Exception handling and term labelling

Gilles Bernot[1] Pascale Le Gall[2]

Abstract: We propose a new algebraic framework for exception handling which is powerful enough to cope with many exception handling features such that recovery, implicit propagation of exceptions, etc. This formalism is capable of treating all the exceptional cases, including the following ones: "intrinsic" exceptions which are related to the underlying data structure (for instance, popping an emptstack or applying predecessor on zero for natural numbers), exceptions which are relied on "dynamic" properties (as an acces to a non-initialized array cell) or else exceptions which are due to certain limitations (mainly bounded data structures). We show that within the already existing frameworks, the case of bounded data structures with certain recoveries of exceptional values remains unsolved.

First, we justify the usefulness of "labelling" some terms in order to easily specify exceptions without inconsistency, and we then define a general framework of *label algebras* which allows us to "type" *terms* instead of *values*. *Exception algebras* and *exception specifications* are defined as a direct application of label algebras. Indeed, the usual inconsistency problems raised by exception handling are avoided by the possibility of labelling terms.

As a conclusion, we also sketch out how far the application domain of label algebras seems to be much more general than exception handling.

Key-words: Algebraic specifications of abstract data types, Error and exception handling, Exception recovery, Bounded data structures, Structured specifications.

1 Introduction

Exception handling is often neglected in software engineering, especially at the specification stage. This results in incomplete specifications and the various choices of "how to treat exceptional cases" are then often made at the programming stage. As usual when specifications are incomplete, this decreases the overall quality of the software. When the exceptional cases are not well specified, the corresponding bugs are very difficult to identify, as they do not cope with the standard verification and validation methods (proving or testing methods). Nevertheless, most of these exceptional cases are fairly easy to classify. An important class of exceptional cases is related to "intrinsic" properties of the underlying abstract data types: access to an empty data structure (e.g. popping an empty stack) or functions which are intrinsically not defined for certain values (e.g. predecessor for 0 in natural numbers). Another important class of exceptional cases relies on "dynamic" properties of the data structure (e.g. access to a non-initialized array cell). In addition, it is very important not to neglect certain limitations, due to the system itself or required by the specifier (mainly bounded data structures). If bounded data structures are not taken into account at the specification level, then almost all the specified properties are actually false; and precisely, in practice, softwares requires a strong verification and validation effort near the bounds of the underlying data structures.

Over the last fifteen years [LZ75][Gut75][GTW78], *algebraic specifications* have been widely advocated as one of the most fruitful formal specification methods. In this paper, a new framework for exception handling within algebraic specifications, covering all the classes of exceptional cases, is proposed. First of all, we will justify the introduction of a new algebraic framework by studying

[1] LIVE, Université d'Evry - Val d'Essonne, Bd des Coquibus, 91025 Evry Cedex, FRANCE. (bernot@lri.fr)

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

different exception handling cases. Then, before defining *exception algebras*, we will introduce a new general framework, the *label algebras*. Exception specifications will be defined as a direct application of label specifications.

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

2 Crucial aspects of exception handling

Obviously, a good exception handling framework must cover all kinds of exceptional cases. Two additional crucial aspects are often neglected, namely *clarity* and *terseness*. For programming languages *exception* handling is actually not only used for *error* handling; it is also a great tool for clarity and terseness. The "rare cases" are extracted from the main program text; they are treated in the exception handler. Thus, the exception handler, as well as the main program text, goes straight to the point; exception handling improves both clarity and terseness of programs. For formal specifications, clarity and terseness are *a fortiori* an important aim of exception handling. A formal framework only treating *error* handling is not sufficient; specification and abstraction require *exception* handling. An exception is not necessarily an error; it simply requires a special treatment which has to be clearly distinguished from the main properties.

Regarding clarity, it is necessary to separate the "exceptional properties" from the "normal behaviour." When this partition is not available, it is necessary to write complex axioms where additional predicates appear in order to restrict the scope of the axioms to normal (resp. exceptional) cases (see also Section 3). Thus, the semantics should *implicitly* restrict the scope of the axioms.

Regarding terseness, the specialized semantics for each part of the syntax (exceptional/normal properties) should be powerful enough to handle obvious general properties of exceptions. For instance, an error propagation rule by default should not have to be explicitly specified.

In addition, the following principles have been widely recognized as crucial for abstract specifications with exception handling [Gog78a][GDLE84][Bid84][Ber86][BBC86][Sch91]: each exceptional (or erroneous) case should be declared with some exception name (or error message) which provides enough information to treat it easily; various recoveries of the exceptional cases must be possible (related to their exception names); more generally, all the relevant properties of exceptional behaviours should be formally specified.

3 Algebraic specifications with exception handling

The main difficulty is that all the "simple" semantics that we can imagine lead to inconsistencies. To illustrate this fact, let us simply consider natural numbers with exception handling.

A simple idea would be to use the classical ADJ semantics [GTW78], adding a new constant *error* of sort *Nat* and the axiom: $pred(0) = error$. We have to face error propagation: what is the value of $succ(error)$? A natural idea is to add, for each operation f of the signature, axioms of the form: $f(\ldots error \ldots) = error$. But the specification also contains the axiom: $x \times 0 = 0$; thus we get $error = 0$ (with $f = \times$, via the assignment $x = error$). This inconsistency has already been shown in [GTW78], where the explicit introduction of a predicate *Ok* (checking if a value is not erroneous) is proposed in order to restrict the scope of the axioms. Unfortunately, as pointed out by the authors in [GTW78]: *"the resulting total specification (. . .) is unbelievably complicated"* because *Ok* is very difficult to specify properly.

Clearly, these difficulties result from the explicit introduction of an *erroneous value* in the signature. A simple idea could be to introduce partial functions [BW82] (e.g. $pred(0)$ being undefined). Unfortunately, specifying exceptions via partial functions is not powerful enough for a full exception handling. Exceptional cases can give rise to ulterior recoveries: even if $pred(0)$ is not defined, we can require for $succ(pred(0))$ to be recovered (e.g. on 0). This is not possible if $pred(0)$ has no semantical meaning.

Another idea is to use subsorting; the *Ok*-part (resp. the error-part) of the sort *Nat* being a subsort *OkNat* (resp. *ErrNat*) of *Nat*. Since the work of Goguen in [Gog78b], order sorted algebras have been advocated to be a solution for exception handling (see also [FGJM85][GM89]). Unfortunately, in this framework, the definition domain of each operation of the signature must be explicitly specified. Even if this approach is workable for several simple erroneous cases, such as the division by 0, it fails with respect to clarity and terseness for operations with a definition domain that is not reduced to a simple Cartesian product, such as the subtraction. (See also Example 1.)

The terseness criterion is better fulfilled when the semantics are based on a declaration of the "*Ok-codomain*" of the operations rather than on their "*Ok-domain*." The reason is simple: in general, all the operations of a data type share the same *Ok*-codomain, while each of them has its own *Ok*-domain. In [GDLE84], the signature Σ is partitioned into "safe" and "unsafe" operations. For example, 0, *succ* and + are safe operations because when they are applied to *Ok*-arguments, they always return *Ok*-results; on the contrary, *pred* is unsafe. Such a simple syntactic classification of functions is sufficient to describe the *Ok*-codomain; the *Ok*-values are those generated by the safe operations. As pointed out in [Ber86][BBC86], bounded data structures cannot be specified in this framework. For example, *succ* and + are not safe for bounded natural numbers; consequently the *Ok*-part of the sort *Nat* would be reduced to 0.

This problem is solved in [BBC86] where exception names are reflected by *labels*, a special label *Ok* being reserved for *Ok*-values. The labels are carried by values: the predicate $v \epsilon \, TooLarge$ means that the value v is labelled by *TooLarge*. This approach allows us to specify bounded data structures; unfortunately, certain recoveries lead to write inconsistent specifications:

Example 1 Let us assume that every value of the form $succ^i(0)$ $(i \geq 9)$ is labelled by *TooLarge*.[1] Let us consider an exception handler that recovers every *TooLarge*-value on $succ^8(0)$. A possible way of expressing this recovery is *"if the operation succ raises the exception TooLarge, then do not perform it."* It is formally specified as:

$$succ(n) \, \epsilon \, TooLarge \implies succ(n) = n$$

As the term $succ^9(0)$ is recovered on $succ^8(0)$, they have the same value; thus both of them are labelled by *TooLarge*. By applying the axiom with the assignment $n = succ^7(0)$ we get then the inconsistency[2] $succ^8(0) = succ^7(0)$. The point here is that, even if the *terms* $succ^9(0)$ and $succ^8(0)$ have the same *value*, they should not carry the same labels. Notice that subsorting [FGJM85] gives rise to a similar paradox because sorts are attached to values. Two terms having the same value must share the same subsorts; consequently $succ^8(0)$ and $succ^9(0)$ cannot be distinguished, and this example would lead to the same inconsistency.

This is indeed the case for all existing algebraic framework for exception handling, because exception names (if provided) are always carried by values.

Example 1 reveals the difference between "exception handling" and "error handling." The term $succ^9(0)$ is not erroneous but it is exceptional; even if the term $succ^9(0)$ is recovered on $succ^8(0)$, the exception name *TooLarge* should not be propagated to $succ^8(0)$. Exception names should be carried by terms, not by values. Roughly speaking, exception handling requires a special "typing" of *terms* (according to our terminology distinguishing exception handling from error handling). We shall call *labels* these special "types." Labels will form a third component of the signature. From this point

[1] $succ^i(0)$ stands for $succ(succ(\ldots(succ(0))..))$, the operation *succ* being applied i times.

[2] In fact, this inconsistency propagates in the same way, and all values are equal to 0.

of view, label algebras are an extension of more standard algebraic approaches with "multityping" such as order sorted algebras [Gog78b][FGJM85] (for a comparison with Equational Type Logic [MSS90], unified algebras [Mos89] or G-algebras [Meg90], see [BL91a]). In [Ber86][BBC86], only the label Ok is carried by terms, and in a restricted manner. This leads to complicated semantics and Example 1 cannot be treated.

The rest of this paper is devoted to label algebras, label specifications and their applications to exception handling.

4 Label algebras

4.1 Introduction

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^*; an object (algebra) of the category $Alg(\Sigma)$ is a heterogeneous set A, partitioned as $A = \{A_s\}_{s \in S}$, and with, for each operation name "$f : s_1 \ldots s_n \to s$" in Σ $(0 \leq n)$, a total function $f_A : A_{s_1} \times \ldots \times A_{s_n} \to A_s$; the morphisms of $Alg(\Sigma)$ (Σ-morphisms) being obviously the sort preserving, operation preserving applications.

As a consequence of our approach, (labelled) terms must also be considered as "first class citizen objects." Given an algebra A, the satisfaction relation 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). 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 implementations. How is one to deal with both terms and unreachable values ? This question is solved by the free Σ-term algebra $T_\Sigma(A)$.

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 Σ-morphisms) such that $V \subseteq T_\Sigma(V)$.*

Since V is not necessarily finite or enumerable, we can consider in particular $T_\Sigma(A)$ for every algebra A. An element of $T_\Sigma(A)$ is a Σ-term such that each leaf can contain either a constant of the signature, or a value of A.

The main technical point underlying our framework is to systematically use $T_\Sigma(A)$. Intuitively, a term reflects the "history" of a value; it is a "sequence of calculations" which results in a value. 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. The canonical evaluation morphism $eval_A : T_\Sigma(A) \to A$, 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. However, the considered assignments can be precisely restricted to certain kinds of terms/histories *before* checking equalities on values, and consequently, all 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.*

4.2 Definitions and results

For lack of space, the results are not proved here. Complete proofs can be found in [BL91b][LeG93].

Definition 1 *A* label signature *is a triplet $\Sigma L =< S, \Sigma, L >$ where $< S, \Sigma >$ is a (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: $\forall l \in L$, $\overline{h}(l_A) \subseteq l_B$. The category of all ΣL-algebras is denoted by $Alg_{Lbl}(\Sigma L)$.*

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.

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

Definition 3 *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 $(w \,\epsilon\, l)$ 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.)[3]

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 4 *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. It is of first importance to consider assignments with range in $\overline{A} = T_\Sigma(A)$ (terms) instead of A (values):

Definition 5 *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 in $\overline{A}$, means that $eval_A(u) = eval_A(v)$ [the symbol "$=$" being the set-theoretic equality in the carrier of A].*
- *$\mathcal{A}$ satisfies $(w \,\epsilon\, l)$, 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 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 "atomic satisfaction" defined above and the truth tables of the connectives.*

A label algebra satisfies a label specification SP if and only if it satisfies all its axioms. The full subcategory of $Alg_{Lbl}(\Sigma L)$ containing all the algebras satisfying SP is denoted by $Alg_{Lbl}(SP)$.

The classical results of [GTW78] can be extended to the framework of label algebras:

Theorem 1 *Let SP be a positive conditional ΣL-specification. Let $\mathcal{X}$ be a ΣL-algebra. Let R be a binary relation over X compatible with the sorts of the signature (i.e. R is a subset of $\bigcup_{s\in S} X_s \times X_s$).*

[3] Allowing existential quantifiers is not difficult [LeG93], but this extension is not required for defining exception algebras.

There is a least ΣL-algebra $\mathcal{Y}$ in $Alg_{Lbl}(SP)$ such that there exists a label morphism $h_Y : \mathcal{X} \to \mathcal{Y}$ and $(\mathcal{Y}, h_Y)$ is compatible with R (i.e. $\forall x, y \in X, x \; R \; y \Rightarrow h_Y(x) = h_Y(y)$).

Corollary 1 *If SP is positive conditional, then $Alg_{Lbl}(SP)$ has an initial object $\mathcal{T}_{SP}$.*

Corollary 2 *Let SP_1 and SP_2 be two positive conditional label specifications such that $SP_1 \subseteq SP_2$ (i.e. $S_1 \subseteq S_2$, $\Sigma_1 \subseteq \Sigma_2$, etc.). The forgetful functor $U : Alg_{Lbl}(SP_2) \to Alg_{Lbl}(SP_1)$ exists and has a left adjoint functor $F : Alg_{Lbl}(SP_1) \to Alg_{Lbl}(SP_2)$.*

As usual, the adjunction unit $I_{\mathcal{A}} : \mathcal{A} \to U(F(\mathcal{A}))$ can be used to define *hierarchical consistency* (i.e. the "no-collapse" property) and *sufficient completeness* (i.e. the "no-junk" property) for structured specifications. In addition, pushouts and colimits exist (from Theorem 1), thus parameterization can be handled without difficulty.

5 Exception algebras

The framework of exception algebras is a specialization of the one of label algebras, where the labels are used for exception handling purposes.

5.1 Label algebras and exception algebras

The particular label Ok will be distinguished to characterize the normal cases; exception names and error messages will be reflected by the other labels. This allows us to take exception names into account in the axioms; thus, an extremely wide spectrum of exception handling and error recovery cases can be specified. Intuitively, in an exception algebra $\mathcal{A}$, $t \in l_A$ with $l \neq Ok$ will mean that the calculation defined by t leads to the exception name l; and $t \in Ok_A$ will mean that the calculation defined by t is a "normal" calculation (i.e. it does not need an exceptional treatment and the calculation is successful).

As shown in Section 3, when specifying a data structure with exception handling features, the specifier has first to declare the desired Ok-part. Let us assume for example that all the terms $succ^i(0)$ with $i \leq 8$ 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 the terseness principle, labelling by Ok must be *implicitly* propagated through the axioms kept for normal cases. These axioms will be called Ok-axioms, and this implicit propagation rule will be an important component of their semantics, as described in Section 6. Since label algebras have no implicit aspects, the semantics of exception specifications must be more elaborated than the semantics of label specifications.

Another important implicit aspect is the "common future" property. Let us consider $\mathcal{A}$ reflecting the natural numbers bounded by 8, the terms $succ^i(0)$ with $i \leq 8$ being labelled by Ok. Let us assume that $succ^9(0)$ is recovered on $succ^8(0)$. Once this recovering is done, we want everything to happen as if the exception $succ^9(0)$ were never raised; this is the very meaning of the word recovery. The same succession of operations applied to $succ^8(0)$ or to $succ^9(0)$ should return the same value and raise the same exception names. If $succ^9(0)$ is labelled by $TooLarge$, then the term $t = succ^{10}(0)$ should also be labelled by $TooLarge$, since $succ^9(0) = t[succ^9(0) \leftarrow succ^8(0)]$. [4]

In a label algebra $\mathcal{A}$, $eval_A(u) = eval_A(v)$ implies that, for every term t containing u as strict subterm, 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.

[4] The term $t[u \leftarrow v]$ is the term of $\overline{A}$ obtained by replacing the occurrence of u in t by v.

5.2 Exception signature

Definition 6 *An* exception signature ΣExc *is a label signature* $< S, \Sigma, L >$ *such that* Ok *does not belong to* L*. The elements of* L *are called* exception labels *or* exception names*.*

In the sequel, $\tilde{L}$ *will denote* $L \cup \{Ok\}$*, and* $\Sigma\tilde{L}$ *will be the label signature* $< S, \Sigma, \tilde{L} >$*.*

Example 2 $NatExc =< \{Nat\}, \{succ\_, pred\_\}, \{Negative, TooLarge\} >$ is a possible exception signature for an exception specification of bounded natural numbers.

Definition 7 *Let* $\mathcal{A}$ *be a label algebra and* l *a label.* $\mathcal{A}$ *satisfies the* common future property *for* l *if and only if, for all terms* u *and* v *of* $\overline{A}$ *such that* $eval_A(u) = eval_A(v)$*, we have for all terms* t *with* u *as a strict subterm:* $t \in l_A \iff t[u \leftarrow v] \in l_A$ *.*
("strict" means that u *is a subterm of* t *distinct from* t*; in this case,* t *is called a* future *of* u*.)*

Definition 8 *An* exception algebra *over the exception signature* ΣExc *is a label algebra* $\mathcal{A}$ *over the signature* $\Sigma\tilde{L}$ *that satisfies the common future property for all exception labels* $l \in L$*.*

We have carefully excluded the term u from the set of all the futures of u by considering only strict subterms. If we accept $t = u$ as a future of u then everything happens exactly as if labelling were attached to values. We have shown in Example 1 that this is not suitable. The common future property is a weaker constraint than the labelling of values. For instance Example 1 does not lead to inconsistencies with our formalism (see Section 7.1).

Remark 1 The label Ok is not concerned with the common future property. Otherwise, if $pred(0)$ is recovered on 0 and if $succ(pred(x)) = x$ is an Ok-axiom, we would have: $succ(pred(0))$ is an Ok-term. This would lead to the inconsistency $succ(0) = 0$. Clearly, the term $succ(pred(0))$ is recovered but it must remain exceptional because an exceptional treatment has been required in its history (see also Section 6).

Example 3 According to the exception signature $NatExc$ defined above, we can consider the exception algebra $\mathcal{A} = (A, \{l_A\}_{l \in \tilde{L}})$ defined by:

$$A = \{\ldots, -2, -1, 0, 1, 2, \ldots, 8\}$$

$succ_A$ and $pred_A$ being defined as usual on integers with the restriction $succ_A(8) = 8$;

$$Negative_A = \left\{ \begin{array}{cccccc} \ldots, & \ldots, & pred(pred(0)), & pred(0), & succ(pred(0)), & \ldots, \\ \ldots, & -3, & -2, & -1, & succ(-1), & succ(succ(-1)), \\ \ldots, & \ldots, & succ(-3), & succ(-2), & succ(succ(-2)), & \ldots, \end{array} \right\}$$

$Negative_A$ contains here at the same time negative values and terms. All these terms have a negative value by classical evaluation in the set of integers or else have at least a subterm which would have a negative value by evaluation.

$$TooLarge_A = \{succ^9(0), succ(8), succ(succ(8)), \ldots\}$$

$$Ok_A = \left\{ \begin{array}{rrrr} \ldots, & succ(0), & succ(1), & \ldots, \\ 0, & 1 & 2, & 3, \\ pred(1), & pred(2), & pred(3), & \ldots, \end{array} \right\}$$

Definition 9 *Given an exception signature* ΣExc*, an* exception morphism $\mu : \mathcal{A} \to \mathcal{B}$ *is a* $\Sigma\tilde{L}$*-morphism from* $\mathcal{A}$ *to* $\mathcal{B}$*. The category of all* ΣExc*-algebras is denoted by* $Alg_{Exc}(\Sigma Exc)$*.*

Theorem 2 *Let ΣExc be an exception signature. Let $SP_{\Sigma\tilde{L}}$ be the positive conditional label specification which contains all the $\Sigma\tilde{L}$-axioms of the form:*

$$x_1 = y_1 \ \wedge \ldots \wedge \ x_n = y_n \ \wedge \ f(x_1, \ldots, x_n) \ \epsilon \ l \quad \Longrightarrow \quad f(y_1, \ldots, y_n) \ \epsilon \ l$$

where f is any (non-constant) operation of Σ, x_i and y_i are variables of sorts given by the arity of f, and l is any exception label of L.

The label specification $SP_{\Sigma\tilde{L}}$ specifies the ΣExc-algebras, i.e. $Alg_{Exc}(\Sigma Exc) = Alg_{Lbl}(SP_{\Sigma\tilde{L}})$.

Consequently, $Alg_{Exc}(\Sigma Exc)$ has an initial object, denoted $\mathcal{T}_{\Sigma Exc}$.

6 Exception specifications

Following the arguments given in Section 2, the axioms of an exception specification will be separated in two parts in order to preserve clarity and terseness.

The first part, called *GenAx*, is mainly devoted to exception handling. Its first purpose concerns labelling of terms. The axioms with a conclusion of the form $t \ \epsilon \ Ok$ (resp. $t \ \epsilon \ l$ 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.

The second part, called *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. It will both treat equalities between *Ok*-terms and carefully propagate labelling by *Ok* through these equalities (following the arguments given in Section 5.1).

Two examples of exception specifications are given in Section 7.

Definition 10 *Let ΣExc be an exception signature. A set of* generalized axioms *is a set GenAx of positive conditional label axioms with respect to the label signature $\Sigma\tilde{L}$.*

An exception algebra $\mathcal{A}$ satisfies GenAx if and only if its underlying label algebra satisfies GenAx, regarded as a set of label axioms.

Definition 11 *Let ΣExc be an exception signature. A set of* Ok-axioms *is a set OkAx of positive conditional $\Sigma\tilde{L}$-axioms with a conclusion of the form: $v = w$.*

Definition 12 *Let ΣExc 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,*[5] *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 $\Sigma\tilde{L}$-algebra satisfies the "ground" label axiom $\sigma(P)$), the two following properties hold:*

1. *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 (p may be equal to 0), then both $\sigma(v)$ and $\sigma(w)$ belong to Ok_A.*

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

$\mathcal{A}$ satisfies OkAx if and only if $\mathcal{A}$ satisfies all the Ok-axioms of OkAx.

The first property of the definition reflects a propagation of the label *Ok* (which starts from the *Ok*-terms declared in *GenAx*); a term can be labelled by *Ok* through an *Ok*-axiom only if all

[5] P may be empty.

the arguments of its heading function are already labelled by Ok. This rule allows us to carefully propagate the label Ok. Intuitively, such an innermost evaluation reflects an implicit propagation of exceptions: if t is not an Ok-term then $f(\ldots t \ldots)$ cannot be turned into an Ok-term via the Ok-axioms. (However f is not necessarily a strict function; lazyness can be specified via the generalized axioms, where $f(\ldots t \ldots)$ can be recovered.)

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.

Definitions 13 *An* exception specification *is a triplet $SPEC =< \Sigma Exc, GenAx, OkAx >$ where ΣExc is an exception signature, $GenAx$ a set of generalized axioms and $OkAx$ a set of Ok-axioms.*

A ΣExc-algebra $\mathcal{A}$ satisfies $SPEC$ if and only if it satisfies $GenAx$ and $OkAx$, as sets of generalized axioms and Ok-axioms respectively.

We denote by $Alg_{Exc}(SPEC)$ the full subcategory of $Alg_{Exc}(\Sigma Exc)$ containing all the algebras satisfying $SPEC$.

Lemma 1 *Let ΣExc be an exception signature. Let α be an Ok-axiom. There is a set of positive conditional $\Sigma\tilde{L}$-axioms, $Tr(\alpha)$, such that for every ΣExc-algebra $\mathcal{A}$, $\mathcal{A}$ satisfies the Ok-axiom α if and only if the underlying $\Sigma\tilde{L}$-algebra of $\mathcal{A}$ satisfies $Tr(\alpha)$.*

$Tr(\alpha)$ is obtained from α by adding certain preconditions reflecting Definition 12; $Tr(\alpha)$ may thus contain a great number of label axioms (related to the number of operations of $\Sigma\tilde{L}$).

Theorem 3 *Let $SPEC =< \Sigma Exc, GenAx, OkAx >$ be an exception specification. Let $Tr(SPEC)$ be the label specification defined by the label signature $\Sigma\tilde{L}$ and the set of label axioms containing: the axioms of $SP_{\Sigma\tilde{L}}$ (defined in Theorem 2), $GenAx$ and all the $Tr(\alpha)$ for $\alpha \in OkAx$ (mentioned in Lemma 1 above). We have $Alg_{Exc}(SPEC) = Alg_{Lbl}(Tr(SPEC))$. $Tr(SPEC)$ is called the* translation *of $SPEC$ into a label specification.*

$Tr(SPEC)$ only contains positive conditional axioms. Thus, from Corollary 1 we have:

Theorem 4 *Let $SPEC$ be a ΣExc-specification. $Alg_{Exc}(SPEC)$ has an initial object $\mathcal{T}_{SPEC}$.*

Remark 2 Given two exception specifications $SPEC_1$ and $SPEC_2$ such that $SPEC_1 \subseteq SPEC_2$, the forgetful functor $U : Alg_{Exc}(SPEC_2) \to Alg_{Exc}(SPEC_1)$ exists and has a left adjoint functor F.

7 Examples

Section 7.1 contains an example with "intrinsic" exceptional cases and bounds. Section 7.2 contains an example with bounds and "dynamic" exceptional cases. Thus, all the classes of exceptional cases, as classified in Section 1, are covered.

7.1 A short example

Let $NatExc =< \{Nat\}, \{succ\_, pred\_\}, \{TooLarge, Negative\} >$ be the exception signature given in Section 5.2. An exception specification of natural numbers bounded by 8 is given below:

$$
\begin{array}{ll}
\textbf{GenAx}: & succ^8(0) \mathrel{\epsilon} Ok \\
 & succ(n) \mathrel{\epsilon} Ok \implies n \mathrel{\epsilon} Ok \\
 & succ^9(0) \mathrel{\epsilon} TooLarge \\
 & pred(0) \mathrel{\epsilon} Negative \\
 & succ(n) \mathrel{\epsilon} TooLarge \implies succ(n) = n \\
\textbf{OkAx} \ : & pred(succ(n)) = n
\end{array}
$$

Where : $n \ : \ Nat$

The first two axioms of *GenAx* specify the *Ok* part of *Nat*. It is not necessary to declare *all* the *Ok*-terms (the label *Ok* will be automatically propagated to terms such as $pred(succ(0))$ via the *Ok*-axiom). It is only desirable to declare at least one term for each intended *Ok*-value. The meaning of the third and fourth generalized axioms is that the operation *succ* (resp. *pred*) raises the exception *TooLarge* (resp. *Negative*) when applied to $succ^8(0)$ (resp. 0). The last generalized axiom recovers $succ^9(0)$ on $succ^8(0)$ (as well as all its successors, from the common future property). The inconsistency described in Example 1 does not occur any more, as $succ^8(0)$ is not labelled by *TooLarge*. Then, *OkAx* only has to specify the operation *pred* in all normal cases; it is actually terse and clear.

Moreover, the instance $pred(succ^8(0)) = pred(succ^9(0)) = succ^8(0)$ is no longer an instance of the *Ok*-axiom because $succ^9(0)$, and therefore $pred(succ^9(0))$, is not required to be an *Ok*-term in our framework (even though $eval_A(succ^9(0)) = eval_A(succ^8(0))$). Thus, $pred(succ^9(0)) = succ^8(0)$ is *not* a consequence of *OkAx*. This is a good example of our restricted propagation of the label *Ok* through the *Ok*-axioms; it shows how the semantics of *Ok*-axioms reflect an implicit propagation of exceptions.

Let us note that the exception algebra described in the Section 5.2 satisfies this specification.

7.2 A more elaborated example

We give a specification of bounded arrays of natural numbers,[6] where a new array is not initialized. The specifications of natural numbers and booleans are not given in this example; it is not difficult, for instance, to complete the specification of Section 7.1 with the operations *eq* and $\leq$.

$$\begin{array}{lll}
S & : & Array \\
\Sigma & : & create\_\_ \ : \ Nat\ Nat \rightarrow Array \\
 & & store\_\_\_ \ : \ Nat\ Array\ Nat \rightarrow Array \\
 & & fetch\_\_ \ : \ Array\ Nat \rightarrow Nat \\
 & & lower\_ \ : \ Array \rightarrow Nat \\
 & & upper\_ \ : \ Array \rightarrow Nat \\
L & : & BadRange,\ OutOfRange,\ NonInitialized
\end{array}$$

$$\begin{array}{ll}
\textbf{GenAx}: & low \ \epsilon\ Ok \ \wedge up \ \epsilon\ Ok \ \wedge \ low < up = true \ \Longrightarrow \ create(low, up) \ \epsilon\ Ok \\
 & a \ \epsilon\ Ok \ \wedge \ ind \ \epsilon\ Ok \ \wedge \ x \ \epsilon\ Ok \ \wedge \ lower(a) \leq ind = true \ \wedge \ ind \leq upper(a) = true \ \Longrightarrow \\
 & \qquad\qquad\qquad\qquad\qquad\qquad store(x, a, ind) \ \epsilon\ Ok \\
 & low < up = false \ \Longrightarrow \ create(low, up) \ \epsilon\ BadRange \\
 & ind < lower(a) = true \ \Longrightarrow \ store(x, a, ind) \ \epsilon\ OutOfRange \\
 & upper(a) < ind = true \ \Longrightarrow \ store(x, a, ind) \ \epsilon\ OutOfRange \\
 & ind < lower(a) = true \ \Longrightarrow \ fetch(a, ind) \ \epsilon\ OutOfRange \\
 & upper(a) < ind = true \ \Longrightarrow \ fetch(a, ind) \ \epsilon\ OutOfRange \\
 & lower(a) \leq ind = true \ \wedge \ ind \leq upper(a) = true \ \Longrightarrow \\
 & \qquad\qquad\qquad\qquad\qquad\qquad fetch(create(low, up), ind)) \ \epsilon\ NonInitialized \\
 & eq(ind1, ind2) = false \ \wedge \ fetch(a, ind1) \ \epsilon\ NonInitialized \ \Longrightarrow \\
 & \qquad\qquad\qquad\qquad\qquad\qquad fetch(store(x, a, ind2), ind1) \ \epsilon\ NonInitialized
\end{array}$$

$$\begin{array}{ll}
\textbf{OkAx}\ : & lower(create(low, up)) = low \\
 & upper(create(low, up)) = up \\
 & lower(store(x, a, ind)) = lower(a) \\
 & upper(store(x, a, ind)) = upper(a) \\
 & store(x, store(y, a, ind), ind) = store(x, a, ind) \\
 & eq(ind1, ind2) = false \ \Longrightarrow \ store(x, store(y, a, ind1), ind2) = store(y, store(x, a, ind2), ind1) \\
 & fetch(store(x, a, ind), ind) = x
\end{array}$$

[6] For simplicity, both indexes and elements of arrays are of sort *Nat*.

Where : $low,\ up,\ ind,\ ind1,\ ind2,\ x,\ y\ :\ Nat\ ;\ \ a\ :\ Array$

The term $create(low, up)$ creates a new array of range $[low, up]$. Notice that if low or up is exceptional, then $create(low, up)$ is exceptional too (exception propagation). The operations *lower* and *upper* retrieve the acceptable range of an array. The exception name *OutOfRange* is raised when a *store* or a $fetch$ is performed outside of the acceptable range. Thus, the label *OutOfRange* intersect several sorts (*Array* and *Nat*).

8 Conclusion

We have introduced a distinction between what we call "exception handling" and "error handling." We have shown that *exception handling* requires a refined notion of the satisfaction relation for algebraic specifications. The scope of an axiom should 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 is 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 have also shown that exception names, or error messages, should be carried by terms, and that they are advantageously reflected by *labels*. 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 a label axiom is carefully delimited by labels which serve as special marks on terms.

Then, we have proposed a new algebraic framework for exception handling, based on label algebras, which is powerful enough to cope with all suitable exception handling features such as implicit propagation of exceptions, possible recoveries, declaration of exception names, etc. All the usual exceptional cases can easily be specified ("intrinsic" exceptions of an abstract data type, "dynamic" exceptional cases and bounded data structures). This approach solves some weaknesses of existing frameworks (see Section 3) and succeeds with respect to *clarity* and *terseness*, that are two crucial criteria for formal specifications with exception handling.

Although we have introduced the theory of label algebras as a general frame for exception handling purpose, 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 mentioned in Section 3 that label algebras can be shown as an extension of more standard algebraic approaches based on "multityping." Similarly to exception handling, partial functions [BW82] or observability issues [Hen89][BB91] can also be described in the same way by some well chosen forms of label specifications. However, all the specific applications of label algebras require certain *implicit* label axioms in order to preserve clarity and terseness. 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. observational specifications or exception specifications) can be specified by a well chosen set of label axioms.

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." Several other extensions, such as higher order label specifications, may be dealt with in future works.

Acknowledgements: We would like to thank Pierre Dauchy and Anne Deo-Blanchard for a careful reading of the draft version of this paper. This work has been partially supported by CNRS GRECO de Programmation and EEC Working Group COMPASS.

References

[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.)

[Ber86] Bernot G. *Une sémantique algébrique pour une spécification différenciée des exceptions et des erreurs : application à l'implémentation et aux primitives de structuration des spécifications formelles.* Thèse de troisième cycle, Université de Paris-Sud, Orsay, February 1986.

[BL91a] Bernot G., Le Gall P. *Label algebras : a systematic use of terms.* "8th International Workshop on Abstract Data Types", Dourdan, August 1991. To appear in LNCS. (also LRI Report 719, Orsay, Dec. 1991.)

[BL91b] Bernot G., Le Gall P. *Label algebras and exception handling.* Draft Version (also in Habilitation Thesis of Bernot G., University of Orsay, Paris XI, Feb. 1992.)

[Bid84] Bidoit M. *Algebraic specification of exception handling by means of declarations and equations.* Proc. 11th ICALP, Springer-Verlag LNCS 172, July 1984.

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

[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.

[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.

[Gog78a] Goguen J.A. *Abstract errors for abstract data types.* Formal Description of Programming Concepts, E.J. NEUHOLD Ed., North Holland, pp.491-522, 1978.

[Gog78b] 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.

[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.)

[Gut75] Guttag J.V. *The specification and application to programming.* Ph.D. Thesis, University of Toronto, 1975

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

[LeG93] Le Gall P. *Les algèbres étiquetées : une sémantique fondée sur une utilisation systématique des termes. Application au test de logiciel avec traitement d'exceptions.* forthcoming thesis, University of Orsay, 1993.

[LZ75] Liskov B., Zilles S. *Specification techniques for data abstractions.* IEEE Transactions on Software Engineering, Vol.SE-1 n.1, March 1975.

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

[Meg90] Mégrelis A. *Algèbre galactique - Un procédé de calcul formel, relatif aux semi-fonctions, à l'inclusion et à l'égalité.* Ph.D. Thesis, University of Nancy I, Sept. 1990.

[Mos89] Mosses P. *Unified algebras and Institutions.* Proc. of IEEE LICS'89, Fourth Annual Symposium on Logic in Computer Science, June 1989, Asilomar, California.

[MSS90] Manca V., Salibra A. and Scollo G. *Equational Type Logic.* Theoretical Computer Science, Vol.77, p.131-149, 1990.

[Sch91] Schobbens P.Y. *Clean algebraic exceptions with implicit propagation.* Proc. of AMAST-2, Second Conference of Algebraic Methodology and Software Technology, Iowa City, Iowa, USA, May 1991.