ÉTOILE-specifications: an object-oriented way of specifying systems

Marc Aiguier[1] and Gilles Bernot[1]

August 19, 2008

Extended Abstract

1 Introduction

The ability to provide a formal specification language based on an object-oriented methodology has been, and is still, a great challenge. Several formal semantics to cope with reactive objects have been proposed [AZ92, Gur91, G92]. Several variants of logics based on Kripke semantics have been proposed and carefully studied [FM92, FCSM91] in order to model concurrent aspects of object-oriented programming. Lastly, several algebraic specification formalisms have been proposed in order to extend classical algebraic specifications to concurrent systems with hidden states [GD92, AB95, Aig95]. These approaches generally reached a sufficient expressive power to model concurrent objects. Our particularity is that we try to propose a language allowing to write short and easily readable specifications [CP94, CSS89, JSHS91, JSHS95, SR94]. ÉTOILE-specifications put the emphasis on system modularization. The syntax of ÉTOILE-specifications has been designed in collaboration with several kinds of "naive specifiers," mainly people used to object-oriented programming languages, hardware designers and people used to classical formal specifications.

It turned out to be natural to distinguish between "class" specifications and system specifications. In ÉTOILE-specifications, a "class" is called an *object-type*. The specification of an object type is a description of the typical behaviour of one representative object of the class (conventionally called *self*). Then, the specifications of a system is, roughly speaking, a set of object-type specifications with certain "gluing constraints" and some additional invariants.

When specifying an object-type, the experience showed us that it is convenient to partially specify the objects used by *self* in order to perform its own functionalities (or *methods*). Thus, an object-type specification is structured like a star, the center of which is the object of interest *self*, the branches being peripheral objects which provide some services to *self*. The word "étoile" is the French translation of "star."

2 Object type specifications

An object type specification describes "a view" (the one of *self*) like a star. Intuitively, *self* is fully specified while the branches only outline the functionalities used by the center, *as they are seen by self*. The branches play the role of parameters, they are not fully described, and their signatures are not exhaustive. In the classical algebraic modular approaches, such parameter parts are also often distinguished in a module [BEP87, EG94, NOS95]. An object type specification is defined by a signature (Section 2.1) and a set of formulas (Section 2.2).

2.1 Signature

Definition 2.1 *A* ÉTOILE-set $\mathcal{S} = (c, O, D)$ *consists of the object type of interest* c*, a set of object types* O *and a set of data types* D *such that:* $D \cap O = \emptyset$ *and* $c \notin D$.
$\mathcal{S} = (c, O, D)$ *being given, we note* $S = O \cup D$.

[1]Laboratoire de Mathématiques et d'Informatique, Université d'Evry, Cours Monseigneur Romero, 91025 Evry cedex, France. E-mail: {aiguier,bernot}@lami.univ-evry.fr, Fax: 33 1 69 47 74 72.

c is the center of the star and each $o \in O$ can be considered as one branch of the star. Roughly, the data types of D can be understood as shared types for the communication between objects in a system.

In this article, to clarify things, we will take as running example the specification of dynamic queues. The contents of a queue will be included in its internal state. The elements stored in the queue will be any arbitrary object-type (called *elem*), so that a queue will indeed store the element identities. After giving the abstract specification of the object-type *queue* in this Section 2, we specify a system in Section 3 that implements queues as chained cells. Lastly, in Section 4, we outline the proof of a property of this system.

Example 2.2 *To specify queues, the type of interest is $c = queue$. Moreover, since queues store elements, the object self (of type queue) will use some objects of type elem. No other object-type is needed at this level, thus $O = \{elem\}$. Lastly, Booleans and natural numbers will be useful, thus $D = \{bool, nat\}$.*
Consequently, $\mathcal{S} = (queue, \{elem\}, \{bool, nat\})$.

In a similar way as there is a distinction between "pure data types" in D and object-types in O, it has proved useful to distinguish "purely functional operations" from operations whose semantics can depend on local states and/or can modify them. The first ones are called *functions* whilst the second ones are called *methods*. Moreover, a method can be executed by the objects of a given object type $o \in O$. On the contrary, functions are not attached to an object-type.

Definition 2.3 *A* ÉTOILE-signature $\Theta = <\mathcal{S}, F, \mathcal{M}>$ *consists of a* ÉTOILE-*set $\mathcal{S} = (c, O, D)$, a set F of function names with an arity of the form $(\alpha \to \beta)$ where $\alpha \in S^*$ and $\beta \in S^+$, and a family $\mathcal{M} = \{M^o\}_{o \in O \cup \{\mathcal{C}\}}$ such that M^o is a set of method names with an arity of the form $(\alpha \to \beta)$ where $\alpha, \beta \in S^*$. Moreover, it is required for $M^{\mathcal{C}}$ to contain a method called new.*

F is the set of function names which denote pure functions (without side-effect). For every $o \in O$, the set M^o only contains the methods that *self* needs from objects of type o. The set $M^{\mathcal{C}}$ contains *all* the methods offered by the object-type c (by analogy with programming languages, we can see it as the *interface* of the object type c). Lastly, if a method is named "new" in M^o for $o \in O \amalg \{c\}$ then it is supposed to be devoted to create objects of type o.

Example 2.4 *For the queue example:*
F contains the whole set of usual functions associated to the data types bool and nat:
$F = \{true :\to bool, \quad false :\to bool, \quad 0 :\to nat, \quad succ : nat \to nat, \ldots\}$.
$M^{queue} = \{new :\to, \quad add : elem \to, \quad remove :\to elem, \quad is\_empty :\to bool, \quad length :\to nat\}$
Moreover, it is not necessary to know anything about elements in order to store them into a queue. The queue self never uses an element method: $M^{elem} = \emptyset$

2.2 Terms and formulas

We suppose given a ÉTOILE-signature $\Theta = <\mathcal{S}, F, \mathcal{M}>$ and a set of variables $V = \coprod_{\alpha \in S^+} V_\alpha$.

Terms are inductively defined as follows:

1. if x is a variable of type $\alpha \in S^+$ then x is a term of type α.
2. if each t_i is a term of type α_i then $(t_1, \ldots, t_n)$ (resp. $(t_1 \ ; \ \ldots \ ; \ t_n)$) is a term of type $\alpha_1 \ldots \alpha_n$.
   n may be 0; in that case the corresponding term is denoted by "_" of type ε. "**,**" and "**;**" are associative.
3. if t is a term of type α then $f(t)$ where $(f : \alpha \to \beta) \in F$ is a term of type β.
4. if t is a term of type α, $(m : \alpha \to \beta) \in M^o$, and t' a term of type $o \in O$ then $t'.m(t)$ is a term of type β.
5. if t is a term of type α then $m(t)$ where $(m : \alpha \to \beta) \in M^{\mathcal{C}}$ is a term of type β.
6. if t is a term of type $s_1 \ldots s_n$ with $s_i \in S$ then $<t>^\omega$ where $\omega = i_1 \ldots i_m \in \{1, \ldots, n\}^*$ is a term of type $s_{i_1} \ldots s_{i_m}$.

Intuitively, in 2., the comma "**,**" is the parallel evaluation of terms, whilst the semi-colon "**;**" is the sequential evaluation. The result of both $(t_1, \ldots, t_n)$ and $(t_1 \ ; \ \ldots \ ; \ t_n)$ is the tuple composed by the concatenation of the results of the t_i. In 4., t' is a term that computes an identity: the identity of the object which will perform m. In 5., $m(t)$ has to be understood as $self.m(t)$: since *self* is *the* specified object, it is convenient to leave *self* implicit. The construction in 6. allows to rearrange the tuple which results from the computation of a term.

From terms, we can build atoms. There are two kinds of equational atoms:

1. $t = u$ where t and u are terms of the same type.
2. $< t >_{t'} = < u >_{u'}$ where t and u are terms of the same type, and, t' and u' are terms of the same type belonging to O^*.

Moreover, in order to specify deadlocks and creation/deletion of objects, we introduce two supplementary predicates; we have the following atoms:

- **wait** (t) where t is a term.
- **alive** (t) where t is a term of a type belonging to O, as well as **alive** $(self)$.

Formulas are inductively defined from the atoms above, usual connectives and usual quantifiers. Moreover, in order to take into account an implicit notion of time, we introduce: **after** $[t]$ (φ) where t is a term and φ is a formula.

As usual, a ÉTOILE-specification SP consists of a ÉTOILE-signature Θ and a set of formulas ("axioms").

Example 2.5 *We specify the behaviour of dynamic queue by the following axioms:*

1. **after** $[new]$ $(is\_empty = true)$
2. **after** $[add(e)]$ $(is\_empty = false)$
3. $is\_empty = true \implies$ **wait** $(remove)$
4. $is\_empty = true \implies add(e)$ **;** $remove = e$
5. $is\_empty = false \implies add(e)$ **;** $remove = remove$ **;** $add(e)$
6. **after** $[new]$ $(length = 0)$
7. $length = n \implies$ **after** $[add(e)]$ $(length = succ(n))$

2.3 Semantic hints

2.3.1 Models

For each object type $o \in O$, we associate a set A_o which should be understood as the set of all possible object identities usable by *self* and a set $\underset{\leftrightarrow}{A_o}$ of *possible states* of any object of type o. The set $\underset{\leftrightarrow}{A_o}$ contains the view that *self* has of the possible behaviours of objects of type o in its environment. For the object type of interest c, we associate the set $\underset{\leftrightarrow}{A_C}$ which should be understood as the set of all possible *true local states* of *self* (by analogy with object oriented programming languages, it simulates the attributes of *self*; the states in $\underset{\leftrightarrow}{A_C}$ are abstractions of the value vectors of the *self* attributes). Formally:

Definition 2.6 *Given a* ÉTOILE*-signature* $\Theta = < \mathcal{S}, F, \mathcal{M} >$*, a* ÉTOILE*-model* A *consists of:*

- *a triplet* $(A, \underset{\leftrightarrow}{A}, \mathcal{G}_A)$ *where* A *is a* S*-indexed family,* $\underset{\leftrightarrow}{A}$ *is a* $O \amalg \{c\}$*-indexed family and* $\mathcal{G}_A$ *is a set of partial functions* $\gamma : \coprod_{o \in O} A_o \amalg \{self\} \rightarrow \underset{\leftrightarrow}{A}$ *such that: if* $\gamma(a)$ *is defined then* $\gamma(a) \in \underset{\leftrightarrow}{A_o}$ *for every* $a \in A_o$*, and if* $\gamma(self)$ *is defined then* $\gamma(self) \in \underset{\leftrightarrow}{A_C}$*.*

- *Given* $\alpha = s_1 \dots s_n \in S^*$, *we note* $A_\alpha = \prod_{i=1}^{n} A_{s_i}$ *and we let* $A_\varepsilon = \{\mathbb{1}\}$, *where* $\mathbb{1}$ *is neutral in a tuple, i.e.,* $(a_1, \dots, \mathbb{1}, \dots, a_n) = (a_1, \dots, a_n)$.
- *A total function* $f^{\mathcal{A}} : A_\alpha \to A_\beta$ *for every* $(f : \alpha \to \beta) \in F$.
- *A partial function* $m_\eta^{\mathcal{A}} : A_\alpha \to A_\beta \times \underset{\leftrightarrow}{A_o}$ *(resp.* $m_\gamma^{\mathcal{A}} : A_\alpha \to A_\beta \times \mathcal{G}_A$*) for every* $(m : \alpha \to \beta) \in M^o$ *(resp.* $(m : \alpha \to \beta) \in M^{\mathcal{C}}$*) and every* $\eta \in \underset{\leftrightarrow}{A_o}$ *(resp.* $\gamma \in \mathcal{G}_A$*)*
- *For the particular case of the methods named new,* $new^{\mathcal{A}} : A_\alpha \to A_\beta$ *does not depend on* η *(resp.* γ*).*

Intuitively, given an object identity $a \in A_o$, $\gamma(a)$ is the state of a. If $\gamma(a)$ is undefined, then a is not alive. If a is alive, and $\eta = \gamma(a)$, then the semantics of a method m performed by a is given by $m_\eta^{\mathcal{A}}$. Let $(r, \eta') = m_\eta^{\mathcal{A}}(t)$, $r \in A_\beta$ denotes the result of $a.m(t)$ and η' denotes the state of a after m has been performed. For the particular case of $a.new(t)$, it means that a has been created and, r and η' only depends on t ($\gamma(a)$ being possibly undefined, $\eta = \gamma(a)$, thus $new_\eta^{\mathcal{A}}$, would have no sense).
When a method is performed by *self* ($m \in M^{\mathcal{C}}$), it can call other methods in other objects. Consequently, the semantics of m depend on, and can modify, the state of any object in the star view of *self*. Thus, we consider $m_\gamma^{\mathcal{A}}$ instead of $m_\eta^{\mathcal{A}}$.

2.3.2 Term evaluation

Let $\mathcal{A}$ be a Θ-model. Roughly, a term can be seen as something that takes an initial state γ as input, and returns a tuple r as value, as well as a resulting state γ'. The evaluation of terms follows a "bottom-up" strategy. Things are unfortunately rather complex because concurrency can lead to deadlocks or non-deterministic results. For example, to evaluate $t_0.m(t_1, t_2)$, it is possible that, after the evaluation of t_0, t_1 and t_2, $\gamma(t_0)$ is undefined; then, the evaluation leads to a deadlock. Moreover, because t_1 and t_2 can have side-effects on the state γ, it is also possible that evaluating t_1 after t_2 leads to a different semantics of m than evaluating t_2 after t_1 ... and so on. Thus, the evaluation of a term t leads indeed to a set $eval_\gamma(t)$ of pairs (r, γ'). There is a deadlock if $eval_\gamma(t)$ is empty, and we say that t is *waiting*. Lastly, terms of the form $x.new(t)$ lead to a special treatment of substitutions, since they define the identity assigned to x.

2.3.3 The satisfaction relation

As usual, the satisfaction of formulas is inductively defined from the satisfaction of atomic formulas, the truth tables of usual connectives and the semantics associated to the new operator **after**.
Intuitively, an equational atom of the form $t = u$ is satisfied if for any initial state γ under consideration, the sets $eval_\gamma(t)$ and $eval_\gamma(u)$ are equal and reduced to a singleton. In a similar way, an equational atom of the form $< t >_{t'} = < u >_{u'}$ is satisfied if for any γ_1 resulting from the evaluation of t, for any γ_2 resulting from the evaluation of u, for any r_1 resulting from the evaluation of t' and for any r_2 resulting from any evaluation of u', we have: $\gamma_1(r_1) = \gamma_2(r_2)$. An atom of the form **wait** (t) is satisfied if $eval_\gamma(t) = \emptyset$. An atom of the form **alive** (t) is satisfied if for any value r resulting from the evaluation of t, $\gamma(r)$ is defined.
The satisfaction of classical connectives and quantifiers is handled as usual.
Lastly, a formula of the form **after** $[t]$ (φ) is satisfied if for any γ resulting from the evaluation of t, the formula φ is true.

3 System

3.1 Intuition

A system is obtained by gluing together several object-types. Remember that object types are stars whose branches are partial views of other object types. Thus a system will be given by a set of actualizations; each branch of a star ($\approx$ the virtual parameter) has to be actualized by the center of another star ($\approx$ the actual parameter) of the system.

3.2 Syntax

Definition 3.1 *A* system signature $\widetilde{\Theta}$ *is a set* $\{\Theta_1, \ldots, \Theta_n\}$ *of* ÉTOILE*-signatures such that the following conditions hold for every* $i, j \in [1, n]$*:*

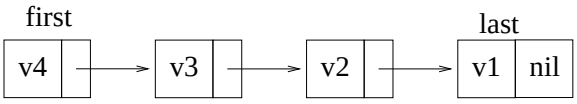
1. if *we note* $S_{i,j} = (S_{\Theta_i} \cap S_{\Theta_j})$ then*:* $\forall \alpha \in S^*_{i,j},\ \forall \beta \in S^+_{i,j},\ \forall (f : \alpha \to \beta),\ (f \in F_{\Theta_i} \iff f \in F_{\Theta_j})$
2. $c_j \in O_{\Theta_i} \Longrightarrow M^{\mathcal{C}_j}_{\Theta_i} \subseteq M^{\mathcal{C}_j}_{\Theta_j}$.

The first condition ensures the consistency between shared data types. The second condition says that the methods of the virtual parameter always belong to the methods of the actual parameter.

Terms and formulas to specify the behaviour of a system are defined from the union of the signatures Θ_i. We follow the same induction than Section 2.2 except that we do not consider terms of the form $m(t)$ because there is no reason to privilege a *self* object in the system.

A system specification $\widetilde{SP}$ is defined by a set $\{SP_1, \ldots, SP_n\}$ of ÉTOILE-specifications and a set of invariants, which are formulas on the underlying system signature $\widetilde{\Theta}$.

Example 3.2 *Let us specify a system that implements queue by chained cells.*



The system is composed by 3 object types: elem, cell and implemented queue.

1. **Cell specification:**

   $\mathcal{S} = (cell, \{elem, cell\}, \emptyset)$

   We consider a constant identity of type cell, called nil, which will denote a never alive cell. Since the name nil does not depend on any state, nor modify any state, it belongs to F*.*

   $F = \{nil :\to cell\}$

   $M^{cell} = \{new : elem \times cell \to\ ,\ \ value :\to elem\ ,\ \ next :\to cell\ ,\ \ setvalue : elem \to,\ \ setnext : cell \to\}$

   $M^{elem} = \emptyset$

   Notice that this is an example where $c \in O$ *because self uses other objects of type cell (cf. next and setnext methods).*

   axioms:

   (a) **after** $[new(e, c)]$ $(value = e \ \ \wedge \ \ next = c)$

   (b) **after** $[setvalue(v)]$ $(value = v)$

   (c) **after** $[setnext(c)]$ $(next = c)$

2. **Implemented queue specification:**

   The signature is similar to the one of Example 2.4 except that some additional types and operations are needed to manage the concrete representation of a queue by chained cells.

   $F = \{nil :\to cell,\ \ true :\to bool,\ \ false :\to bool,\ \ 0 :\to nat,\ \ succ : nat \times nat \to nat, \ldots\}$

   $M^{queue} = \{new :\to,\ \ add : elem \to,\ \ remove :\to elem,\ \ is\_empty :\to bool,\ \ length :\to nat,$
   $first :\to cell,\ \ last :\to cell,\ \ setfirst : cell \to,\ \ setlast : cell \to,\ \ setlength : nat \to\}$

   $M^{cell} = \{new : elem \times cell \to,\ \ value :\to elem,\ \ next :\to cell,\ \ setnext : cell \to\}$

   $M^{elem} = \emptyset$

   Notice that setvalue is useless in the queue specification.

   axioms:

(a) **after** $[setfirst(x)]$ $(first = x)$

(b) **after** $[setlast(x)]$ $(last = x)$

(c) **after** $[setlength(n)]$ $(length = n)$

(d) $new \quad = \quad setfirst(nil)$; $setlength(0)$

(e) **alive** $(first) \implies add(e) = x.new(e, nil)$; $last.setnext(x)$; $setlast(x)$; $setlength(succ(length))$

(f) ¬**alive** $(first) \implies add(e) = x.new(e, nil)$; $setlast(x)$; $setfirst(x)$; $setlength(succ(length))$

(g) $remove \quad = \quad first.value$; $setfirst(first.next)$; $setlength(length - 1)$

(h) $first = nil \iff is\_empty = true$

3. **Invariants:**

(a) ¬**alive** (nil)

This axiom is given here instead of within the specification of cell because it does not concern the cell representative object self. It is an invariant observed by all objects of the system.

3.3 Semantic hints

3.3.1 Models

For every $i, j \in [1, n]$, if $c_j \in O_i$ then the object *self* of type c_i has only an abstract view of the object behaviour of type c_j. This abstract view is represented by a surjective application $abs^{\widetilde{\mathcal{A}}}_{j,i}$ called abstraction. Formally:

Definition 3.3 *Let $\widetilde{\Theta} = \{\Theta_1, \ldots, \Theta_n\}$ be an exhaustive system signature. A $\widetilde{\Theta}$-*model *$\widetilde{\mathcal{A}}$ is defined by a set $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ where $\mathcal{A}_i$ is a Θ_i-model for $i \in [1, n]$ and satisfying the following conditions for every $i, j \in [1, n]$:*

- *for every $s \in (S_i \cap S_j)$, $(A_i)_s = (A_j)_s$.*
- *for every $(f : \alpha \to \beta) \in F_i \cap F_j$, $f^{\mathcal{A}_i} = f^{\mathcal{A}_j}$.*

and with for every $i, j \in [1, n]$ such that $c_j \in O_i$, a surjective application $abs^{\widetilde{\mathcal{A}}}_{j,i} : \mathcal{G}_{A_j} \to \underset{\leftrightarrow}{(A_i)}_{\mathcal{C}_j}$ such that for every $(m : \alpha \to \beta) \in M_i^{\mathcal{C}_j}$, every $a \in (A_j)_\alpha$ and every $\gamma \in \mathcal{G}_{A_j}$ if $\eta = abs^{\widetilde{\mathcal{A}}}_{j,i}(\gamma)$ then:

1. *$m^{\mathcal{A}_j}_\gamma(a)$ is defined $\iff$ $m^{\mathcal{A}_j}_\eta(a)$ is defined*
2. *$(m^{\mathcal{A}_i}_\eta(a)$ is defined $\implies m^{\mathcal{A}_i}_\eta(a) = (p_1(m^{\mathcal{A}_j}_\gamma(a)), abs^{\widetilde{\mathcal{A}}}_{j,i}(p_2(m^{\mathcal{A}_j}_\gamma(a))))$*
   where $p_1 : (A_j)_\beta \times \mathcal{G}_{A_j} \to (A_j)_\beta$ and $p_2 : (A_j)_\beta \times \mathcal{G}_{A_j} \to \mathcal{G}_{A_j}$ are the projections.

3.3.2 Term evaluation and the satisfaction condition

Naturally, a state of the system under consideration will be defined as the collection of the states of each object of the system. We associate to every object of type c_i, a state belonging to $\mathcal{G}_{A_i}$. Some instantiation constraints have to be defined in order to only consider *feasible global states*. Formally:

Definition 3.4 *Let $\widetilde{\Theta} = \{\Theta_1, \ldots, \Theta_n\}$ be an exhaustive system signature. Let $\widetilde{\mathcal{A}}$ be a $\widetilde{\Theta}$-model. We note $\mathcal{G}_{\widetilde{A}}$ the set of partial functions $\widetilde{\gamma} : \coprod_{o \in \widetilde{O}} \widetilde{A}_o \to \coprod_{i \in [1,n]} \mathcal{G}_{A_i}$ such that for every $i \in [1, n]$ and for every $a \in \widetilde{A}_{\mathcal{C}_i}$ the following conditions are verified:*

$$\widetilde{\gamma}(a) \in \mathcal{G}_{A_i}$$

$$\forall c_j \in O_i,\ \forall b \in \widetilde{A}_o : \begin{array}{ll} \widetilde{\gamma}(a)(b) \text{ is undefined} & \text{if } \widetilde{\gamma}(b)(self) \text{ is undefined} \\ \widetilde{\gamma}(a)(b) = abs^{\mathcal{A}}_{j,i}(\widetilde{\gamma}(b)) & \text{otherwise} \end{array}$$

As in Section 2.3.2, evaluations of terms follow a "bottom-up" strategy by reduction of terms. So, the evaluation of a term t from a state $\widetilde{\gamma}$ in a model $\widetilde{\mathcal{A}}$ leads also to a set $eval_{\widetilde{\gamma}}(t)$ of pairs $(r, \widetilde{\gamma'})$.

Lastly, the satisfaction relation follows the same rules than in Section 2.3.3.

4 Proving properties

Let us consider the system specification described in Example 3.2, and let us prove the following formula φ:

$$is\_empty = true \implies \textbf{wait}\ (remove)$$

Sketch of the proof: The inference rules we use will be introduced here on a "call-by-need" basis, just before their first use. Moreover, we will ignore the use of any rule about classical connectives ($\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$).

From the two following axioms of the implemented buffer specification:

- $first = nil \iff is\_empty = true$
- $remove = first.value\ ;\ setfirst(first.next)\ ;\ setlength(length - 1)$

and from the following rules: $\frac{t=t'}{<t>^{\omega}=<t'>^{\omega}}$ and $\frac{\textbf{wait}\ (t) \ \wedge \ <t>^{\varepsilon}=<t'>^{\varepsilon}}{\textbf{wait}\ (t')}$ the formula φ is equivalent to the following formula:

$$first = nil \implies \textbf{wait}\ (first.value\ ;\ setfirst(first.next)\ ;\ setlength(length - 1))$$

From the system specification, we have:

- $\neg\textbf{alive}\ (nil)$

Thus from the three following inference rules:

$$\frac{}{\_=\_} \quad \text{and} \quad \frac{t=t' \quad u=u}{<t>_u=<t'>_u} \quad \text{and} \quad \frac{<t>\_=<t'>\_}{\textbf{alive}\ (t) \Longrightarrow \textbf{alive}\ (t')}$$

we can write:

- $first = nil \implies \neg\textbf{alive}\ (first)$

Thus from the rule: $\frac{\neg\textbf{alive}\ (t)}{\textbf{wait}\ (t.m(t'))}$ we can write:

- $first = nil \implies \textbf{wait}\ (first.value)$

And we conclude from the rule: $\frac{\textbf{wait}\ (t_i)}{\textbf{wait}\ (t_1\ ;\ \ldots\ ;\ t_i\ ;\ \ldots\ ;\ t_n)}$

- $first = nil \implies \textbf{wait}\ (first.value\ ;\ setfirst(first.next)\ ;\ setlength(length - 1))$

This ends the proof. Unfortunately, to fully prove the correctness of our implementation of queue by cells, the proof of formulas involving equality between queue states are more complex. We need observability technics, similar to context induction [Hen91].

References

[Aig95] M. Aiguier, : "*Spécifications algébriques par objets : une proposition de formalisme et ses applications à l'implantation abstraite*," PhD thesis, University of Paris-Sud, Orsay, France, Junuary 1995.

[AB95] M. Aiguier, and G. Bernot, : "*Algebraic semantics of object type specifications*," In Information Systems Correctness and Reusability, Selected papers from the IS-CORE workshop, September 1994, World Scientific Publishing, R.J.Wieringa R.B.Feenstra eds (Germany), 1995.

[AZ92] 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.

[BEP87] E.K. Blum, and H. Ehrig, and F. Parisi-Presicce : "*Algebraic specification of modules and their basic interconnections*," Journal of Computer Systems Science, Vol.34, p.293-339, 1987.

[CP94] P. Carmo, and P. Penedo : "*GNOME compiler*," Research report, Instituto Superior Técnico, Secçao de Ciência da Computaçao, Departamento de Matematica, 1096 Lisboa, Portugal, 1994.

[CSS89] J.-F. Costa, and A. Sernadas, and C. Sernadas : "*OBLOG user manual*," Technical report, INESC, Rua Alves Redol 9, 1000 Lisboa, Portugal, 1989.

[EG94] H. Ehrig, and M. Grosse-Rhode : "*Functorial theory of parameterized specifications in a general specification framework*," Theoretical Computer Science (TCS), Vol.135, p.221-266, Elsevier Science Pub. B.V. (North-Holland), 1994.

[FCSM91] 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.

[FM92] J. Fiadeiro, and T. Maibaum, : "*Temporal Theories as Modularization Units for Concurrent System Specification*," Formal aspects of Computing 4(3), pp. 239-272, 1992.

[G92] J.A. Goguen, : "*Sheaf Semantics for Concurrent Interacting Objects*," Mathematical Structures in Computer Science, 2(2), pp.159-191, 1992.

[GD92] J.A. Goguen, and , R. Diaconescu, : "*Towards an Algebraic Semantics for the Object Paradigm*," Recent Trends in Data Type Specification, Selected Papers of the 5th Workshop on Specifications of Abstract Data Types joint with 4th COMPASS Workshop, Caldes de Malavella, Spain, pp.1-29, October 1992.

[Gur91] Y. Gurevitch, : "*Evolving Algebras, A tutorial Introduction*," Bull. EATCS 43, pp. 264-284, 1991.

[Hen91] R. Hennicker, : "*Context induction: A proof principle for behavioural abstractions and algebraic implementations*," Formal aspects of Computing, Vol.3, No.4, p.326-345, 1991.

[JSHS91] R. Jungclaus, and G. Saake, and T. Hartmann, and C. Sernadas, : "*Object-Oriented specification of information systems: The TROLL language*," Technical University of Brauschweig, 1991.

[JSHS95] R. Jungclaus, and G. Saake, and T. Hartmann, and C. Sernadas, : "*TROLL - a language for object-oriented specification of information systems*," ACM Transactions on Information Systems, 1995.

[NOS95] M. Navarro, and F. Orejas, and A. Sanchez, : "*On the correctness of modular systems*," Theoretical Computer Science (TCS), Vol.140, p.139-177, Elsevier Science Pub. B.V. (North-Holland), 1992.

[SR94] A. Sernadas, and J. Ramos, : "*GNOME: Sintaxe, semantica e calculo*," Research report, Instituto Superior Técnico, Secçao de Ciência da Computaçao, Departamento de Matematica, 1096 Lisboa, Portugal, 1994.