Object Certification*

Amílcar Sernadas†
Cristina Sernadas‡

July 20, 1994

Abstract

A brief overview is made of the use of temporal logic formalisms for specifying and verifying concurrent systems in general and information systems in particular. The requirements imposed by object-orientation on such formalisms are examined. A logic is proposed fulfilling those requirements (except concerning non-monotonic features), allowing the uniform treatment of both local and global properties of systems with concurrent, interacting components organized in classes, and supporting specialization. A semantics and a calculus (following an axiomatic, Hilbert style) are presented in detail. The calculus includes rules for the sound inheritance and reflection of theorems between classes. Practical aspects of the usage of such a logic for both specification and verification are considered. To this end a set of metatheorems is provided for expediting the proof of invariants. Finally, the need and availability of automatic theorem proving for systems querying is briefly discussed.

1 Introduction

Object-oriented programming has become increasingly popular thanks to the expected advantages in software development and maintenance productivity. Such advantages are presented as corollaries of the intensive use of the data encapsulation and reuse facilities made available by the object-oriented development languages and systems. Object-orientation seems also to pay-off at the level of software specification and design. Therefore, one should ask what kind of object-oriented specification language should be adopted.

We defend a *formal approach* to specification since the price of lower flexibility at the early stages is well counterbalanced by the advantages of a rigourous semantics and correctness verifiability. The latter imposes more than a formal specification language: a deductive system is needed also, making it a specification/verification logic. Indeed, object-orientation promises a lot at the level

*This work was partly supported by CEC under ESPRIT-III BRA WG 6071 IS-CORE (Information Systems - COrrectness and REusability) and BRA WG 6112 COMPASS (COMPrehensive Algebraic approach to System Specification and development) and by ESDI under research contract OBLOG (OBject LOGic).

†IST, Department of Mathematics, Lisbon, Portugal, E-mail:acs@inesc.pt
‡IST, Department of Mathematics, Lisbon, Portugal, E-mail:css@inesc.pt

of the deductive system: it should be possible to carry out "local" proofs and later on assess if and how the resulting theorems carry over to the "whole".

More specifically, we defend a *temporal logic approach* to object specification, in the sequel of the intensive work triggered by [Pnu77] in the general direction of using temporal logic for concurrent system specification. Substantial effort was dedicated to the compositionality of temporal specifications [HO83, BKP84, BKP85, BKP86, NDOG86]. It is also worhtwhile to look at the effort towards the specification and verification of liveness properties [Man82, MCS82, Jon87, MP91]. The recent text book [MP92] is a quite useful general reference. The interested reader should also consult [Hai82]. For an overview see [Eme90]. Closer to the application area of information systems, some effort triggered by [Ser80] has been dedicated to establishing appropriate temporal logic frameworks for the specification and verification of information systems [FS88, CS88, CS91].

More recently several attempts have been made towards the development of a suitable temporal logic around the concepts of *object-orientation*, mainly within the scope of the Esprit BRA IS-CORE [SFSE89a, SSC92, FSMS92, FM92, FM94]. Some interesting results have been obtained concerning compositionality [FCSM93, SS93] within an institutional setting. Indeed, it is clear that one should try to find a suitable institution [GB84, GB92] in order to establish the envisaged degree of compositionality of specifications and theories.

Herein we present a revised and extended version of the logic of object classes and object aspects presented in [SSC92]. The proposed logic is an extension of a rather standard linear temporal, many-sorted, first-order predicate logic with equality. The extensions where designed to be as simple as possible while fulfilling the requirements imposed by object-orientation. The logic allows the uniform treatment of both local and global properties of systems with concurrent, interacting components organized in classes, and supports specialization. A semantics and a calculus (following an axiomatic, Hilbert style) are presented in detail. The calculus includes rules for the inheritance and reflection of theorems between classes. Practical aspects of the usage of such a logic for both specification and verification are considered. To this end a set of metatheorems is provided for expediting the proof of invariants. Finally, the need and availability of automatic theorem proving for systems querying is briefly discussed.

We assume that the reader is conversant with the field of temporal logic specification (for instance at the level of [MP92]). We also use a little bit of category theory (the reader may find all relevant concepts in the introductory chapters of any textbook on the matter, eg [AHS90]): the more exotic notion of freely generated cartesian category is described in [SSC92]. We also assume that the reader is familiar with the use of many-sorted formalisms for specification purposes, as in the algebraic abstract data type setting (see [EM85]).

Before entering the presentation of the proposed logic of object classes, Section 2 discusses the underlying object framework (following [ES91]) and outlines the main requirements for an object-oriented logic. The logic itself is established in Section 3: language, semantics and deductive system. Section 4 provides some additional techniques for *practical* use of the logic in object system specification and verification with emphasis on invariants.

2 Requirements of object-orientation

At first sight, an object is an entity with an internal state (reflected in the values of its slots) that may interact with other objects. While so interacting an object displays some behaviour. That is, depending on its internal state, it is not always ready to provide the same services to the others and/or to ask the same services of others. The most basic unit of interaction is the event: for instance, the event open-door may be shared by a person and a car. Therefore, the behaviour of an object may be seen as corresponding to a state-dependent menu of enabled actions (the set of actions that the object is "ready" to execute).

At this naive level of analysis, already adopted in [SFSE89b, SE91, FSMS92], the specification of an object should include the list of its slots (with the indication of their result sorts), the list of its actions (with the indication of their parameter sorts if any) and the constraints on its behaviour. The life of an object would correspond to a trajectory of the events that happened since its birth.

As an illustration consider books as objects. Among their slots we may find: code (taking values in the set of natural numbers), registered-in (taking values in the set of books), and available (taking boolean values). For instance, available=true is supposed to mean that a book is available for borrowing. Among the actions of books we may consider: is-taken (with a parameter taking values in the set of users) and is-returned (with a similar parameter). For instance, the occurrence of the event is-taken(John) is supposed to mean that the book under consideration was at that point taken out by user John. Clearly, that event may only happen in a state where the book is available.

Actually, the picture is a little bit more complex: what we have been talking about so far is just an *object aspect* [ES91]. Indeed, if we consider an object like "Abstract and Concrete Categories" in some library, it may have several aspects: it is a book (having all properties of books in that library), but it may also be a reserved-book (having the added properties of the so called reserved books that may not be taken away from the library). The other objects may interact with it as a book, disregarding the additional services of a reserved book if any, unknowing that its behaviour may be constrained because it is also a reserved book. Of course, they may also interact with it as a reserved book. In short, "Abstract and Concrete Categories" may play the role of a book although it is a reserved book. Indeed, it might even play other roles like thing. This mechanism (polymorphism by inclusion) is essential to reuse, one of the key features of object-orientation.

Therefore, an object is, in general, a collection of aspects consistent with each other. From a specification point of view, it is clear that we should specify only object aspects! That is, objects are specified through their aspects. Therefore, we need an object aspect specification logic for "local" reasoning about a given aspect disregarding the other aspects.

But the interplay between the different aspects of the same object is to be carried out at the "global" level where we recognize relationships between different aspects: for instance, the specialization relationship we have been illustrating. Furthermore, at the global level we can also impose the interactions that may exist between different aspects: for instance, when a book is-taken we know that a user takes it. Thus, interaction constraints are expressed by formulae relating events in different aspects.

To this end, it is important to consider *aggregation* aspects, such as the composite of a book and an user where the interactions between that book and that user are to be specified. Then, the same specification and verification techniques can be used for specifying local and global (interaction) behaviour.

Note also that similar aspects constitute aspect types or *classes*. Clearly, it makes more sense to specify classes instead of aspects per se. Moreover, we may need attributes that take values in such classes (besides data types). For illustration consider the class *book* composed of all books, including "Abstract and Concrete Categories". It is important to stress that we are dealing here with aspect types (classes) and not object types. Indeed, each object may have several related aspects (recall the example of "Abstract and Concrete Categories" with two aspects: a book aspect and a reserved-book aspect). Each of its aspects is an instance of a class.

Specialization appears at the class level as follows: for instance we say that *reserved-book* is a subclass of *book*, or *book* is a superclass of *reserved-book*. Aggregation is reflected at the class level also. We need aggregation classes, such as $book \otimes user$ whose instances are the pairs of books and users.

Therefore, the envisaged specification logic must provide the means for referring to individuals of different types (eg *integer*, *book*, *user*, $book \otimes user$, *reserved-book*), hence the many-sorted first-order approach. Furthermore, the logic should provide an ordering of classes so that specialization can be made explicit. The specification logic must also provide the means for describing the behaviour of each individual, hence the temporal approach. The signature of each class should include attributes and actions.

With respect to the interplay between the local and global levels of reasoning, we should expect some form of inheritance mechanisms, eg for specialization (allowing the inclusion of theorems about a superclass into the set of theorems about a subclass). Moreover, we should also include reflection mechanisms allowing the theorems to be obtained in a class by taking into account its embedding in the community, (eg if we know that every user returns the books that he/she takes away, we can conclude that every book is returned). Therefore, the theories and semantics of the different classes are closely related in a rich structure. The presentation and exploitation of this structure is much simplified by the use of a modicum of techniques from category theory.

3 Object specification logic

We now proceed with the development of the envisaged object specification logic (OSL), starting with signatures (alphabets), followed by the syntax (of terms and formulae), interpretation structures and satisfaction. We conclude with specifications and theories plus the deductive system.

3.1 Signatures

We assume as given once and for all the underlying data signature $\Sigma_{DT} = \langle S_{DT}, OP \rangle$. We also assume that *bool* $\in S_{DT}$ and $OP_{\epsilon,bool} = \{false, true\}$.

Definition 3.1 An *inheritance net* is a finite partial order $N = \langle PC, \leq \rangle$.

In the sequel we assume a given inheritance net $N = \langle PC, \leq \rangle$. We denote by C_N the cartesian category freely generated by N. Each element of PC is called a *prime class*. Each element of $|C_N|$ is called a *class*. Non prime classes are said to be *composite*. For each pair $c, c' \in PC$ such that $c' \leq c$ we denote by $\iota : c' \to c$ the corresponding morphism and call it a *class inclusion*. The terminal class is denoted by 1. We refer to a product of classes by $c_1 \otimes ... \otimes c_m$ with *class projections* $\pi_i : c_1 \otimes ... \otimes c_m \to c_i$ for $i = 1, ..., m$.

Definition 3.2 The set of *sorts* induced by N is the set $\overline{S}_N = S_N \cup S_N^\tau$ where $S_N = S_{DT} \cup |C_N|$ and $S_N^\tau = \{\tau_c : c \in PC\}$.

The elements of S_N are called *proper sorts*. The elements of S_N^τ are called *action sorts*.

Definition 3.3 A *class signature* over N is a triple $\langle c, ACT, ATT \rangle$ such that:

- $c \in |C_N|$;
- ACT is an $S_N^* \otimes S_N^\tau$-indexed family of finite sets such that whenever $c \in PC$:
  - $\perp \in ACT_{\epsilon,\tau_c}$;
  - $ACT_{w,\tau_{c'}} = \emptyset$ provided that $c \not\leq c'$;
  - $ACT_{w,\tau_{c'}} \subseteq ACT_{w,\tau_c}$ provided that $c \leq c'$;
- ATT is an $\overline{S}_N^* \otimes \overline{S}_N$-indexed family of finite sets such that whenever $c \in PC$:
  - $ATT_{\tau_c,bool} = \{enb\}$;
  - $ATT_{\epsilon,\tau_c} = \{occ\}$;
- all these sets are pairwise disjoint.

Given a class signature $\langle c, ACT, ATT \rangle$, c is said to be the *underlying* class, each element $z \in ACT_{w,s}$ is said to be an *action symbol* with parameter sorts w and result sort s, and each element $a \in ATT_{w,s}$ is said to be an *attribute symbol* with parameter sorts w and result sort s. An attribute symbol in $ATT_{\epsilon,s}$ for $s \in S_N$ is said to be a *slot symbol*.

Definition 3.4 A *class signature morphism* over N

$$\sigma : \langle c, ACT, ATT \rangle \to \langle c', ACT', ATT' \rangle$$

is a triple $\langle \sigma_{id}, \sigma_{ACT}, \sigma_{ATT} \rangle$ where:

- $\sigma_{id} : c' \to c$ is a morphism in C_N;
- σ_{ACT} is an $S_N^* \otimes S_N^\tau$-indexed family of inclusion maps $\sigma_{ACT_{w,s}} : ACT_{w,s} \hookrightarrow ACT'_{w,s}$;
- σ_{ATT} is an $\overline{S}_N^* \otimes \overline{S}_N$-indexed family of maps $\sigma_{ATT_{w,s}} : ATT_{w,s} \to ATT'_{w,s}$.

Class signatures and their morphisms over N constitute the category $ASig_N$. A class signature morphism σ is said to be an *inclusion* iff σ_{id} is a class inclusion and the map $\sigma_{ATT_{w,s}}$ is an inclusion. And σ is said to be an *injection* iff σ_{id} is a class projection and the map $\sigma_{ATT_{w,s}}$ is an injection.

Definition 3.5 An *object community signature* is a pair $\Sigma = \langle N, Sg \rangle$ where:

- $N = \langle PC, \leq \rangle$ is an inheritance net;
- $Sg : C_N^{op} \to ASig_N$ is a functor preserving op-inclusions and finite coproducts and such that:
  - $Sg(c)_1 = c$;
  - $Sg(\sigma)_{id} = \sigma$.

From now on we concentrate on a given object community signature $\Sigma = \langle N, Sg \rangle$ and write $\Sigma^c = \langle c, ACT^c, ATT^c \rangle$ for $Sg(c)$. Clearly:

- $Sg(1) = \langle 1, \emptyset, \emptyset \rangle$;
- $Sg(c_1 \otimes ... \otimes c_m)$ endowed with the injections $Sg(\pi_i : c_i \to c_1 \otimes ... \otimes c_m)$ for $i = 1, ..., m$ is a coproduct of $Sg(c_1), ..., Sg(c_m)$;
- $Sg(\iota : c' \to c)$ is an inclusion.

3.2 Terms and formulae

We assume as given once and for all an $\overline{S}_N$-indexed family X of infinite, pairwise disjoint sets of variables endowed with a choice map $\delta_s : \wp(X_s) \to X_s$ such that $\delta_s(Y) \in Y$ for each $s \in \overline{S}_N$.

Definition 3.6 Let c be a class. Then, T^c is an $\overline{S}_N$-indexed family of sets of *terms* inductively defined as follows:

- $y \in T_s^c$, provided that $y \in X_s$;
- $f(t_1, ..., t_n) \in T_s^c$, provided that $f \in OP_{s_1 \dots s_n, s}$ and $t_i \in T_{s_i}^c$ for every $i = 1, ..., n$;
- $z(t_1, ..., t_n) \in T_s^c$, provided that $z \in ACT_{s_1 \dots s_n, s}^c$ and $t_i \in T_{s_i}^c$ for every $i = 1, ..., n$;
- $a(t_1, ..., t_n) \in T_s^c$, provided that $a \in ATT_{s_1 \dots s_n, s}^c$ and $t_i \in T_{s_i}^c$ for every $i = 1, ..., n$;

- $\mu(t) \in T^c_{s'}$, provided that $\mu : s \to s'$ is a class morphism in C_N and $t \in T^c_s$.

Each element of T^c_s is called a term of sort s on class c. A term without action symbols and without attribute symbols is called a *data term.*

Definition 3.7 The set Q^c of *formula schemata* on class c is inductively defined as follows:

- $t \in Q^c$ provided that $t \in T^c_{bool}$;
- $(t = t') \in Q^c$ provided that $t, t' \in T^c_s$;
- $\star \in Q^c$;
- $(\neg\, q), (\mathsf{X}\, q), (\mathsf{G}\, q), (\mathsf{Y}\, q), (\mathsf{H}\, q) \in Q^c$ provided that $q \in Q^c$;
- $(q \Rightarrow q') \in Q^c$ provided that $q, q' \in Q^c$;
- $(\forall y\ q) \in Q^c$ provided that $q \in Q^c$ and $y \in X_s$ for some sort s.

As usual we are free to introduce abbreviations. In the sequel we use: $(q \vee q')$ for $((\neg\, q) \Rightarrow q')$, $(q \wedge q')$ for $(\neg((\neg\, q) \vee (\neg\, q')))$, $(q \Leftrightarrow q')$ for $((q \Rightarrow q') \wedge (q' \Rightarrow q))$, $(\exists y\ q)$ for $(\neg(\forall y\ (\neg\, q)))$, $(\mathsf{G}_\circ\, q)$ for $(q \wedge (\mathsf{G}\, q))$, and $(\mathsf{Y}^?\, q)$ for $(\star \vee (\mathsf{Y}\, q))$. In the sequel $Var_s(q)$ denotes the set of variables of sort s occurring in q. A formula schema without action symbols, without attribute symbols and without temporal operators is called a *rigid data formula schema.*

Definition 3.8 The set of *formulae* on class c is as follows:

$$L^c = \{x.q : x \in X_c, q \in Q^c\}.$$

A *slot constraint* is a formula without temporal operators and without attribute symbols besides the slot symbols. We denote by SC^c the set of slot constraints on class c. It is convenient to write $x.q$ replacing in q each attribute symbol a by $x.a$.

Example 3.9 Consider the following signature and assertions about flipflops, assuming that $flipflop \in PC$ and $x \in X_{flipflop}$:

- $ATT_{\epsilon,bool} = \{on\}$;
- $ATT_{\epsilon,flipflop} = \{another\}$;
- $ATT_{\tau_{flipflop},bool} = \{enb\}$;
- $ATT_{\epsilon,\tau_{flipflop}} = \{occ\}$;
- $ACT_{\epsilon,\tau_{flipflop}} = \{\bot, flip, flop\}$;
- $(\neg(x.another = x))$;
- $(x.on \Leftrightarrow x.enb(flip))$;

- $((\neg\, x.on) \Leftrightarrow x.enb(flop))$;
- $(((x.occ = \perp) \wedge \alpha) \Rightarrow (\mathsf{X}\, \alpha))$ for any $\alpha \in SC^{flipflop}$;
- $(((x.occ = flip) \wedge \alpha^{x.on}_{false}) \Rightarrow (\mathsf{X}\, \alpha))$ for any $\alpha \in SC^{flipflop}$;
- $(((x.occ = flop) \wedge \alpha^{x.on}_{true}) \Rightarrow (\mathsf{X}\, \alpha))$ for any $\alpha \in SC^{flipflop}$;
- $(\forall y\, ((x.occ = y) \Rightarrow x.enb(y)))$.

Example 3.10 Consider the following assertion about two flipflops assuming that $v \in X_{flipflop \otimes flipflop}$:

- $((v.another_1 = \pi_2(v)) \Rightarrow ((v.occ_1 = flip) \Rightarrow (v.occ_2 = flip)))$

  where:

  - $occ_i = Sg(\pi_i)_{ATT_{\epsilon,\tau_{flipflop}}}(occ)$ for $i = 1, 2$;
  - $another_1 = Sg(\pi_1)_{ATT_{\epsilon,flipflop}}(another)$.

3.3 Interpretation structures

We assume a given algebra dt over $\Sigma_{dt} = \langle S_{DT}, OP\rangle$ such that

$$dt_{bool} = \{false_{dt}, true_{dt}\}.$$

Definition 3.11 A functor $Pop : C_N \to Set$ over $\Sigma = \langle N, Sg\rangle$ preserving inclusions and finite products is said to be a class *population functor*.

In the sequel we assume a given population functor Pop. We write Pop_c for $Pop(c)$ and extend the notation so that Pop_s stands for dt_s when $s \in S_{dt}$. Clearly, Pop_1 is a singleton, say $\{1\}$.

Definition 3.12 The set $act(c)$ of *actions* on prime class c over Σ for Pop is as follows:

$$\bigcup_{\langle s_1,\ldots,s_n\rangle \in S_N^*} \{z(u_1, ..., u_n) : z \in ACT^c_{s_1\ldots s_n,\tau_c}, u_i \in Pop_{s_i} \text{ for } i = 1, ..., n\}.$$

In the sequel we write Pop_{τ_c} for $act(c)$ for each $c \in PC$.

Definition 3.13 An *instance* over Σ for Pop is a pair $\langle c, u\rangle$ where c is a class and $u \in Pop_c$.

Definition 3.14 An *instance morphism* over Σ for Pop

$$\sigma : \langle c, u\rangle \to \langle c', u'\rangle$$

is a class morphism $\sigma : c \to c'$ such that $Pop(\sigma)(u) = u'$.

Instances and their morphisms over Σ for Pop constitute the category $Inst_{\Sigma,Pop}$.

Theorem 3.15 The category $Inst_{\Sigma,Pop}$ is cartesian.

Proof:

- the terminal object is $\langle 1, 1\rangle$;
- the product of $\langle c_1, u_1\rangle$ and $\langle c_2, u_2\rangle$ is $\langle c_1 \otimes c_2, u\rangle$ endowed with the projections π_1, π_2, provided that $Pop(\pi_i)(u) = u_i$ for $i = 1, 2$. $\square$

Definition 3.16 The set $att(c)_s$ of *attributes of sort s* on c over Σ for Pop is as follows:

$$\bigcup_{\langle s_1 \ldots s_n\rangle \in \overline{S}_N^*} \{a(u_1, ..., u_n) : a \in ATT^c_{s_1 \ldots s_n, s}, u_i \in Pop_{s_i} \text{ for } i = 1, ..., n\}.$$

Definition 3.17 The set $att(c)$ of *attributes* on c over Σ for Pop is as follows:

$$\bigcup_{s \in \overline{S}_N} att(c)_s.$$

For each attribute $b \in att(c)_s$, we say that Pop_s is the *domain* D_b of b.

Definition 3.18 The cone of *snapshots* on class c over Σ for Pop is the product in Set with vertex

$$snp(c) = \bigotimes_{b \in att(c)} D_b$$

and projections $p_b : snp(c) \to D_b$ for $b \in att(c)$.

Clearly, $snp(1)$ is a singleton, say $\{1\}$.

Definition 3.19 Given a class morphism $\sigma : c' \to c$ and a snapshot $V' \in snp(c')$ over Σ for Pop, the *image* $\sigma(V')$ of V' under σ is the unique $V \in snp(c)$ such that

$$p_{a(u_1,\ldots,u_n)}(V) = p_{Sg(\sigma)_{ATT_{s_1 \ldots s_n, s}}(a)(u_1,\ldots,u_n)}(V')$$

for each $a \in ATT_{s_1 \ldots s_n, s}$ and $u_i \in Pop_{s_i}$ for $i = 1, ..., n$.

Definition 3.20 A *run* over Σ for Pop is a pair $\langle c, \lambda\rangle$ where c is a class and $\lambda : \mathbb{N} \to snp(c)$ such that whenever $c \in PC$ and $k \in \mathbb{N}$:

- $p_{enb(\perp)}(\lambda_k) = true_{dt}$;
- if $p_{occ}(\lambda_k) = \perp$ then $p_b(\lambda_{k+1}) = p_b(\lambda_k)$ for each $b \in att(c)$, $b \neq occ$;
- if $p_{occ}(\lambda_k) = z(u_1, ..., u_n)$ then $p_{enb(z(u_1,\ldots,u_n))}(\lambda_k) = true_{dt}$.

A run $\langle c, \lambda\rangle$ is said to be a run on class c. Clearly, there is only one run on the terminal class 1. We denote it by $\langle 1, 1\rangle$.

Definition 3.21 A *run morphism* over Σ for Pop

$$\sigma : \langle c, \lambda\rangle \to \langle c', \lambda'\rangle$$

is a class morphism $\sigma : c \to c'$ such that $\lambda'_k = \sigma(\lambda_k)$ for every $k \in \mathbb{N}$.

Runs and their morphisms over Σ for Pop constitute the category $Run_{\Sigma,Pop}$.

Theorem 3.22 The category $Run_{\Sigma,Pop}$ is cartesian.

Proof:

- the terminal object is $\langle 1, 1\rangle$;
- the product of $\langle c_1, \lambda_1\rangle$ and $\langle c_2, \lambda_2\rangle$ is $\langle c_1 \otimes c_2, \lambda\rangle$ endowed with the projections π_1, π_2 provided that:

$$p_{Sg(\pi_i)_{ATT^{c_i}_{s_1\ldots s_n,s}}(a)(u_1,\ldots,u_n)}(\lambda_k) = p_{a(u_1,\ldots,u_n)}(\lambda_{i_k})$$

for every $k \in I\!N$ and $i = 1, 2$. $\square$

Definition 3.23 A *community run* over Σ for Pop is a functor

$$\Lambda : Inst_{\Sigma,Pop} \to Run_{\Sigma,Pop}$$

mapping each $\langle c, u\rangle$ into a run on class c and preserving finite products.

Definition 3.24 An *object community* over Σ is a pair $oc = \langle Pop, \Lambda\rangle$ where

- $Pop : C_N \to Set$ is a population functor over Σ;
- $\Lambda : Inst_{\Sigma,Pop} \to Run_{\Sigma,Pop}$ is a community run over Σ for Pop.

From now on we concentrate on an object community $oc = \langle Pop, \Lambda\rangle$ over the object community signature Σ.

3.4 Satisfaction

Definition 3.25 An *assignment* θ into oc is an $\overline{S}_N$-indexed family of maps $\theta_s : X_s \to Pop_s$.

Definition 3.26 Let $y \in X_s$ and θ, θ' be assignments into oc. We say that θ and θ' are *y-equivalent* iff for every $s' \in \overline{S}_N$ and $y' \in X_{s'}$ $\theta_{s'}(y') = \theta'_{s'}(y')$ whenever $y' \neq y$.

Definition 3.27 The *interpretation* of terms on class c for assignment θ at point $k \in I\!N$ of $u \in Pop_c$ in oc is inductively defined as follows:

- $[\![y]\!]^{\theta}_{k,u,oc} = \theta_s(y)$;
- $[\![f(t_1, ..., t_n)]\!]^{\theta}_{k,u,oc} = f_{dt}([\![t_1]\!]^{\theta}_{k,u,oc}, ..., [\![t_n]\!]^{\theta}_{k,u,oc})$;
- $[\![z(t_1, ..., t_n)]\!]^{\theta}_{k,u,oc} = z([\![t_1]\!]^{\theta}_{k,u,oc}, ..., [\![t_n]\!]^{\theta}_{k,u,oc})$;
- $[\![a(t_1, ..., t_n)]\!]^{\theta}_{k,u,oc} = p_{a([\![t_1]\!]^{\theta}_{k,u,oc},\ldots,[\![t_n]\!]^{\theta}_{k,u,oc})}(\Lambda(\langle c, u\rangle)_{2,k})$;
- $[\![\mu(t)]\!]^{\theta}_{k,u,oc} = Pop(\mu)([\![t]\!]^{\theta}_{k,u,oc})$.

We note that data terms are interpreted within dt. We denote by $[\![t]\!]^{\theta}_{dt}$ the value of such a term t for θ in dt.

Definition 3.28 The *satisfaction* of formula schemata on class c for assignment θ at point $k \in I\!N$ of $u \in Pop_c$ in oc is inductively defined as follows:

- $\theta, k, u, oc \Vdash_c t$ iff $[\![t]\!]^{\theta}_{k,u,oc} = true_{dt}$;
- $\theta, k, u, oc \Vdash_c (t = t')$ iff $[\![t]\!]^{\theta}_{k,u,oc} = [\![t']\!]^{\theta}_{k,u,oc}$;
- $\theta, k, u, oc \Vdash_c \star$ iff $k = 0$;
- $\theta, k, u, oc \Vdash_c (\neg\, q)$ iff not $\theta, k, u, oc \Vdash_c q$;
- $\theta, k, u, oc \Vdash_c (\mathsf{X}\, q)$ iff $\theta, k+1, u, oc \Vdash_c q$;
- $\theta, k, u, oc \Vdash_c (\mathsf{G}\, q)$ iff $\theta, k', u, oc \Vdash_c q$ for every $k' > k$;
- $\theta, k, u, oc \Vdash_c (\mathsf{Y}\, q)$ iff $k > 0$ and $\theta, k-1, u, oc \Vdash_c q$;
- $\theta, k, u, oc \Vdash_c (\mathsf{H}\, q)$ iff $\theta, k', u, oc \Vdash_c q$ for every $k' < k$;
- $\theta, k, u, oc \Vdash_c (q \Rightarrow q')$ iff not $\theta, k, u, oc \Vdash_c q$ or $\theta, k, u, oc \Vdash_c q'$;
- $\theta, k, u, oc \Vdash_c (\forall y\; q)$ iff $\theta', k, u, oc \Vdash_c q$ for every θ' y-equivalent to θ.

We note that the satisfaction of rigid data formula schemata is established within dt. We write $\theta, dt \Vdash q$ for the satisfaction of such a rigid data formula schema q for θ in dt.

Definition 3.29 The *satisfaction* of formulae on class c for assignment θ at point $k \in I\!N$ of $u \in Pop_c$ in oc is defined as follows: $\theta, k, u, oc \Vdash_c x.q$ iff if $\theta_c(x) = u$ then $\theta, k, u, oc \Vdash_c q$.

Definition 3.30 We say that the object community oc *satisfies* the formula φ on class c, written $oc \Vdash_c \varphi$, iff $\theta, k, u, oc \Vdash_c \varphi$ for every assignment θ, $k \in I\!N$, $u \in Pop_c$.

Definition 3.31 Let $\Psi \subseteq L^c$ and $\varphi \in L^c$. We say that Ψ *entails* φ on class c, written $\Psi \vDash_c \varphi$, iff for every object community oc if $oc \Vdash_c \psi$ for every $\psi \in \Psi$ then $oc \Vdash_c \varphi$.

Definition 3.32 Let $\Psi \subseteq L^c$. The *closure by entailment* of Ψ on class c is the set $\Psi^{\vDash_c} = \{\psi : \Psi \vDash_c \psi\}$.

Definition 3.33 Let $A = \{A^c\}_{c \in |C_N|}$ with $A^c \subseteq L^c$. We say that oc *satisfies* A, written $oc \Vdash A$, iff $oc \Vdash_c \psi$ for each $\psi \in A^c$.

Definition 3.34 Let $A = \{A^c\}_{c \in |C_N|}$ with $A^c \subseteq L^c$. The *closure by entailment* of A is the family $A^{\vDash} = \{A^{\vDash c}\}_{c \in |C_N|}$ such that

$$A^{\vDash c} = \{\psi \in L^c : \text{ if } oc \Vdash A \text{ then } oc \Vdash_c \psi\}.$$

Note that one should not confuse $A^{\vDash c}$ with $A^{c\,\vDash_c}$. The latter is included in the former. In general, the inclusion is strict.

Definition 3.35 Let $A = \{A^c\}_{c \in |C_N|}$ and $H = \{H^c\}_{c \in |C_N|}$ with $A^c, H^c \subseteq L^c$. The *entailment of* H on A is the family $A^{H\vDash} = \{A^{H\vDash c}\}_{c \in |C_N|}$ such that

$$A^{H\vDash c} = (A \cup H)^{\vDash c}.$$

3.5 Community institution

Definition 3.36 Let $\sigma : c' \to c$ be a class morphism. The *translation maps* induced by σ are as follows:

- $\sigma_{T,s} : T_s^c \to T_s^{c'}$ inductively defined as follows for $s \in \overline{S}_N$:
  - $\sigma_{T,s}(y) = y$;
  - $\sigma_{T,s}(f(t_1, ..., t_n)) = f(\sigma_{T,s_1}(t_1), ..., \sigma_{T,s_n}(t_n))$;
  - $\sigma_{T,s}(z(t_1, ..., t_n)) = z(\sigma_{T,s_1}(t_1), ..., \sigma_{T,s_n}(t_n))$;
  - $\sigma_{T,s}(a(t_1, ..., t_n)) = Sg(\sigma)_{ATT_{s_1 \ldots s_n,s}}(a)(\sigma_{T,s_1}(t_1), ..., \sigma_{T,s_n}(t_n))$;
  - $\sigma_{T,s'}(\mu(t)) = \mu(\sigma_{T,s}(t))$.
- $\sigma_Q : Q^c \to Q^{c'}$ inductively defined as follows:
  - $\sigma_Q(t) = \sigma_{T,bool}(t)$;
  - $\sigma_Q((t = t')) = (\sigma_{T,s}(t) = \sigma_{T,s}(t'))$;
  - $\sigma_Q(\star) = \star$;
  - $\sigma_Q((\neg\, q)) = (\neg(\sigma_Q(q)))$;
  - $\sigma_Q((\mathsf{X}\, q)) = (\mathsf{X}(\sigma_Q(q)))$;
  - $\sigma_Q((\mathsf{G}\, q)) = (\mathsf{G}(\sigma_Q(q)))$;
  - $\sigma_Q((\mathsf{Y}\, q)) = (\mathsf{Y}(\sigma_Q(q)))$;
  - $\sigma_Q((\mathsf{H}\, q)) = (\mathsf{H}(\sigma_Q(q)))$;
  - $\sigma_Q((q \Rightarrow q')) = (\sigma_Q(q) \Rightarrow \sigma_Q(q'))$;
  - $\sigma_Q((\forall y\; q)) = (\forall y\; \sigma_Q(q))$.
- $\sigma_L : L^c \to L^{c'}$ defined as follows:
  - $\sigma_L(x.q) = x'.\sigma_Q(q^x_{\sigma(x')})$ where x' is $\delta_{c'}(X_{c'} \setminus Var_{c'}(q))$ and $q^x_{\sigma(x')}$ is the formula schema obtained by replacing all free occurrences of x by $\sigma(x')$ in q.

Lemma 3.37 Let $\sigma : c' \to c$ be a class morphism, θ an assignment, $k \in I\!N$, $u' \in Pop_{c'}$, $t \in T_s^c$ and $q \in Q^c$. Then:

- $[\![t]\!]^\theta_{k,Pop(\sigma)(u'),oc} = [\![\sigma_{T,s}(t)]\!]^\theta_{k,u',oc}$;
- $\theta, k, Pop(\sigma)(u'), oc \Vdash_c q$ iff $\theta, k, u', oc \Vdash_{c'} \sigma_Q(q)$.

Lemma 3.38 Let $\sigma : c' \to c$ be a class morphism, $k \in I\!N$, $u' \in Pop_{c'}$, $x \in X_c$, x' be $\delta_{c'}(X_{c'} \setminus Var_{c'}(q))$ and $q \in Q^c$. Then, $\theta, k, Pop(\sigma)(u'), oc \Vdash_c x.q$ for every assignment θ iff $\theta, k, u', oc \Vdash_{c'} x'.\sigma_Q(q^x_{\sigma(x')})$ for every assignment θ.

Theorem 3.39 The signature Σ, the family X and the community oc induce the institution $I_{\Sigma,X,oc} = \langle C_N^{op}, Sen, Int, \Vdash \rangle$ such that:

- $Sen(c) = L^c$;

- $Sen(\sigma : c \to c') = \sigma_L : L^c \to L^{c'}$;
- $Int(c) = Pop_c$;
- $Int(\sigma : c \to c') = Pop(\sigma) : Pop_{c'} \to Pop_c$;
- $u \Vdash\!\vdash_c \varphi$ iff $\theta, k, u, oc \Vdash_c \varphi$ for every assignment θ and $k \in I\!N$.

In the sequel we shall use the *satisfaction condition* that we just established:

$$u' \Vdash\!\vdash_{c'} \sigma_L(\varphi) \text{ iff } Pop(\sigma)(u') \Vdash\!\vdash_c \varphi$$

provided that $\sigma : c' \to c$ in C_N.

3.6 Specifications and theories

Definition 3.40 A *class specification* over Σ is a pair $\langle c, \Phi \rangle$ where $c \in |C_N|$ and $\Phi \subseteq L^c$. A *class theory* over Σ is a class specification $\langle c, \Phi \rangle$ such that $\Phi^{\models_c} = \Phi$.

Definition 3.41 A *class theory morphism* over Σ

$$\sigma : \langle c, \Phi \rangle \to \langle c', \Phi' \rangle$$

is a class morphism $\sigma : c' \to c$ such that $\sigma_L(\varphi) \in \Phi'$ whenever $\varphi \in \Phi$.

Class theories and their morphisms over Σ constitute the category CTh_Σ.

Theorem 3.42 The category CTh_Σ is cocartesian.

Proof:

- the initial object is $\langle 1, \emptyset^{\models_1} \rangle$;
- the coproduct of $\langle c_1, \Phi_1 \rangle$ and $\langle c_2, \Phi_2 \rangle$ is

  $$\langle c_1 \otimes c_2, (\{\pi_{1\,L}(\varphi) : \varphi \in \Phi_1\} \cup \{\pi_{2\,L}(\varphi) : \varphi \in \Phi_2\})^{\models_{c_1 \otimes c_2}} \rangle$$

  endowed with the injections π_1, π_2. $\square$

Definition 3.43 An *object community specification* is a pair $spec = \langle \Sigma, Ax \rangle$ where

- $\Sigma = \langle N, Sg \rangle$ is an object community signature;
- $Ax = \{Ax^c\}_{c \in |C_N|}$ with $Ax^c \subseteq L^c$ for each c.

An *object community theory* is an object community specification $spec = \langle \Sigma, Ax \rangle$ such that $Ax^{\models} = Ax$.

Definition 3.44 Let $spec = \langle \Sigma, Ax \rangle$ be an object community specification. The *closure by entailment* of $spec$ is $spec^{\models} = \langle \Sigma, Ax^{\models} \rangle$.

Theorem 3.45 An object community specification $spec = \langle \Sigma, Ax \rangle$ induces the functor $Th : C_N^{op} \to CTh_\Sigma$ such that:

- $Th(c) = Ax^{\vdash c}$;
- $Th(\sigma) = \sigma_L$.

Remark 3.46 Only special "interaction" axioms may be included in the specification of a composite class $c_1 \otimes c_2$, namely axioms of the form

$$(cond_1 \Rightarrow ((x.occ_1 = z_1(t_1, ..., t_n)) \Rightarrow (x.occ_2 = z_2(t'_1, ..., t'_k))))$$

where $cond_1$ is a condition not involving symbols from c_2, z_1 is an action symbol from c_1, and z_2 is an action symbol from c_2.

In the sequel we concentrate on a given object community specification $spec = \langle \Sigma, Ax \rangle$.

3.7 Hilbert calculus

Definition 3.47 The *closure by consequence* of *spec* is the pair $spec^\vdash = \langle \Sigma, Ax^\vdash \rangle$ where $Ax^\vdash = \{Ax^{\vdash c}\}_{c \in |C_N|}$ is inductively defined as follows:

1. $\varphi \in Ax^{\vdash c}$
   provided that $\varphi \in Ax^c$;
2. $x.q \in Ax^{\vdash c}$
   provided that q is an axiom of a floating linear temporal, first-order with equality calculus (for details see for instance [MP92, SSC92]);
3. $x.q' \in Ax^{\vdash c}$
   provided that $x.q \in Ax^{\vdash c}$ and $x.(q \Rightarrow q') \in Ax^{\vdash c}$;
4. $x.(\mathsf{X}\, q), x.(\mathsf{G}\, q), x.(\mathsf{Y}^?\, q), x.(\mathsf{H}\, q), x.(\forall y\ q) \in Ax^{\vdash c}$
   provided that $x.q \in Ax^{\vdash c}$;
5. $x.(\mu_n(...(\mu_1(y)...) = \mu(y)) \in Ax^{\vdash c}$
   provided that $\mu = \mu_n \circ ... \circ \mu_1$;
6. $x.q \in Ax^{\vdash c}$
   provided that q is a rigid data formula schema such that $\theta, dt \Vdash q$ for every assignment θ;
7. $x.((z(x_1, ..., x_n) = z(x'_1, ..., x'_n)) \Rightarrow ((x_1 = x'_1) \wedge ... \wedge (x_n = x'_n))) \in Ax^{\vdash c}$
   provided that $z \in ACT^c_{s_1 \ldots s_n, s}$;
8. $x.(\neg(z(x_1, ..., x_n) = z'(x'_1, ..., x'_{n'}))) \in Ax^{\vdash c}$
   provided that $z \in ACT^c_{s_1 \ldots s_n, s}$, $z' \in ACT^c_{s'_1 \ldots s'_{n'}, s}$ and $z \neq z'$;
9. $x.(\forall y\ (\bigvee_{s_1 \ldots s_n \in S_N^*} (\bigvee_{z \in ACT^c_{s_1 \ldots s_n, s}} (\exists y_1\ (...(\exists y_n (y = z(y_1, ..., y_n)))...)))) \in Ax^{\vdash c}$
   for each sort $s \in S_N^\tau$;

10. $x.enb(\bot) \in Ax^{\vdash c}$
provided that $c \in PC$;

11. $x.((occ = \bot) \Rightarrow ((a(x_1, ..., x_n) = y) \Rightarrow (\mathsf{X}(a(x_1, ..., x_n) = y)))) \in Ax^{\vdash c}$
provided that $c \in PC$, $a \in ATT^c_{s_1 \ldots s_n, s}$ and $a \neq occ$;

12. $x.enb(occ) \in Ax^{\vdash c}$
provided that $c \in PC$;

13. $\sigma_L(\varphi) \in Ax^{\vdash c'}$
provided that $\sigma : c' \to c$ is a class morphism and $\varphi \in Ax^{\vdash c}$;

14. $\varphi \in Ax^{\vdash c_i}$
for $i = 1, ..., m$ provided that $\pi_{i\,L}(\varphi) \in Ax^{\vdash c_1 \otimes \ldots \otimes c_m}$.

Rule 1 states that every axiom is a theorem: *AX*. Rule 3 is *modus ponens*: *MP*. Rule 4 establishes the *necessitations*: *NEC*. Rules 5 depicts a property of population functors: *POP*. Rule 6 brings in all the valid formulae in the underlying data algebra: *DT*. Rules 7–9 reflect that actions are not interpreted and are generated: *ACT*. Rules 10–12 depict properties of the runs: *RUN*. Rule 13 corresponds to *inheritance*: *INH*. Rule 14 corresponds to *reflection*: *REF*. Note that *INH* and *REF* lead to

$$\pi_{i\,L}(\varphi) \in Ax^{\vdash c_1 \otimes \ldots \otimes c_m} \text{ iff } \varphi \in Ax^{\vdash c_i}.$$

Definition 3.48 Let $H = \{H^c\}_{c \in |C_N|}$ where $H^c \subseteq L^c$. The *consequence of* H on *spec* is the pair $spec^{H\vdash} = \langle \Sigma, Ax^{H\vdash} \rangle$ where $Ax^{H\vdash} = \{Ax^{H\vdash c}\}_{c \in |C_N|}$ is inductively defined as follows:

- $\varphi \in Ax^{H\vdash c}$ provided that $\varphi \in Ax^{\vdash c}$;
- $\varphi \in Ax^{H\vdash c}$ provided that $\varphi \in H^c$;
- $x.q' \in Ax^{H\vdash c}$ provided that $x.q \in Ax^{H\vdash c}$ and $x.(q \Rightarrow q') \in Ax^{H\vdash c}$;

Theorem 3.49 *(Soundness)* Let $H = \{H^c\}_{c \in |C_N|}$ where $H^c \subseteq L^c$. Then, $Ax^{H\vdash c} \subseteq Ax^{H\vDash c}$.

Hilbert calculi can be made more practical by using metatheorems. Indeed, for proving special classes of theorems (eg invariant slot constraints) it is useful to develop a special calculus with specific metatheorems. We also find convenient to use the metatheorem of deduction. To this end we need the following notation: $(H \cup \{\varphi\})$ denotes the family $\{(H \cup \{\varphi\})^c\}_{c \in |C_N|}$ where $(H \cup \{\varphi\})^c = H^c \cup \{\varphi\}$ and $(H \cup \{\varphi\})^{c'} = H^{c'}$ for $c' \neq c$, assuming that $H^c \subseteq L^c$ and $\varphi \in L^c$.

Theorem 3.50 *(Deduction rule: DED)* Let $H = \{H^c\}_{c \in |C_N|}$ where $H^c \subseteq L^c$, $q, q' \in Q^c$ and $x \in X_c$. Then:

- $x.(q \Rightarrow q') \in Ax^{H\vdash c}$ provided that $x.q' \in Ax^{(H \cup \{x.q\})\vdash c}$.

Theorem 3.51 *(Absurdum: ABS)* Let $H = \{H^c\}_{c \in |C_N|}$ where $H^c \subseteq L^c$, $q, q' \in Q^c$ and $x \in X_c$. Then:

- $x.(\neg q) \in Ax^{H \vdash c}$ provided that $x.q', x.(\neg q') \in Ax^{(H \cup \{x.q\}) \vdash c}$.

In the sequel, when presenting derivations, we refrain from giving the details of the reasoning in first order logic with equality as long as *NEC* (for quantification) is not used: we indicate *FOL* as the justification. Clearly, we can still apply the metatheorems above.

4 OSL at work

We now look at the practical impact of the proposed logic (OSL for Object Specification Logic). Namely, we consider its applications, introduce a practical invariant calculus and examine the need and availability of tools for automatic theorem proving.

4.1 What for

Clearly, OSL can be used as a "minimal" stand-alone object specification language. However, the lack of syntactic sugar does not make that type of application very promising, as it is shown in the next Subsection. Instead, one should adopt a "real" object specification language (such as OBLOG [SGS92, ESD93], TROLL [JSHS91] and GNOME [SR94]) and provide its semantics in terms of OSL (as it is done for instance for TROLL in [Jun93]).

The advantages of providing the semantics of the chosen object specification language in terms of OSL are clear: such a "semantics" endows the language with a verification calculus. The availability of a deductive system seems to be the single most important argument in favour of using formal specifications. Indeed, we can then check the consistency of specifications and the compliance of subsequent implementations (assuming that the verification calculus deals with reification mechanisms such as those considered for temporal logic in [FM94]). Of course, these verification tasks can be huge for any practical application. Therefore, automatic theorem proving tools are needed. They are discussed at the end of this Section. But even with machine support, exhaustive verification may not be feasible at all. Of more practical and immediate interest are answers to "systems queries". One may want to know if a given property of the application holds and if not in which situations it may be violated and if so how should the specification be modified. To this end abduction techniques seem to provide important results [GS94].

When checking if given properties hold it is convenient to distinguish between safety properties (such as invariants) and liveness properties (such as reactiveness). The former are easier to establish and the techniques for doing so are by now well understood. In Subsection 4.3 below we illustrate such techniques for OSL. We refrain to consider in this short paper the problem of verifying liveness properties. The insterested reader should start by consulting [MP91].

It is important to stress at this stage that OSL is a "monotonic" logic. Therefore, it is not trivial to explain in OSL the semantics of "non-monotonic" constructs of the chosen object-oriented specification language, if possible at all in a convenient way. For that purpose one should consider a "default" version of OSL with the capability to express and deal with defaults (besides absolute axioms). The interested reader should start by consulting [BL91, BLR93].

4.2 Specification example

Consider the following specification example that will also be used later on for illustrating proofs of invariants. Though rather simple and abstract it is sufficiently rich for our purposes, namely concerning the impact of interaction on some interesting invariants.

- $N = \langle\{c_1, c_2, c_3\}, \{(c_3, c_2), (c_1, c_1), (c_2, c_2), (c_3, c_3)\}\rangle$;
- Σ^{c_1} is such that:
  - $ACT^{c_1}_{\epsilon,\tau_{c_1}} = \{Z\}$;
  - $ATT^{c_1}_{\epsilon,int} = \{A\}$;
- Σ^{c_2} is such that:
  - $ACT^{c_2}_{\epsilon,\tau_{c_2}} = \{Z\}$;
  - $ATT^{c_2}_{\epsilon,int} = \{A\}$;
- Σ^{c_3} is such that:
  - $ACT^{c_3}_{\epsilon,\tau_{c_3}} = \{Z, Z'\}$;
  - $ATT^{c_3}_{\epsilon,int} = \{A, A'\}$;
- Ax^{c_1} is composed of:
  - $(\star \Rightarrow (x_1.A = 0))$;
  - $(((x_1.occ = Z) \wedge \alpha^{x_1.A}_{x_1.A+1}) \Rightarrow (\mathsf{X}\,\alpha))$ provided that $\alpha \in SC^{c_1}$;
- Ax^{c_2} is composed of:
  - $(\star \Rightarrow (x_2.A = 0))$;
  - $(((x_2.occ = Z) \wedge \alpha^{x_2.A}_{x_2.A-1}) \Rightarrow (\mathsf{X}\,\alpha))$ provided that $\alpha \in SC^{c_2}$;
- Ax^{c_3} is composed of:
  - $(\star \Rightarrow (x_3.A' = 0))$;
  - $(((x_3.occ = Z) \wedge \alpha) \Rightarrow (\mathsf{X}\,\alpha))$ provided that $\alpha \in (SC^{c_3} \setminus SC^{c_2})$;
  - $(((x_3.occ = Z') \wedge \alpha^{x_3.A'}_0) \Rightarrow (\mathsf{X}\,\alpha))$ provided that $\alpha \in (SC^{c_3} \setminus SC^{c_2})$;
- $Ax^{c_1 \otimes c_2}$ is composed of:
  - $((x_{12}.occ_1 = Z) \Rightarrow (x_{12}.occ_2 = Z))$;
  - $((x_{12}.occ_2 = Z) \Rightarrow (x_{12}.occ_1 = Z))$.

4.3 Invariant calculus

The goal is to verify that $(\mathsf{G}_\circ\, \alpha) \in Ax^{\vdash c}$, assuming that α is a slot constraint. To this end we need the following four metatheorems. The first one is the OSL version of the well known "invariant rule". The other metatheorems allow us to combine local "axioms" stating the effects of actions on the slots values. For the sake of readability we use a "vectorial" notation for the parameters and slots that should be self-explanatory.

Theorem 4.1 *(Invariant rule: INV)* For each $\alpha \in SC^{c_1 \otimes \ldots \otimes c_m}$:

- $(\mathsf{G}_0\, \alpha) \in Ax^{\vdash c_1 \otimes \ldots \otimes c_m}$ provided that:
  - $(\star \Rightarrow \alpha) \in Ax^{\vdash c_1 \otimes \ldots \otimes c_m}$;
  - for each tuple $z_1, \ldots, z_m$ of action symbols, $((\bigwedge_{i=1}^{m}(occ_i = z_i(\vec{y_i})) \wedge \alpha) \Rightarrow (\mathsf{X}\, \alpha)) \in Ax^{\vdash c_1 \otimes \ldots \otimes c_m}$.

That is, in order to verify that a slot constraint is invariant we have to check that it is established initially and that it is preserved by any combination of actions.

Theorem 4.2 *(Combination rule 1: CMB1)* For each $\beta \in SC^{c_1 \otimes \ldots \otimes c_m}$:

- $x.((\bigwedge_{i=1}^{m}(occ_i = z_i(\vec{y_i})) \wedge \beta^{\pi_1(\vec{A_1}), \ldots, \pi_m(\vec{A_m})}_{\pi_{1T}(\vec{t_1}), \ldots, \pi_{mT}(\vec{t_m})}) \Rightarrow (\mathsf{X}\, \beta)) \in Ax^{\vdash c_1 \otimes \ldots \otimes c_m}$ provided that:
  - for each $i = 1, \ldots, m$ and $\alpha \in SC^{c_i}$, $x_i.(((occ = z_i(\vec{y_i})) \wedge \alpha^{\vec{A_i}}_{\vec{t_i}}) \Rightarrow (\mathsf{X}\, \alpha)) \in Ax^{\vdash c_i}$.

Theorem 4.3 *(Combination rule 2: CMB2)* Let $c' \leq c$. For each action symbol z of c and $\beta \in SC^{c'}$:

- $x.(((occ = z(\vec{y})) \wedge \beta^{\vec{A}, \vec{A'}}_{\vec{t}, \vec{t'}}) \Rightarrow (\mathsf{X}\, \beta)) \in Ax^{\vdash c'}$ provided that:
  - for each $\alpha \in SC^{c}$, $x.(((occ = z(\vec{y})) \wedge \alpha^{\vec{A}}_{\vec{t}}) \Rightarrow (\mathsf{X}\, \alpha)) \in Ax^{\vdash c}$.
  - for each $\alpha' \in (SC^{c'} \setminus SC^{c})$, $x.(((occ = z(\vec{y})) \wedge \alpha'^{\vec{A'}}_{\vec{t'}}) \Rightarrow (\mathsf{X}\, \alpha')) \in Ax^{\vdash c'}$.

Theorem 4.4 *(No side effects rule: NSE)* Let $c' \leq c$. For each action symbol z of c' but not of c and $\alpha \in SC^{c}$:

- $x.(((occ = z(\vec{y})) \wedge \alpha) \Rightarrow (\mathsf{X}\, \alpha)) \in Ax^{\vdash c'}$.

This last rule depicts one of the consequences of the "monotonic" nature of OSL: though in a specialization we can add more slots and actions, the inherited slots may not be affected by the additional actions.

4.4 Verification example

We proceed now to illustrate the use of OSL (with the metatheorems above) for establishing an invariant property of the system specified in Subsection 4.2.

$$(\mathsf{G}_\circ(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3')) \in Ax^{\vdash c_1 \otimes c_3}?$$

Proof (fragment):

(A) $(\star \Rightarrow (x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3')) \in Ax^{\vdash c_1 \otimes c_3}?$

0.	$\star \in Ax^{\vdash c_1 \otimes c_3}$	Hyp
1.	$(\star \Rightarrow (x_1.A = 0)) \in Ax^{\vdash c_1}$	AX
2.	$(\star \Rightarrow (x_{13}.A_1 = 0)) \in Ax^{\vdash c_1 \otimes c_3}$	$INH : 1$
3.	$(x_{13}.A_1 = 0) \in Ax^{\vdash c_1 \otimes c_3}$	$MP : 0, 2$
4.	$(\star \Rightarrow (x_2.A = 0)) \in Ax^{\vdash c_2}$	AX
5.	$(\star \Rightarrow (x_2.A = 0)) \in Ax^{\vdash c_3}$	$INH : 4$
6.	$(\star \Rightarrow (x_{13}.A_3 = 0)) \in Ax^{\vdash c_1 \otimes c_3}$	$INH : 5$
7.	$(x_{13}.A_3 = 0) \in Ax^{\vdash c_1 \otimes c_3}$	$MP : 0, 6$
8.	$(\star \Rightarrow (x_3.A' = 0)) \in Ax^{\vdash c_3}$	AX
9.	$(\star \Rightarrow (x_{13}.A_3' = 0)) \in Ax^{\vdash c_1 \otimes c_3}$	$INH : 8$
10.	$(x_{13}.A_3' = 0) \in Ax^{\vdash c_1 \otimes c_3}$	$MP : 0, 9$
11.	$(x_{13}.A_1 + x_{13}.A_3 = 0 + 0) \in Ax^{\vdash c_1 \otimes c_3}$	$FOL : 3, 7$
12.	$(0 + 0 = 0) \in Ax^{\vdash c_1 \otimes c_3}$	DT
13.	$(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3') \in Ax^{\vdash c_1 \otimes c_3}$	$FOL : 11, 12, 10$
14.	$(\star \Rightarrow (x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3')) \in Ax^{\vdash c_1 \otimes c_3}$	$DED : 0, 13$

(B) $(((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = Z) \wedge (x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3'))\Rightarrow$
$(\mathsf{X}(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3'))) \in Ax^{\vdash c_1 \otimes c_3}?$

0.	$((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = Z) \wedge$ $(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3')) \in Ax^{\vdash c_1 \otimes c_3}$	Hyp
1.	$(((x_1.occ = Z) \wedge \alpha^{x_1.A}_{1_{x_1.A+1}}) \Rightarrow (\mathsf{X}\,\alpha_1)) \in Ax^{\vdash c_1}$	AX
2.	$(((x_2.occ = Z) \wedge \alpha^{x_2.A}_{2_{x_2.A-1}}) \Rightarrow (\mathsf{X}\,\alpha_1)) \in Ax^{\vdash c_2}$	AX
3.	$(((x_3.occ = Z) \wedge \alpha_3) \Rightarrow (\mathsf{X}\,\alpha_3)) \in Ax^{\vdash c_3}$	AX
4.	$(((x_3.occ = Z) \wedge \alpha^{x_3.A-1}_{x_3.A}) \Rightarrow (\mathsf{X}\,\alpha)) \in Ax^{\vdash c_3}$	$CMB2 : 2, 3$
5.	$(((x_3.occ = Z) \wedge \alpha) \Rightarrow (\mathsf{X}\,\alpha)) \in Ax^{\vdash c_3}$	AX
6.	$((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = Z) \wedge$ $(x_{13}.A_1 + 1 + x_{13}.A_3 - 1 = x_{13}.A_3'))\Rightarrow$ $(\mathsf{X}(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3'))) \in Ax^{\vdash c_1 \otimes c_3}$	$CMB1 : 1, 4, 5$
7.	$(x + x' = x + 1 + x' - 1) \in Ax^{\vdash c_1 \otimes c_3}$	DT
8.	$((x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3')\Rightarrow$ $(x_{13}.A_1 + 1 + x_{13}.A_3 - 1 = x_{13}.A_3')) \in Ax^{\vdash c_1 \otimes c_3}$	$FOL : 7$
9.	$((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = Z) \wedge (x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3'))\Rightarrow$ $((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = Z) \wedge$ $(x_{13}.A_1 + 1 + x_{13}.A_3 - 1 = x_{13}.A_3'))) \in Ax^{\vdash c_1 \otimes c_3}$	$FOL : 8$
10.	$((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = Z) \wedge$ $(x_{13}.A_1 + 1 + x_{13}.A_3 - 1 = x_{13}.A_3')) \in Ax^{\vdash c_1 \otimes c_3}$	$MP : 0, 9$
11.	$(\mathsf{X}(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3')) \in Ax^{\vdash c_1 \otimes c_3}$	$MP : 10, 6$
12.	$(((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = Z) \wedge$ $(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3'))\Rightarrow$ $(\mathsf{X}(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A_3'))) \in Ax^{\vdash c_1 \otimes c_3}$	$DED : 0, 11$

(C) $(((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = \bot) \wedge (x_{13}.A_1 + x_{13}.A_3 = x_{13}.A'_3)) \Rightarrow (\mathsf{X}(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A'_3))) \in Ax^{\vdash c_1 \otimes c_3}$?

0. $((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = \bot) \wedge (x_{13}.A_1 + x_{13}.A_3 = x_{13}.A'_3)) \in Ax^{\vdash c_1 \otimes c_3}$ — Hyp
1. $(x_{13}.occ_3 = \bot) \in Ax^{\vdash c_1 \otimes c_3}$ — $FOL : 0$
2. $(x_{13}.occ_1 = Z) \in Ax^{\vdash c_1 \otimes c_3}$ — $FOL : 0$
3. $((x_{12}.occ_1 = Z) \Rightarrow (x_{12}.occ_2 = Z)) \in Ax^{\vdash c_1 \otimes c_2}$ — AX
4. $((x_{13}.occ_1 = Z) \Rightarrow (x_{13}.occ_3 = Z)) \in Ax^{\vdash c_1 \otimes c_3}$ — $INH : 3$
5. $(x_{13}.occ_3 = Z) \in Ax^{\vdash c_1 \otimes c_3}$ — $MP : 2, 4$
6. $x_{13}.(\bot = Z) \in Ax^{\vdash c_1 \otimes c_3}$ — $FOL : 1, 5$
7. $x_{13}.(\neg(\bot = Z)) \in Ax^{\vdash c_1 \otimes c_3}$ — ACT
8. $(\neg((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = \bot) \wedge (x_{13}.A_1 + x_{13}.A_3 = x_{13}.A'_3))) \in Ax^{\vdash c_1 \otimes c_3}$ — $ABS : 0, 6, 7$
9. $(((x_{13}.occ_1 = Z) \wedge (x_{13}.occ_3 = \bot) \wedge (x_{13}.A_1 + x_{13}.A_3 = x_{13}.A'_3)) \Rightarrow (\mathsf{X}(x_{13}.A_1 + x_{13}.A_3 = x_{13}.A'_3))) \in Ax^{\vdash c_1 \otimes c_3}$ — $FOL : 8$

The rest of the proof is left to the interested reader: namely, the other action symbols pairs to be considered for the application of the invariant rule.

4.5 Automatic theorem proving

We conclude the Section by examining very briefly the problem of automatic theorem proving. Roughly we can subdivide the problem into three almost independent domains: temporal predicate reasoning; equational reasoning; and numerical reasoning. According to our experience it is worthwhile to adopt a tableaux confutation approach to temporal predicate reasoning (see for instance [GSGA93]). Then, when doing a confutation we are lead to a set of equalities and inequalities. Their joint satisfiability can be checked using rewriting techniques on the abstract data types at hand [DJ90] and simplex techniques on the real numbers [NO79]. The cooperation between these techniques can be achieved as proposed in [NO79].

One should stress that the state-of-the-art seems to be still far away from an inference machine that can be used for the "complete" verification of a "real system" specification. As we pointed out before, the most we can envisage in practice at this stage is to provide automatic support to "systems querying" in the small.

5 Concluding remarks

We described very briefly the requirements of object-orientation on the temporal approach to specification/verification of concurrent, reactive systems. We outlined a logic fulfilling those requirements except concerning non-monotonic features. The proposed logic allows the uniform treatment of both local and global properties of systems with concurrent, interacting components organized in classes and taking into account (monotonic) specialization. We provided both a semantics and a calculus (following an axiomatic, Hilbert style). This calculus includes rules for inheritance and reflection of theorems between classes.

We also examined the more practical aspects of the usage of such a logic both for specification and verification. To this end we provided a set of metatheorems for expediting the proof of invariants. A similar effort can and should be done for expediting the proof of liveness properties. We concluded by considering the need and availability of automatic theorem proving for systems querying.

At this stage we see five main research lines in the area of temporal, object-oriented specification and verification of concurrent, interactive systems:

- reification: such a logic must be enriched towards supporting reification (implementation) – see [ES90, SGG+92, FM94];
- defaults: it should also be enriched towards supporting defaults and default reasoning – see [BL91, BLR93];
- abduction: theorem proving is not enough – we also want to be able to detect why some property does not hold and to suggest where to make the necessary changes in the specification – see [GS94];
- modularization: the object seems to be too small for a modularization unit – parameterized modules are needed in temporal specifications of objects – see [FM92, SRGS91];
- automation: formal specification will not come of age before automatic theorem provers (and abducters) are usable in practical, real problems.

Acknowledgments

The authors are grateful to their colleagues in the IS-CORE and OBLOG projects for many rewarding discussions on temporal specification of objects, and specially to Jaime Ramos for feedback on early versions of the paper.

References

[AHS90] J. Adámek, H. Herrlich, and G. Strecker. *Abstract and concrete categories*. John-Wiley, 1990.

[BKP84] H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal logic specifications. In *Proceedings of the 16th ACM Symposium on Theory of Computing*, pages 51–63, 1984.

[BKP85] H. Barringer, R. Kuiper, and A. Pnueli. A compositional temporal approach to csp-like language. In E. Neuhold and G. Chroust, editors, *Formal Models of Programming*, pages 207–227. North-Holland, 1985.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Proceedings of the 13th ACM Symposium on Principles of Programming Languages*, pages 173–183, 1986.

[BL91] S. Brass and U. Lipeck. Semantics of inheritance in logical object specifications. In *Proceedings of the 2nd International Conference on Deductive and Object-oriented Databases*, 1991.

[BLR93] S. Brass, U. Lipeck, and P. Resende. Specification of object behaviour with defaults. In U. Lipeck and G. Koschorreck, editors, *Proceedings of the International Workshop on Information Systems – Correctness and Reusability*, pages 155–177. University of Hannover, 1993.

[CS88] J. Carmo and A. Sernadas. A temporal logic framework for a layered approach to systems specification and verification. In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal Aspects in Information Systems*, pages 31–46. North-Holland, 1988.

[CS91] J. Carmo and A. Sernadas. Formal techniques for systems specification and verification. *Information Systems*, 16:245–272, 1991.

[DJ90] N. Dershowitz and J. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science B: Formal Models and Semantics*, pages 243–320. Elsevier, 1990.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification I.* Springer-Verlag, 1985.

[Eme90] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science B: Formal Models and Semantics*, pages 995–1072. Elsevier, 1990.

[ES90] H.-D. Ehrich and A. Sernadas. Algebraic implementation of objects over objects. In J. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, pages 239–266. Springer-Verlag, 1990.

[ES91] H.-D. Ehrich and A. Sernadas. Object concepts and constructions. In G. Saake and A. Sernadas, editors, *Proceedings of the IS-CORE Workshop 91*, pages 1–24. Technical University of Braunschweig, 1991.

[ESD93] ESDI, Av. Alvares Cabral, 41, 7., 1200 Lisboa. *OBLOG-CASE V1.0 User's Guide*, 1993. Supplied with the OBLOG-CASE V1.0 product kit.

[FCSM93] J. Fiadeiro, J. Costa, A. Sernadas, and T. Maibaum. Process semantics of temporal logic specification. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification: 8th Workshop on Specification of Abstract Data Types – Selected papers*, pages 236–253. Springer-Verlag, 1993.

[FM92] J. Fiadeiro and T. Maibaum. Temporal theories as modularization units for concurrent system specification. *Formal Aspects of Computing*, 4:239–272, 1992.

[FM94] J. Fiadeiro and T. Maibaum. Sometimes tomorrow is sometime: Action refinement in a temporal logic of objects. In *Proceedings of the 1st International Conference on Temporal Logic*. Springer-Verlag, 1994. In print.

[FS88] J. Fiadeiro and A. Sernadas. Specification and verification of database dynamics. *Acta Informatica*, 25:625–661, 1988.

[FSMS92] J. Fiadeiro, C. Sernadas, T. Maibaum, and A. Sernadas. Describing and structuring objects for conceptual schema development. In P. Loucopoulos and R. Zicari, editors, *Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development*, pages 117–138. John Wiley, 1992.

[GB84] J. Goguen and R. Burstall. Introducing institutions. In E. Clarke and D. Kozen, editors, *Proceedings of the Logics of Programming Workshop*, pages 221–256. Springer-Verlag, 1984.

[GB92] J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[GS94] P. Gouveia and C. Sernadas. Abduction in temporal object specification. Research report, Section of Computer Science, Department of Mathematics, Instituto Superior Técnico, 1096 Lisboa, Portugal, 1994. To be presented at IS-CORE Workshop 94.

[GSGA93] P. Gouveia, C. Sernadas, J. Gomes, and J. Apolinário. Tableaux for reasoning about objects. In *Theorem Proving with Analytic Tableaux and Related Methods*, pages 113–125, 1993. Max Plank Institut fur Informatik.

[Hai82] B. Hailpern. *Verifying Concurrent Processes Using Temporal Logic*. Springer-Verlag, 1982.

[HO83] B. Hailpern and S. Owicki. Modular verification of computer communication protocols. *IEEE Transactions on Communications*, 1:56–68, 1983.

[Jon87] B. Jonsson. Modular verification of asynchronous networks. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, pages 152–166, 1987.

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

[Jun93] R. Jungclaus. *Modeling of Dynamic Object Systems: A Logic-based Approach*. Vieweg, 1993.

[Man82] Z. Manna. Verification of sequential programs: Temporal axiomatization. In M. Broy and G. Schmidt, editors, *Theoretical Foundations of Programming Methodology*, pages 53–102. D. Reidel, 1982.

[MCS82] J. Misra, K. Chandy, and T. Smith. Proving safety and liveness of communicating processes with examples. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 157–164, 1982.

[MP91] Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83:97–130, 1991.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[NDOG86] V. Nguyen, A. Demers, S. Owicki, and D. Gries. A modal and temporal proof system for networks of processes. *Distributed Computing*, 1:7–25, 1986.

[NO79] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1:245–257, 1979.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[SE91] A. Sernadas and H.-D. Ehrich. What is an object after all? In R. Meersman, W. Kent, and S. Khosla, editors, *Object-oriented Databases: Analysis, Design and Construction*, pages 39–69. North-Holland, 1991.

[Ser80] A. Sernadas. Temporal aspects of logical procedure definition. *Information Systems*, 5:167–187, 1980.

[SFSE89a] A. Sernadas, J. Fiadeiro, C. Sernadas, and H.-D. Ehrich. Abstract object types: A temporal perspective. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, pages 324–350. Springer-Verlag, 1989.

[SFSE89b] A. Sernadas, J. Fiadeiro, C. Sernadas, and H.-D. Ehrich. The basic building blocks of information systems. In E. Falkenberg and P. Lindgreen, editors, *Information Systems Concepts: An In-depth Analysis*, pages 225–246. North-Holland, 1989.

[SGG+92] C. Sernadas, P. Gouveia, J. Gouveia, A. Sernadas, and P. Resende. The reification dimension in object-oriented data base design. In D. Harper and M. Norrie, editors, *Specification of Data Base Systems*, pages 275–299. Springer-Verlag, 1992.

[SGS92] C. Sernadas, P. Gouveia, and A. Sernadas. Oblog: Object-oriented, logic-based conceptual modeling. Research report, Section of Computer Science, Department of Mathematics, Instituto Superior Técnico, 1096 Lisboa, Portugal, 1992.

[SR94] A. Sernadas and J. Ramos. *The GNOME language: Syntax, semantics and calculus*. Instituto Superior Técnico, 1096 Lisboa, Portugal, 1994. In Portuguese.

[SRGS91] C. Sernadas, P. Resende, P. Gouveia, and A. Sernadas. In-the-large object-oriented design of information systems. In F. van Assche, B. Moulin, and C. Rolland, editors, *The Object-oriented Approach in Information Systems*, pages 209–232. North-Holland, 1991.

[SS93] A. Sernadas and C. Sernadas. Denotational semantics of object specification within an arbitrary temporal logic institution. Research report, Section of Computer Science, Department of Mathematics, Instituto Superior Técnico, 1096 Lisboa, Portugal, 1993. Presented at IS-CORE Workshop 93 - Submitted for publication.

[SSC92] A. Sernadas, C. Sernadas, and J. Costa. Object specification logic. Research report, Section of Computer Science, Department of Mathematics, Instituto Superior Técnico, 1096 Lisboa, Portugal, 1992. Revised 1993 - Submitted for publication.