I. Havel and V. Koubek, editors, Mathematical Foundations of Computer Science (MFCS’92), pages 200–208. Springer Verlag, 1992

Object Interaction

J.F.Costa[1], A.Sernadas[1], C.Sernadas[1], and H.-D.Ehrich[2]

[1]INESC, Apartado 10105, 1017 Lisboa Codex, PORTUGAL
[2]Tech. Univ. Braunschweig, Postfach 3329, 3300 Braunschweig, GERMANY

Abstract. Objects are presented as processes endowed with state-dependent slots. Active objects have non-quiescent states. A morphism between two objects expresses that the source object is part-of the target object, in such a way that the possible observations of the part, considered in isolation, are constrained within the whole. Our semantic domain for object-oriented concepts includes a cocomplete category of objects where several kinds of dynamic interaction like event and memory sharing are reflected by colimits.

1 Introduction

According to the object-oriented view [10]: (1) an object has an internal state that can be changed by certain actions (events) and observed through slots (attributes); and (2) an object displays behaviour (at any time some actions are possible and some are not). Therefore, an object is, basically, a process endowed with trace-dependent slots. In [4,5] we find contributions to a semantic domain for objects dealing with aggregation of interacting objects as well as object classes and inheritance.

A semantic domain for objects must start with the adoption of a semantic domain for processes. In this paper we will work with a trace-based model of processes, called herein the *quiescence model*. But traces alone do not tell everything about an object. The behaviour of a stack is described satisfactorily if we describe what we can observe, depending on what happened to the stack before. The observations we can make about a stack are the values of its top entry. The top value is determined by the finite sequence of events that happened so far. Formally, if we have a set S of slots and a set obs(S) of observations over S, the observable behaviour of an object is modelled by a slot observation mapping $\alpha:E^*\rightarrow obs(S)$ associating observations with traces. Summing up, our process-oriented model for objects consists of a set E of events, a set S of slots, a set $\Lambda \subseteq E^*$ of quiescent traces, and a slot observation map $\alpha:E^*\rightarrow obs(S)$. This model represents adequately the causal relationship between event traces and attribute observations.

The main ideas on categorial fundamentals of object theory could be found in [4,5,7,8]: (a) morphisms express inheritance, ie, an object P inherits the behaviour of an object Q iff there is an object morphism $f:Q\rightarrow P$; (b) communities of objects are seen as diagrams in a suitable category; (c) joint behaviour is colimit. In this paper we take the program originally set-up in [4] to complete satisfaction within a new categorial framework.

In section 2 we introduce the concept of object on top of the concept of process starting with some motivation of the adopted concept of process. Sections 3 and 4 are dedicated to the notions of morphism between processes and morphism between objects, respectively. In section 5 we deal with parallel composition of interacting processes. In section 6 we consider object interaction with emphasis in the special case of functional objects, ie, objects where the slot observation map returns a singleton set for each slot at any state.

We make moderate use of the theory of categories. The reader may find all the necessary category-theoretic notions in the book by Jiří Adámek *et al* [1].

2 Processes and Objects

The adopted model of processes is obtained from the traditional *trace model* (PS model, from *prefix-closed set*) by dropping the prefix-closure condition. Let us consider a chocolate vending MACHINE=μx.coin•!choc•x. The traditional semantics of such a process term is the pair <E,Λ> with E={coin,choc} and Λ is a subset of E* denoted by the regular expression (coin choc)*(coin + ε). But, this semantics provides no information about the process liveness (indicated by the !), namely, that a chocolate must always be delivered by the machine once a coin is inserted.

The idea of quiescence, developed in [9] in the context of a logic to reason about specifications, and developed in [3] within an algebraic framework, provides a solution for the problem of representing state-dependent liveness in a process. We say that a process (or a community of processes) is in a *quiescent state* iff it is not able to produce any action by itself unless triggered by the outside. Returning to our example, the quiescent traces of the MACHINE are those that are defined with an equal number of coin and choc events, ie, the set of traces is given by the regular expression (coin choc)*. If we are willing to include recursive definitions, we have to consider infinite traces, because an infinite trace is quiescent. For instance, a CLOCK = μx.!tick•x that produces a nonterminating tick signal cannot be in a quiescent state, but could be represented by an infinite trace tick$^{\omega}$. However only short term commitments will be considered in this paper (see [2] for completeness).

A process is represented by the set of its quiescent traces. And a trace of the community is quiescent iff it can be projected onto a quiescent trace of every of its components. Indeed, the whole community is in a quiescent state iff all its components are in a quiescent state.

According to the model just described, which we denote by QS (from *quiescent set*), given a finite set of events (the *alphabet*) E, any pair <E,Λ>, where Λ is a set of traces over E, is a *process*. For instance, the pair <{coin,choc},(coin choc)*> is a QS-process, but not a PS-process, since it is not prefix-closed. Clearly, every PS-process is also a QS-process. Such a process is considered to be without liveness or passive, since it is always quiescent. Those QS-processes that are not PS-processes display some liveness. Indeed, we say that the process <{coin,choc},(coin choc)*> above is active in the sense that after coin, for instance, it is willing to make choc happen.

Definition 2.1. A *process* $P=<E,\Lambda>$ is a pair consisting of a set E and a set $\Lambda \subseteq E^*$.

Given a process $P=<E,\Lambda>$ we refer to E as alph(P) — the alphabet of P — and we refer to Λ as lang(P) — the quiescent language of P.

Definition 2.2. A *slot alphabet* is a pair $A=<S,cod>$ where S is a set and cod is a total map from S into a given class D of carrier sets.

The idea is that, for each slot a in S, cod returns its codomain, ie, an element of D. In this way cod(a) is the set of the possible values that a can have. Given a slot alphabet $A=<S,cod>$ we define (i) slt(A)=S and (ii) $\partial(A)=cod$.

Definition 2.3. An slot morphism $f:A_1 \rightarrow A_2$ is a total map $f_S:slt(A_1) \rightarrow slt(A_2)$ such that $cod_1(a)=cod_2(f_S(a))$, for every $a \in slt(A_1)$.

Proposition 2.4. The slot alphabets and slot morphisms constitute a cocomplete category Slt (called the category of slots).

Proof: Colimits are reflected by the forgetful functor υ:Slt$\rightarrow$Set. □

Definition 2.5. An observation over A is a set of slot-value pairs $\{(a_1:d_1),...,(a_n:d_n)\}$ where $a_i \in slt(A)$ and $d_i \in \partial(A)(a_i)$ for $1 \leq i \leq n$. The class of observations over A is denoted by obs(A). An observation structure over a given process $<E,\Lambda>$ is a pair $<A,\alpha>$ consisting of a slot alphabet A and a total slot observation map $\alpha:E^* \rightarrow obs(A)$.

An observation provides values for some of the slots. Given an observation structure $<A,\alpha>$ over some process $<E,\Lambda>$ we define $alph(<A,\alpha>)=A$ and $map(<A,\alpha>)=\alpha$.

Definition 2.6. An object $ob=<<E,\Lambda>,<A,\alpha>>$ is a pair consisting of a process $<E,\Lambda>$ and an observation structure $<A,\alpha>$ over $<E,\Lambda>$.

For any slot $a \in slt(A)$, we can also introduce a map $\alpha(a)$ that returns a set of values in the codomain of a for each trace, ie, $\alpha(a)=\lambda u.\{d \in \partial(A)(a): (a:d) \in \alpha(u)\}$.

Given a sequence u of events, for each $a \in slt(A)$, if $\alpha(a)(u)$ is empty we say that the slot a is undefined after the sequence u. It may also happen that a slot is given more than one value. Thus, wrt slot values this model allows for nondeterminism.

Given an object $ob=<<E,\Lambda>,<A,\alpha>>$ let $bhv(ob)=<E,\Lambda>$ be the behaviour of ob and let $ost(ob)= <A,\alpha>$ be the observation structure of ob.

3 The Category of Processes

The category Pfn of *partial functions* is defined as Set but with partial functions as morphisms. It has sets, as before, as objects. Identity functions are the identity morphisms, isomorphisms are (total) bijections and composition of partial functions is taken as the composition of morphisms in Pfn. Therefore, Pfn is a subcategory of Set. In particular we have that $\emptyset$ is a zero object of Pfn (both initial and terminal), and $X+_{Set}(X\times_{Set}Y)+_{Set}Y$ is, up to an isomorphism, the categorial product of sets X and Y with projections that are partial maps. We can extend the existence of products to the existence of all small limits and so Pfn is small complete.

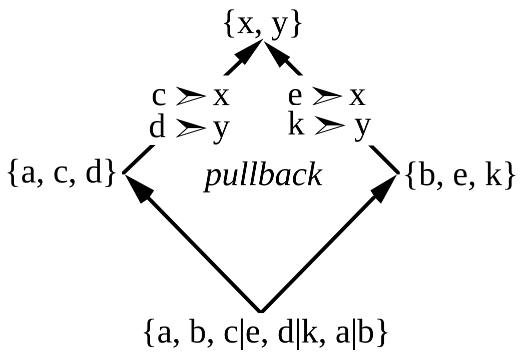

Fig. 1.1. Limits within Pfn.

Extension of h: $A \rightarrow B$ to A^* is the function h^*: $A^* \rightarrow B^*$ inductively defined as follows: (i) $h^*(\varepsilon)=\varepsilon$ and (ii) $h^*(as)$ = if $h(a)\downarrow$ then $h(a)h^*(s)$ else $h^*(s)$. By $h(a)\downarrow$ we mean that h is defined at a. Otherwise we say that h is undefined and we write $h(a)\uparrow$. The function h^* is in turn extended to sets (and also denoted by h^*) by taking the set of images. Let f be a partial function from a set A to a set B and let g be a partial function from a set B to a set C. Then it is straightforward to prove that $(g \circ f)^* = g^* \circ f^*$. Adapting from [11] we arrive to the following definition:

Definition 3.1. A *process morphism* h: $<E_1,\Lambda_1> \rightarrow <E_2,\Lambda_2>$ is a partial event map $h_E{:}E_1 \rightarrow E_2$ such that $h_E^*(\Lambda_1) \subseteq \Lambda_2$.

Now we establish the category of processes whose elements are processes and whose morphisms are process morphisms. Recall first that a *concrete category* over a base category B is a pair <C,U>, where C is a category and $U{:}C \rightarrow B$ is a faithful functor.

Proposition 3.2. The processes and process morphisms constitute a concrete category Proc over Pfn.

Proof: Let $f{:}P_1 \rightarrow P_2$ and $g{:}P_2 \rightarrow P_3$ be process morphisms and let $h_E=g_E \circ f_E$. We know that $f_E^*(\Lambda_1) \subseteq \Lambda_2$ and $g_E^*(\Lambda_2) \subseteq \Lambda_3$. Thus $g_E^*(f_E^*(\Lambda_1)) \subseteq \Lambda_3$ and it follows that $(g_E \circ f_E)^*(\Lambda_1) \subseteq \Lambda_3$. Thus, process morphisms compose and, moreover, this composition is associative. It is also trivial to recognize the identity morphisms. The faithful functor is the forgetful functor $U{:}\mathrm{Proc} \rightarrow \mathrm{Pfn}$ that maps each process onto its alphabet. □

A composite process is formed by putting processes together so that each of the latter is a *part-of* the composite process. Processes may have common parts, and processes with common parts may be composed again. In this way, a rather involved structure may arise. A useful categorial concept for studying this is that of a limit. We hope to obtain the joint behaviour of two independent processes through the product construction and the joint behaviour of two processes sharing an event through a pullback construction.

A *source* in a category C is a pair $<X,\{f_i\}_{i \in I}>$, usually denoted by $(f_i{:}X \rightarrow X_i)_{i \in I}$, consisting of an object X and a family of morphisms $f_i{:}X \rightarrow X_i$ with domain X, indexed by some class I. A source $(f_i{:}X \rightarrow X_i)_{i \in I}$ in a concrete category <C,U> over B is called *initial* provided that a B-morphism $f{:}UY \rightarrow UX$ is a C-morphism whenever each composite

$f_i \circ f{:}UY \rightarrow UX_i$ is a C-morphism. Let <C,U> be a concrete category over B. A limit L of a diagram $D{:}I \rightarrow C$ is called a *concrete limit* of D in <C,U> provided that it is preserved by U. A concrete limit can be constructed in two steps: first, form the limit of the underlying diagram in the base category, and, second, provide an initial lift of this underlying limit. In other words, if <C,U> is a concrete category over B and $D{:}I \rightarrow C$ is a diagram then $L=(f_i{:}L \rightarrow D_i)_{i \in |I|}$ is a concrete limit in <C,U> iff U(L) is a limit of $U \circ D$ and L is an initial source in <C,U>.

The concept dual to that of *source* is called *sink*. Dual concepts of *concrete colimit* and *final sink* are left to the reader. A concrete colimit can be also constructed in two steps: first, form the colimit of the underlying diagram in the base category, and, second, provide a final lift of this underlying colimit.

Consider two processes P and Q and assume that they share no events. That is, the alphabet of their parallel composition P||Q should correspond to the set-theoretic disjoint union of the two alphabets plus the Set categorial product of these alphabets, ie,

$$\mathrm{alph}(P||Q)=\mathrm{alph}(P)\times_{Pfn}\mathrm{alph}(Q)=\mathrm{alph}(P)+_{Set}(\mathrm{alph}(P)\times_{Set}\mathrm{alph}(Q))+_{Set}\mathrm{alph}(Q)$$

and the quiescent traces of the product are those quiescent traces, over the product of the two alphabets, that are projected onto the quiescent traces of the process components, ie,

$$\mathrm{lang}(P||Q)=\{s\in \mathrm{alph}(P||Q)^*\text{: } s{\downarrow}\mathrm{alph}(P)\in \mathrm{lang}(P) \text{ and } s{\downarrow}\mathrm{alph}(Q)\in \mathrm{lang}(Q)\}$$

This equality reads as follows: a trace of the product is quiescent iff it can be projected onto a quiescent trace of its components. Thus, the product is in a quiescent state iff all its components are in a quiescent state. The elements of $\mathrm{alph}(P)\times_{Set}\mathrm{alph}(Q)$ are unordered pairs of elements of alph(P) and alph(Q). Such pairs stand for concurrent executions of P and Q in a distributed computation. That is, the process P||Q would be able, at each instant, to be involved either in an event of P (while Q remains idle), or in an event of Q (P remaining idle), or in both an event of P and one of Q. We shall denote the elements of $\mathrm{alph}(P)\times_{Set}\mathrm{alph}(Q)$ through $a|b$. The parallel composition of two processes having disjoint alphabets (sharing no events), P||Q, allows P and Q to proceed independently and, additionally, if P can perform an event a and Q can perform an event b then P||Q can perform the joint event $a|b$. The next proposition show how to build a limit in Proc by lifting from the corresponding source in Pfn.

Proposition 3.3. Every source $(f_{i_E}{:}E \rightarrow U{<}E_i,\Lambda_i{>})_{i\in I}$, with I a finite set, in Pfn can be lifted to an initial source in Proc.

Proof: With $\Lambda=\{s\in E^*\text{: for every } i\in I,\ f_{i_E}{}^*(s)\in \Lambda_i\}$, we trivially recognize that the pair $<E,\Lambda>$ is a process and that $(f_i{:}{<}E,\Lambda{>}\rightarrow{<}E_i,\Lambda_i{>})_{i\in I}$ is a source in Proc. We show that this source is initial. Let $<E',\Lambda'>$ be an arbitrary process and $g_E{:}E' \rightarrow E$ a partial function. We have to prove that if $f_i \circ g$ is a process morphism then $g{:}{<}E',\Lambda'{>}\rightarrow{<}E,\Lambda{>}$ is a process morphism. Since, for every $i\in I$, $f_{i_E}{}^*(g_E{}^*(\Lambda'))\subseteq \Lambda_i$ it is straightforward to establish that $g_E{}^*(\Lambda')\subseteq \Lambda$. □

This constructive proof explains why a process community $({<}E_i,\Lambda_i{>})_{i\in I}$ is represented by the set of quiescent traces $\Lambda=\{s\in E^*\text{: for every } i\in I,\ f_{i_E}{}^*(s)\subseteq \Lambda_i\}$, if we take E to be the resulting alphabet of the whole community.

Proposition 3.4. Proc is finitely complete.

Proof: This is a consequence of U being a faithful functor that lifts limits of the complete category Pfn. □

4 The Category of Objects

We assume that an object is a collection of slots plus a collection of events that modify the values of the slots. Therefore we accept the following *atomic object hypothesis*: the smallest portion of the community that still retains the object's characteristics is an event or a *slot plus the set of all events that participate in its alteration*. Furthermore, *if an object is part of a whole, then in isolation the former has possibly more observations than within the whole*. Accordingly, we are taken to the following notion of object morphism:

Definition 4.1. A morphism h from an object $<<E_1,\Lambda_1>,<A_1,\alpha_1>>$ to an object $<<E_2,\Lambda_2>,<A_2,\alpha_2>>$ is a pair $<h_P,h_A>$ where $h_P:<E_2,\Lambda_2>\rightarrow<E_1,\Lambda_1>$ is a process morphism and $h_A:A_1\rightarrow A_2$ is a slot morphism such that the following *observation condition* holds: for every $u\in E_2^*$ and for every $a\in slt(A_1)$, $\alpha_1(a)(h_E{}^*(u))\subseteq\alpha_2(h_S(a))(u)$.

Let $h_S(\alpha(u))$ denote $\{(h_S(a_1):d_1),...,(h_S(a_n):d_n)\}$ whenever $\alpha(u)$ is $\{(a:d_1),...,(a_n:d_n)\}$. Then we can rewrite the observation condition as $h_S\alpha_1h_E{}^*\subseteq\alpha_2$.

Proposition 4.2. Objects and object morphisms constitute a concrete category Ob over $Proc^{op}\times Slt$.

Proof: Object morphisms compose and this composition is associative. Moreover, it is trivial to recognize the identity morphisms. The faithful functor is the forgetful functor V into $Proc^{op}\times Slt$ such that V(ob)=<bhv(ob),alph(ost(ob))> on objects and $V(f:ob_1\rightarrow ob_2)=<f_P,f_A>$ on morphisms. □

Now we are interested in the colimits of Ob since they correspond to limits of the underlying processes. As we shall see in sections 5 and 6, colimits correspond to the aggregation of (possibly) interacting objects.

Proposition 4.3. The category Ob has an initial object.

Proof: Take $\mathbf{I}=<<\emptyset,\{\varepsilon\}>,<\emptyset,\lambda x.\emptyset>>$. For every Ob-object $<<E,\Lambda>,<A,\alpha>>$ the completely undefined map $\bot_E:E\rightarrow\emptyset$ induces a process morphism $\bot:<E,\Lambda>\rightarrow<\emptyset,\{\varepsilon\}>$ because $\bot_E{}^*(\Lambda)\subseteq\{\varepsilon\}$. Taking $\emptyset_S:\emptyset\rightarrow slt(A)$ we see that the observation condition is trivially satisfied. Thus $\bot_E$ and $\emptyset_S$ induce a unique object morphism. □

Proposition 4.4. Every sink $(<f_{iP},f_{iS}>:V<<E_i,\Lambda_i>,<A_i,\alpha_i>>\rightarrow<<E,\Lambda>,A>)_{i\in I}$, with I a finite set, can be lifted to a final sink in Ob.

Proof: Define the object $<<E,\Lambda>,<A,\alpha>>$, with $\alpha=\cup_{i\in I}f_{iS}\alpha_if_{iE}{}^*$. We recognize that $(f_i:<<E_i,\Lambda_i>,<A_i,\alpha_i>>\rightarrow<<E,\Lambda>,<A,\alpha>>)_{i\in I}$ is a sink in Ob. We show that this sink is final. Let us consider an arbitrary object $<<E',\Lambda'>,<A',\alpha'>>$ and a morphism in $Proc^{op}\times Slt$, $<g_P,g_S>:<<E,\Lambda>,A>\rightarrow<<E',\Lambda'>,A'>$. If $g\circ f_i$ is a object morphism then, for every $i\in I$, $g_Sf_{iS}\alpha_if_{iE}{}^*g_E{}^*\subseteq\alpha'$. Then we have $g_S\alpha g_E{}^*\subseteq\alpha'$. It follows that, for every a in slt(A), $\alpha(a)g_E{}^*\subseteq\alpha'g_S(a)$. □

Proposition 4.5. Ob is finitely cocomplete.

Proof: This is a consequence of V being a faithful functor that lifts colimits of the cocomplete category $\mathrm{Proc}^{\mathrm{op}}\times\mathrm{Slt}$. □

5 Interaction and Process Communities

Let us start by considering the interconnection of processes. Pullbacks are a special kind of limit expressing the cooperative composition of two processes, say P and Q, sharing common events. We introduce a middle process that contains only the events that should be shared by the two processes and with the most liberal quiescent traces over the sharing alphabet. Let us denote this middle object by R. The following corollary is a characterization of pullbacks up to an isomorphism.

Proposition 5.1. The vertex of the pullback of $P\xrightarrow{f}R\xleftarrow{g}Q$ is, up to an isomorphism, the process $P\|_R Q$, with $\mathrm{alph}(P\|_R Q)=\{e\in \mathrm{alph}(P\|Q):f(\pi_P(e))=g(\pi_Q(e))\}$ and $\mathrm{lang}(P\|_R Q)=\{s\in \mathrm{alph}(P\|_R Q)^*:\pi_{P_E}{}^*(s)\in \mathrm{lang}(P)$ and $\pi_{Q_E}{}^*(s)\in \mathrm{lang}(Q)\}$. Processes P and Q share the events in $f_E(\mathrm{alph}(P))\cap g_E(\mathrm{alph}(Q))$.

Proof: Given the processes P and Q, $P\|_R Q$ is the domain of the initial lifting of the corresponding pullback source in Pfn. □

$$\begin{tikzcd}
& \langle\{a\},\{a\}^*\rangle & \\
\langle\{x, y\},\{\varepsilon,xy\}\rangle \arrow[ur, "x \succ a"] & \textit{pullback} & \langle\{z,w\},\{\varepsilon,zw\}\rangle \arrow[ul, "z \succ a"'] \\
& \langle\{a,y,w,y|w\},\{\varepsilon,ayw,awy,a(y|w)\}\rangle \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Fig. 5.1. A partial, cooperative parallel composition of two processes.

Consider fig. 5.1. Let us calculate the pullback process $P\|_{<\{a\},\{a\}^*>}Q$. Note that the up vertex is a process with one event in the alphabet (the event *a* we want to share) and containing all possible sequences of *a*'s. In this way we ensure that the edges of the peak are indeed process morphisms. We have concurrency and cooperation in the parallel composition of both processes. The alphabet of the composite process, synchronizing at the event *a*, will contain all the events of the disjoint parallel composition except the ones that are mapped onto one of the events of a synchronization pair but not to the other. Hence, in this case, we ruled out, for instance, *x*|*w* because it contained one of the events of the synchronization pair *x*|*y* but not the other.

6 Interaction and Functional Object Communities

Let us turn our attention now to the problem of object interconnection. Like processes, objects may be combined into larger objects that display the joint behaviour of the components. This operation is known as aggregation in the field of object-oriented programming. At the process level, it corresponds to parallel composition. The proposition 5.1 and the constructive proof of proposition 4.4, show how we can aggregate objects even in the presence of memory sharing.

Proposition 6.1. The vertex of the pushout of diagram $ob_1 \xleftarrow{f} ob \xrightarrow{g} ob_2$ is the object $ob_1 \|_{ob} ob_2 = <P_1 \|_P P_2, <A_1 \cup A_2, \alpha_1 \cup \alpha_2>>$, where $A_1 \cup A_2$ is the vertex of the pushout in Slt of diagram $A_1 \xleftarrow{f_S} A \xrightarrow{g_S} A_2$ and $\alpha_1 \cup \alpha_2 = inj_{1_S} \alpha_1 \pi_{1_E}{}^* \cup inj_{2_S} \alpha_2 \pi_{2_E}{}^*$, where inj_i, i=1,2, are the pushout injections in Slt and π_i, i=1,2, are the pullback projections in Proc.

If I is the initial Ob object then $ob_1 \|_{\mathsf{I}} ob_2$ stands for the aggregation (parallel composition) of the two independent objects ob_1 and ob_2. Interaction between objects means sharing events and slots, ie, event and memory sharing. Both event and memory sharing can be accomplished by a lifting from $Proc^{op} \times Slt$ into Ob. Actually, event and slot sharing are special cases of object sharing.

In practice we tend to work only with objects with functional observation maps. Therefore, although a special case of the general (relational) objects defined above, functional objects deserve some attention.

Definition 6.2. An object $<<E,\Lambda>,<A,\alpha>>$ is said to be *functional* whenever $\alpha(a)(u)$ is a singleton set, for every $a \in slt(A)$ and for every $u \in E^*$.

Proposition 6.3. If h is a morphism in category Ob from a functional source object $<<E_1,\Lambda_1>,<A_1,\alpha_1>>$ into a functional target object $<<E_2,\Lambda_2>,<A_2,\alpha_2>>$ then $\alpha_1(a)(h_E{}^*(u)) = \alpha_2(h_S(a))(u)$, for every $u \in E_2{}^*$ and for every $a \in slt(A_1)$.

Proof: Straightforward! Inclusion $\alpha_1(a)(h_E{}^*(u)) \subseteq \alpha_2(h_S(a))(u)$ degenerates into equality because $\alpha_1(a)(h_E{}^*(u))$ and $\alpha_2(h_S(a))(u)$ are singleton sets. □

Therefore, we can introduce the following subcategory of Ob with the functional observation condition above for morphisms between its elements:

Definition 6.4. The category of functional objects fOb is the full subcategory of Ob whose elements are functional.

Proposition 6.5. The category fOb is finitely cocomplete.

Proof: Let $<<E_1,\Lambda_1>,<A_1,\alpha_1>> \xrightarrow{f1} <<E,\Lambda>,<A,\alpha>> \xleftarrow{f2} <<E_2,\Lambda_2>,<A_2,\alpha_2>>$ be a pushout of $<<E_1,\Lambda_1>,<A_1,\alpha_1>> \xleftarrow{g1} <<E',\Lambda'>,<A',\alpha'>> \xrightarrow{g2} <<E_2,\Lambda_2>,<A_2,\alpha_2>>$ envolving only functional objects. If $f_{1_S}(a_1) = f_{2_S}(a_2)$, for some $a_1 \in slt(A_1)$ and $a_2 \in slt(A_2)$, then there is a slot $a \in slt(A')$ such that $g_1(a) = a_1$ and $g_2(a) = a_2$. We have that $\alpha_1(g_1(a)) f_{1_E}{}^* = \alpha'(a) g_{1_E}{}^* f_{1_E}{}^* = \alpha'(a) g_{2_E}{}^* f_{2_E}{}^* = \alpha_2(g_2(a)) f_{2_E}{}^*$, ie, $\alpha_1(a_1) f_{1_E}{}^* = \alpha_2(a_2) f_{2_E}{}^*$. The existence of functional pushouts in fOb follows now straightforward from proposition 4.4. Moreover the initial Ob object is functional. □

7 Concluding Remarks

We have presented a mathematical model for objects displaying deterministic behaviour. We have shown that the category of objects is finitely cocomplete and that we can explain several types of object interconnection as colimits. Moreover, we have also shown that these nice properties are also valid when working with functional objects. Further object-oriented concepts (such as object class and inheritance) can be established on top of this category of objects (following the ideas in [6]).

Acknowledgements

This work was partially supported by the JNICT Project PMCT/C/TIT/178/90 (FAC3) and by the Esprit Basic Research Action 3023 (IS-CORE).

References

1. J.Adámek, H.Herrlich and G.Strecker: *Abstract and Concrete Categories*. Wiley, 1990
2. J.-F.Costa and A.Sernadas: Process Models within a Categorial Framework. *Research Report*, INESC, 1990 (submitted)
3. J.-F.Costa and A.Sernadas: Progress Assumption in Concurrent Systems. *Research Report*, INESC, 1990 (submitted)
4. H.-D.Ehrich, A.Sernadas and C.Sernadas: From Data Types to Object Types. In: *Journal of Information Processing and Cybernetics*, EIK 26(1/2), 1990, pp. 33-48
5. H.-D.Ehrich, J.Goguen and A.Sernadas: A Categorial Theory of Objects as Observed Processes. In: J.W.deBakker, W.P.deRoever and G.Rozenberg (eds): *Proc. of the REX90/Workshop on Foundations of Object-Oriented Languages*. LNCS 489, Springer-Verlag, 1991, pp. 203-228
6. H.-D.Ehrich and A.Sernadas: Object Concepts and Constructions. In: G.Saake and A.Sernadas (eds): *Proc. of the IS-CORE Workshop'91*. Informatik-Berichte 91-03, Tech. Univ. Braunschweig, 1991, pp. 1-24
7. J.Goguen: Sheaf Semantics of Concurrent Interacting Objects. To appear in *Mathematical Structures in Computer Science*.
8. J.Goguen and S.Ginali: A Categorical Approach to General Systems Theory. In: G.Klir (ed): *Applied General Systems Research,* Plenum 1978, pp. 257-270
9. B.Jonsson: A Model and Proof System for Asynchronous Networks. In: *Proc. of the 4th Annual ACM Symposium on Principles on Distributed Computing*, Minaki, Canada, 1985, pp. 49-58
10. P.Wegner: Learning the language. In: *Byte* 14, 1989, pp. 245-253
11. G.Winskel: Synchronization Trees. In: *Theoretical Computer Science* 34, 1984