Math. Struct. in Comp. Science (1999), *vol.* 9, *pp.* 227–252. Printed in the United Kingdom

Proof-theoretic foundations for the design of actor systems

CARLOS H. C. DUARTE[†]

Department of Computing, Imperial College, 180 Queen's Gate, London, U.K., SW7 2BZ
E-mail: cd7@doc.ic.ac.uk, *tel:* +44 171 594 8341, *fax:* +44 171 581 8024

Received 3 November 1997, revised 12 November 1998

The pioneering work of Hewitt and Baker on the foundations of concurrency during the seventies has inspired the development of a promising object-based framework for understanding open distributed systems, the actor model. So far, theoretical research on actors has focused on identifying the basic primitives of the model and on characterising the operational behaviour of distributed programming languages in terms of actor components. In this paper, we show that the actor model can also be used as a faithful basis for rigorously designing open distributed systems. We argue that a proof-theoretic approach is better suited to this purpose. An abstract data type like axiomatisation of the actor primitives is proposed to support composing and reasoning from specifications of actor communities within a temporal logical system.

1. Introduction

Since the pioneering work of Hewitt and Baker (1977) on the foundations of concurrency, a promising model of open distributed systems has been developed, initially by Clinger (1981) and lately by Agha (1986), Talcott (1997) and others. The so-called actor model regards open distributed systems as communities of objects with encapsulated state, which may only be changed by performing local computations. Interaction between actors is via buffered, point-to-point, asynchronous message passing, based on a localised naming scheme. As a result of processing messages, new concurrent actors can be created, local computations can be performed and actor names can be communicated. With these characteristics, actor systems possess desirable run-time features such as configurability and extensibility. In addition, the actor model integrates the object-based and functional approaches to software development, enforcing in this way design principles such as modularity and incrementability.

Considering the characteristics above, it seems to be a natural research direction to abstract from previous work in which the model was realised in diverse programming languages and semantic domains in order to examine the step-by-step development, and here in particular the design, of open distributed systems in terms of actors. Agha (1986)

[†] This work was supported by CNPq, the Brazilian National Research Council.

identified the basic primitives to support the model and outlined a generic operational semantics for actor languages. Agha *et al.* (1997) developed an operational semantics for a complete language along with criteria for dynamically composing interacting actor components. Alternative semantic domains defined in terms of the inference rules of rewriting and linear logic were studied by Talcott (1996a), Darlington and Guo (1995), respectively. Talcott (1997) also studied many semantic domains capturing actor components at different levels of abstraction. All these works have focused on describing in an operational manner the behaviour of actor systems.

We believe that proof-theoretic approaches such as (Talcott 1996a, Darlington and Guo 1995) are particularly well-suited to describing not only how parts of a system perform computation but also all the other properties the whole system is required to fulfil. These are important too because, in a stepwise development process, designers and programmers have to deal in a rigorous and systematic manner with syntactic constructions such as programs as well as with specifications that may or may not be realised in a computational sense. Despite the recent advances in designing object-based systems using logical systems, it appears that few results can be used in the definition of a formal actor theory. Sernadas *et al.* (1995) proposed a linear time logical system for object-oriented systems design where interaction between objects was synchronous. Similarly, America and de Boer (1996) proposed an exogenous logical system based on the synchronous CSP primitives for dealing with object creation and reconfiguration in a subset of POOL, sticking to the specific programming formalism in this way. Wieringa *et al.* (1995), Parisi-Presicce and Pierantonio (1994) studied naming, roles and classes, although the treatment of interaction and dynamic reconfiguration was not addressed.

The actor model seems to demand a specific logical system to support the design of open distributed systems — the meaning of the actor primitives can be encoded in axioms and inference rules of a specific proof calculus in this way. Objects can thus be specified by theory presentations of the logic and specifications be interconnected by means of interpretations between theories. Techniques for defining such formalisms have been popularised by Goguen and Burstall (1992) in their study of Institutions applied to the theory of abstract data types (ADTs) using some sort of equational logic. Here, however, we have to point out that, when concurrency comes to place, and particularly because the actor model makes some fairness assumptions, the use of a temporal logical system appears to be almost unavoidable.

These remarks leave us very close to the theory-centred, modular view advocated by Fiadeiro and Maibaum (1992) for the design of concurrent systems. Inspired by their work, we organise actor specifications in terms of signatures and presentations of temporal theories. We propose an axiomatisation of the actor primitives for sending and receiving messages as well as for creating objects, deriving inference rules to support reasoning about actor behaviour in terms of safety and liveness properties. Manna and Pnueli (1983) have also applied, at lower levels of abstraction, this idea of particularising temporal logical systems. Having developed our own formalism, we see the main contribution of this work as a logical system that establishes a firm proof-theoretic basis for actor specification, composition and verification, which follows to some extent previous work of the ADT school.

We proceed by discussing some relevant issues in designing a temporal proof-theory for actors. Subsequently, we describe our approach to the specification and verification of actor systems, illustrating the involved technicalities by means of a simple example. Our concluding remarks and prospects for future research are presented in the final section.

2. Issues in the design of a proof-theory for actors

Because we are interested in capturing the constituent entities of the actor model and our approach to design is logical, we need to determine the characteristics of a logical system to make possible the representation of all these entities. To begin with, an *actor* deals with distinct sets of *values* in message passing and computation. Values may be considered as actors with unserialised behaviour (Agha 1986), which are not history sensitive and have a fixed meaning in every computation. Here, however, in order to keep a clear distinction between values and actors, we represent the first family as objects of a sort in a many-sorted language, instead of using an unsorted language. In a way, sorts define types for values, which is indeed the usual representation of properties of fixed meaning objects in programming languages. Actors, in turn, have observable behaviour, state-independent identity and can be regarded as objects. Hence, they are specified using theory presentations as suggested by Fiadeiro and Maibaum (1992).

Actors interact via buffered message passing. Since the work of Sistla *et al.* (1984), later extended by Koymans (1987), temporal logic has been the preferred framework for studying *buffered communication*. Even among such logics, there are many possibilities to choose. Due to the infinite character of some *data domains* of values transmitted in messages, propositional logic cannot be used. Because the actor model requires the delivery and consumption of a message to be guaranteed whenever it remains possible often enough for the target actor to deliver such functionality, *fairness* requirements which demand specifying when these events may occur as it is impossible to determine *a priori* how the environment will evolve, branching time logic has to be used, since it is impossible to express possibility using a linear time frame only (Lamport 1983).

To complete the picture, we need to address the *naming* and *creation* schemes adopted in the actor model. Producing a specification, we are in fact defining a template for the behaviour of a population of similar actors so that each of them receives a distinct mail address at creation time to serve as a name in any communication. The usual way of representing this is to regard the specification as implicitly parameterised by a sort of names, extending the original specification (Ehrich *et al.* 1988). In addition, to avoid conflicts between the creation of new actors and the satisfiability of Barcan formulas, which state that the quantification domain of variables do not vary with the passing of time, every actor specification needs to carry an auxiliary existential boolean attribute symbol. According to this approach, objects that have not been created, i.e., their respective attribute is equal to false, do not play any role, paraphrasing America and de Boer (1996).

Considering this rationale, actor specifications should look like Figure 1. Therein, buffer cells are specified which dynamically allocate a new cell for each stored integer number. Attribute symbols represent the actor state whereas messages and local computations are

Actor BufferCell
data types addr, bool, int (T, F : bool)
attributes val : int; nxt : addr; $void, lst, up$: bool
actions $nil, item(\mathsf{int})$: **local** + **extrn birth**;
$go, cons, link(\mathsf{addr})$: **local computation**;
$put(\mathsf{int}), get(\mathsf{addr})$: **local** + **extrn message**;
$reply(\mathsf{int})$: **extrn message**
axioms k, n : addr; v : int; x, y : bool

$$nil \rightarrow void = \mathrm{T} \wedge lst = \mathrm{T} \wedge up = \mathrm{F} \quad (1.1)$$
$$item(v) \rightarrow val = v \wedge void = \mathrm{F} \wedge lst = \mathrm{T} \wedge up = \mathrm{F} \quad (1.2)$$
$$nil \vee item(v) \rightarrow \mathbf{X}(go) \quad (1.3)$$
$$go \rightarrow \mathbf{X}(up = \mathrm{T}) \quad (1.4)$$
$$go \wedge val = v \wedge void = x \wedge nxt = n \wedge lst = y \rightarrow \mathbf{X}(val = v \wedge void = x \wedge nxt = n \wedge lst = y) \quad (1.5)$$
$$cons \wedge nxt = n \wedge lst = x \wedge up = y \rightarrow \mathbf{X}(void = \mathrm{T} \wedge nxt = n \wedge lst = x \wedge up = y) \quad (1.6)$$
$$link(n) \rightarrow \mathbf{X}(nxt = n \wedge lst = \mathrm{F}) \quad (1.7)$$
$$link(n) \wedge val = v \wedge void = x \wedge up = y \rightarrow \mathbf{X}(val = v \wedge void = x \wedge up = y) \quad (1.8)$$
$$put(v) \wedge lst = \mathrm{T} \rightarrow \mathbf{X}(\exists n \cdot \mathbf{new}(item, n, v) \wedge link(n)) \quad (1.9)$$
$$put(v) \wedge lst = \mathrm{F} \wedge nxt = n \rightarrow \mathbf{X}(\mathbf{send}(put, n, v)) \quad (1.10)$$
$$get(n) \wedge void = \mathrm{F} \wedge val = v \rightarrow \mathbf{X}(\mathbf{send}(reply, n, v) \wedge cons) \quad (1.11)$$
$$get(n) \wedge void = \mathrm{T} \wedge lst = \mathrm{F} \wedge nxt = k \rightarrow \mathbf{X}(\mathbf{send}(get, k, n)) \quad (1.12)$$
$$\exists n \cdot \mathbf{new}(item, n, v) \vee link(n) \leftarrow put(v) \wedge lst = \mathrm{T} \quad (1.13)$$
$$\mathbf{send}(reply, n, v) \vee cons \leftarrow get(n) \wedge val = v \wedge void = \mathrm{F} \quad (1.14)$$
$$\mathbf{send}(put, k, v) \leftarrow put(v) \wedge nxt = k \wedge lst = \mathrm{F} \quad (1.15)$$
$$\mathbf{send}(get, k, n) \leftarrow get(n) \wedge nxt = k \wedge void = \mathrm{T} \wedge lst = \mathrm{F} \quad (1.16)$$
$$up = \mathrm{T} \rightarrow \mathbf{FE}(\mathbf{deliv}(put, v)) \wedge \mathbf{FE}(put(v)) \wedge \mathbf{FE}(\mathbf{deliv}(get, n)) \wedge \mathbf{FE}(get(n)) \quad (1.17)$$

End

Fig. 1. Specification of integer buffer cells.

represented by action symbols. The operators **E**, **X**, **F** and $\leftarrow$ are temporal connectives to state respectively that a property holds in some behaviour, in the next instant, sometime in the future or only if preceded by the occurrence of another property. For instance, axiom (1.9) states that, if a message $put(v)$ is consumed by the last cell of the buffer ($lst = \mathrm{T}$), in the next instant another cell containing the value v will be created and linked to the current one ($\mathbf{new}(item, n, v) \wedge link(n)$). Subsequently, the buffer will have reconfigured accordingly. Axiom (1.13) determines that neither of the two events above happen unless the proper cell consumes a *put* message first. We shall continue to explain this example in Section 3.

3. An axiomatisation of the actor model

3.1. Representing actors

We use theory signatures to define the symbols that can be used in writing each specification. Signatures bring both the notion of scope and interface to the logic, by forcing every used symbol to be declared locally and by enabling the definition of language translations in order to connect specifications. Theory signatures for actor specification are defined as follows:

Definition 1. (Actor Signature) An *actor signature* $\Delta = (\Sigma, \mathcal{A}, \Gamma)$ is a triple of disjoint and finite families of symbols such that:

— $\Sigma = (S, \Omega)$ is a universe signature, i.e., S is a set of rigid sort symbols and Ω is an $S^*_{fin} \times S$-indexed family of rigid function symbols[†]. We also require that $\mathsf{addr} \in S$, representing the sort of mail addresses (or actor names);
— $\mathcal{A}$ (or $\mathcal{A}_l$) is an $S^*_{fin} \times S$-indexed family of flexible attribute symbols;
— $\Gamma = (\Gamma_e, \Gamma_l, \Gamma_c)$ is a triple of S^*_{fin}-indexed families of flexible action symbols such that $(\Gamma_e \cup \Gamma_l) \cap \Gamma_c = \{\}$. Γ_c is a set of local computation symbols. The elements of Γ_e and Γ_l represent, respectively, events to be requested from the environment and provided locally[‡]. Each of these two sets contains distinguished sub-sets of message and birth symbols, e.g. $\Gamma_l - \Gamma_{l_b}$ and Γ_{l_b}.

For ϵ as the empty sequence, we write $\epsilon \times s$-indexed families of signature symbols as if s were their single index. Given a set or sequence of such symbols X, we write as $X_{\langle s_1,\ldots,s_n\rangle,s}$ the sub-set or sub-sequence of X containing symbols of type $\langle s_1, \ldots, s_n\rangle \to s$ only. To make reference to specific sets of signature symbols, we operate with subscripts to denote operations on sub-sets. For instance, $\Gamma_{e_b} \cap \Gamma_{l_b}$ is written as $\Gamma_{e_b \cap l_b}$.

In the example specification of Figure 1, addr, bool and int are the sort symbols that constitute, together with their implicitly specified constants and operations, the universe signature Σ. Clearly, the sort of mail addresses addr has to be part of every signature. Otherwise, some specified actors would be useless without the ability of exchanging messages or creating new actors. Still in the example, *val* (current value), *nxt* (next cell address), *void* (consumed content), *lst* (last cell) and *up* (live cell) are the attribute symbols in $\mathcal{A}$. In the particular terminology of the actor model, they are called acquaintances, which may be determined at creation time or in performing local computations.

Action symbols represent instantaneous occurrences of local computations; the dispatch, delivery and consumption of messages; actor births and their respective requests. All these occurrences can be regarded as events in the sense studied by Hewitt and Baker (1977). The structure of the set of action symbols adopted here differs from those of Sernadas *et al.* (1995), Fiadeiro and Maibaum (1992), who advocate similar logics. Each actor specification may guarantee the occurrence of externally required events and may determine that the occurrence of some events is required from the environment. Actor specifications may also define local computations. Because of these distinctions, the set of action symbols is divided into Γ_l, Γ_e and Γ_c, respectively. The first two of these are partitioned into sub-sets of symbols representing messages and births, Γ_{e-e_b} and Γ_{e_b} for instance. Actors interact via asynchronously transmitted messages, denoted by the symbols in $\Gamma_{(l-l_b)\cup(e-e_b)}$, which are used in many different ways. For instance, $put(v)$ represents the consumption of a message put carrying v as its contents and $\mathbf{send}(put, n, v)$ specifies that the same message and contents are transmitted to an object whose mail address is n. Signature symbols also have distinct uses in the creation of actors, through the primitive **new** and the subsequent occurrence of birth actions in $\Gamma_{l_b \cup e_b}$. All these events can only occur carrying a finite number of acquaintances and are exemplified by the action symbols in Figure 1.

[†] We usually consider that the enumerated constants are all different from each other.
[‡] Because actors may self-address requests, Γ_e and Γ_l should not be disjoint in general.

As is usual in a proof-theoretic approach, cf. Wieringa *et al.* (1995), we extend signatures with new logical symbols. The situation here resembles the use of hidden symbols in algebraic specifications (Ehrig and Mahr 1985). Therein, the specifier may need to use an externally unavailable language to specify complex data types. Herein, we use a simpler language to specify complex patterns of behaviour presented by every actor, defined in terms of a more complex language. This extended language will be used to provide an implicit proof-theoretic semantics for the actor primitives and that is why it should not be required from the specifier of each signature.

Definition 2. (Extended Actor Signature) Given an actor signature $\Delta = (\Sigma, \mathcal{A}_l, \Gamma)$ such that $\Sigma = (S, \Omega)$ and $\Gamma = (\Gamma_e, \Gamma_l, \Gamma_c)$, the triple $\lambda\Delta = (\lambda\Sigma, \lambda\mathcal{A}, \lambda\Gamma)$ is said to be the *extended signature* of Δ if and only if:

1 $\lambda\Sigma = (S \cup \{\mathsf{bool}\}, \Omega \cup \{\mathrm{T}_{\mathsf{bool}}, \mathrm{F}_{\mathsf{bool}}, \mathrm{NOT}_{\mathsf{bool}\to\mathsf{bool}}\})$;

2 $\lambda\mathcal{A} = (\mathcal{A}_l, \mathcal{A}_i, \mathcal{A}_s, \mathcal{A}_d)$, such that (i) for each $c \in \Gamma_{l_b}$ of sort $\langle s_1, \ldots, s_n\rangle$ there is an $init_c \in \mathcal{A}_{i_{\langle s_1,\ldots,s_n\rangle,\mathsf{bool}}}$; (ii) for each $c \in \Gamma_{(e-e_b)\cup(l-l_b)}$ of sort $\langle s_1, \ldots, s_n\rangle$ there is a $sent_c \in \mathcal{A}_{s_{\langle s_1,\ldots,s_n\rangle,\mathsf{bool}}}$, and (iii) for each $c \in \Gamma_{l-l_b}$ of sort $\langle s_1, \ldots, s_n\rangle$ there is a $delivd_c \in \mathcal{A}_{d_{\langle s_1,\ldots,s_n\rangle,\mathsf{bool}}}$. All the symbols in the respective components of $\lambda\mathcal{A}$ are due to (i), (ii) and (iii);

3 $\lambda\Gamma = (\Gamma_e, \Gamma_{out}, \Gamma_l, \Gamma_{in}, \Gamma_c, \Gamma_{rcv})$, where (i) for each $c \in \Gamma_e$ of sort $\langle s_1, \ldots, s_n\rangle$ there is an $out_c \in \Gamma_{out_{\langle \mathsf{addr},\mathsf{addr},s_1,\ldots,s_n\rangle}}$; (ii) for each $c \in \Gamma_l$ of sort $\langle s_1, \ldots, s_n\rangle$ there is an $in_c \in \Gamma_{in_{\langle \mathsf{addr},\mathsf{addr},s_1,\ldots,s_n\rangle}}$, and (iii) for each $c \in \Gamma_{l-l_b}$ of sort $\langle s_1, \ldots, s_n\rangle$ there is a $rcv_c \in \Gamma_{rcv_{\langle s_1,\ldots,s_n\rangle}}$ such that $\Gamma_{(in\cup out)\cap rcv} = \{\,\}$ and that $in_c = out_c$ if and only if $c \in \Gamma_{e\cap l}$. All the symbols in the respective components of $\lambda\Gamma$ are due to (i), (ii) and (iii).

That is to say, the original universe signature is extended with a boolean sort symbol, new attribute symbols are provided to deal with the existence of actors and buffering of messages, and new action symbols are introduced to handle creation and interaction. Hereafter, we will not make any distinction between extended and actor signatures.

A central feature of actors is interaction. Here, it is simulated using the action symbols out_c and in_d which happen simultaneously for any $c \in \Gamma_e$ and $d \in \Gamma'_l$ belonging to the actor communities, populations of objects complying with the same specification, requesting and providing the event respectively. These symbols correspond either to the dispatch of a message or the request of an actor birth. The occurrence of these logical actions plays the role of the interaction steps of Talcott (1996b). For an interaction represented by c between actors of the same community, hence by a member of $\Gamma_{e\cap l}$, the occurrence of the new actions above is obliged to be synchronous by the second constraint in (3.iii) of Definition 2. Otherwise, this synchronisation must be supported by the existence of a morphism identifying these symbols as shared by the distinct signatures, as discussed in Section 3.5. Asynchrony in message transmission is guaranteed by forcing $out_c|in_d$ to happen strictly before rcv_d, which in turn has to occur strictly before d itself. The two last symbols correspond to the occurrence of the delivery and consumption of the message, respectively. Finally, (double) buffering is captured by the attribute $delivd_d$ ($sent_c$) becoming true for some values whenever these values are delivered (sent) in a message. Of course, these new symbols do not explicitly appear in specifications but their

behavioural constraints will have to be captured by our axiomatisation. Also, according to the definition above, ill formed messages are not allowed — as action symbols, messages always have a locally correct representation at the sender — and dispatched messages which do not belong to the language available to the target actor are never delivered.

Following America and de Boer (1996), we consider that in a given point in time it is only possible to deal with the existing actors at that moment. Accordingly, an object will have some $init_c$ attribute equalised to T(RUE) for some sequence of terms $\vec{v_c}$ only if the occurrence of an action $in_c(\vec{v_c})$, $c \in \Gamma_{l_b}$, gives rise to its birth. The structure of communities of actors which comply with the same specification, each of which having a distinguished mail address, is defined below:

Definition 3. (Actor Community Signature) Given a signature $\Delta = (\Sigma, \mathcal{A}, \Gamma)$, a *community signature* Δ^{P} is obtained by "parameterising" Δ with sort P. That is, $\Sigma^{\mathsf{P}} \stackrel{\text{def}}{=} (S \cup \{\mathsf{P}\}, \Omega)$; $\mathcal{A}^{\mathsf{P}}$ is obtained from $\mathcal{A}$ by adding the parameter sort P to each of its attribute symbols; and Γ^{P} is obtained from Γ by adding the parameter sort P to each action symbol in Γ_e, Γ_l, Γ_c and Γ_{rcv}. The other symbols of Δ remain the same in Δ^{P}.

Clearly, the parameter sort P of every community should be addr. Indeed, as identified by Talcott (1996b), actor semantics should be parameterised by sets of actor addresses. Due to our definition, a new argument is added to the appropriate signature symbols and its instances will be actor names. In this way, the basic operations on object references identified by America and de Boer (1996), *equality test* and *dereferencing*, are supported. However, signatures alone do not support a modular design discipline, obliging the entire structure of complex systems to be represented as single entities. The required means of composition shall be studied in Section 3.5.

3.2. Specifying actor behaviours

Actor specifications correspond to the behaviour definitions of Agha (1986) at higher levels of abstraction. In their definition, we assume that a countably infinite family of rigid variables with a partial classification Ξ according to the set of sorts of each signature are given. For a sort symbol s of a given signature, we also represent the set of s-classified variables as Ξ_s.

Terms stand for meaningful values. In their definition, a signature Δ and a classification Ξ are used. These are assumed to be given in the sequel.

Definition 4. (Terms) The S-indexed set of *terms* $T_\Delta(\Xi)$ is defined as follows, provided that $x \in \Xi_s \cup \Omega_s \cup \mathcal{A}_s$, $f \in \Omega_{\langle s_1,\ldots,s_n\rangle,s}$, $g \in \mathcal{A}_{\langle s_1,\ldots,s_n\rangle,s}$ and $t_i \in T_\Delta(\Xi)_{s_i}$:

$$t ::= x \mid f(t_1, \ldots, t_n) \mid g(t_1, \ldots, t_n)$$

That is, a term is a variable, a nullary function or attribute symbol, or a term is a function or attribute symbol applied to a sequence of terms. We usually write each sequence of terms $t_1, \ldots, t_n$ as $\vec{t}$. The classification of variables and the sort of each signature symbol induce a type over each expression of the language. Functional constructions taking arguments of sort $\langle s_1, \ldots, s_n\rangle$ and producing a result of sort s are said to have type $\langle s_1, \ldots, s_n\rangle \to s$.

As explained previously, to give an account of actor behaviour in terms of (sets of) formulas, first-order branching time logic with equality is required. In what follows, we take the usual connectives of classical first-order logic and add to this set the occurrence of an instantaneous action, of the initial instant (**beg**), the occurrence of a formula q_1 in any instant that could succeed the actual past history ($\mathbf{A}q_1$) or the occurrence of q_1 strictly in the future such that formula q_2 happens from the next instant until but not necessarily including then ($q_1 \mathbf{V} q_2$). This is formally stated as follows:

Definition 5. (Formulas) The set $F_\Delta(\Xi)$ of *formulas* is defined by the production rule below, provided that $c \in \Gamma_{\langle s_1,\ldots,s_n\rangle}$, $t_i \in T_\Delta(\Xi)_{s_i}$, $x \in \Xi_s$ and $q_i \in F_\Delta(\Xi)$:

$$q ::= (t_1 = t_2) \mid q_1 \rightarrow q_2 \mid \neg q_1 \mid \forall x \cdot q_1 \mid c(\vec{t}) \mid \mathbf{beg} \mid \mathbf{A}(q_1) \mid (q_1)\mathbf{V}(q_2)$$

Other required connectives are defined as usual. Concerning the temporal and modal operators, this means that: $p\mathbf{U}q \stackrel{\text{def}}{=} q \vee (p \wedge q\mathbf{V}p)$ (p happens until q does), $\mathbf{X}p \stackrel{\text{def}}{=} p\mathbf{V}\bot$ (p happens in the next instant), $\mathbf{F}p \stackrel{\text{def}}{=} \top\mathbf{U}p$ (p happens sometime in the future), $\mathbf{G}p \stackrel{\text{def}}{=} \neg\mathbf{F}(\neg p)$ (henceforth p happens), $p\mathbf{W}q \stackrel{\text{def}}{=} \mathbf{G}p \vee p\mathbf{U}q$ (p happens until q does, if it ever happens), $q_1 \stackrel{i}{\leftarrow}_p q_2 \stackrel{\text{def}}{=} p \rightarrow (\neg q_1)\mathbf{W}(q_2 \wedge \neg q_1)$ (after p happens, q_2 precedes q_1), $q_1 \leftarrow_p q_2 \stackrel{\text{def}}{=} q_1 \stackrel{i}{\leftarrow}_p q_2 \wedge q_1 \rightarrow \mathbf{X}(q_1 \stackrel{i}{\leftarrow}_\top q_2)$ (after p happens, q_2 always precedes q_1) and $\mathbf{E}p \stackrel{\text{def}}{=} \neg\mathbf{A}(\neg p)$ (p is possible, i.e., happens in some behaviour). The temporal connectives future, always, until and unless are all defined to be non-strict — they also talk about the current instant. Conversely, next and the precedence connectives are strict and can be used to define causality relationships between events; for instance that an occurrence of a *get* causes the dispatch of a *reply* (1.11) and this does not happen otherwise, also forbidding the simultaneous occurrence of such events (1.14). The precedence connective indexes appearing in each specification, normally the **beg** connective defined above, are omitted.

A formal definition of actor specifications, exemplified here by BUFFERCELL, is:

Definition 6. (Actor Specification) An *actor specification* is a pair $\Phi = (\Delta, \Psi)$ where Δ is an actor signature and Ψ is a finite set of Δ-formulas (the specification axioms).

We consider that free variables in axioms are implicitly universally quantified. Moreover, due to the parameterisation of each signature by `addr`, we are allowed to use the conventional object-based notation of prefixing the name of an object to the logical expressions pertaining to it. Therefore, we write $q(n, \vec{v_q})$ as $n.q(\vec{v_q})$ for any $q \in \Omega \cup \mathcal{A} \cup \Gamma$. For any formulas q_1 and q_2, we also have, say, $n.q_1 \rightarrow n.q_2 \equiv n.(q_1 \rightarrow q_2)$ and $\neg n.q_1 \equiv n.(\neg q_1)$. The actor primitives defined below are also admissible in specifications and proofs:

FOR	IN	FORMULA	READS	REPRESENTS
—	—	**init**	initialisation	$\bigvee \{\exists \vec{v_c} \cdot c(\vec{v_c}) \mid c \in \Gamma_{l_b}\}$
$n_i, \vec{v_c}$	$T_\Delta(\Xi)$	$n_1.\mathbf{new}(c, n_2, \vec{v_c})$	actor creation	$out_c(n_1, n_2, \vec{v_c})$, if $c \in \Gamma_{e_b}$ $in_c(n_1, n_2, \vec{v_c})$, if $c \in \Gamma_{l_b}$
$n_i, \vec{v_c}$	$T_\Delta(\Xi)$	$n_1.\mathbf{send}(c, n_2, \vec{v_c})$	message dispatch	$out_c(n_1, n_2, \vec{v_c})$, if $c \in \Gamma_{e-e_b}$ $in_c(n_1, n_2, \vec{v_c})$, if $c \in \Gamma_{l-l_b}$
$n, \vec{v_c}$	$T_\Delta(\Xi)$	$n.\mathbf{deliv}(c, \vec{v_c})$	message delivery	$n.rcv_c(\vec{v_c})$, if $c \in \Gamma_{l-l_b}$

There exists just another actor primitive not treated here: **become**, which prescribes that an actor will subsequently behave according to a distinct specification determined *a priori*. In fact, local computations in Γ_c like *cons* of our example together with a selective use of attribute symbols simulate this in an awkward manner. Indeed, the whole BUFFERCELL specification could have been split so that each cell could become both a linked and an empty one according to the processing of previously received messages[§]. It would be easy to present **become** as another definition, by introducing death actions in signatures and by defining the primitive as the death of an actor and its subsequent resurrection with a distinct behaviour, keeping the same mail address in this process. However, we have reasons to avoid treating this here: in the first place, in order to simplify our presentation, and secondly because the primitive, with the meaning described above, does not increase the expressive power of the model, as identified by Agha (1986).

3.3. Understanding actor specifications

To design actor systems in a rigorous manner, we prefer to adopt a logic which is different from the usual full branching time logic CTL* defined by Emerson (1990). Here, the branching modality **E** defines the occurrence of a formula in some alternative behaviour with identical past history but not necessarily including the current moment. In CTL*, this condition is strictly inclusive and **E** acquires an undesirable denotation in our specifications. We provide in what follows an outline of the model theory of this logic.

Discrete semantic models for branching time such as transition systems and event structures abound in the literature. The definition below is of the first kind:

Definition 7. (Branching Time Structure) A *branching time structure* or *frame* is a tuple $(\alpha, \alpha_0, \rho, \Lambda)$ where:

— α and $\alpha_0 \subseteq \alpha$ are sets of worlds and initial worlds respectively;
— $\rho : \alpha \to \mathcal{P}(\alpha)$ is the accessibility relation (a powerset function);
— Λ is a non-empty set of possible behaviours. Each $L \in \Lambda$ is a function such that: (i) $\mathsf{dom}\ L \subseteq \alpha$ and $\mathsf{cod}\ L \stackrel{\text{def}}{=} \mathbf{N}$; (ii) $L(w) = 0$ iff $w \in \alpha_0$; (iii) $\forall w, w' \in \mathsf{dom}\ L \cdot L(w) = L(w') \to w = w'$; (iv) $\forall n \in \mathsf{cod}\ L \cdot \exists w \in \mathsf{dom}\ L \cdot L(w) = n$; and (v) $\forall w, w' \in \mathsf{dom}\ L \cdot L(w') = L(w) + 1 \to w' \in \rho(w)$.

The sequences of worlds which determine behaviours in Λ (not necessarily of any program) are in a one to one correspondence with the natural numbers, according to (iii-iv). Hence, each $L \in \Lambda$ is invertible and we use this fact to define the meaning of **A**.

Based on branching time structures, signature symbols are interpreted as follows:

Definition 8. (Interpretation Structure) An *interpretation structure* for a signature $\Delta = (\Sigma, \mathcal{A}, \Gamma)$, $\Sigma = (S, \Omega)$, is a tuple $\theta = (T, U, G, A)$ where:

— T is a branching time structure;
— U maps each $s \in S$ to a non-empty collection s_U and each $f \in \Omega$, $type(f) = \langle s_1, \ldots, s_n \rangle \to s$, to $f_U : s_{1_U} \times \ldots \times s_{n_U} \to s_U$;

[§] Note that, since $|\Gamma_{l_b}| \in [0, \omega_0[$, we allow actors to have "multiple constructors".

— G maps each $g \in \mathcal{A}$, $type(g) = \langle s_1, \ldots, s_n \rangle \to s$, to $G(g) : s_{1_U} \times \ldots \times s_{n_U} \to \alpha \to s_U$;
— A maps each $a \in \Gamma$, $type(a) = \langle s_1, \ldots, s_n \rangle$, to $A(a) : s_{1_U} \times \ldots \times s_{n_U} \to \mathcal{P}(\alpha)$.

We adopt interpretation structures as models of logical formulas. Hence, whenever a formula has a model, the sets of worlds α and α_0 in the underlying frame are not empty. That α appears as an argument in the interpretation of some symbols is related to their flexible, time-dependent meaning: sorts are interpreted as constant sets and operations as constant functions whereas attributes vary with time. Interpreting flexible symbols in Γ also shows that the events in this set may happen in parallel among themselves, in which case this is specified through the conjunction of their symbols, or with respect to other events of the environment. This is in keeping with the open but not necessarily interleaving semantics adopted in (Barringer 1987, Fiadeiro and Maibaum 1992).

We interpret terms as defined below. Because we have a first-order logic, we need to define first how logical variables are assigned to the elements of quantification domains:

Definition 9. (Assignment) Given an interpretation structure θ for a signature $\Delta = ((S, \Omega), \mathcal{A}, \Gamma)$, an *assignment* N for θ maps each set Ξ_s, $s \in S$, to s_U.

Definition 10. (Interpretation of Terms) Given an interpretation structure $\theta = (T, U, G, A)$ for a signature $\Delta = ((S, \Omega), \mathcal{A}, \Gamma)$ and an assignment N for θ, $[\!]^{\theta,N} : \alpha \to s_U$ defined below is an *interpretation of terms* of sort $s \in S$ at a world $w \in \alpha$:

— $[\![x]\!]^{\theta,N}(w) \stackrel{\text{def}}{=} N(x)$ if $x \in \Xi_s$;
— $[\![f(t_1, \ldots, t_n)]\!]^{\theta,N}(w) \stackrel{\text{def}}{=} f_U([\![t_1]\!]^{\theta,N}(w), \ldots, [\![t_n]\!]^{\theta,N}(w))$;
— $[\![g(t_1, \ldots, t_n)]\!]^{\theta,N}(w) \stackrel{\text{def}}{=} (G(g)([\![t_1]\!]^{\theta,N}(w), \ldots, [\![t_n]\!]^{\theta,N}(w)))(w)$.

Our branching modality is interpreted with the help of an equivalence relation $\simeq$ over behaviour prefixes. We define this relation in a pointwise manner, saying that two words of a frame are related if and only if all the attribute and action symbols have identical interpretation in both worlds. The satisfaction of logical formulas is defined below:

Definition 11. (Satisfaction of Formulas) Given a signature Δ, the *satisfaction* of a Δ-formula at world w_i of a behaviour L (i.e., $w_i \in \mathsf{dom}\ L$) by a structure $\theta = (T, U, G, A)$ with assignment N is defined as follows:

— $(\theta, N, L, w_i) \models a(t_1, \ldots, t_n)$ iff $w_i \in A(a)([\![t_1]\!]^{\theta,N}(w_i), \ldots, [\![t_n]\!]^{\theta,N}(w_i))$;
— $(\theta, N, L, w_i) \models \neg p$ iff it is not the case that $(\theta, N, L, w_i) \models p$;
— $(\theta, N, L, w_i) \models p \to q$ iff $(\theta, N, L, w_i) \models p$ implies $(\theta, N, L, w_i) \models q$;
— $(\theta, N, L, w_i) \models \forall x \cdot p$ iff for every $v \in \mathsf{cod}\ N$ and every assignment N_v for θ such that $N_v(y) = N(y)$ if $y \neq x$ and $N_v(y) = v$ otherwise, $(\theta, N_v, L, w_i) \models p$;
— $(\theta, N, L, w_i) \models (t_1 = t_2)$ iff $[\![t_1]\!]^{\theta,N}(w_i) = [\![t_2]\!]^{\theta,N}(w_i)$;
— $(\theta, N, L, w_i) \models \mathbf{beg}$ iff $L(w_i) = 0$;
— $(\theta, N, L, w_i) \models p\mathbf{V}q$ iff there is $w_j \in \mathsf{dom}\ L$ with $L(w_i) < L(w_j)$, $(\theta, N, L, w_j) \models p$ and $(\theta, N, L, w_k) \models q$ for any $w_k \in \mathsf{dom}\ L$ where $L(w_i) < L(w_k) < L(w_j)$;
— $(\theta, N, L, w_i) \models \mathbf{A}p$ iff for every $L_j \in \Lambda$ such that $w_k \simeq (L_j^{-1} \circ L)(w_k)$ for each $w_k \in \mathsf{dom}\ L$ with $L(w_k) < L(w_i)$, $(\theta, N, L_j, (L_j^{-1} \circ L)(w_i)) \models p$.

The definition of satisfiability above determines a floating interpretation for our logic. That is, the initial instant has no special significance in the interpretation, even though it is represented as the logical connective **beg**.

3.4. Axiomatising actor behaviours

In this section, we develop a proof calculus for reasoning about actors. The associated notion of model is taken from the class of structures defined in Section 3.3 which also satisfy our extended axiomatisation. We assume that an axiomatisation of the underlying branching time logic is given — a sound one appears in (Duarte 1998) — and thus we can focus just on the actor model here.

We develop an axiomatisation of a consequence relation $\vdash_\Delta$, which is indexed by a signature Δ because this relation is defined in a way that depends on the symbols of the given signature. We assume that $\Delta = (\Sigma, \mathcal{A}, \Gamma)$ is given. We also use the variable n for actor names, decorated with indexes whenever necessary. Moreover, for a given $c \in \Gamma$, $type(c) = \langle s_1, \ldots, s_n \rangle$, $n \in \vec{v_c}$ abbreviates $\bigvee\{n = v_{c_i} | type(v_{c_i}) = \mathsf{addr}; 1 \leq i \leq n\}$ and $\vec{v_c} = \vec{u_c}$ abbreviates $\bigwedge\{v_{c_i} = u_{c_i} | 1 \leq i \leq n\}$. The following notation is used to express the invariance of an expression; that a required actor name has become known due to the delivery of a message, the birth of the actor or the creation of new objects; that a property does not occur until a specific actor name becomes known; and a strong fairness requirement over the occurrence of a particular formula:

FOR	IN	FORMULA	REPRESENTS
t	$T_\Delta(\Xi)$	$Inv(t)$	$\forall k \cdot t = k \rightarrow \mathbf{X}(t = k)$
p	$\mathcal{G}(\Delta)$	$Inv(p)$	$(p \wedge \mathbf{X}p) \vee (\neg p \wedge \mathbf{X}(\neg p))$
n	$T_\Delta(\Xi)_{\mathsf{addr}}$	$Acq(n)$	$\bigvee\{\exists \vec{v_d} \cdot \mathbf{deliv}(d, \vec{v_d}) \wedge n \in \vec{v_d} \mid d \in \Gamma_{l-l_b}\}$ $\bigvee\{\exists \vec{v_d} \cdot d(\vec{v_d}) \wedge n \in \vec{v_d} \mid d \in \Gamma_{l_b}\}$ $\bigvee\{\exists \vec{v_d} \cdot \mathbf{new}(d, n, \vec{v_d}) \mid d \in \Gamma_{e_b}\}$
n, p	$T_\Delta(\Xi)_{\mathsf{addr}}, \mathcal{G}(\Delta)$	$Wait(n, p)$	$(\neg p)\mathbf{W}(\mathbf{init}) \wedge (\neg p)\mathbf{W}(Acq(n))$
p	$\mathcal{G}(\Delta)$	$Fair(p)$	$\mathbf{F}(p \vee \mathbf{GA}(\neg p))$

As identified by Hewitt and Baker (1977), *locality* is an essential characteristic of the actor model. This is also a crucial assumption in object-based logics to support modular specification and reasoning (Fiadeiro and Maibaum 1992, Sernadas *et al.* 1995). Generally speaking, locality requires that state changes of an actor be effected only by the events related to the object itself. This means in particular that each actor has encapsulated state. We choose to capture locality through the axioms below:

($\overline{\mathbf{L1}}$) $\bigvee_{c \in \Gamma_c} \exists \vec{v_c} \cdot n.c(\vec{v_c}) \vee \bigwedge_{f \in \mathcal{A}_l} \forall \vec{v_f} \cdot n.Inv(f(\vec{v_f}))$

(**L2**) $\bigwedge_{c \in \Gamma_{l_b}} \forall \vec{v_c} \cdot \exists n_1 \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c}) \vee n_2.Inv(init_c(\vec{v_c}))$

(**L3**) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot \exists n_2 \cdot n_2.\mathbf{send}(c, n_1, \vec{v_c}) \vee n_1.\mathbf{deliv}(c, \vec{v_c}) \vee n_1.Inv(sent_c(\vec{v_c}))$

(**L4**) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot n_1.\mathbf{deliv}(c, \vec{v_c}) \vee n_1.c(\vec{v_c}) \vee n_1.Inv(delivd_c(\vec{v_c}))$

The first axiom says that either an actor performs a local computation or its extra-logical attributes all remain invariant. In the BUFFERCELL example, this means that either *cons*, *link* or *go* occur or else the values of *val*, *nxt*, *void*, *lst* and *up* do not change. According to the second axiom, either an object is created with a certain name or the existence of an actor with such a name is not disturbed. The other two logical axioms are to guarantee that buffering attributes vary only when message passing takes place.

The following axioms constrain the *occurrence* of events:

($\overline{\mathbf{O1}}$) $\bigwedge_{c \in \Gamma_{e-e_b}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow \mathbf{G}(\neg n_1.\mathbf{init}) \vee \bigwedge_{n \in n_2 : \vec{v_c}} n_1.Wait(n, \mathbf{send}(c, n_2, \vec{v_c}))$

($\overline{\mathbf{O2}}$) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow (\neg n.\mathbf{deliv}(c, \vec{v_c}))\mathbf{W}(n.\mathbf{init})$

($\overline{\mathbf{O3}}$) $\bigwedge_{c \in \Gamma_{(l-l_b) \cup e}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow (\neg n.c(\vec{v_c}))\mathbf{W}(n.\mathbf{init})$

($\overline{\mathbf{O4}}$) $\bigwedge_{c \in \Gamma_{e_b}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow \mathbf{G}(\neg n_1.\mathbf{init}) \vee \bigwedge_{n \in \vec{v_c}} n_1.Wait(n, \exists n_2 \cdot \mathbf{new}(c, n_2, \vec{v_c}))$

(**O5a**) $\bigwedge_{\substack{b,c, \in \Gamma_{l_b} \\ d \in \Gamma l - l_b}} \forall \vec{v_c}, \vec{v_d} \cdot \exists n_1, \vec{v_b} \cdot n_1.\mathbf{new}(b, n_2, \vec{v_b}) \rightarrow n_2.init_d(\vec{v_c}) = n_2.sent_d(\vec{v_d}) = n_2.delivd_d(\vec{v_d}) = \mathrm{F}$

(**O5b**) $\bigwedge_{c \in \Gamma_{l_b}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow (n.c(\vec{v_c}) \leftrightarrow n.init_c(\vec{v_c}) = \mathrm{T})$

($\overline{\mathbf{O6a}}$) $\bigwedge_{c \in \Gamma_{l_b}} \exists n_1, n_2, \vec{v_c} \cdot \mathbf{E}(n_1.\mathbf{new}(c, n_2, \vec{v_c}))$

($\overline{\mathbf{O6b}}$) $\bigwedge_{c \in \Gamma_{l_b}} \mathbf{G}(\exists! \, n_2 \cdot \exists n_1, \vec{v_c} \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c})) \rightarrow \forall n_2 \cdot \mathbf{F}(\exists n_1, \vec{v_c} \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c}))$

($\overline{\mathbf{O7a}}$) $\bigwedge_{c \in \Gamma_{l_b}} \forall \vec{v_c} \cdot \exists n_1 \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c}) \rightarrow \mathbf{XF}(n_2.c(\vec{v_c}))$

($\overline{\mathbf{O7b}}$) $\bigwedge_{c \in \Gamma_{l_b}} \forall \vec{v_c} \cdot \mathbf{beg} \rightarrow \mathbf{X}((\neg n_2.c(\vec{v_c}))\mathbf{W}(\neg n_2.c(\vec{v_c}) \wedge \exists n_1 \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c})))$

($\overline{\mathbf{O8}}$) $\bigwedge_{\substack{c,d \in \Gamma_{l_b} \\ d \neq c}} \forall \vec{v_c} \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c}) \rightarrow \nexists n_3, \vec{u_c}, \vec{v_d} \cdot n_3 : \vec{u_c} \neq n_1 : \vec{v_c} \wedge n_3.\mathbf{new}(c, n_2, \vec{u_c}) \vee n_3.\mathbf{new}(d, n_2, \vec{v_d})$

(**O9**) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot n.\mathbf{deliv}(c, \vec{v_c}) \rightarrow n.sent_c(\vec{v_c}) = \mathrm{T}$

($\overline{\mathbf{O10}}$) $\bigwedge_{\substack{c,d \in \Gamma_{l-l_b} \\ d \neq c}} \forall \vec{v_c} \cdot n.\mathbf{deliv}(c, \vec{v_c}) \rightarrow \nexists \vec{u_c}, \vec{v_d} \cdot \vec{u_c} \neq \vec{v_c} \wedge n.\mathbf{deliv}(c, \vec{u_c}) \vee n.\mathbf{deliv}(d, \vec{v_d})$

(**O11**) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \rightarrow n.delivd_c(\vec{v_c}) = \mathrm{T}$

($\overline{\mathbf{O12}}$) $\bigwedge_{\substack{c,d \in \Gamma_{(l-l_b) \cup e} \\ c \neq d}} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \rightarrow \nexists \vec{u_c}, \vec{v_d} \cdot \vec{u_c} \neq \vec{v_c} \wedge n.c(\vec{u_c}) \vee n.d(\vec{v_d})$

O1-4 state that, before the birth of an actor, not only the dispatch, delivery and consumption of messages but also local computations and requests for creation are forbidden. Note that **O1** and **O4** are more liberal than the other axioms if the respective actor is never created but are more restrictive otherwise by requiring that each actor name becomes known due to the delivery of a message, the birth of the actor or the creation of another object before the name can be used in the respective task. These restrictions are to prevent the use of arbitrary names and modes of interaction such as broadcasting which are distinct from point-to-point message passing. On the other hand, the same axioms are permissive concerning unborn actors because we are capturing an open mode of interaction, which cannot be totally constrained by the local semantics. An actor complying with some specification, say, does not have to be created in this context, but may need to dispatch some messages which are mentioned therein. Therefore, the occurrence of these events should not be logically forbidden. The situation above is dual to that described by Fiadeiro and Maibaum (1997) wherein read-only attributes are adopted as a means of capturing an open synchronous mode of interaction. Such attributes cannot be constrained locally, but only at a global level where the respective components are put together and interfere with the behaviour of one another.

The subsequent set of logical axioms above relates the creation of new actors, the

occurrence of birth actions and the existence of other objects. **O5a** and the other axioms imply that an actor can only be created once and also that messages are not sent or delivered to the object before its birth. Moreover, according to **O5b**, the actor birth occurs in the beginning of time if the object always exists. **O6a** says that it is always possible for some actor to create a new object and **O6b** states that all the actor names will be used if exactly one object is created at each instant. It is important to mention that, because of the specific characteristics of the adopted time flows, the former axiom implies that the set of actor names is infinite while the latter implies that the same set is countable. **O7a** and **O7b** state that the occurrence of births and requests for creation are always causally connected after the initial moment.

We have also proposed a set of axioms stating mutual exclusion. Most of these properties are particular to the actor model, whereas a few are due to decisions in the design of our formalism. **O8** specifies that actors with the same name cannot be concurrently created; **O9** says that messages can be delivered only if they were previously sent; **O10** determines that only one message can be delivered to an actor at each instant; **O11** says that messages can be consumed only if they were previously delivered; and finally, according to **O12**, message consumption and local computations of an actor are totally ordered, meaning that two such events cannot occur in parallel. Concerning this last axiom, we could have allowed instead actors with full internal concurrency while ensuring attribute consistency through additional axioms. We prefer the simpler formulation here to facilitate specification and reasoning. Note that the specified actors can always present some internal concurrency anyway: they can, for instance, create many other objects and send several messages at the same time.

Many logical attributes are introduced in the extension of actor signatures. The modification of their values according to the occurrence of the respective actions is defined by the following *valuation* axioms:

(**V1**) $\bigwedge_{c \in \Gamma_{l_b}} \forall \vec{v_c} \cdot \exists n_1 \cdot n_1.\mathbf{new}(c, n_2, \vec{v_c}) \rightarrow \mathbf{X}(n_2.init_c(\vec{v_c}) = \mathrm{T})$

(**V2**) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot \exists n_1 \cdot n_1.\mathbf{send}(c, n_2, \vec{v_c}) \rightarrow \mathbf{X}(n_2.sent_c(\vec{v_c}) = \mathrm{T})$

(**V3**) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot n.\mathbf{deliv}(c, \vec{v_c}) \rightarrow \mathbf{X}(n.sent_c(\vec{v_c}) = \mathrm{F} \wedge n.delivd_c(\vec{v_c}) = \mathrm{T})$

(**V4**) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot n.c(\vec{v_c}) \rightarrow \mathbf{X}(n.delivd_c(\vec{v_c}) = \mathrm{F})$

According to **V1**, if the creation of an actor has been requested, there will exist a new actor in the next instant. Moreover, axioms **V2** and **V3** say that if a message is dispatched, it will be buffered for output, and likewise the message will be removed from the output and transferred to the input buffer whenever it is delivered. Furthermore, each processed message will be subsequently removed from the input buffer as stated in axiom **V4**. Note that the delay in buffering messages, in the next instant only, rules out the existence of Zeno actors, which could receive, compute and reply infinitely fast.

Finally, *fairness* axioms are required to guarantee a correct collective behaviour. Without fairness, it could be the case that a message is not delivered even if the target actor is always willing to receive it, e.g., because of a transmission failure, and likewise that received messages are never consumed.

(**F1**) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot n.delivd_c(\vec{v_c}) = \mathrm{T} \wedge \mathbf{E}(n.c(\vec{v_c})) \rightarrow n.Fair(c(\vec{v_c}))$

(**F2**) $\bigwedge_{c \in \Gamma_{l-l_b}} \forall \vec{v_c} \cdot n.sent_c(\vec{v_c}) = \mathrm{T} \wedge \mathbf{E}(n.\mathbf{deliv}(c, \vec{v_c})) \rightarrow n.Fair(\mathbf{deliv}(c, \vec{v_c}))$

The first axiom says that, if the processing of a single message is obliged, because the message was delivered and has been locally buffered, and it is also enabled, i.e., possible, the message will be processed or else the actor will become always disabled for processing, unable to consume the pending message. *Mutatis mutandis*, this is what the second axiom says for message delivery. These axioms capture assumptions that can be classified in between those of perfect and initially perfect buffers as described by Koymans (1987).

A crucial simplification has been made here concerning message passing. We should have treated the fact that messages may be exchanged in sequence or concurrently and some of them could be lost or duplicated in this way. The usual treatment of this problem is to attach tags to messages so that they become distinct from each other. To avoid obliging the specifier to deal with such details, a logical treatment could have been defined here, much in the way that object naming is dealt with through auxiliary attributes. Details are omitted.

All the properties discussed above have already been stated in the literature on the actor model, e.g. by Clinger (1981), Hewitt and Baker (1977), despite the lack of a formally stated axiomatisation. Hereafter, we name the full set of logical axioms as $Ax \stackrel{\text{def}}{=} \{$**L1-4**, **O1-12**, **V1-4**, **F1-2**$\}$. The set $\overline{Ax}$, on the other hand, contains only the axioms with barred labels, wherein logical attribute symbols do not appear. The axiomatisation of the actor model allows us to derive the following more or less standard temporal logical rules for reasoning about the concurrent behaviour of object communities:

Proposition 12. (Derived Rules of Inference) Given an actor specification $\Phi = (\Delta, \Psi)$, $\Delta = (\Sigma, \mathcal{A}, \Gamma)$, the following inference rules are derivable for existing objects in the Φ community, provided that $\{k, n_1, n_2\} \subset \mathcal{V}_{\mathsf{addr}}$, p_1 and q are local state formulas parameterised by n_1 and p_2 is a local state formula parameterised by n_2:

(**EXIST**) $b \in \Gamma_{e_b}$

1. $p_2[k] \rightarrow \exists \vec{v_b} \cdot n_2.\mathbf{new}(b, k, \vec{v_b})$
2. $p_1[k] \rightarrow q \vee \bigvee_{c \in \Gamma_{l_b}} \exists \vec{v_c} \cdot n_1.\mathbf{new}(c, k, \vec{v_c})$

$$p_2[k] \rightarrow \mathbf{XG}(p_1[k] \rightarrow q)$$

(**SAFE**)

1. $\bigwedge_{b \in \Gamma_{l_b}} \forall \vec{v_b} \cdot n_1.b(\vec{v_b}) \rightarrow q$
2. $\bigwedge_{c \in \Gamma_e} \forall \vec{v_c} \cdot n_1.c(\vec{v_c}) \wedge q \rightarrow \mathbf{X}q$

$$\mathbf{G}q$$

(**INV**)

1. $\bigwedge_{c \in \Gamma_e} \forall \vec{v_c} \cdot n_1.c(\vec{v_c}) \wedge q \rightarrow \mathbf{X}q$

$$q \rightarrow \mathbf{G}q$$

(RESP)

1. $\bigwedge_{c \in \Gamma_c} \forall \vec{v_c} \cdot n_1.c(\vec{v_c}) \wedge p_1[\vec{v_d}] \rightarrow \mathbf{X}(p_1[\vec{v_d}] \vee n_1.d(\vec{v_d}))$
2. $n_1.d(\vec{v_d}) \rightarrow \mathbf{F}(q[\vec{v_d}])$
3. $p_1[\vec{v_d}] \rightarrow \mathbf{FE}(n_1.d(\vec{v_d}))$

$d \in \Gamma_{l-l_b}$

$$n_1.\mathbf{deliv}(d, \vec{v_d}) \rightarrow \mathbf{X}(\mathbf{F}(p_1[\vec{v_d}]) \rightarrow \mathbf{F}(q[\vec{v_d}]))$$

(COM)

1. $\bigwedge_{c \in \Gamma_c} \forall \vec{v_c} \cdot n_1.c(\vec{v_c}) \wedge p_1[\vec{v_d}] \rightarrow \mathbf{X}(p_1[\vec{v_d}] \vee n_1.\mathbf{deliv}(d, \vec{v_d}))$
2. $n_1.\mathbf{deliv}(d, \vec{v_d}) \rightarrow \mathbf{F}(q[\vec{v_d}])$
3. $p_1[\vec{v_d}] \rightarrow \mathbf{FE}(n_1.\mathbf{deliv}(d, \vec{v_d}))$

$d \in \Gamma_{e-e_b}$ and $d \in \Gamma_{l-l_b}$

$$n_2.\mathbf{send}(d, n_1, \vec{v_d}) \rightarrow \mathbf{X}(\mathbf{F}(p_1[\vec{v_d}]) \rightarrow \mathbf{F}(q[\vec{v_d}]))$$

(NRESP)

1. $p_2[n_1] \rightarrow \exists \vec{v_b} \cdot n_2.\mathbf{new}(b, n_1, \vec{v_b})$
2. $\exists \vec{v_b} \cdot n_1.b(\vec{v_b}) \rightarrow \forall v_d \cdot p_1[\vec{v_d}]$
3. $n_1.d(\vec{v_d}) \rightarrow q[\vec{v_d}]$
4. $\exists n \cdot n.\mathbf{deliv}(d, \vec{v_d}) \rightarrow \mathbf{X}(q[\vec{v_d}])$
5. $n_1.d(\vec{v_d}) \rightarrow \mathbf{X}(p_1[\vec{v_d}] \vee q[\vec{v_d}])$

$b \in \Gamma_{l_b \cap e_b}$

$d \in \Gamma_{l-l_b}$

$$p_2[n_1] \rightarrow \mathbf{XG}(p_1[\vec{v_d}] \rightarrow (\neg n_1.d(\vec{v_d}))\mathbf{W}(n_1.\mathbf{deliv}(d, \vec{v_d})))$$

(NCOM)

1. $p_2[n_1] \rightarrow \exists \vec{v_b} \cdot n_2.\mathbf{new}(b, n_1, \vec{v_b})$
2. $\exists \vec{v_b} \cdot n_1.b(\vec{v_b}) \rightarrow \forall v_d \cdot p_1[\vec{v_d}]$
3. $n_1.\mathbf{deliv}(d, \vec{v_d}) \rightarrow q[\vec{v_d}]$
4. $\exists n \cdot n.\mathbf{send}(d, n_1, \vec{v_d}) \rightarrow \mathbf{X}(q[\vec{v_d}])$
5. $n_1.\mathbf{deliv}(d, \vec{v_d}) \rightarrow \mathbf{X}(p_1[\vec{v_d}] \vee q[\vec{v_d}])$

$b \in \Gamma_{l_b \cap e_b}$, $d \in \Gamma_{e-e_b}$ and $d \in \Gamma_{l-l_b}$

$$p_2[n_1] \rightarrow \mathbf{XG}(p_1[\vec{v_d}] \rightarrow (\neg n_1.\mathbf{deliv}(d, \vec{v_d}))\mathbf{W}(\exists n \cdot n.\mathbf{send}(d, n_1, \vec{v_d})))$$

The rules above can be derived using the axiomatisation of the branching time logic and our logical axioms about the actor model. These rules are more convenient to use because the logical attributes have been eliminated. Rule **EXIST**, based on the fact that a name cannot be reused once it is given to some actor, guarantees a local safety property from the configuration of the actors in the environment. **SAFE** and **INV** are the usual rules for verifying safety and invariance properties. Rules **COM** and **RESP** capture the fairness requirements on actor behaviours. They should be applied to verify that the consequences of delivering or consuming a message are eventually obtained whenever the recipient actor becomes enabled often enough to guarantee the occurrence of the respective event. The slightly more complex rules for absence of communication and response, **NCOM** and **NRESP**, respectively, need to be ground on the creation of new actors since our axiomatisation admits initially present messages addressed to originally existing objects. Their conclusions are that, once the actor is created, whenever there are no pending messages for delivery or processing, messages will be delivered or consumed

only if preceded by the occurrence of their triggering events. All these inference rules may be simplified by a careful instantiation of the adopted schematic variables.

Let us illustrate the application of our specific proof calculus to the verification of local properties of individual actors. From the BUFFERCELL specification, it is easy to see that once a cell is created, it may be consumed or linked to another cell of the buffer afterwards. If a cell has been consumed and it is not the last element of the list, the cell will never perform such local computations again. Hence, if there exists a subsequent buffer element, the cell will send forward every incoming message. Assuming familiarity with temporal logic, this is stated and verified as follows:

$$\vdash_{\text{BUFFERCELL}} void = \mathrm{T} \wedge lst = \mathrm{F} \rightarrow \mathbf{G}(\neg cons \wedge \neg link(n)) \tag{1}$$

Proof.

1.	$ready \wedge void = \mathrm{T} \rightarrow \mathbf{X}(void = \mathrm{T})$	(1.5)
2.	$cons \wedge void = \mathrm{T} \rightarrow \mathbf{X}(void = \mathrm{T})$	(1.6)
3.	$link(n) \wedge void = \mathrm{T} \rightarrow \mathbf{X}(void = \mathrm{T})$	(1.8)
4.	$void = \mathrm{T} \rightarrow \mathbf{G}(void = \mathrm{T})$	**INV** 1, 2, 3
5.	$cons \rightarrow \mathbf{X}((\neg cons)\mathbf{W}(get(n) \wedge void = \mathrm{F} \wedge \neg cons))$	(1.14), **DEF** $\leftarrow$
6.	$(\neg cons)\mathbf{W}(get(n) \wedge void = \mathrm{F} \wedge \neg cons) \rightarrow (\mathbf{G}(void = \mathrm{T}) \rightarrow \mathbf{G}(\neg cons))$	**DEF W**, **UIF**, bool Ax
7.	$cons \rightarrow \mathbf{XG}(void = \mathrm{T})$	2, 4, **G**-$\mathcal{I}$, **MON-GX**
8.	$cons \rightarrow \mathbf{XG}(\neg cons)$	5, 6, 7, **G**-$\mathcal{I}$, **MON-G**
9.	$\mathbf{beg} \rightarrow (\neg cons)\mathbf{W}(get(n) \wedge void = \mathrm{F} \wedge \neg cons)$	(1.14), **DEF** $\leftarrow$
10.	$\mathbf{beg} \rightarrow \mathbf{G}(void = \mathrm{T} \rightarrow \neg cons)$	9, **UIF**, **TRAN W**, **DEF G**
11.	$void = \mathrm{T} \rightarrow \neg cons$	**beg**-$\mathcal{E}$ 10
12.	$\mathbf{G}(void = \mathrm{T}) \rightarrow \mathbf{G}(\neg cons)$	**G**-$\mathcal{I}$ 11, **MON-G**
13.	$get(n) \wedge void = \mathrm{F} \rightarrow \mathbf{X}(void = \mathrm{F} \wedge cons)$	(1.11), **O12**, **L1**
14.	$get(n) \wedge void = \mathrm{F} \rightarrow \mathbf{G}(void = \mathrm{T} \rightarrow \neg cons)$	13, **MON-GX**, **FIX-G**
15.	$void = \mathrm{T} \wedge lst = \mathrm{F} \rightarrow \mathbf{G}(\neg cons)$	12, 14, **MON-G**

using *modus ponens*, temporal generalisation (**G**-$\mathcal{I}$) and **beg**-elimination (**beg**-$\mathcal{E}$) as inference rules of the underlying logical system and the definition (**DEF**), monotonicity (**MON**), transitivity (**TRAN**) and fixed point characterisation (**FIX**) of some connectives, apart from the replacement of until by the eventual occurrence of its second argument (**UIF**). In a similar way, it can be shown that $\vdash_{\text{BUFFERCELL}} void = \mathrm{T} \wedge lst = \mathrm{F} \rightarrow \mathbf{G}(\neg link(n))$. Conjoining these partial results and using the fact that $\mathbf{G}p \wedge \mathbf{G}q \rightarrow \mathbf{G}(p \wedge q)$, we conclude that the property above is derivable. □

3.5. Composing actor specifications

In Section 3.1 we discovered that, to give an account of what is usually considered to be a complex component in the actor model, we need at least to be able to put distinct signatures together to represent the linguistic structure of yet another component or an entire system. More generally, the view that complex descriptions should be defined in terms of simpler descriptions put together has been developed within the theory of Institutions by Goguen and Burstall (1992) and requires the definition of basic entities to be regarded as design units. In our case, they will be actor specifications.

It is also necessary to provide means of connecting object descriptions to each other. Traditionally, in a proof-theoretic approach to design, this is achieved by providing translations between the languages of the related theories (Maibaum and Turski 1984). If a symbol-to-symbol mapping, i.e., a morphism, between two actor signatures is given, the existence of a compositional relation of translation between the respective languages can be guaranteed.

Definition 13. (Actor Signature Morphism) Given two actor signatures $\Delta_1 = (\Sigma_1, \mathcal{A}_1, \Gamma_1)$ and $\Delta_2 = (\Sigma_2, \mathcal{A}_2, \Gamma_2)$, an *actor signature morphism* $\tau : \Delta_1 \to \Delta_2$ consists of:

— a morphism of algebraic structures $\tau_v : \Sigma_1 \to \Sigma_2$ such that $\tau_v(\mathsf{addr}_1) = \mathsf{addr}_2$;

— for each $f \in \mathcal{A}_{1_{\langle s_1,\ldots,s_n\rangle,s}}$, an attribute symbol $\tau_\alpha(f) : \tau_v(s_1) \times \ldots \times \tau_v(s_n) \to \tau_v(s)$ in $\mathcal{A}_2$;

— for each $c \in \Gamma_{1_{\langle s_1,\ldots,s_n\rangle}}$, an action symbol $\tau_\gamma(c) : \tau_v(s_1) \times \ldots \times \tau_v(s_n)$ in Γ_2 such that: (i) $\tau_\gamma(\Gamma_{c_1}) \subseteq \Gamma_{c_2}$; (ii) $\tau_\gamma(\Gamma_{e_{b_1}}) \subseteq \Gamma_{e_{b_2}}$; (iii) $\tau_\gamma(\Gamma_{e_1-e_{b_1}}) \subseteq \Gamma_{e_2-e_{b_2}}$; (iv) $\tau_\gamma(\Gamma_{l_{b_1}}) \subseteq \Gamma_{l_{b_2}}$; (v) $\tau_\gamma(\Gamma_{l_1-l_{b_1}}) \subseteq \Gamma_{l_2-l_{b_2}}$ and (vi) $\tau_\gamma(\Gamma_{l_1-e_1}) \subseteq \Gamma_{l_2-e_2}$.

It is straightforward to provide a compositional definition for the translation of classifications, terms, formulas and sets thereof under τ.

Translations that necessarily relate the distinguished symbol of each signature, as defined above concerning addr, have been called pointed morphisms in the literature (Parisi-Presicce and Pierantonio 1994). Since renaming is possible in translating the other signature symbols, morphisms capture the relabelling operation proposed by Agha (1986) to equalise identifiers in distinct descriptions. In addition, it is possible to use signature morphisms to allow some external symbols, members of Γ_e, to become local as well. This stems from the fact that, in a complex configuration, there may be events required from the environment of a component which are not provided by the environment of the whole configuration, because they are ensured by another component of the same configuration. It is not difficult to see that any given actor signature morphism induces other morphisms between the corresponding extended and parameterised signatures, by translating their additional symbols according to the way the original symbols are translated by the given morphism. This means that the specifier, in defining a morphism to connect two signatures, does not need to be concerned with the new symbols introduced in their extension or parameterisation.

We would like to always be able to combine any finite number of actor signatures so as to ensure the necessary structure to support interaction. This can be accomplished if we can show that actor signatures and morphisms determine a finitely co-complete category:

Theorem 14. (Category of Actor Signatures) Actor signatures and morphisms constitute a finitely co-complete category $\mathbf{Sig}^{\mathbf{Act}}$.

Proof. To ensure that we have a category, we must show that identities exist and composition is associative. Considering that morphisms are set-valuated functions, the only difficulty that may arise in verifying the existence of identity is due to the non-disjoint sets of action symbols. But, for $\Delta \stackrel{\mathbf{id}}{\to} \Delta$, if $c \in \Gamma_{e_b}$, (a) $\mathbf{id}(c) \in \Gamma_{e_b}$, from (ii) and (iii) in the definition of signature morphisms. Now, if it is also the case that $c \in \Gamma_{l_b}$, (b) $\mathbf{id}(c) \in \Gamma_{l_b}$, from (iv) and (v). Due to (a) and (b), $\mathbf{id}(c) \in \Gamma_{e_b \cap l_b}$ whenever $c \in \Gamma_{e_b \cap l_b}$. The

same argument applies to any $c \in \Gamma_{(e-e_b)\cap(l-l_b)}$ and therefore $\mathbf{Sig^{Act}}$ admits identity, the constant function on sets. The associativity of signature morphisms follows directly from their set-theoretic definition.
The initial element of this category is $\Delta_\perp = ((\{\mathsf{addr}\}, \{\}), \{\}, \{\})$. Given a pair of morphisms $\Delta \xrightarrow{\tau_1} \Delta_1$, $\Delta \xrightarrow{\tau_2} \Delta_2$, their pushout is defined up to isomorphism by any pair of morphisms $\Delta_1 \xrightarrow{\tau_{1'}} \Delta'$, $\Delta_2 \xrightarrow{\tau_{2'}} \Delta'$ such that $S' = \tau_{1'}(S_1) \oplus_{\overline{\tau}(S)} \tau_{2'}(S_2)$, $\Omega' = \tau_{1'}(\Omega_1) \oplus_{\overline{\tau}(\Omega)} \tau_{2'}(\Omega_2)$, $\mathcal{A}' = \tau_{1'}(\mathcal{A}_1) \oplus_{\overline{\tau}(\mathcal{A})} \tau_{2'}(\mathcal{A}_2)$ and $\Gamma' = \tau_{1'}(\Gamma_1) \oplus_{\overline{\tau}(\Gamma)} \tau_{2'}(\Gamma_2)$, where $\overline{\tau} = \tau_1' \circ \tau_1 = \tau_2' \circ \tau_2$. The existence of the initial element and pushouts is sufficient to guarantee finite co-completeness. □

Specification morphisms induced by the signature morphisms above do not capture the expected enrichment of object behaviour as usual in Institutions (Goguen and Burstall 1992). This happens because they do not translate our additional logical axioms, which are needed to guarantee a correct collective behaviour. This shows that such morphisms do not determine interpretations between theories. To support this, the following morphisms are used:

Definition 15. (Actor Specification Morphisms) Given two actor specifications $\Phi_1 = (\Delta_1, \Psi_1)$ and $\Phi_2 = (\Delta_2, \Psi_2)$, a *specification morphism* $\tau : \Phi_1 \to \Phi_2$ is a signature morphism lifted to sentences such that $\vdash_{\Phi_2} \tau(g)$ for every $g \in \Psi_1 \cup Ax_{\Phi_1}$.

The inclusion of the translated logical axioms $\tau(Ax_{\Phi_1})$ into Φ_2 is necessary as they represent properties which are not always a consequence of Ax_{Φ_2}, since some of these axioms rely on the existence of the original signature symbols only. Once the signature is augmented with new symbols using a morphism, the respective properties may fail to hold. The locality property, for instance, is not preserved by the translation, as shown by Fiadeiro and Maibaum (1992).

Our finite co-completeness result concerning the category of actor signatures easily lifts to categories of extended and parameterised signatures. Much in the same way, it can be transported to a category of actor specifications with the morphisms defined above:

Proposition 16. (Category of Actor Specifications) Actor specifications and morphisms constitute a finitely co-complete category $\mathbf{Spec^{Act}}$. □

A comparison between our notion of composability and that of Agha *et al.* (1997) and Talcott (1996b) is in order. Composition is realised here by computing co-limits, or pushouts in the particular case of two connected specifications. Given a set of specifications with their pairwise shared sub-components fixed, pushouts of specification morphisms are commutative and have $(\Delta_\perp, \{\})$ as their identity. In addition, all their possible compositions in any order are isomorphic among themselves, which yields associativity up to isomorphism. Nevertheless, these are the only similarities with their semantic notion. The composability notion in their work is dynamic and fails to put together components having in common identical names of existing actors. This is syntactically immaterial, though, since there is a canonical way of relating actor syntax and semantics, as hinted by Agha (1986) and followed here, obliging the composed specifications to entail configurations with disjoint sets of existing actor addresses. We treat the dynamic composition of actor components while developing rely-guarantee proofs, as outlined below.

We also need to compare the composition of actor specifications using the morphisms

Actor TERMINAL
data types addr, cmd
attributes bf : addr
actions
ter(addr) : **local** + **extrn birth**;
rd(cmd) : **local computation**;
tr(cmd) : **extrn message**
axioms n : addr; v : cmd
$ter(n) \rightarrow bf = n$ (2.1)
$\mathbf{FG}(\forall v \cdot \neg rd(v))$ (2.2)
$rd(v) \wedge bf = n \rightarrow \mathbf{X}(bf = n)$ (2.3)
$rd(v) \wedge bf = n \rightarrow \mathbf{X}(\mathbf{send}(tr, n, v))$ (2.4)
$\mathbf{send}(tr, n, v) \leftarrow rd(v) \wedge bf = n$ (2.5)
End

Actor PROCESSOR
data types addr, cmd (NEX, BEG : cmd)
attributes in, id : addr; prv : cmd
actions pro(addr, addr) : **local** + **extrn birth**;
exc(int) : **local computation**;
nop, rec(int) : **local** + **extrn message**;
req(addr) : **extrn message**
axioms n, p : addr; v : cmd
$pro(n, p) \rightarrow id = n \wedge in = p \wedge prv = \text{NEX}$ (3.1)
$pro(n, p) \rightarrow \mathbf{X}(exc(\text{BEG}) \wedge \mathbf{send}(nop, n))$ (3.2)
$exc(v) \rightarrow \mathbf{X}(prv = v)$ (3.3)
$exc(v) \wedge id = n \wedge in = p \rightarrow \mathbf{X}(id = n \wedge in = p)$ (3.4)
$nop \wedge id = n \wedge in = p \rightarrow \mathbf{X}(\mathbf{send}(req, p, n))$ (3.5)
$nop \wedge id = n \rightarrow \mathbf{X}(\mathbf{send}(nop, n))$ (3.6)
$rec(v) \rightarrow \mathbf{X}(exc(v))$ (3.7)
$exc(v) \leftarrow rec(v) \vee \exists n, p \cdot pro(n, p) \wedge v = \text{BEG}$ (3.8)
$\mathbf{send}(nop, n) \leftarrow id = n \wedge \exists p \cdot pro(n, p) \vee nop$ (3.9)
$\mathbf{send}(req, p, n) \leftarrow nop \wedge id = n \wedge in = p$ (3.10)
$\mathbf{E}(rec(v)) \rightarrow v \neq \text{NEX}$ (3.11)
$prv \neq \text{NEX} \rightarrow \mathbf{FE}(\mathbf{deliv}(nop)) \wedge \mathbf{FE}(nop)$ (3.12)
$prv \neq \text{NEX} \wedge v \neq \text{NEX} \rightarrow \mathbf{FE}(\mathbf{deliv}(rec(v)))$ (3.13)
$prv \neq \text{NEX} \wedge v \neq \text{NEX} \rightarrow \mathbf{FE}(rec(v))$ (3.14)
End

Fig. 2. Specification of terminals and processors.

above to the similar usage of categorical notions in Fiadeiro and Maibaum (1992). It is particularly important to mention that, because of the implicit parameterisation of actor signatures by a sort of mail addresses and the restricted use of logical action symbols to support interaction, it is not possible to express at the local level any form of extra-logical sharing of signature symbols. This means that at this point interaction is supported logically, always by the synchronised actions introduced in the extension of actor signatures, which may occur simply because the interaction is between actors belonging to the same community or, conversely, because they belong to distinct communities and the designer decided to define morphisms to support their interaction. This is in keeping with the local discipline imposed by the actor model, which precludes any form of interaction other than by object creation and asynchronous message passing.

3.6. A toy example

Using the constructions described in the previous section, we can now study communities of heterogeneous actors. A good example is obtained by composing a buffer as described in Section 2, a processor and a set of terminals in order to represent a uniprocessor time-sharing architecture. The intended behaviour of the respective component, which complies with a specification called UTSA, is to allow commands typed by terminal users to be always processed eventually. The specification of terminal and processor actors for this purpose appear in Figure 2.

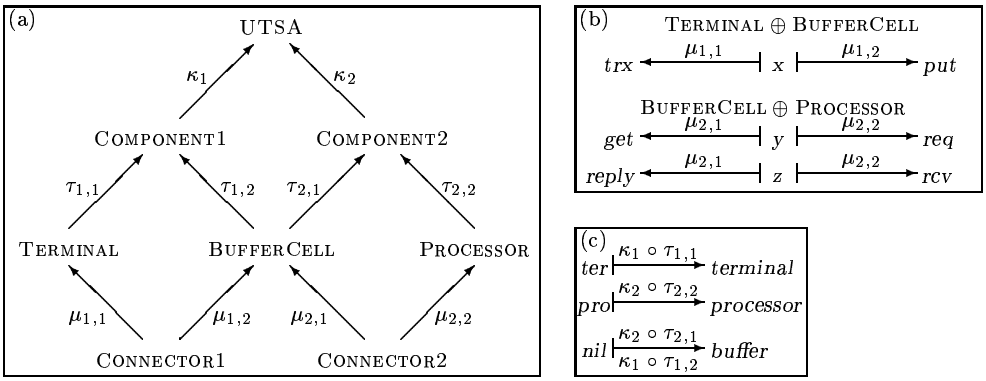

Fig. 3. Static configuration of the system.

A terminal becomes aware of the mail address of a cell which will serve as a buffer at creation time (2.1). Afterwards, the terminal always transmits typed commands to the buffer so that they can wait for processing (2.4). The reading capability of terminals, however, is finite according to (2.2). Processors, in turn, have a more complex behaviour since they have to request commands from the buffer at any possible occasion (3.5-3.6). Valid commands may always be eventually delivered to the processor after initialisation (3.13). That is, any command except NEX, which stands for a not executable command, can be delivered to the processor after this object becomes live executing the first BEG. Once received, any command is subsequently executed (3.7). The computation cycle of the processor alternates among doing nothing and processing the messages *nop* (3.5, 3.6), *rec* (3.7) and the local computation *exc* (3.3, 3.4). This cycle starts just after the occurrence of the actor birth denoted by the symbol *pro* (3.2).

Clearly, the presentations above cannot specify a single component unless the proper interconnections between them are provided. Morphisms establish "physical shared channels" to make message passing possible, as defined in Figure 3, part (a). COMPONENT1, COMPONENT2 and UTSA, which result from the composition of the three specifications, are all defined up to isomorphism by the pushout of the given morphisms. This means that any name for each of their symbols suffices as long as the symbols to be shared and only them are equalised. They are defined according to the two connectors and the morphisms in Figure 3, part (b). The signature of CONNECTOR1 contains one external message symbol only, x, which is mapped to the *tr* action of terminals and to the *put* action of buffers. CONNECTOR2 has two such symbols, y and z, which are mapped to *get* and *reply* at the buffer side and to *req* and *rec* at the processor side, respectively. The set of axioms in both specifications is empty. Assuming that the underlying algebraic morphisms map the sort symbol of mail addresses accordingly and associate integers to commands, these morphisms clearly satisfy the requirements of Definition 15.

If the uniprocessor time-sharing architecture described above is to present the outlined behaviour, that user commands are always processed eventually, we must stipulate under which circumstances this property is expected. Certainly, there are situations in

$$\mathbf{beg} \rightarrow \neg \mathit{Reach}(x, y) \tag{2}$$
$$k.\mathbf{new}(\mathit{nil}, x) \vee \exists v \cdot k.\mathbf{new}(\mathit{item}, x, v) \rightarrow \mathbf{X}(\mathit{Reach}(x, x)) \tag{3}$$
$$(k.\mathbf{new}(\mathit{nil}, x) \vee \exists v \cdot k.\mathbf{new}(\mathit{item}, x, v)) \wedge \mathit{Reach}(y, k) \rightarrow \mathbf{X}(\mathit{Reach}(y, x)) \tag{4}$$
$$k.\mathbf{new}(\mathit{nil}, x) \vee \exists v \cdot k.\mathbf{new}(\mathit{item}, x, v) \vee \mathit{Inv}(\mathit{Reach}(x, x)) \tag{5}$$
$$(k.\mathbf{new}(\mathit{nil}, x) \vee \exists v \cdot k.\mathbf{new}(\mathit{item}, x, v)) \wedge \mathit{Reach}(y, k) \vee x \neq y \wedge \mathit{Inv}(\mathit{Reach}(y, x)) \tag{6}$$

Fig. 4. Definition of *Reach*.

which this is not established. Assume that a finite number of terminals is connected to a single processor via a buffer. This is the minimal condition we require to ensure that the characteristic property makes sense. Without loss of generality, we postulate that there are exactly two terminals in this configuration. If other arbitrary objects apart from the processor could remove commands from the buffer, if this last component could ignore commands from a specific terminal indefinitely, the characteristic property would not be established. Considering such properties as part of a single rely-guarantee assertion, we can prove that the characteristic property is indeed obtained.

Adopting the translations of birth action symbols in Figure 3, part (c), the definition $\forall x : y \cdot p[x] \stackrel{\text{def}}{=} \forall x \cdot \mathit{Reach}(y, x) \rightarrow p[x]$ and a similar one for $\exists$, both based on the auxiliary flexible symbol *Reach* defined in Figure 4, we state the properties above as follows:

$$k.\mathbf{new}(\mathit{buffer}, n) \wedge k.\mathbf{new}(\mathit{processor}, m, m, n) \wedge k.\mathbf{new}(\mathit{terminal}, t_1, n) \wedge k.\mathbf{new}(\mathit{terminal}, t_2, n) \tag{7}$$
$$\mathbf{G}(\forall v \cdot \forall y : n \cdot y.\mathit{put}(v) \rightarrow \mathbf{XG}(\neg y.\mathit{put}(v))) \tag{8}$$
$$\mathbf{G}(\forall y \cdot (\exists v \cdot y.\mathbf{send}(\mathit{put}, n, v)) \rightarrow y = t_1 \vee y = t_2) \tag{9}$$
$$\mathbf{G}(\forall v \cdot \forall x, y : n \cdot \exists z : x \cdot z = y \vee (\neg x.\mathbf{send}(\mathit{put}, x.\mathit{nxt}, v))\mathbf{W}(y.\mathbf{send}(\mathit{put}, y.\mathit{nxt}, v))) \tag{10}$$

The formula (7) says that the buffer, processor and terminals are considered to be initially created and linked. This illustrates that the designer, in order to verify any global property, is required not only to define morphisms, allowing actors in different communities to share part of the same language, but also to assume the existence of some "logical shared channels", names which bind actors to each other and enable message passing. The formulas above say in addition that cells of the buffer can only consume each distinct command once (8) — a simplifying assumption which allows us to ignore dispatched messages containing identical commands — that *put* messages are dispatched to the initial buffer cell n solely by one of the two terminals (9), and that each cell dispatches a command to the subsequent buffer element only if all the previous cells of the buffer dispatched the same command in the past (10). The last two properties are static configuration constraints. It is important to mention that all these properties only make sense in an extension of UTSA where the axioms concerning the actor model do not affect the symbol *Reach* as an auxiliary flexible definition. This is obtained by assuming the existence of a functor mapping our logical system into the underlying branching time system in a way that translates such logical axioms into an extra-logical part of each theory presentation. We ignore such technicalities for the sake of simplicity.

The conjunction of the formulas above defines what we call an initialisation condition, corresponding to a property that must be true at some point in time to ensure the occurrence of another property from the subsequent instant onwards. In other words, it determines a temporal context in which we want to be sure that another property is always the case, possibly when also taking into account other dynamic constraints called rely conditions. We propose the following such condition for UTSA,

$$\forall y \cdot (\exists x : n \cdot x.get(y)) \rightarrow y = m \tag{11}$$

which should be true until we obtain a post-condition, if this even happens, provided that a pre-condition has already happened. Formula (11) means that the buffer is assumed to be only requested to send commands to the processor m. To capture the characteristic property of our architecture, we propose the following pre and post conditions:

$$\exists y \cdot y.rd(v) \wedge (y = t_1 \vee y = t_2) \wedge v \neq \text{NEX} \tag{12}$$

$$m.exc(v) \tag{13}$$

That is, an executable user command is read from some terminal (12) and, sometime in the future of this occurrence, the command is processed (13). Naming the extension of UTSA wherein *Reach* is defined as UTSA_1, we can formalise the intuition above putting the proposed formulas p_i, $7 \leq i \leq 13$, together in a single temporal assertion:

$$\vdash_{\text{UTSA}_1} \bigwedge \{p_i | i \in [7, 10]\} \rightarrow \mathbf{XG}(p_{12} \wedge (p_{11})\mathbf{W}(p_{13}) \rightarrow \mathbf{F}(p_{13})) \tag{14}$$

What is asserted in (14) is an instance of the so-called Fair Merge Problem. That is, the processing of sequences of commands from each user must be fair; in other words, that each of them must not have the completion of its execution indefinitely delayed. To understand the validity of this assertion, first note that the respectively linked buffer cells are organised as a reversed queue. Each cell either processes incoming messages or these are forwarded to the remainder of the buffer, because the cell has already been consumed or is not the last element of the queue, or else each message is ignored, because the entire buffer is empty. Now, because the buffer is required by the initialisation condition to receive messages from the two terminals only and these actors eventually stop producing commands according to (2.2), the buffer itself will always be finite in any behaviour, meaning that commands will be fairly stored in and retrieved from this component. Furthermore, since our rely assumption is that the buffer is hidden from the environment with respect to receiving *get* messages, only the processor will recurrently request commands and possibly receive a reply from the buffer. Each command dispatched by a terminal will be eventually processed in this way.

The assertion above is verified based on the definition of a (relative) well-founded relation. We propose an extension of UTSA_1 (named UTSA_2) and choose to include in this conservative extension the following definition concerning buffer cells:

$$R(n, x, y) \leftrightarrow Reach(n, x) \wedge Reach(n, y) \wedge y.nxt = x \wedge y.lst = \text{F} \tag{15}$$

In general, R does not define a well-founded relation. If we take into account just the cells related to the initial buffer element n as the first argument of R, a flexible well-founded relation is defined. In this context, R is first shown to be irreflexive, which means that no buffer cell is related by R to itself, and stable, meaning that cells remain related forever. These involve applications of **EXISTS**, **INV** and **SAFE**. Some results concerning R are also needed in the proof: that buffer cells reachable from n are prevented from being reachable from and related to any other fixed cell at the same time, and that R is both transitive and acyclic. These results are used in conjunction with **NRESP** and **NCOM** to show that R eventually stops changing, i.e., eventually new cells are never added to the buffer, and if we transverse any chain of cells a final element is always found. These properties guarantee the well-foundedness of R with n as the first argument.

The verification of (14) depends on the use of an inference rule for (relative) well-founded induction, **WELL**, admissible in first-order temporal logical systems (Emerson 1990) as is our case. A number of interaction properties is derived with the help of **COM** and **RESP** to obtain the premise of this rule: that executable commands dispatched by the buffer to the processor are eventually executed; that the processor keeps requesting commands from the buffer regularly; and that similar properties also hold concerning the interaction between terminals and buffer. We infer our induction assertion saying that whenever a valid command is delivered to a buffer cell x and the same cell always eventually receives requests from the processor, provided that just the processor is allowed to consume the contents of such cell or its sucessor, in the future either there is a cell in the buffer dispatching the deposited command to the processor or there is another cell y related to x for which the same property obtains. The conclusion of **WELL** leads to our rely-guarantee assertion. The detailed verification appears in (Duarte 1998).

4. Concluding remarks

In this paper, we have laid proof-theoretic foundations, in the form of a set of axioms and inference rules, to support the rigorous specification and verification of open distributed systems in terms of the actor model. In addition, we used notions from category theory to define interpretations between temporal theories and compose actor specifications in this way. The use of rely-guarantee constructions was also illustrated as a means of addressing the dynamic properties of actor components. Not disregarding model-theory but attributing to it an auxiliary role in the development process, we believe that we have defined a logical system which is useful in practice to design open distributed systems in a rigorous and modular manner. We have already illustrated in Duarte (1997) how to design mobile systems according to this approach. Other object-based logical systems do not provide built-in support to the essential features of the actor model.

It seems to be worthwhile investigating in the future systematic ways of applying the logical system defined here at many stages of the development process. So far, we have only addressed design but the refinement of open distributed systems in terms of the actor model appears to be promising too. Another direction for future research is to consider extending the model so as to treat other interesting properties of real distributed systems. We are currently studying the inclusion of reflection in this framework.

Acknowledgements

A preliminary version of this paper was presented at the Esprit WG Workshop *Modelage* at Certosa di Pontignano, Italy, in January, 1997. The author wishes to thank Carolyn Talcott for many helpful discussions on the actor model and the anonymous referees for their constructive criticism.

References

G. Agha, I. A. Mason, S. Smith, and C. Talcott (1997). A foundation for actor computation. *Journal of Functional Programming* **7** (1) 1–72.

G. Agha (1986). *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press.

P. America and F. de Boer (1996). Reasoning about dynamically evolving process structures. *Formal Aspects of Computing* **6** (3) 269–316.

H. Barringer (1987). The use of temporal logic in the compositional specification of concurrent systems. In: A. Galton (ed.), *Temporal Logics and their applications*, Academic Press 53–90.

W. D. Clinger (1981). *Foundations of Actor Semantics.* PhD thesis, Massachusetts Institute of Technology.

J. Darlington and Y. K. Guo (1995). Formalising actors in linear logic. In: D. Patel, Y. Sun, and S. Patel (eds.), *Proc. International Conference on Object-Oriented Information Systems (OOIS'94)*, Springer-Verlag 37–53.

C. H. C. Duarte (1997). A proof-theoretic approach to the design of object-based mobility. In: H. Bowman and J. Derrick (eds.), *Proc. 2nd IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'97)*, Chapman and Hall 37–53. Canterbury, UK.

C. H. C. Duarte (1998). *Proof-Theoretic Foundations for the Design of Extensible Software Systems.* PhD thesis, Department of Computing, Imperial College, London, UK.

H.-D. Ehrich, A. Sernadas, and C. Sernadas (1988). Objects, object types and object identity. In: H. Ehrig (ed.), Categorical Methods in Computer Science with Aspects from Topology, *Springer Verlag Lecture Notes in Computer Science* **334** 142–156.

H. Ehrig and B. Mahr (1985). *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, *EATCS Monographs in Theoretical Computer Science* **6**. Springer Verlag.

E. A. Emerson (1990). Temporal and modal logic. In: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, Elsevier 996–1072.

J. Fiadeiro and T. Maibaum (1992). Temporal theories as modularisation units for concurrent systems specification. *Formal Aspects of Computing* **4** (3) 239–272.

J. Fiadeiro and T. Maibaum (1997). Categorical semantics of parallel program design. *Science of Computer Programming* **28** (2–3) 111–138.

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

C. Hewitt and H. Baker (1977). Laws for communicating parallel processes. In: G. Bruce (ed.), *Information Processing 77: 7th IFIP World Congress*, Elsevier 987–992.

R. Koymans (1987). Specifying message passing systems requires extending temporal logic. In: *6th ACM Symposium on Principles of Distributed Computing*, ACM Press 191–204.

L. Lamport (1983). What good is temporal logic? In: R. E. A. Mason (ed.), *Information Processing 83: 9th IFIP World Congress*, Elsevier 657–668.

T. Maibaum and W. Turski (1984). On what exactly is going on when software is developed step-by-step. In: *Proc. 7th International Conference on Software Engineering (ICSE'84)*, IEEE Computer Society Press 525–533.

Z. Manna and A. Pnueli (1983). How to cook a temporal proof system for your pet language. In: *Proc. 10th Symposium on Principles of Programming Languages*, ACM Press 141–154.

F. Parisi-Presicce and A. Pierantonio (1994). An algebraic theory of class specification. *ACM Transactions on Software Engineering and Methodology* **3** (2) 166–199.

A. Sernadas, C. Sernadas, and J. F. Costa (1995). Object specification logic. *Journal of Logic and Computation* **5** (5) 603–630.

A. P. Sistla, E. M. Clarke, N. Francez, and A. R. Meyer (1984). Can message buffers be axiomatised in linear temporal logic? *Information and Computation* **63** 88–112.

C. Talcott (1996). An actor rewriting theory. In: 1st International Workshop on Rewriting Logic and its Applications, *North Holland Electronic Notes in Theoretical Computer Science* **4** 360–383.

C. Talcott (1996). Interaction semantics for components of distributed systems. In: E. Najm and J.-B. Stefani (eds.), *Proc. 1st IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'96)*, Chapman and Hall.

C. Talcott (1997). Composable semantic models for actor theories. In: M. Abadi and T. Ito (eds.), Theoretical Aspects of Computer Science: 3rd International Symposium (TACS'97), *Springer Verlag Lecture Notes in Computer Science* **1281** 321–364.

R. J. Wieringa, W. de Jonge, and P. Spruit (1995). Using dynamic classes and role classes to model object migration. *Theory and Practice of Object Systems* **1** (1) 61–83.