, there exists a $\Sigma\widetilde{L}$-morphism $\mu_Z : \mathcal{Y} \to \mathcal{Z}$ defined by $\forall x \in X, \mu_Z(h_Y(x)) = h_Z(x)$. Consequently, if $(\mathcal{Y}, h_Y)$ belongs to F then it is its smallest element. It is then not too difficult to prove that $(\mathcal{Y}, h_Y)$ is compatible with R and that $\mathcal{Y}$ satisfies $\widehat{SP}$. This concludes the proof of the theorem. ■

Theorem 2 *Let $\widehat{SP}$ be a positive conditional label specification. $Alg_{Lbl}(\widehat{SP})$ and $Gen_{Lbl}(\widehat{SP})$ have an initial object $\mathcal{T}_{\widehat{SP}}$. $\mathcal{T}riv$ is final in $Alg_{Lbl}(\widehat{SP})$ (and in $Gen_{Lbl}(\widehat{SP})$ if $\widehat{\Sigma\widetilde{L}}$ is sensible).*

Sketch of proof: by using Theorem 1 with $R = \emptyset$, $\mathcal{X} = \mathcal{T}_{\Sigma\widetilde{L}}$, we get $\mathcal{Y} = \mathcal{T}_{\widehat{SP}}$. ■

The purpose of the remainder of this section is to give some basic tools for manipulating structured positive conditional label specifications. We first define the *forgetful functor* U associated with a structured presentation, then the *synthesis functor* F, and we prove that F is left adjoint for U.

Definition 12 *Let $\widehat{\Sigma\widetilde{L}_1}$ and $\widehat{\Sigma\widetilde{L}_2}$ be such that $\widehat{\Sigma\widetilde{L}_1} \subseteq \widehat{\Sigma\widetilde{L}_2}$ (i.e. $S_1 \subseteq S_2$, $\Sigma_1 \subseteq \Sigma_2$, $L_1 \subseteq L_2$ and $C_1 \subseteq C_2$). The forgetful functor $U : Alg_{Lbl}(\widehat{\Sigma\widetilde{L}_2}) \to Alg_{Lbl}(\widehat{\Sigma\widetilde{L}_1})$ is defined as follows:*

- *for each $\widehat{\Sigma\widetilde{L}_2}$-algebra $\mathcal{A}$, $U(\mathcal{A})$ is the $\widehat{\Sigma\widetilde{L}_1}$-algebra $\mathcal{B}$ defined by:*
$$\forall s \in S_1,\ B_s = A_s\ ;\quad \forall l \in \widetilde{L_1},\ l_B = l_A \cap \overline{B}\ ;\quad \text{and}\ \ \forall f \in \Sigma_1,\ f_B = f_A\ ;$$
- *for each $\widehat{\Sigma\widetilde{L}_2}$-morphism $\mu : \mathcal{A} \to \mathcal{A}'$, $U(\mu) : U(\mathcal{A}) \to U(\mathcal{A}')$ is the $\widehat{\Sigma\widetilde{L}_1}$-morphism μ restricted to $\mathcal{B} = U(\mathcal{A})$ and co-restricted to $\mathcal{B}' = U(\mathcal{A}')$.*

Remark 2 $\mathcal{B} = U(\mathcal{A})$ satisfies the partial evaluation constraint with respect to $\widehat{L_1}$ and $\overline{U(\mu)}$ clearly preserves the labels of $\widetilde{L_1}$.

Theorem 3 *Let $\widehat{SP_1}$ and $\widehat{SP_2}$ be two label specifications such that $\widehat{SP_1} \subseteq \widehat{SP_2}$. Let U be the forgetful functor from $Alg_{Lbl}(\widehat{\Sigma\widetilde{L}_2})$ to $Alg_{Lbl}(\widehat{\Sigma\widetilde{L}_1})$. The restriction of U to $Alg_{Lbl}(\widehat{SP_2})$ can be co-restricted to $Alg_{Lbl}(\widehat{SP_1})$. More generally, given two signatures $\widehat{\Sigma\widetilde{L}_1} \subseteq \widehat{\Sigma\widetilde{L}_2}$, for all $\widehat{\Sigma\widetilde{L}_2}$-algebras $\mathcal{A}$ and for all $\widehat{\Sigma\widetilde{L}_1}$-axioms φ we have:* $\quad \mathcal{A} \models \varphi \implies U(\mathcal{A}) \models \varphi$

Remark 3 Theorem 3 does not require for the axiom φ to be positive conditional.

The reverse implication is not valid in general, as shown in the following example. Consequently, the so-called "satisfaction condition" does not hold for label algebras; the framework

of label algebras is not an institution [GB84], at least with the natural definitions of signature morphisms and axiom translations.

Example 2 Let ΣL_1 be the label signature defined by $S_1 = \{ thesort \}$, $\Sigma_1 = \{ cst1 : \rightarrow thesort \}$ and $L_1 = \{ thelabel \}$ (it does no matter if *thelabel* is constrained or not in this example). Let ΣL_2 be the label signature defined by $S_2 = S_1$, $\Sigma_2 = \{ cst1 : \rightarrow thesort , cst2 : \rightarrow thesort \}$ and $L_2 = L_1$. We clearly have $\Sigma L_1 \subset \Sigma L_2$. Let $\mathcal{A}$ be the ΣL_2-algebra defined by $A = \{a = cst1_A = cst2_A\}$ (A is a singleton) and $thelabel_A = \{ a , cst1 \}$ (let us remind that $T_{\Sigma_2}(A) = \{ a , cst1 , cst2 \}$). The ΣL_1-algebra $U(\mathcal{A})$ is then characterized by $U(A) = \{a = cst1_{U(A)}\}$ and $thelabel_{U(A)} = \{ a , cst1 \}$; thus, $thelabel_{U(A)} = T_{\Sigma_1}(U(A))$. Consequently, $U(\mathcal{A})$ satisfies the ΣL_1-axiom "$x \epsilon thelabel$" while $\mathcal{A}$ does not (as $cst2$ does not belong to $thelabel_A$).

Theorem 4 *Let $\widehat{SP_1}$ and $\widehat{SP_2}$ be two positive conditional label specifications such that $\widehat{SP_1} \subseteq \widehat{SP_2}$. There exists a synthesis functor $F : Alg_{Lbl}(\widehat{SP_1}) \rightarrow Alg_{Lbl}(\widehat{SP_2})$ which is a left adjoint for U.*

Sketch of proof: For every $\widehat{SP_1}$-algebra $\mathcal{A}$, $F(\mathcal{A})$ is a quotient of $T_{\Sigma_2}(\mathcal{A})$ defined using Theorem 1, with a well chosen initial labelling for $\widehat{L_1}$. The adjunction can be proved with the Yoneda lemma [McL71]. ■

As usual, the adjunction $I_{\mathcal{A}} : \mathcal{A} \rightarrow U(F(\mathcal{A}))$ can be used to define *hierarchical consistency* ("no-collapse" property) and *sufficient completeness* ("no-junk" property) for structured specifications.

Remark 4 We have shown in this subsection that the framework of label algebras does not form an institution [GB84], even if restricted to positive conditional axioms because Example 2 shows that we do not have the reverse implication of the satisfaction condition given in the Theorem 3. Indeed, we have proved that the framework of positive conditional label algebras form a specification logic which has free constructions [EBO91][EBCO91]. Let us point out that the specification logic of label algebras has *not* amalgamations (as defined in [EBCO91]). The reason *a priori* is that we show in Section 5.1 that observational semantics can be reflected within label algebras, and [EBCO91] has proved that observational semantics have not amalgamations in general. It is the same for extensions (at least if we do not restrict the definition of morphisms).

5 Some applications of label algebras

We have mentioned so far that labels can be used within frameworks devoted to observability or exception handling. Indeed, labels provide a great tool to express several other features already developed in the field of (first order) algebraic specifications.

5.1 Some possible applications

We have mentioned in the introduction that the framework of label algebras can be shown as an extension of more standard algebraic approaches based on "multityping." More precisely, we can *specify multityping* by means of label specifications. Indeed the difference between a label and a type is that labels are carried by terms (in $\overline{A}$) while type names are carried by values (in A). Thus a label l can easily play the role of a type name: it is sufficient to saturate each fiber of $eval_A : \overline{A} \rightarrow A$ which contains a term labelled by l. This is easily specified by a ΣL-axiom of the form:

$$x \epsilon l \wedge x = y \implies y \epsilon l$$

where x and y are variables. For every model A satisfying such axioms for l belonging to L, two terms u and v of $\overline{A}$ having equal values in A are necessarily labelled by the same labels, thus labels can play the role of types. Notice that we should write one axiom of this form for each

sort belonging to S because the variables x and y are typed with respect to S in our framework. Nevertheless, as far as we intend to simulate types by labels, S should be a singleton. Thus, the "typing" of terms, as well as variables, becomes explicit in the precondition of each axiom. Therefore, this approach leads to consider typing as "membership constraints." For finitely generated algebras, such a specification style facilitates theorem proving, as demonstrated in [Smo86][Com90].

An advantage of such an approach is that additional properties about types, according to the needs of the considered application, can be easily specified within the same framework. For example, let us consider a property such as $s \leq s'$ between two sorts in the framework of *order sorted algebras* [FGJM85]. It can be specified within the framework of label specifications by $x \epsilon s \Rightarrow x \epsilon s'$ where s and s' are labels which simulate the corresponding (sub)sorts. In the same way, it is possible to specify *dependent types* such as binary search tree (the specifications of natural numbers and booleans are supposed already written by another way):

$S = \{All\}$
$\Sigma = \{empty, node\_\ \_\ \_, root\_, max\_, min\_\}$
$L = \{Bool, Nat, Notdefined, Bst, Sta, Gta\}$[3]

with the following axioms:

$empty \epsilon Bst$
$max(empty) \epsilon Sta$
$min(empty) \epsilon Gta$
$x \epsilon Sta \wedge n \epsilon Nat \Longrightarrow x \leq n = true$
$x \epsilon Gta \wedge n \epsilon Nat \Longrightarrow n \leq x = true$
$a \epsilon Bst \wedge b \epsilon Bst \wedge n \epsilon Nat \wedge max(a) \leq n = true \wedge n \leq min(b) = true \Longrightarrow node\ a\ n\ b \epsilon Bst$
$root(empty) \epsilon Notdefined$
$node\ a\ n\ b \epsilon Bst \Rightarrow root\ (node\ a\ n\ b) = n$

Algebraic specifications with *partial functions* can also be reflected via label specifications. They often rely on an additional predicate D which is used to specify the definition domain of each operation of the signature ([BW82] and others). Thus, atoms are either equalities, or of the form $D(t)$, where t is a term with variables. It is of course not difficult to translate $D(t)$ to $(t \epsilon IsDefined)$; we simply have to specify the propagation of the definition domains with respect to any operation f of the signature:

$$f(x_1, \ldots, x_n) \epsilon IsDefined \implies x_1 \epsilon IsDefined \wedge \ldots \wedge x_n \epsilon IsDefined$$

Then, the label $IsDefined$ can be used in the preconditions of the axioms defining the partial operations in such a way that every label algebra $\mathcal{A}$ satisfying the resulting label specification has the property that $eval_A(IsDefined_A)$ is a subset of A that behaves like a partial algebra satisfying the original specification (see also [AC91]).

In the same way, labels can be used to give a refined semantics of the predefined *predicates of specification languages*. For example in *Pluss* [Bid89], an expression of the form "t `is defined when` *something*" can be reflected by the following label axiom: $something \Rightarrow t \epsilon IsDefined$

More generally, labels are indeed unary predicates on terms; thus, they can be at least used as *predicates* on values (using the label axiom already mentioned for multityping). The advantage of such predicates is that their semantics is not defined via a hidden boolean sort: using booleans to define predicates is often unsatisfactory because it assumes that the specification is consistent with respect to boolean values. In this way, labels can advantageously be used in a specification to provide additional informations about the specified data types. For instance, we can write:

[3] *Bst* for Binary Search Tree; *Sta* for *Smaller-Than-All* and *Gta* for *Greater-Than-All*

$0 \in Even$
$n \in Even \implies succ(n) \in Odd$
$n \in Odd \implies succ(n) \in Even$
$exp(n, 0) = succ(0)$
$succ(m) \in Odd \implies exp(n, succ(m)) = exp(n, m) \times n$
$m \in Even \implies exp(n, m) = exp(n \times n\ ,\ m\ div\ succ(succ(0))\ )$

Let us return to the application of label algebras to observability (see Section 2). Clearly, the predicate *Obs* used by [Hen89] can be reflected by a label. More generally, labels can be used to characterize observable *terms*. By distinguishing a subset ΣObs of Σ, the framework of [BB91] introduces two distinct notions that reflect a hierarchy in the definition of observability. The terms that only contain operations belonging to ΣObs are said to "allow observability" (the other ones can never be observed). Then, a term "allowing observability" really becomes "observable" only if it belongs to an observable sort. It is not difficult to reflect the observational hierarchy defined in [BB91] by using two distinct labels denoted *AllowsObs* and *Obs*. For each operation f allowing observability (i.e. belonging to the considered subset ΣObs of the signature), it is sufficient to consider the following label axiom:

$$x_1 \in AllowsObs\ \wedge \ldots \wedge\ x_n \in AllowsObs \implies f(x_1, \ldots, x_n) \in AllowsObs$$

The fact that a term allowing observability becomes observable if and only if it belongs to an observable sort s can easily be specified by the label axiom (one axiom for each observable sort): $x \in AllowsObs \Rightarrow x \in Obs$ where x is a variable of sort s. As shown in [BB91], this label *Obs* is carried by terms, not by values contrarily to the predicate *Obs* used in [Hen89]. Hopefully, the advantages of the Hennicker's approach could be preserved, since they mainly rely on the explicit specification of the predicate *Obs*.

5.2 Application to exception handling

Let us define the framework of exception algebras as a specialization of the one of label algebras, where the labels are used for exception handling purposes. The particular label *Ok* will be distinguished to characterize the normal cases; exception names and error messages will be reflected by all the other labels. This allows us to take exception names into account in the axioms. Intuitively, in an exception algebra $\mathcal{A}$, $t \in l_A$ with $l \neq Ok$ will mean that the calculus defined by t leads to the exception name l; and $t \in Ok_A$ will mean that the calculus defined by t is a "normal" calculus (i.e. it does not need an exceptional treatment and the calculus is successful). An exception signature is then by definition a label signature $\widehat{\Sigma L}$ with $\widehat{L} = (L, \{Ok\})$ where L is the set of exception names. In order to reflect the specific behaviours of data with exception handling, we add some implicit label axioms.

An important implicit aspect is the "common future" property. Let us consider $\mathcal{A}$ reflecting the natural numbers bounded by *maxint*, the terms $succ^i(0)$ with $i \leq maxint$ being labelled by *Ok*. Let us assume that $succ^{maxint+1}(0)$ is recovered on $succ^{maxint}(0)$. Once this recovering is done, we want everything to happen as if the exception $succ^{maxint+1}(0)$ were never raised; this is the very meaning of the word recovery. The same succession of operations applied to $succ^{maxint}(0)$ or to $succ^{maxint+1}(0)$ should return the same value and raise the same exception names. If $succ^{maxint+1}(0)$ is labelled by *TooLarge*, then the term $t = succ^{maxint+2}(0)$ should also be labelled by *TooLarge*, since $succ^{maxint+1}(0) = t[succ^{maxint+1}(0) \leftarrow succ^{maxint}(0)]$. In a label algebra $\mathcal{A}$, $eval_A(u) = eval_A(v)$ implies for every term t containing u as strict subterm, that t and $t[u \leftarrow v]$ have the same value, but it does not imply that they have the same labels. On the contrary, such a property will be required for exception names in exception algebras. An exception algebra is then by definition a $\widehat{\Sigma L}$-algebra which satisfies the common future property.

As usual, when specifying a data structure with exception handling features, the specifier first declares the desired Ok-part. Let us assume that all the terms $succ^i(0)$ with $i \leq maxint$ are labelled by Ok and that the specification contains also the following "normal axiom:" $pred(succ(n)) = n$. Then, for instance, the term $pred(succ(0))$ should also belong to the Ok-domain because its calculus does not require any exceptional treatment and leads to the Ok-term 0 via the previous normal axiom. By a terseness principle, labelling by Ok must be *implicitly* propagated through the axioms kept for normal cases. Since label algebras have no implicit aspects, the semantics of exception specifications must implicitly add label axioms reflecting these properties.

As Ok-axioms require special semantics, it is necessary to separate the axioms concerning exceptional cases (given by $GenAx$) from the Ok-axioms which concern normal cases. Thus an exception specification $SPEC$ is defined as a triple $< \widehat{\Sigma L}, GenAx, OkAx >$ where $GenAx$ is a set of generalized axioms (which are positive conditional $\widehat{\Sigma L}$-axioms) and where $OkAx$ is a set of Ok-axioms (which are positive conditional $\widehat{\Sigma L}$-axioms with a conclusion of the form $u = v$).

- $GenAx$ is mainly devoted to exception handling. Its first purpose concerns labelling of terms. The axioms with a conclusion of the form $t \in Ok$ (resp. $t \in l$ with $l \in L$) mean that t is a normal term (resp. the heading function of the term t raises the exception name l). The second purpose of $GenAx$ is to handle the exceptional cases, in particular to specify recoveries, according to the previous labelling of terms. The corresponding axioms will have a conclusion of the form $u = v$. As the axioms of $GenAx$ concern all terms, exceptional or not, the satisfaction of such axioms will simply be the same as for label axioms.

- $OkAx$ is entirely devoted to the normal cases, and will only concern terms labelled by Ok. The semantics of $OkAx$ must be carefully restricted to Ok-assignments, in order to avoid inconsistencies (see Section 2). It will both treat equalities between Ok-terms and carefully propagate labelling by Ok through these equalities.

(For an example of exception specification, see Example 3 below)

Definition 13 *Let $\widehat{\Sigma L}$ be an exception signature. An exception algebra $\mathcal{A}$ satisfies an Ok-axiom of the form $P \Rightarrow v = w$, where P is the precondition,*[4] *if and only if for all assignments σ with range in $\overline{A}$ (covering all the variables of the axiom) which satisfy the precondition (i.e. $\mathcal{A}$ as $\widehat{\Sigma L}$-algebra satisfies the "ground" label axiom $\sigma(P)$), the two following properties hold:*

*Ok-*propagation*: if at least one of the terms $\sigma(v)$ or $\sigma(w)$ belongs to Ok_A and the other one is of the form $f(t_1, \ldots, t_p)$ with all the t_i belonging to Ok_A*[5]*, then both $\sigma(v)$ and $\sigma(w)$ belong to Ok_A.*

*Ok-*equality*: if $\sigma(v)$ and $\sigma(w)$ belong to Ok_A then $eval_A(\sigma(v)) = eval_A(\sigma(w))$.*

The first property of the definition reflects a careful propagation of the label Ok (which starts from the Ok-terms declared in $GenAx$). Intuitively, such an innermost evaluation reflects an implicit propagation of exceptions because a term can be labelled by Ok through an Ok-axiom only if all the arguments of its heading function are already labelled by Ok. The second property specifies the equalities that must hold for the normal cases. Two terms can get the same evaluation through an Ok-axiom only if they are both labelled by Ok.

We denote by $Alg_{Exc}(SPEC)$ the full subcategory of $Alg_{Lbl}(\widehat{\Sigma L})$ containing all the exception algebras satisfying $SPEC$ (i.e. which satisfy each axiom of $SPEC$).

Theorem 5 *Let $SPEC =< \widehat{\Sigma L}, GenAx, OkAx >$ be an exception specification. There exists a positive conditional label specification $Tr(SPEC)$, over the same label signature $\widehat{\Sigma L}$ such that $Alg_{Exc}(SPEC) = Alg_{Lbl}(Tr(SPEC))$.*

[4] P may be empty.

[5] p may be equal to 0

Sketch of proof: It is sufficient to add well chosen label axioms which reflect the common future property and to add for each Ok-axiom of $SPEC$ a set of label axioms reflecting the Definition 13. ■

Remark 5 $Tr(SPEC)$ may contain a great number of label axioms (related to the number of operations and labels of $\widehat{\Sigma L}$).

$Tr(SPEC)$ only contains positive conditional axioms. Thus, from Section 4 we have:

Theorem 6 *Let $SPEC$ be a exception specification. $Alg_{Exc}(SPEC)$ has an initial object $\mathcal{T}_{SPEC}$.*

Moreover, given two exception specifications $SPEC_1$ and $SPEC_2$ such that $SPEC_1 \subseteq SPEC_2$, the forgetful functor $U : Alg_{Exc}(SPEC_2) \rightarrow Alg_{Exc}(SPEC_1)$ exists and has a left adjoint functor F.

Example 3 Let $BoundedNat = < \{Nat\}, \{0, succ\_, pred\_\}, (\{TooLarge, Negative\}, \{Ok\}) >$ be the exception signature. An example of exception specification over this signature is given by:

$$
\begin{array}{lll}
GenAx & : & succ^{maxint}(0) \in Ok \\
 & & succ(n) \in Ok \Rightarrow n \in Ok \\
 & & succ^{maxint+1}(0) \in TooLarge \\
 & & pred(0) \in Negative \\
 & & succ(n) \in TooLarge \implies succ(n) = n \\
OkAx & : & pred(succ(n)) = n \\
\multicolumn{3}{l}{Where\ :\ n\ : Nat}
\end{array}
$$

The two first axioms specify the Ok domain of Nat. It is not necessary to declare *all* the Ok-terms (the label Ok will be automatically be propagated to terms such as $pred(succ(0))$ via the Ok-axiom). Even if it is generally easier to recursively specify the Ok domain, it is not mandatory; it is only desirable to declare at least one term for each intended Ok-value. The third and fourth axioms declare exception names. Their meaning is that the operation *succ* (resp. *pred*) raises the exception *TooLarge* (resp. *Negative*) when applied to *maxint* (resp. 0). Finally, the last axiom of *GenAx* recovers the terms labelled by *TooLarge* by expressing the following exceptional treatment: *"if the operation succ raises the exception TooLarge, then do not perform it."*.

When compared to the formalism of [BBC86], our formalism is much simpler because its semantics can be translated into the one of the label algebras which is more easily understandable. Moreover, we justify in Section 2 the necessity of labelling terms instead of values with respect to the label Ok. But, it is also crucial to label terms (and not values) by the exception names. In [BBC86], exception names are carried by values instead of terms (while the label Ok is rightly carried by terms). If we consider the last axiom of *GenAx* of the Example 3, this would lead to inconsistencies. The term $succ(maxint)$ being equal to the term *maxint*, both of them are labelled by *TooLarge*. Let $m = maxint - 1$; since *maxint* is labelled by *TooLarge*, we get the following inconsistency: $maxint = succ(m) = m$. This inconsistency propagates in the same way, and all values are equal to 0. Thus, our formalism of exception algebras is not only simpler than [BBC86] but it is also better than [BBC86] because Example 3 correctly behaves with respect to our semantics, while it is inconsistent with respect to the semantics of [BBC86].

As already mentioned (see page 6), it would make no sense to consider a partial evaluation constraint with respect to $\widehat{L} = (\emptyset, L \cup \{Ok\})$. In the Example 3, the partial evaluation constraint for the label *TooLarge* would lead to label the constant *maxint* with *TooLarge* since *maxint* is a partial evaluation of $succ(maxint)$. Thus, everything would happen as if exception names were carried by values, leading to the same inconsistency as for the semantics of [BBC86].

Summing up, as for the application to exception handling, all the possible applications mentioned in Section 5.1 require some generic label axioms which must be *implicit*. These axioms

should be considered as modifiers of the semantics. Thus, the framework of label algebras provides us with "low level" algebraic specifications. When an algebraic specification $SPEC$ is written according to some special semantics (e.g. observational specifications or exception algebras), it has to be "compiled" (translated) to a label specification $Tr(SPEC)$.

6 Conclusion

We have shown that observational approaches or exception handling require a refined notion of the satisfaction relation for algebraic specifications. The scope of an axiom must be restricted to carefully chosen patterns, because a satisfaction relation based on assignments with range in values often raises inconsistencies. A more elaborated notion of assignment must be considered: assignment with range in terms. This allows us to restrict the scope of an axiom to certain suitable patterns, and solves the inconsistencies raised by exception handling. We use *labels* to characterize these suitable patterns. In order to avoid inconsistencies, labels must not go through equational atoms; thus, two terms having the same value do not necessarily carry the same labels. We have first defined the framework of *label algebras*, that defines suitable semantics for labels. The scope of the label axioms is carefully delimited by labels which serve as special marks on terms.

We have applied with success the formalism of label algebras in order to avoid usual inconsistencies raised by exception handling. This approach is powerful enough to cope with all suitable exception handling features such as implicit propagation of exceptions, possible recoveries, declaration of exception names, bounded data structures, etc.

The application domain of label algebras seems to be much more general than exception handling. Indeed, labels provide a great tool to express several other features developed in the field of (first order) algebraic specifications. We have outlined in Section 5.1 how label algebras can be used to specify several more standard algebraic approaches such as order sorted algebras [Gog83], partial functions [BW82] or observability issues [Hen89][BB91]. However, as for the application to exception handling, all the specific applications of label algebras require certain *implicit* label axioms or constraints. Thus, the framework of label algebras provides us with "low level" algebraic specifications: in a generic way, the specific semantical aspects of a given approach (e.g. subsorting or exception handling) can be specified by a well chosen set of label axioms.

We have shown in Section 4 that the framework of label algebras restricted to positive conditional axioms, and consequently the one of exception algebras, form a specification logic which has free constructions [EBO91][EBCO91].[6] These results provide us with a first basis to study modularity for label specifications. However, modularity should be studied according to the specific application under consideration (behavioural specifications, exception specifications, etc). Indeed, it can be shown that the existing frameworks for modularity do not cope with exception handling because structured exception specification must allow erroneous "junk" (see [BBC86]).

Several other extensions of the framework of label algebras will probably give promising results. Intuitively, labels are unary predicates on terms. In order to facilitate certain applications of label algebras, we plane to generalize labels to "labels of strictly positive arity." Theorem proving methods according to the semantics of label algebras should be studied in future works. Higher order label specifications may also be dealt with in future works. Last, but not least, let us mention that bounded data structures play a crucial role in the theory of *testing* because test data sets should contain many elementary tests near the bounds. One of our current researches is to extend the theory of test data selection from algebraic specifications described in [BGM91] to exception specifications.

[6] but it does not form a liberal institution [GB84].

Acknowledgements: Several discussions with Marie-Claude Gaudel and Michel Bidoit have been helpful. This work has been partially supported by CNRS GRECO de Programmation and EEC Working Group COMPASS.

References

[AC91] Astesiano E., Cerioli M. *Non-strict don't care algebras and specifications.* TAPSOFT CAAP, Brighton U.K., April 1991, Springer-Verlag LNCS 493, p.121-142.

[BB91] Bernot G., Bidoit M. *Proving the correctness of algebraically specified software: Modularity and Observability issues.* Proc. of AMAST-2, Second Conference of Algebraic Methodology and Software Technology, Iowa City, Iowa, USA, May 1991.

[BBC86] Bernot G., Bidoit M., Choppy C. *Abstract data types with exception handling : an initial approach based on a distinction between exceptions and errors.* Theoretical Computer Science, Vol.46, n.1, pp.13-45, Elsevier Science Pub. B.V. (North-Holland), November 1986. (Also LRI Report 251, Orsay, Dec. 1985.)

[BGM91] Bernot G., Gaudel M.C., Marre B. *Software testing based on formal specifications: a theory and a tool.* to appear in Software Engineering Journal, December 1991.

[BL91] Bernot G., Le Gall P. *Label algebras and exception handling.* To appear in LRI Research Report, University of Orsay Paris XI, 1991.

[Bid89] Bidoit M. *Pluss, un langage pour le développement de spécifications algébriques modulaires.* Thèse d'état, University of Orsay Paris XI, 1989.

[Bre91] Breu M. *Bounded Implementation of Algebraic Specifications.* 8th Workshop on Specification od Abstract DataTypes, Dourdan, 1991.

[BW82] Broy M., Wirsing M. *Partial abstract data types.* Acta Informatica, Vol.18-1, Nov. 1982.

[Com90] Comon H. *Equational formulas in order-sorted algebras.* Proc. ICALP, Warwick, Springer-Verlag, July 1990.

[EBCO91] Ehrig H., Baldamus M., Cornelius F., Orejas F. *Theory of algebraic module specification including behavioural semantics, constraints and aspects of generalized morphisms.* Proc. of AMAST-2, Second Conference of Algebraic Methodology and Software Technology, Iowa City, Iowa, USA, May 1991.

[EBO91] Ehrig H., Baldamus M., Orejas F. *New concepts for amalgamation and extension in the framework of specification logics.* Research report Bericht-No 91/05, Technische Universitat Berlin, May 1991.

[EM85] Ehrig H., Mahr B. *Fundamentals of Algebraic Specification 1. Equations and initial semantics.* EATCS Monographs on Theoretical Computer Science, Vol.6, Springer-Verlag, 1985.

[FGJM85] Futatsugi K., Goguen J., Jouannaud J-P., Meseguer J. *Principles of OBJ2.* Proc. 12th ACM Symp. on Principle of Programming Languages, New Orleans, january 1985.

[GB84] Goguen J.A., Burstall R.M. *Introducing institutions.* Proc. of the Workshop on Logics of Programming, Springer-Verlag LNCS 164, pp.221-256, 1984.

[GDLE84] Gogolla M., Drosten K., Lipeck U., Ehrich H.D. *Algebraic and operational semantics of specifications allowing exceptions and errors.* Theoretical Computer Science 34, North Holland, 1984, pp.289-313.

[GM89] Goguen J.A., Meseguer J. *Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations.* Technical Report SRI-CSL-89-10, SRI, July 1989.

[Gog78] Goguen J.A. *Order sorted algebras: exceptions and error sorts, coercion and overloading operators.* Univ. California Los Angeles, Semantics Theory of Computation Report n.14, Dec. 1978.

[Gog83] Gogolla M. *Algebraic specification with partially ordered sorts and declarations.* Research report Forschungsbericht No 169, University of Dormund, 1983.

[Gog87] Gogolla M. *On parametric algebraic specifications with clean error handling.* Proc. Joint Conf. on Theory and Practice of Software Development, Pisa (1987), Springer-Verlag LNCS 249, pp.81-95.

[GTW78] Goguen J.A., Thatcher J.W., Wagner E.G. *An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types.* Current Trends in Programming Methodology, ed. R.T. Yeh, Printice-Hall, Vol.IV, pp.80-149, 1978. (Also IBM Report RC 6487, October 1976.)

[Hen89] Hennicker R. *Implementation of Parameterized Observational Specifications.* TapSoft, Barcelona, LNCS 351, vol.1, pp.290-305, 1989.

[LG86] Liskov B., Guttag J. *Abstraction and specification in program development.* The MIT Press and McGraw-Hill Book Company, 1986.

[Mos89] Mosses P. *Unified algebras and action semantics.* STACS 89, Paderborn, R. Cori (Ed.), Springer-Verlag LNCS 349, pp.17-35.

[McL71] Mac Lane S. *Categories for the working mathematician.* Graduate texts in mathematics, 5, Springer-Verlag, 1971

[Smo86] Smolka G. *Order-sorted horn logic: semantics and deduction.* Research report SR-86-17, Univ. Kaiserslautern, Oct. 1986.

[WB80] Wirsing M., Broy M. *Abstract data types as lattices of finitely generated models.* Proc. of the 9th Int. Symposium on Mathematical Foundations of Computer Science (MFCS), Rydzyna, Poland, Sept. 1980.