ALGEBRAIC SEMANTICS OF OBJECT TYPE SPECIFICATIONS

MARC AIGUIER and GILLES BERNOT
LaMI *(**La**boratoire de **M**athématiques et d'**I**nformatique)*
Université Evry-Val d'Essonne, Bd des Coquibus
F-91025 Evry cedex, France
E-mail: {aiguier,bernot}@lami.univ-evry.fr

1. Introduction

Object oriented programming languages are more and more popular and the "object oriented approaches" seem to be increasingly appreciated for software engineering tasks. We believe that it would be a pity if classical formal specification languages would not follow this evolution. A lot of works have already been done [11,7,8,5,2]; they often address the problem of defining models to reflect object oriented issues, or to provide a suitable logic. Our goal is to take up the challenge of defining a theory for *object oriented algebraic specifications*.

An abstract specification should describe *what* the system is supposed to do; it should not describe *how* it is supposed to do it. Consequently, the axioms of an object oriented algebraic specification should be considered as *requirements*; our goal is *not* to define a formalism allowing to explicitly solve concrete implementation problems such as concurrency (e.g. safety, liveness,...) (although the axioms can imply the satisfaction of such properties). Nevertheless, we will make use of certain connectives[a] in order to express some temporal constraints on the system evolution.
Axioms express properties about the behaviour of a system or of an object type. However, as concurrency can modify system (or object) behaviours, it should be taken into account. So, we will only specify the observable consequences of concurrency, which can be represented by non-determinism.

A major difference between object oriented algebraic specification and classical algebraic specifications is the introduction of *methods* whose semantics can change according to an implicit internal state. So, a peculiarity of our approach is to consider local states as dynamic modifiers of the method semantics. While Dauchy and Gaudel[4] only allow one global state, we allow several local states. We get advantage of having as many states as objects in the system. With respect to the semantic side, another difference of our approach with other works[4,2,10] is that states are simple elements of one algebra instead of considering one different algebra by state.

We also introduce the concept of *object type* which can be seen as an extension of the concept of specification module. Object types can be combined in order to build a system. Dependences between object types within a system allow cycles, as

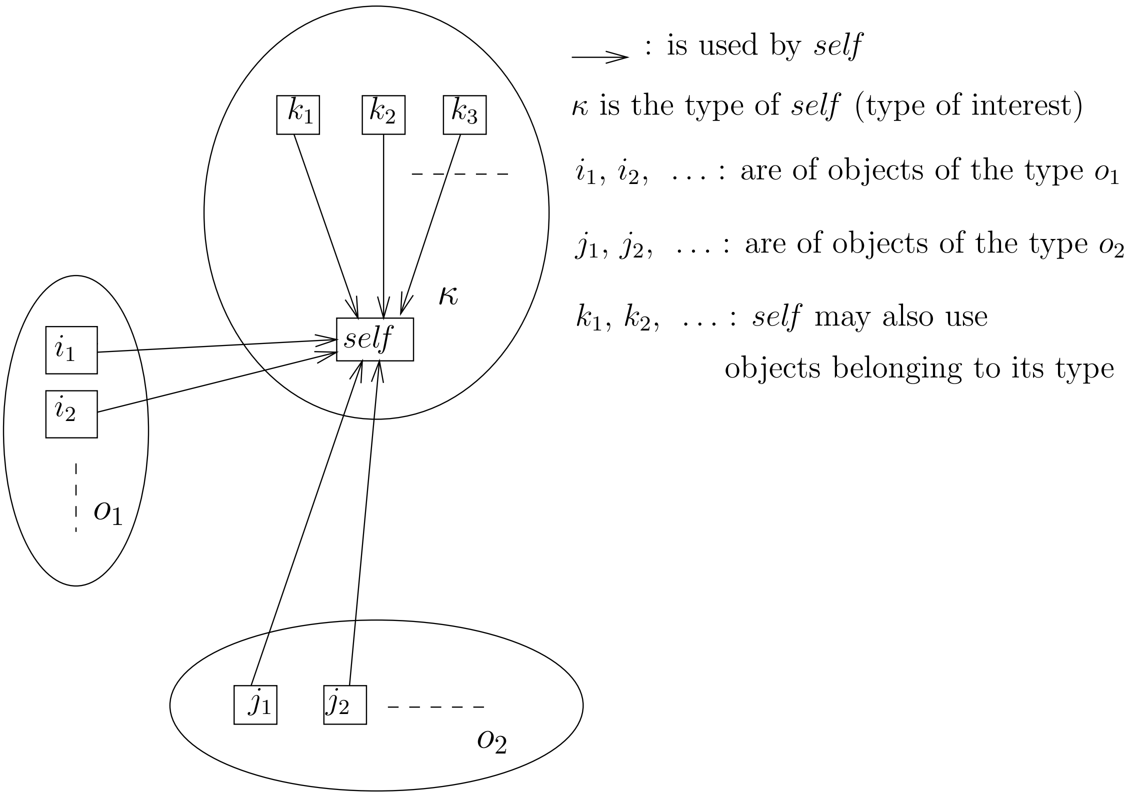
[a] **after** and **when**

opposed to the classical notion of module. In this article, we focus on the specification of object types only (for more details, see the Aiguier's thesis[1]).

The following sections contain the main definitions of object types without extended comments. A running example should help the reader to understand the definitions.

2. Object type specifications

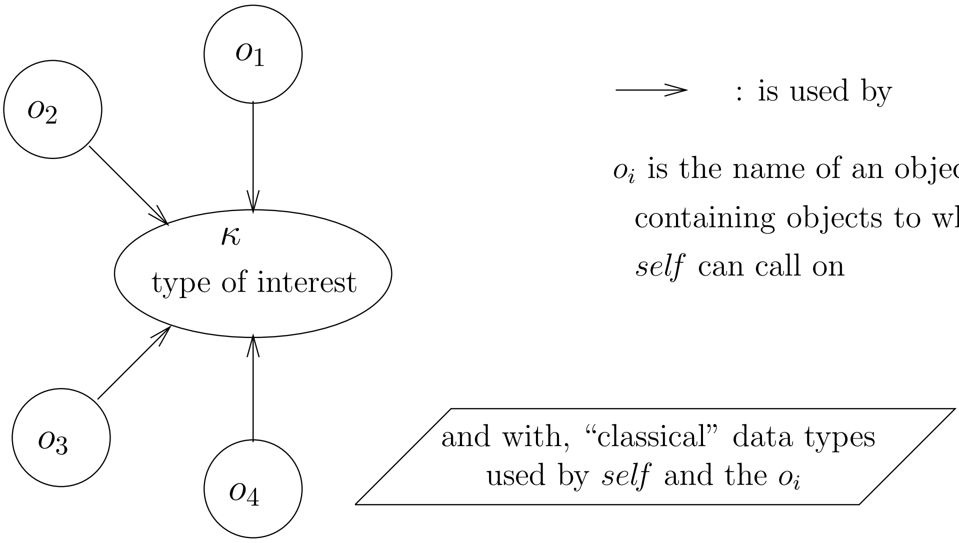
Intuitively, an object type specification will define the behaviour of an *arbitrary* object of this type, called *self*. Consequently, an object type specification will describe an "egocentric *view*", the center of which is *self*. The objects belonging to the view of *self* are those which can provide services to *self* in order to help it to define its behavior.



An object type specification is defined by a signature (Section 2.1) and a set of formulas (Section 2.2).

2.1. Signature

Intuitively, an object type signature is structured like a "star": its "center" is the object type of interest κ (the one we are specifying, i.e. the one of *self*); the "branches" represent an abstract, simplified view of the object types that provide some functionalities to *self*. The center is supposed to be fully specified (through *self*) while the branches only outline the functionalities used by the center.



An object type signature declares some sorts and operation names.

Definition 1 *An* object type signature Θ *is a tuple* $<\mathcal{S}, F, \{M^o\}_{o \in O}>$ *where:*

- $\mathcal{S} = (\kappa, O, S)$ *is a "star set" i.e.: S is a set, $O \subseteq S$ and $\kappa \in O$.*

  Every element belonging to S is called a sort*, every sort belonging to O is called an* object sort*, the sort κ is called the* sort of interest *and every sort belonging to $S \setminus O$ is called a* data sort.

- *F is a set of operation names with an arity of the form $(\alpha \to \beta)$ where $\alpha \in S^*$ and $\beta \in S$.*

  (Intuitively, F describes a set of classical operations as in ADJ[9], the semantics of which do not depend on the states of the objects).

- *for every $o \in O$, M^o is a set of operation names with an arity of the form $(\alpha \to \beta)$ where $\alpha \in S^*$ and β is either an element of S, or of the form new_o' where $o' \in O$, or empty.*

  The operations of M^o will be called methods.

Comments

- The sort κ is the sort of *self*. The sorts belonging to O are the object types for which *self* uses objects to define its behaviour. The others sorts (i.e. the sorts belonging to $S \setminus O$) are data types used by *self* or the objects of the others object types o_i.

- For any $o \in O$, o can be understood as the sort of all object identities of type o, while *new_o* is only a notation which indicates the creation of an object of type o (*new_o* is not a new sort).

- The set F contains the operations associated to data types. We do not restrict the arities to the classical data type part ($S \setminus O$). Indeed, we want to manipulate identities as classical data. For example, we may want to specify a classical data type "queue" where the elements added to the queue are identities of object.

- For every $o \in (O \setminus \{\kappa\})$, the set M^o only contains the methods that *self* can use in an object of sort o. It does not necessarily contain all the methods defined in the full specification of the type o. On the contrary, the set M^κ contains all the methods that the object type κ can give to the outside.

- Lastly, we do not distinguish, among the methods of a signature, those which modify the state of objects and those which only observe them. If we specify the dynamic stacks, we can have a method "*top&pop*" with an arity of the form "$\rightarrow$ *elem*" (the stack is implicitly contained in the state) whose behaviour is to return the first element and to remove it from the stack. Such a method observes the sate and modifies it.

Example 1 *Obviously, we specify the "Hanoï tower" game by means of two object type signatures, respectively called "Tower" and "Disk". The signature of Tower is defined by:*

Tower signature*:*

$\mathcal{S}$ *is defined by:* *sort of interest:* $\kappa = tower$
other object sorts: disk (i.e. $O = \{tower, disk\}$*)*
data sorts: int, bool (i.e. $S = \{tower, disk, int, bool\}$*)*

F *contains all operations belonging to the abstract data types int and bool.*

$$
\begin{aligned}
M^{tower} = \ & \{empty : \ \rightarrow bool, \\
& push : disk \rightarrow \ , \\
& pop : \ \rightarrow \ , \\
& top : \ \rightarrow disk \ , \\
& height : \ \rightarrow int\} \\
M^{disk} = \ & \{diameter : \ \rightarrow int\}
\end{aligned}
$$

Let us remark that Disk is another type signature where, this time, $\kappa = disk$*:*

Disk signature:

$\mathcal{S}$ *is defined by:* *sort of interest:* $\kappa = disk$
other object sorts: (none)
data sorts: int

F *contains all operations belonging to the abstract data type int.*

$M^{disk} = \{diameter : \ \rightarrow int,$
$weight : \ \rightarrow int\}$

Let us remark that *Tower* does not use the *weight* method of *Disk*. The advantage of putting in the definition of an object type signature the *methods* that *self* can use (and not only the object *sorts* that *self* can use) is to define autonomous semantics to object type specifications, without knowing the surrounding system. Moreover, only the minimal requirements about the context are specified. This allows to give "modular" specifications of systems, in which each object description is somehow "parameterized" on the context.

2.2. Terms and formulas

The first syntactical element of formulas is the notion of term. The terms are built inductively from the operations belonging to an object type signature and a set of variables.

Let $\Theta = <\mathcal{S}, F, \{M^o\}_{o \in O}>$ be an object type signature and V be a S-indexed set of variables.

- Of course, every variable is a term, and we admit all the usual terms of the form $f(t_1, \ldots, t_n)$ where $f : s_1, \ldots, s_n \rightarrow s$ belongs to F and t_i are terms of sort s_i. We also admit the conventional constant *self*, of sort κ.
- With regard to a method (i.e. an operation belonging to M^o with $o \in O$) we have to consider the identity of the object which performs it. Consequently, we introduce the key word "**in**". So, we write terms of the form: $(m(t_1, \ldots, t_n)\ \mathbf{in}\ t)$ where $m : s_1, \ldots, s_n \rightarrow s$ (or $s_1, \ldots, s_n \rightarrow \varepsilon$) belongs to M^o, t_i are terms of sort s_i and t is a term of sort o. The term t denotes the identity of an object able to perform the method m.
- Lastly, to consider the methods of the form $m : s_1, \ldots, s_n \rightarrow new\_o'$ which create an object of sort o', we use the key word "**as**" to introduce the new object identity. We consider the following terms: $(m(t_1, \ldots, t_n)\ \mathbf{as}\ x\ \mathbf{in}\ t)$ where x is a variable of sort o'. This variable will intuitively capture the identity of the new object.

Moreover, since we consider implicit states, the order in which terms are performed is significant. Thus, we introduce the notation "**;**" which defines a sequence of terms and we allow terms of the form: $(t_1;\ldots;t_n)$, called *sequence terms*, where the t_i are well-formed terms. We will consider that such a sequence term denotes the value of the last term of the sequence (i.e. t_n) after that all the t_i have been performed sequentially.

Lastly, we can be interested in the state of an object after the term $(t_1;\ldots;t_n)$ has been performed instead of looking at the result of t_n. Consequently, we introduce the notation "$\downarrow$" to write terms such as: $(t_1;\ldots;t_n)_{\downarrow_t}$ the value of which is the state of t (or rather: the state of the object whose identity is the value of t) after the sequence $(t_1;\ldots;t_n)$ has been performed. Such terms are called *projective terms*.

The canonical object *self* is provided with a fictitious identity, also denoted *self*. To facilitate the reading of formulas, we leave the identity *self* implicit. Consequently, instead of writing $(m(t_1,\ldots,t_n)$ **in** *self*$)$ and $(t_1;\ldots;t_n)_{\downarrow_{self}}$, we will respectively write $m(t_1,\ldots,t_n)$ or $(t_1;\ldots;t_n)_{\downarrow}$ for short.

From terms, we can build atoms. We have two kinds of atoms:

1. $(t_1;\ldots;t_n) = (u_1;\ldots;u_m)$ where t_n and u_m have the same sort $s \in S$ which is intuitively satisfied if the value of t_n after that the t_i have been performed, is equal to the value of u_m after that the u_i have been performed.

2. $(t_1;\ldots;t_n)_{\downarrow_t} = (u_1;\ldots;u_m)_{\downarrow_u}$ where t and u have the same sort $o \in O$ which is intuitively satisfied if the state of the object t after that the sequence $(t_1;\ldots;t_n)$ has been performed, is equal to (or is not distinguishable from) the state of u after that the sequence $(u_1;\ldots;u_m)$ has been performed.

The formulas are inductively defined from the atoms, usual connectives belonging to $\{\neg, \wedge, \vee\}$ and usual quantifiers belonging to $\{\forall, \exists\}$. Moreover, since the notion of implicit states induces an implicit notion of evolution w.r.t. time, we introduce two new connectives: **after** and **when**. We choose the following syntactic notation for these formulas:

- **after** $[(t_1;\ldots;t_n)]\ (\varphi)$
- **when** $[\varphi_1]\ (\varphi_2)$

where φ, φ_1 and φ_2 are formulas.

Intuitively, the formula **after** $[(t_1,\ldots,t_n)]\ (\varphi)$ means that the formula φ must be true immediately after the sequence term $(t_1;\ldots;t_n)$ has been performed. The formula **when** $[\varphi_1]\ (\varphi_2)$ means that at each time φ_1 *would be* true, the formula φ_2 *must be* true. For example, for the *Tower* specification, we can write the following properties:

1. $top_{\downarrow} = \_{\downarrow}$ % *top* is an observer that does not change the states,
   % "_" denotes the empty sequence term

2. **when** $[\mathit{empty} = \mathit{true}]\ (\mathit{height} = 0)$

3. **when** $[(\mathit{diameter}\ \mathbf{in}\ d) \leq (\mathit{diameter}\ \mathbf{in}\ \mathit{top}) = \mathit{true}]\ ((\mathit{push}(d)\ \mathbf{;}\ \mathit{pop})_{\downarrow} = \text{-}_{\downarrow})$

4. **when** $[(\mathit{height} = h)\ \wedge\ \mathit{empty} = \mathit{false}]\ (\mathbf{after}\ [\mathit{pop}]\ (\mathit{height} = h - 1))$

Finally, an *object type specification* SP is a couple (Θ, Ax) where Θ is an object type signature and Ax is a set of formulas built on Θ.

3. Semantics

3.1. Object type algebras

In the remainder, we will assume that Δ is an additional symbol which does not belong to the set of sorts S.

Definition 2 *Let $\mathcal{S} = (\kappa, O, S)$ be a star set. A pre-carrier defined on $\mathcal{S}$ is a tuple $(A, \underset{\leftrightarrow}{A}, \prec^A)$ where:*

- *A is a (heterogeneous) set partitioned as subsets (A_s) indexed by S.*
  (Or equivalently, A is a S-indexed family of disjoint sets A_s).

- *$\underset{\leftrightarrow}{A}$ is a (heterogeneous) set partitioned as subsets $(\underset{\leftrightarrow}{A}_o)$ indexed by $(O \amalg \{\Delta\})$.*
  (Or equivalently, $\underset{\leftrightarrow}{A}$ is a $(O \amalg \{\Delta\})$-indexed family of disjoint sets $\underset{\leftrightarrow}{A}_o$).

- *$\prec^A$ is a preorder defined on $\underset{\leftrightarrow}{A}$ and such that: $\prec^A \subseteq \coprod_{o \in O \amalg \{\Delta\}} (\underset{\leftrightarrow}{A}_o \times \underset{\leftrightarrow}{A}_o)$ i.e.:*

  $$\forall (\eta_1, \eta_2) \in (\underset{\leftrightarrow}{A} \times \underset{\leftrightarrow}{A}),\ \eta_1 \prec^A \eta_2 \Longrightarrow (\exists o \in (O \amalg \Delta),\ \eta_1 \in \underset{\leftrightarrow}{A}_o \wedge \eta_2 \in \underset{\leftrightarrow}{A}_o)$$

  (Or equivalently, $\prec^A$ is a $(O \amalg \{\Delta\})$-indexed family of preorders on the sets $\underset{\leftrightarrow}{A}_o$).

Comments

- For $s \in (S \setminus O)$, A_s should be understood as the set of all classical data of sort s, as for the classical (ADJ[9]) approach. Respectively, for $s \in O$, A_s should be understood as the set of all possible identities for objects of type o, usable by *self*.

- For $o \in O$, $\underset{\leftrightarrow}{A}_o$ should be understood as the set of all possible *apparent states* of any object of type o. The set $\underset{\leftrightarrow}{A}_o$ contains the view that *self* has of the possible object states of sort o. A state $\eta \in \underset{\leftrightarrow}{A}_o$ can also be considered as a "modifier" which modifies the semantics of any method of type o (in M^o).

- $\underset{\leftrightarrow}{A}_\Delta$ should be understood as the set of all possible *true local states* of *self* (intuitively, it simulates the attributs of *self* in an object oriented programming language; the states in $\underset{\leftrightarrow}{A}_\Delta$ are abstractions of the values of the attributs).

- The preorder $\prec^A$ takes into account all side effects of every method. It simulates the evolution from a state to another state. Intuitively, $\eta_1 \prec^A \eta_2$ means that if a given object is in the state η_1 then it may get the state η_2 later.

When a method is performed by *self*, the side effects and the used results can concern all objects belonging to its (star) view. Consequently, the behaviour of *self* is defined by its true local state (i.e. an element of $\underset{\leftrightarrow}{A}_\Delta$) and the state of every object belonging to its (star) view. Moreover, we have said that a state can be seen as a "semantic modifier" which defines the behaviour of any method. So, given a pre-carrier $(A, \underset{\leftrightarrow}{A}, \prec^A)$, a (global) state of *self* is characterized by an application γ from the set of all identities viewed by *self* (i.e. $(\coprod_{o \in O} A_o) \amalg \{self\}$) to the corresponding set of possible states (i.e. $\underset{\leftrightarrow}{A}$).

Definition 3 *Let* $(A, \underset{\leftrightarrow}{A}, \prec^A)$ *be a pre-carrier. A* global state *of self in this pre-carrier is an application* $\gamma : (\coprod_{o \in O} A_o) \amalg \{self\} \to \underset{\leftrightarrow}{A}$ *such that :*

- $\forall o \in O,\ \forall a \in A_o,\ \gamma(a) \in \underset{\leftrightarrow}{A}_o.$

- $\gamma(self) \in \underset{\leftrightarrow}{A}_\Delta.$

(Or equivalently, if we conventionally note $A_\Delta = \{self\}$*,* γ *is a* $(O \amalg \{\Delta\})$*-indexed family of applications from* A_o *to* $\underset{\leftrightarrow}{A}_o$*)*

We note $St[A]$ *the set of all global state of self in the pre-carrier* $(A, \underset{\leftrightarrow}{A}, \prec^A)$*, and we extend the preorder* $\prec$ *to* $St[A]$ *by:*

$$\forall (\gamma, \gamma') \in (St[A] \times St[A]),\ \gamma \prec \gamma' \Longleftrightarrow (\forall a \in (\coprod_{o \in O} A_o \amalg \{self\}),\ \gamma(a) \prec^A \gamma'(a))$$

From one hand, the set $St[A]$ defines all possible states of the "local system" that *self* can directly use. An element γ of $St[A]$ suffices to determine the semantics of any method performed by *self* (since it gives the state of all objects that *self* can call for and its "internal attributs" in $\underset{\leftrightarrow}{A}_\Delta$).
From another hand, $\underset{\leftrightarrow}{A}_\kappa$ is the set of all possible *apparent states* (i.e. behaviour) of *self* when used by an "external" object. According to this intuition, we ask for $\underset{\leftrightarrow}{A}_\kappa$ to be an abstraction of $St[A]$. Formally, this abstraction is represented by a surjective application abs_A defined from $St[A]$ to $\underset{\leftrightarrow}{A}_\kappa$.

Definition 4 *Let* $\mathcal{S} = (\kappa, O, S)$ *be a star set. A* $\mathcal{S}$*-carrier is a tuple* $(A, \underset{\leftrightarrow}{A}, \prec^A, abs_A)$ *where:*

- $(A, \underset{\leftrightarrow}{A}, \prec^A)$ *is a pre-carrier built on* $\mathcal{S}$.
- $abs_A : St[A] \rightarrow \underset{\leftrightarrow}{A}_\kappa$ *is a surjective application such that:*

$$\forall(\gamma, \gamma') \in St[A] \times St[A],\ \gamma \prec \gamma' \Longrightarrow abs_A(\gamma) \prec^A abs_A(\gamma')$$

Definition 5 *Let* $\Theta =< \mathcal{S}, F, \{M^o\}_{o \in O} >$ *be an object type signature. A* Θ*-algebras* $\mathcal{A}$ *is defined by:*

- *a* $\mathcal{S}$*-carrier* $(A, \underset{\leftrightarrow}{A}, \prec^A, abs_A)$.
- *for every operation* $(f : \alpha \rightarrow \beta) \in F$, *an application*[b] $f^{\mathcal{A}} : A_\alpha \rightarrow A_\beta$.
- *for every* $o \in O$, *every method* $(m : \alpha \rightarrow \beta) \in M^o$ *and every state* $\eta \in \underset{\leftrightarrow}{A}_o$*:*
  - *an application:* $\left\{ \begin{array}{ll} m_\eta^{\mathcal{A}} : A_\alpha \rightarrow A_\beta & \text{if } \beta \in S \\ \underset{\longleftrightarrow}{m}{}_\eta^{\mathcal{A}} : A_\alpha \rightarrow \underset{\leftrightarrow}{A}_{o'} & \text{if } \beta = new\_o' \end{array} \right.$
  - *an application:* $\underset{\longrightarrow}{m}{}_\eta^{\mathcal{A}} : A_\alpha \rightarrow \underset{\leftrightarrow}{A}_o$
- *for every method* $(m : \alpha \rightarrow \beta) \in M^\kappa$ *and every global state* $\gamma \in St[A]$*:*
  - *an application:* $\left\{ \begin{array}{ll} m_\gamma^{\mathcal{A}} : A_\alpha \rightarrow A_\beta & \text{if } \beta \in S \\ \underset{\longleftrightarrow}{m}{}_\gamma^{\mathcal{A}} : A_\alpha \rightarrow \underset{\leftrightarrow}{A}_{o'} & \text{if } \beta = new\_o' \end{array} \right.$
  - *an application:* $\underset{\longrightarrow}{m}{}_\gamma^{\mathcal{A}} : A_\alpha \rightarrow St[A]$.

satisfying the following conditions:

- $\forall o \in O,\ \forall (m : \alpha \rightarrow \beta) \in M^o,\ \forall a \in A_\alpha,\ \forall \eta \in \underset{\leftrightarrow}{A}_o,\quad \eta \prec^A \underset{\longrightarrow}{m}{}_\eta^{\mathcal{A}}(a)$
- $\forall (m : \alpha \rightarrow \beta) \in M^\kappa,\ \forall a \in A_\alpha,\ \forall \gamma \in St[A],\quad \gamma \prec \underset{\longrightarrow}{m}{}_\gamma^{\mathcal{A}}(a)$

Comments

- with regard to the methods $m \in M^o$:
  - $m_\eta^{\mathcal{A}}$ represents the behaviour of the method $m^{\mathcal{A}}$ with respect to the state η of an object of the type o.
  - if m does not return a value but creates a new object, the application $\underset{\longleftrightarrow}{m}{}_\eta^{\mathcal{A}}$ gives the initial state for this new object.
  - lastly, $\underset{\longrightarrow}{m}{}_\eta^{\mathcal{A}}$ represents the state evolution: when we perform $m(a)$ from the state η, we obtain the new state $\underset{\longrightarrow}{m}{}_\eta^{\mathcal{A}}(a)$.

[b] Let $A_\alpha = A_{s_1} \times ... \times A_{s_n}$ for $\alpha = s_1...s_n$.

- We have another semantics for methods belonging to M^κ when performed by *self*. The side effects induced by a method performed by *self* do not only concern the internal state of *self*. It also concerns all the objects in its view: in order to perform one of its own method, *self* can ask for any method of any object in its view.

- Let us note that the different local states in the branches of the view of *self* are not interrelated. A method m executed in an object i, with $i \neq self$, cannot make visible a state evolution in another object i'. To ignore this kind of "sharing" between different subobjects is deliberate. The point is that we are specifying *self* and no other object; consequently, every method m performed by another object i is indeed called by a method $m0$ of *self*. If m has to call m' in i', then the side effect of m' will be taken into account by $\underrightarrow{m0}^{\mathcal{A}}$ (roughly speaking, everything goes as if m' were called by $m0$).

The notion of morphisms in the category of Θ-algebras is defined in the Aiguier's thesis[1].

3.2. Evaluations of terms in a model

The evaluations of terms in a Θ-algebra $\mathcal{A}$ is defined on terms with leaves[c] belonging to A and from a global state γ belonging to $St[A]$. The result of every evaluation is either a value for a sequence term or a state for a projective term.

Every evaluation of a term $(t_1;\ldots;t_n)$ or $(t_1;\ldots;t_n)_{\downarrow_t}$ begins by sequentially evaluating the t_i. The evaluation of each t_i is a "bottom-up" evaluation. Moreover, we have to take into account that each evaluation of a term directly evaluable (called *flat term*) modifies the global state from which it is evaluated. Consequently, there is a synchronism on the global states belonging to $St[A]$.

Definition 6 *We call* flat term*, a term of the form:*

- $(m(a_1,\ldots,a_n)$ **in** $a)$
- $(m(a_1,\ldots,a_n)$ **as** a' **in** $a)$

where a_i, a and a' are elements of A.

At each evaluation step, there are choices on the flat terms to evaluate which can provide different evaluations. To cover this kind of concurrency, we adopt a non-deterministic evaluation by considering the set of all possible evaluations.

Moreover, we have two kinds of evaluations: *isolated evaluation* and *normal evaluation*. Intuitively, an isolated evaluation does not consider a possible system which

[c] More precisely, it means that we consider terms inductively defined exactly as in Section 2.2, but replacing the set of variables V by A.

might surround the "star" of *self*. Consequently, the evolutions of global states $\gamma \in St[A]$ are only the result of the operation under consideration in the term. If we rather consider a "normal" evaluation, the global state may have been moved by the surrounding system (not by *self*) before to evaluate a flat term.

Definition 7 *Let t be a flat term. Let $\gamma \in St[A]$ be a global state. $(v, \gamma') \in (A \times St[A])$ is an* isolated reduction *of (t, γ) if and only if:*

- if t *is of the form* $f(a_1, \ldots, a_n)$ then*:*
  - $v = f^{\mathcal{A}}(a_1, \ldots, a_n)$.
  - $\gamma' = \gamma$.
- if t *is of the form* $(m(a_1, \ldots, a_n)$ **in** $a)$ *(resp.* $(m(a_1, \ldots, a_n)$ **as** a' **in** $a)$*)* then*:*
  - *if* $a \neq self$*:*
    - $v = m^{\mathcal{A}}_{\gamma(a)}(a_1, \ldots, a_n)$.
    - $\gamma'(a) = \underrightarrow{m}^{\mathcal{A}}_{\gamma(a)}(a_1, \ldots, a_n)$ *(resp.* $\gamma'(a') = \underleftrightarrow{m}^{\mathcal{A}}_{\gamma(a)}(a_1, \ldots, a_n)$*).*
    - $\forall a'' \in (\coprod_{o \in O} A_o \setminus \{a\})$ *(resp.* $a \in (\coprod_{o \in O} A_o \setminus \{a, a'\})$*),* $\gamma'(a'') = \gamma(a'')$.
  - *if* $a = self$*:*
    - $v = m^{\mathcal{A}}_{\gamma}(a_1, \ldots, a_n)$.
    - $\gamma' = \underrightarrow{m}^{\mathcal{A}}_{\gamma}(a_1, \ldots, a_n)$ *(except for a' where* $\gamma'(a') = \underleftrightarrow{m}^{\mathcal{A}}_{\gamma}(a_1, \ldots, a_n)$*).*

Definition 8 *With the notations of Definition 7, (v, γ') is a* (normal) reduction *of (t, γ) if and only if there exists $\gamma'' \in St[A]$ such that:*

- $\gamma \prec \gamma''$ *with* $\gamma''(self) = \gamma(self)$.
- (v, γ') *is an isolated reduction of* (t, γ'').

The evaluation of a term t_i in an Θ-algebra $\mathcal{A}$ is defined as the set of all values resulting from any sequences of evaluations of flat subterms until we obtain a finale value.

We can extend Definition 7 and Definition 8 to every sequence term and projective term. When we evaluate a term such as $(t_1; \ldots; t_n)$ with a "normal" evaluation, the global state can change independently after the evaluation of t_i, before starting the evaluation of t_{i+1}, for every $i \in [1, n-1]$.

Definition 9 *Let $(t_1; \ldots; t_n)$ be a sequence term. Let $\gamma \in St[A]$ be a global state.*

- (v,γ') *is an isolated reduction of* $((t_1;\ldots;t_n),\gamma)$ *if and only if there exists a finite sequence* $((v_1,\gamma_1),\ldots,(v_n,\gamma_n)) \in \prod_{i=1}^{n}(A \times St[A])$ *such that:*
  - $v = v_n$ *and* $\gamma' = \gamma_n$.
  - *for every* $i \in [1,n]$, (v_i,γ_i) *is an isolated reduction of* (t_i,γ_{i-1}) *where* $\gamma_0 = \gamma$.
- (v,γ') *is a normal reduction of* $((t_1;\ldots;t_n),\gamma)$ *if and only if there exists two finite sequences* $((v_1,\gamma_1),\ldots,(v_n,\gamma_n)) \in \prod_{i=1}^{n}(A \times St[A])$ *and*

  $(\gamma'_1,\ldots,\gamma'_{n-1}) \in \prod_{i=1}^{n-1} St[A]$ *such that:*
  - $v = v_n$ *and* $\gamma' = \gamma_n$.
  - *for every* $i \in [1,n]$, $\gamma_i \prec \gamma'_i$.
  - *for every* $i \in [1,n]$, (v_i,γ_i) *is an isolated reduction of* (t_i,γ'_{i-1}).

We call an isolated evaluation *(resp.* normal evaluation*) of* $((t_1;\ldots;t_n),\gamma)$ *the first component of an isolated reduction (resp. normal reduction) of* $((t_1;\ldots;t_n),\gamma)$. *The second component is simply a global state resulting from the isolated evaluation (resp. normal evaluation) of* $((t_1;\ldots;t_n),\gamma)$.

Lastly, we define the evaluation of the projective terms.

Definition 10 *Let* $(t_1;\ldots;t_n)_{\downarrow_t}$ *be a projective term. Let* $\gamma \in St[A]$ *be a global state.* (v,γ') *is an isolated reduction (resp. normal) of* $((t_1;\ldots;t_n)_{\downarrow_t},\gamma)$ *if and only if:*

- γ' *is a global state resulting from the isolated evaluation (resp. normal evaluation) of* $((t_1;\ldots;t_n),\gamma)$.
- v *is an isolated evaluation of* (t,γ').

3.3. Satisfaction of formulas

We define the semantics of formulas, i.e. the satisfaction relation between algebras and formulas.

3.3.1. Simple satisfaction of formulas

Here, we directly interpret the projective atoms as equalities between the states of the objects under consideration. Such a satisfaction is simpler than a satisfaction defined according to an observational approach (cf. the next Section).

Definition 11 *Let $\mathcal{A}$ be a Θ-algebra. Let V be a set of variables. Let φ be a well formed formula on V and Θ. $\mathcal{A}$* satisfies *φ for an interpretation $I : V \to A$ and a global state $\gamma \in St[A]$ (i.e. $\mathcal{A} \models_{I,\gamma} \varphi$) if and only if:*

- if $\varphi = (t = u)$ *where t and u are sequence terms* then*:*

  $\mathcal{A} \models_{I,\gamma} t = u$ if and only if *for every normal evaluation v_1 of $(I(t), \gamma)$ and every normal evaluation v_2 of $(I(u), \gamma)$, we have: $v_1 = v_2$.*

- if $\varphi = (t = u)$ *where t and u are projective terms* then*:*

  $\mathcal{A} \models_{I,\gamma} t = u$ if and only if *for every normal reduction (v_1, γ_1) of $(I(t), \gamma)$ and every normal reduction (v_2, γ_2) of $(I(u), \gamma)$:*

  - $v_1 = v_2 = self \Longrightarrow \gamma_1 = \gamma_2$.
  - $v_1 = self \wedge v_2 \neq self \Longrightarrow abs_A(\gamma_1) = \gamma_2(v_2)$.
  - $v_1 \neq self \wedge v_2 = self \Longrightarrow \gamma_1(v_1) = abs_A(\gamma_2)$.
  - $v_1 \neq self \wedge v_2 \neq self \Longrightarrow \gamma_1(v_1) = \gamma_2(v_2)$.

- if $\varphi =$ **after** $[t]$ (φ_1) then $\mathcal{A} \models_{I,\gamma} \varphi$ if and only if *for every global state γ' resulting from the evaluation of (t, γ), $\mathcal{A} \models_{I,\gamma'} \varphi_1$.*

- if $\varphi =$ **when** $[\varphi_1]$ (φ_2) then $\mathcal{A} \models_{I,\gamma} \varphi$ if and only if $\mathcal{A} \models^{isol}_{I,\gamma} \varphi_1$ [d] implies $\mathcal{A} \models_{I,\gamma} \varphi_2$.

- *the satisfaction of other connectives and quantifiers is handled as usual.*

A Θ-algebra $\mathcal{A}$ satisfies a formula φ, denoted by $\mathcal{A} \models \varphi$, if and only if for every interpretation $I : V \to A$ and every global state $\gamma \in St[A]$, $\mathcal{A} \models_{I,\gamma} \varphi$.

Lastly, a Θ-algebra $\mathcal{A}$ satisfies an object type specification $SP = <\Theta, Ax>$ if and only if $\mathcal{A}$ satisfies all formulas belonging to Ax.

Comments

1. To satisfy an atom of the form $t = ...$ from a global state γ, we ask in particular for the first element of any reduction (v, γ') of (t, γ) to be the same[e]. So, we considerably reduce the non-determinism of models which satisfy the atom. This constraint has already been proposed in the PhD thesis of A. Deo[3].

2. A formula of the form: **when** $[\varphi_1]$ (φ_2) describes a condition. The formula φ_1 is the precondition, and it describe some "instantaneous condition" about the current state (as viewed by *self*). It means something like *"take a snapshot of*

[d] $\models^{isol}$ means that we do an isolated evaluation.
[e] i.e. all possible evaluations lead in fact to a unique value.

the view of self, verify φ_1 on the snapshot, and then evaluate φ_2 in the real world." Thus, φ_1 is an instant observation. It is the reason why we check the satisfaction of φ_1 via isolated evaluations, while φ_2 has to be considered with respect to normal evaluations.

3.3.2. Observational satisfaction of formulas

The simple satisfaction is not fully satisfactory because it does not reflect the encapsulation principle. More precisely, a state η should be visible only through all the semantics of the methods m_η. This principle leads to an observational equality between projective terms.

Classically, we firstly define the notion of context.

Definition 12 *Let Θ be an object type signature.*

- *A* context *defined on Θ, denoted by C, is a sequence term of a sort $s \in S$ with only one variable x, such that the key word in takes place just before all occurrences of x in the term C.*
- *Given a context C, we denote by $o \to s$ its arity where o is the sort of the variable x and s the sort of the sequence term C.*
- *We denote by C_Θ the whole set of context defined on Θ.*

We only give the observational satisfaction of projectives atoms. For the others atoms and connectives, we follow Definition 11.

Definition 13 *Let $\mathcal{A}$ be a Θ-algebra. Let $t_{\downarrow_{t'}} = u_{\downarrow_{u'}}$ be a projective atom of the sort $o \in O$. $\mathcal{A}$ satisfies $(t_{\downarrow_{t'}} = u_{\downarrow_{u'}})$ for an interpretation $I : V \to A$ and a global state $\gamma \in St[A]$ (i.e. $\mathcal{A} \models_{I,\gamma} (t_{\downarrow_{t'}} = u_{\downarrow_{u'}})$) if and only if:*

for every normal reduction (v_1, γ_1) of $(I(v), \gamma)$, for every normal reduction (u_1, γ_2) of $(I(u), \gamma)$, for every context $(C : o \to s) \in C_\Theta$, for every isolated evaluation v'_1 of $(C(v_1), \gamma_1)$ and for every isolated evaluation u'_2 of $(C(u_1), \gamma_1)$, we have $u'_1 = u'_2$.

Remark The contexts are used as "snapshots" of the state of the objects under consideration in projective terms. It is the reason why their evaluation is isolated in the previous definition.

4. From object types to systems

The definition of a *system* is not addressed in this article. Roughly speaking, a system specification is a collection of object type specifications, the semantics of which is represented by a collection of models, with some compatibility conditions. Moreover, we allow "global formulas" in a system specification in order to express properties which cannot be specified on a "local" basis. For example, the Hanoi Tower Game is a system where three constant identities of sort $Tower$ are declared (say $A,\ B,\ C\ \ :\rightarrow Tower$), and we can specify that the global number of disks is always equal to n:

$$(height\ \mathbf{in}\ A) + (height\ \mathbf{in}\ B) + (height\ \mathbf{in}\ C) = n$$

Assuming that we have specified a method in the system which initializes a pyramid of size n ($pyramid : Nat \times Tower \rightarrow$), and another one which executes a list of elementary moves from the top of a tower to another one ($exec : MoveList \rightarrow$) it is not difficult to characterize the lists of moves l that solve the problem (to move a pyramid from A to C). They are the solutions of the following formula[f]

$$\forall x : Tower, \quad (pyramid(n, A)\ \mathbf{;}\ exec(l))_{\downarrow_x} = pyramid(n, C)_{\downarrow_x}$$

We have also defined the important notion of *abstract implementation* where an object type specification is implemented by a system made of more elementary objects[1]. We have obtained the following compatibility result:
"if a system S satisfies a property φ and if I is a correct abstract implementation of one of the object type T belonging to S, then the system obtained by replacing T by I still satisfies φ".
Such a result for object oriented algebraic specifications replaces usual modularity techniques in classical software engineering.

5. Conclusion

The research briefly exposed here is only our first proposal for an object oriented approach in the framework of algebraic specification. We are aware that our definitions are still rather complex and we daily work to simplify them. Nevertheless, we believe that our framework is actually interesting, because it proves for the first time that terse object oriented specifications can be achieved within an algebraic theory (syntax and semantics).

Our approach has been first to define a syntax powerful enough to specify a collection of examples, and to content colleagues who have the habit of using object oriented programming languages. Then, we have defined the corresponding semantics and satisfaction relation in order to cope with (our and their) intuition. This approach ensures a sufficient expressive power, but it has the drawback to produce complex definitions of the semantics, leaving a great research work to simplify them.

[f] Remember that a specification describes *what* is wanted, and not *how* to obtain it... However it is not difficult in this case to establish, by induction on n, that the usual recursive solution works well...

According to this approach, the two connectives **after** and **when** have been chosen from experimental considerations, as well as the term construction. They have the advantage to cope with the common understanding of programmers, and up to now, we have been able to treat all current examples (e.g. a window manager).

Unfortunately, such an approach does not facilitate our task to provide a corresponding logic, with a finite set of inference rules. Up to now, when trying to establish that some formulas are consequences of some specification, the proofs have been performed by using standard mathematics (directly using the definitions given in this article). The practice shows that there is some analogy between our set $St[A]$ with the possible worlds and our preorder $\prec^A$ with the accessibility relation of Kripke semantics. Also, there is some analogy between our connective **after** and the modalities $[e]t$ and $[e]p$ as in the work of J. Fiadeiro and T. Maibaum[6] for example. Nevertheless, it seems to be only analogies, they are distinct in details. For example, one of the new features of our approach is the distinction between isolated evaluation and normal evaluation, which considerably refines the specification of objects (in particular for the connective **when**).

Acknowledgements
We would like to thank Pascale Le Gall and Elena Zucca for careful proof readings of this article and accurate corrections. We would also like to thank Hans-Dieter Ehrich and Grit Denker for a lot of deep and fruithful discussions in Braunschweig.

6. References

1. M. Aiguier. : *Spécifications algébriques par objets: une proposition de formalisme et ses applications à l'implantation abstraite*, Thèse de Doctorat, Universités d'Orsay–Paris-XI et Evry–Val d'Essonne, January 1995.
2. E. Astesiano, and E. Zucca. : *A semantics model for dynamic systems*, Fourth International Workshop on Foundations of Models and Languages for Data and Objects - Modelling Database Dynamics, Volske (Germany), October 1992.
3. A. Deo, : *Séquence-spécifications. Application à une sémantique pour les itérateurs*, Thèse de Doctorat, Université de Paris-Sud, Orsay, September 1994.
4. P. Dauchy, and M.C. Gaudel. : *Algebraic Specifications with implicit state*, LRI, Université de Paris-Sud, Tech. report n.887,1994.
5. H-D. Erich, M. Gogolla, and A. Sernadas. : *Objects and their Specification*, 8th Workshop on Specification of Abstract Data Types joint with the 3rd COMPASS Workshop, Dourdan Springer-Verlag LNCS 655, pp.40-65, 1991.
6. J. Fiadeiro, and T. Maibaum, : *Towards Object Calculi*, Proc. of the IS-CORE'91 Workshop, London, September 1991.
7. J. Fiadeiro, J.F Costa, A. Sernadas., and T. Maibaum, : *Objects Semantics of Temporal Logic Specification*, 8th Workshop on Specification of Abstract

Data Types joint with the 3rd COMPASS Workshop, Dourdan Springer-Verlag LNCS 655, pp.236-253, 1991.

8. J-A. Goguen, : *Sheaf Semantics for Concurrent Interacting Objects*, Mathematical Structures in Computer Science, 2(2), 159-191, 1992.
9. J.A. Goguen, J.W. Thatcher, and E.G. Wagner, : *An initial algebra approach to the specification, correctness, and implementation of abstract data types* In Current Trends in Programming Methodology, R.T. Yeh Prentice-Hall Editor, volume IV, pp.80-149, 1978. Also IBM Report RC 6487, October 1976.
10. Y. Gurevitch, : *Evolving Algebras, A tutorial Introduction*, Bull. EATCS 43, pp. 264-284, 1991.
11. A. Sernadas., C. Sernadas and J.F Costa, : *Objects Specification Logic*, Internal report, INESC, University of Lisbon, 1992.