Verifiable and Executable Logic Specifications of Concurrent Objects in $\mathcal{L}_\pi$

Luís Caires and Luís Monteiro
{lcaires,lm}@di.fct.unl.pt
Departamento de Informática - Universidade Nova de Lisboa

Abstract. We present the core-$\mathcal{L}_\pi$ fragment of $\mathcal{L}_\pi$ and its program logic. We illustrate the adequacy of $\mathcal{L}_\pi$ as a meta-language for jointly defining operational semantics and program logics of languages with concurrent and logic features, considering the case of a specification logic for concurrent objects addressing mobile features like creation of objects and channels in a simple way. Specifications are executable by a translation that assigns to every specification a model in the form of a core-$\mathcal{L}_\pi$ program. We also illustrate the usefulness of this framework in reasoning about systems and their components.

1 Introduction

The "proof-search as computation" paradigm has inspired the design of several programming and specification languages [6]. In [1] we presented $\mathcal{L}_\pi$, a simple language that supports the reduction and state-oriented style of specification of mobile process calculi without compromising the relational style of specification typical of logic programming. In particular, we have shown that both the π-calculus [10] and a version of the logic of hereditary Harrop formulas [7] can be adequately encoded into $\mathcal{L}_\pi$. On the other hand, $\mathcal{L}_\pi$ can itself be encoded in classical linear logic, in such a way that successful computations correspond to proofs [3]. As illustrated in [2], $\mathcal{L}_\pi$ suits itself well as a specification language of the operational semantics of programming languages with concurrent and logic features; we also proposed $\mathcal{L}_\pi$ as a meta-language for the compositional definition of operational semantics and program verification logics of languages of such a kind. Herein, we pursue this general directions, picking the case of declarative object-oriented specification of concurrent systems.

MOTIVATION. Several embeddings of object-oriented constructs into variants of π-calculus have been presented in the literature [12, 16, 14]. However [11], the π-calculus, while supporting the basic features of concurrency, mobility, abstraction and creation of processes and names, is a low level calculus, so the question arises about what kind of idioms are more suitable to model *and* reason about object systems and their components. In this setting, most of the current approaches focus on operational semantics or types, and the object languages have a definite operational character. In this paper, we develop a contribution in this general theme, stressing the instrumental use of the $\mathcal{L}_\pi$ framework in the spirit stated above. More specifically, we consider the case of a declarative specification language and program logic for concurrent objects that provides a unified scenario for specifying and reasoning about systems. This is achieved in the context

of a well-defined operational semantics that renders specifications executable and makes essential use of the reductive and deductive features of $\mathcal{L}_\pi$.

OVERVIEW. First, we define core-$\mathcal{L}_\pi$, a small yet expressive fragment of $\mathcal{L}_\pi$, and its program logic $\mu\mathcal{L}_\pi$. The logic $\mu\mathcal{L}_\pi$ is based on a many-sorted first-order version of the μ-calculus enriched with connectives enabling the implicit expression of component location and a hiding quantifier. For instance, a property $\varphi \otimes \psi$ holds of a system with two components that satisfy respectively φ and ψ, and $\Sigma_n\varphi$ holds of a system that satisfies φ in a context where a name n is private. Then, we present a specification logic for concurrent objects that actually reduces to an idiom of $\mu\mathcal{L}_\pi$. From the viewpoint of logical specification, our proposal extends existing ones (for instance, [15]) by addressing mobile features like dynamically changing number of components and creation of private channels in an intrinsic way, while supporting both local and global reasoning by relying on the monoidal structure induced by $\otimes$. Moreover, specifications are executable by means of a syntax-directed encoding that assigns to each specification a model in the form of a core-$\mathcal{L}_\pi$ program. This translation relies on a combination of techniques from process calculi and proof-theoretic approaches to logic programming. We also illustrate the usefulness of this framework in reasoning about systems and their components by giving some examples.

OBJECT MODEL. We consider a standard asynchronous model of concurrent objects as closed entities that exhibit computational behaviour by performing mutually exclusive actions over a private state. Actions are triggered by serving request messages from the environment, but are only allowed to ocurr under certain enabling conditions. Actions can change the object's internal state, cause the sending of action requests to other objects or even the creation of new objects or channels, and display internal non-determinism. Thus, any specification of the behaviour of an object must define conditions under which actions can occur, and also what are their effects.

SPECIFICATION LANGUAGE. Since [13], many authors adopted subsets of some temporal logic (TL) as a specification language for concurrent systems. But unrestricted use of TL often leads to unimplementable specifications, and typical approaches pick some restricted class of formulas as dynamic axioms. For instance in [4], one finds safety formulas like $\varphi \wedge A^* \Rightarrow \mathsf{X}\psi$ and liveness formulas like $\varphi \Rightarrow \mathsf{F}A^*$ - where A^* denotes the occurrence of action A, and X and F are respectively the *next* and *eventually* operators of (linear) TL. Concerning liveness properties, in our asynchronous model we will not be able to assert such a strong assertion, but just something like the weaker $\varphi \Rightarrow \mathsf{X}mA^*$ where mA^* asserts the existence of a pending request for action A. Then, whether mA^* implies $\mathsf{F}A^*$ depends on fairness assumptions and on the actual enabling conditions of action A. In summary, we adopt the (branching time) logic inherited from $\mu\mathcal{L}_\pi$ and consider specifications of object classes that enumerate instance attributes, and include dynamic axioms of the form $\varphi \Rightarrow \langle[A]\rangle\,\psi$ and static axioms defining the meaning of non-logical symbols.

STRUCTURE OF THE PAPER. In Sections 2-3 we introduce core-$\mathcal{L}_\pi$ and its program logic. In Section 4, we introduce the object specification language and

present its interpretation in $\mathcal{L}_\pi$ in Section 5. After discussing some properties of this interpretation in Section 6, we present in Section 7 an example of reasoning about a system.

2 The core-$\mathcal{L}_\pi$ Language

Here we introduce core-$\mathcal{L}_\pi$, a small yet expressive fragment of $\mathcal{L}_\pi$ [1, 2].
SYNTAX. Given a set $\mathcal{V}$ of variables, the set $\mathcal{T}(\mathcal{V})$ of untyped terms over $\mathcal{V}$ is defined inductively as follows: variables are terms, if x is a variable and $t_1, \cdots, t_n$ are terms then $x(t_1, \cdots, t_n)$ is also a term t (s.t. $head(t) = x$). In general, we will use x, y, z, i, j for variables, a, b, c, t, u, v for terms, $\tilde{x}$ for a list of distinct variables and $\tilde{t}$ for a sequence $(t_1, \cdots, t_n)$ or a multiset $\{\!|t_1, \ldots, t_n|\!\}$ of terms, depending on context. Sometimes, we will write $x.t$ for $x(t)$ and call *names* to variables. Untyped agents of core-$\mathcal{L}_\pi$ are defined by

$$\mathcal{P} ::= \mathbf{0} \mid a \mid \mathcal{P}|\mathcal{P} \mid \nu x\mathcal{P} \mid \tilde{x} \triangleright \tilde{a}[\mathcal{P}]\mathcal{P} \mid !\tilde{x} \triangleright \tilde{a}[\mathcal{P}]\mathcal{P}$$

Inaction $\mathbf{0}$ denotes the well-terminated process. An atom a can be seen either as an elementary message to be sent asynchronously or as a data element in the shared state. $p|q$ stands for the parallel composition of p and q; we sometimes write $\tilde{m}$ for $m_1|\cdots|m_k$ when the m are atoms. Restriction νx induces generation of private names; $\nu x p$ binds x in p. To explain the behaviour of the input prefix $\tilde{x} \triangleright \tilde{a}[g]p$ we start by considering the particular case where the test g is $\mathbf{0}$, which we abbreviate by $\tilde{x} \triangleright \tilde{a}[]p$. The agent $\tilde{x} \triangleright \tilde{a}[]p$ waits for a message $\tilde{b}$ (a multiset of terms) and then behaves like $\sigma(p)$, if there is a substitution σ with domain $\tilde{x}$ such that $\sigma(\tilde{a}) \equiv \tilde{b}$. In general, in addition to receiving the message, $\tilde{x} \triangleright \tilde{a}[g]p$ must also perform successfully the test $\sigma(g)$. Roughly, a test g succeeds if there is an atomically encapsulated computation sequence starting with g and reaching a success state (defined below). The replicable agent $!\tilde{x} \triangleright \tilde{a}[g]p$ behaves just like it's non-replicable version, except that it does not consume itself upon reduction. In input prefix agents, the input variables $\tilde{x}$ must have free occurrences in either $\tilde{a}$ or g, although not in the head of some a. Note however that in non-replicable input prefix, $\tilde{a}$ may be empty, this corresponds to the testing agent of [1]. In both forms, $\tilde{x}$ may be empty, in which case the prefix $\tilde{x}\triangleright$ may be ommited. For agents, we will use normally p, q, r, g.
TYPING. Here we introduce a simple type system for core-$\mathcal{L}_\pi$. Assume given a set Ω of primitive sorts, containing at least o, and a set Δ of primitive basic types containing say nat, bool. Basic types δ and types τ are given by

$$\delta ::= \Delta \mid (\delta_1, \cdots, \delta_k)\delta \qquad \tau ::= \Omega \mid \delta \mid (\tau_1, \cdots, \tau_k)\tau$$

A type that is not basic is called a sort. The definition of types and sorts is stratified in such a way that no term of basic type δ can contain a subterm of some sort. The motivation for this stratification is to limit scope extrusion to terms of sort kind. Signatures are partial maps from variables to types. We use Σ, Ξ, Γ for signatures, and assume that $x \notin \Sigma$ when writing $\Sigma, x{:}\tau$. Terms and

$$\frac{\Sigma \vdash h : (\tau_1, \cdots, \tau_k)\beta \quad \Sigma \vdash t_i : \tau_i}{\Sigma \vdash h(t_1, \cdots, t_k) : \beta}$$

$$\Sigma \vdash \mathbf{0} \qquad \frac{\Sigma \vdash h(t_1, \cdots, t_k) : o}{\Sigma \vdash h(t_1, \cdots, t_k)} \qquad \frac{\Sigma \vdash p \quad \Sigma \vdash q}{\Sigma \vdash p|q}$$

$$\frac{\Sigma, x : \tau \vdash p}{\Sigma \vdash \nu x : \tau p} \qquad \frac{\Sigma, \tilde{x} : \tilde{\tau} \vdash g \quad \Sigma, \tilde{x} : \tilde{\tau} \vdash p \quad \Sigma, \tilde{x} : \tilde{\tau} \vdash a_i : o}{\Sigma \vdash \tilde{x} : \tilde{\tau} \triangleright \tilde{a}[g]p \;\mid\; \Sigma \vdash !\tilde{x} : \tilde{\tau} \triangleright \tilde{a}[g]p}$$

Fig. 1. Typing of terms and agents.

agents are typed by the rules in Figure 1. We will refer by $\mathcal{T}_\tau(\Sigma)$ the set of terms t such that $\Sigma \vdash t : \tau$ is valid. Note that binding occurrences of variables in νx and $\tilde{x}\triangleright$ are now explicitly assigned a type. In the rule for restriction, τ must be a sort and, in the rules for input prefix agents, any variable x not occurring free in $\tilde{a}$ must be assigned a basic type. If $\Sigma \vdash p$ is valid, we will say that $\Sigma; p$ is a well-typed agent, bringing explicit the typing context. For well-typed agents we will use P, Q, R, and write $\mathcal{P}$ for the set of all well-typed agents.

REDUCTION. The operational semantics of $\mathcal{L}_\pi$ is defined by a relation of reduction between agents. Following [8], we first define a structural congruence relation $\cong$ that abstracts away from irrelevant notational distinctions. $\cong$ is the smallest congruence relation over agents containing α-conversion and closed under equations (i) $p|q \cong q|p$, (ii) $(p|q)|r \cong q|(p|r)$, (iii) $p|\mathbf{0} \cong p$, (iv) $\nu x(p|q) \cong \nu x p|q$ if x is not free in q, (v) $\nu x \mathbf{0} \cong \mathbf{0}$, (vi) $!r \cong !r|!r$ and (vii) $!r|\tilde{x} \triangleright \tilde{a}[q]r \cong !r|\tilde{x} \triangleright \tilde{a}[q|!r]r$ if $\tilde{x}$ are not free in p and $!r$ stands for some replicable input prefix agent. Reduction is the least relation $P \to Q$ between well-typed agents P and Q such that

$$\begin{array}{ll}
\Sigma; \nu x : \tau p \to \Sigma; \nu x : \tau q & \text{if } \Sigma, x : \tau; p \to \Sigma, x : \tau; q \\
\Sigma; \tilde{x} \triangleright [g]p|q \to \Sigma; \sigma(p)|q & \text{if } \Sigma; \sigma(g) \xrightarrow{*} \surd \\
\left.\begin{array}{l} \Sigma; \tilde{m}|\ \tilde{x} : \tilde{\tau} \triangleright \tilde{a}[g]p|q \to \Sigma; \sigma(p)|q \\ \Sigma; \tilde{m}|!\tilde{x} : \tilde{\tau} \triangleright \tilde{a}[g]p|q \to \Sigma; \sigma(p)|!\tilde{x} : \tilde{\tau} \triangleright \tilde{a}[g]p|q \end{array}\right\} & \text{if } \Sigma; \sigma(g) \xrightarrow{*} \surd \wedge\, \tilde{m} \equiv \sigma(\tilde{a})
\end{array}$$

Here σ assigns to each $x : \tau$ a term $t \in \mathcal{T}_\tau(\Sigma)$ and $\surd$ (a success state) is any agent of the form $\Xi; \nu\tilde{y} : \tilde{\tau}(!\mathbf{p})$, where $!\mathbf{p}$ is a composition of a (possibly null) number of replicating agents. As usual, we will refer by $\xrightarrow{*}$ the transitive-reflexive closure of $\to$. Subject reduction holds: if $P \in \mathcal{P}$ and $P \to Q$ then also $Q \in \mathcal{P}$.

LABELLED TRANSITION SYSTEM. To capture the compositional behaviour of agents, an "open" version of the reduction relation must be defined that captures not only the behaviour of agents in isolation but also the behaviour they can exhibit with adequate cooperation of the environment. As usual, this is formalised by means of an action-labelled transition system (LTS), where actions may be output ($\uparrow a$), input ($\downarrow a$) or silent (τ). Several possibilities arise at this point, we must require at least soundness, that is, agreement of $\xrightarrow{\tau}$ with reduction. Additionally, we can also impose a notion of relevance for $\mathcal{P}$, more precisely, that whenever $P \xrightarrow{\downarrow a} Q$ there is $P' \in \mathcal{P}$ such that $P' \xrightarrow{\uparrow a} Q'$, $P|P' \in \mathcal{P}$ and $P|P' \to Q|Q'$, and likewise when $P \xrightarrow{\uparrow a} Q$.

$$\Sigma; a \xrightarrow{\uparrow a} \Sigma; \mathbf{0} \text{ if } head(a) \notin Def \qquad \Sigma; a \xrightarrow{\downarrow !\tilde{x}:\tilde{\tau} \triangleright b[]p} \Sigma; \rho(p) \text{ if } \rho(b) \equiv a \wedge head(a) \in Def$$

$$\Sigma; !p \xrightarrow{\uparrow !p} \Sigma; !p \qquad \frac{\Sigma, \tilde{y} : \tilde{\gamma}; !\mathbf{q}|\sigma(g) \xrightarrow{*} \surd}{\Sigma; \tilde{x} : \tilde{\tau} \triangleright \tilde{a}[g]p \xrightarrow{\downarrow(\tilde{y}:\tilde{\gamma})\sigma(\tilde{a})!\mathbf{q}} \Sigma, \tilde{y} : \tilde{\gamma}; \sigma(p)}$$

$$\frac{\Sigma, x : \tau; p \xrightarrow{a} \Sigma', x : \tau; q}{\Sigma; \nu x : \tau p \xrightarrow{a} \Sigma'; \nu x : \tau q} \text{ if } x \notin a \qquad \frac{\Sigma, x : \tau; p \xrightarrow{\uparrow a} \Sigma'; q}{\Sigma; \nu x : \tau p \xrightarrow{\uparrow(x:\tau)a} \Sigma'; q} \text{ if } x \in a \wedge x \notin head(a)$$

$$\frac{\Sigma; p \xrightarrow{\uparrow(\tilde{x}:\tilde{\tau})a} \Sigma, \tilde{x} : \tilde{\tau}; p' \quad \Sigma; q \xrightarrow{\downarrow(\tilde{x}:\tilde{\tau})a \circledast \downarrow c} \Sigma, \Gamma, \tilde{x} : \tilde{\tau}; q'}{\Sigma; p|q \xrightarrow{\downarrow c} \Sigma, \Gamma; \nu \tilde{x} : \tilde{\tau}(p'|q')}$$

$$[s] \ \frac{\Sigma; p \xrightarrow{a} \Sigma'; p'}{\Sigma'; p|q \xrightarrow{a} \Sigma'; p'|q} \qquad \frac{\Sigma; p \xrightarrow{\uparrow a} \Sigma, \Gamma; p' \quad \Sigma; q \xrightarrow{\uparrow c} \Sigma, \Theta; q'}{\Sigma; p|q \xrightarrow{\uparrow a \circledast \uparrow c} \Sigma, \Gamma, \Theta; p'|q'}$$

Fig. 2. LTS specification for core-$\mathcal{L}_\pi$

ACTIONS. Σ-actions are objects of the forms $\uparrow(\tilde{x} : \tilde{\tau})\tilde{a}!\mathbf{p}$ (output action) or $\downarrow(\tilde{x} : \tilde{\tau})\tilde{a}!\mathbf{p}$ (input action) where $\tilde{x} : \tilde{\tau}$ is a possibly empty signature, each a_i is an atom and each $!p$ a replication such that $\Sigma, \tilde{x} : \tilde{\tau} \vdash a_1 | \cdots | a_n | !p_1 | \cdots | !p_m$ is valid. To grasp the meaning of the binding signature, just consider it a generalisation of π-calculus bound input and bound output actions, in our notation, $\overline{x(y)}$ would be rendered by $\uparrow(y)x(y)$ and $x(y)$ by $\downarrow(y)x(y)$. When all components $\tilde{x}$, $\tilde{a}$ and $!\mathbf{p}$ of an action are void, both $\uparrow$ and $\downarrow$ will be noted τ (the silent action). To highlight the typing context we may write $\Sigma; a$ for Σ-actions a. The set of all actions will be noted Act, and individual actions by a, b, c. We will also define $head((x : \tau)\tilde{a}!\mathbf{h}.\mathbf{p})$ as the set of heads of all $\tilde{a}$ and all $\tilde{h}$. Actions a and b of the same polarity compose into another action $a \circledast b$; composition of $(\tilde{x} : \tilde{\tau})\tilde{a}!\mathbf{p}$ with $(\tilde{y} : \tilde{\gamma})\tilde{c}!\mathbf{q}$ is defined as $(\tilde{x} : \tilde{\tau}, \tilde{y} : \tilde{\gamma})\tilde{a}\tilde{c}!\mathbf{pq}$. Intuitively, $a \circledast b$ is the concurrent execution of a and b; $\circledast$ is commutative, associative and has τ for unit.

LTS SPECIFICATION. In Figure 2, we present a LTS specification for core-$\mathcal{L}_\pi$. Most rules have the form one expects. Substitution σ assigns each $x : \tau$ a term in $\mathcal{T}_\tau(\Sigma, \tilde{y} : \tilde{\gamma})$, and ρ a term in $\mathcal{T}_\tau(\Sigma)$. When we are interested in computations starting from agents $\Gamma; p$ with Γ a given global signature, we let $Def \subset \Gamma$ and define two transitions for an agent $\Sigma; a$, to distinguish the case where a is to be substituted in place by it's definition (when $head(a) \in Def$), from the case where a is to be sent to the environment. This specification defines a sound and relevant transition relation $\xrightarrow{a}$ for all of $\mathcal{P}$, and should be taken as the reference transition relation. Note however that when developing applications of core-$\mathcal{L}_\pi$ (for instance, when studying certain restricted idioms as in this paper) we might be interested just in some subset of $\mathcal{P}$. Since $\xrightarrow{a}$ might not be relevant for that subset, additional provisos may be placed occasionally in some of the rules above, in order to obtain relevance for the fragment under consideration.

$$
\begin{array}{ll}
\Sigma; p \models_{\mathbf{I},v} P(\tilde{t}) & \text{if } \tilde{t} \in \mathbf{I}_\Sigma(P) \\
\Sigma; p \models_{\mathbf{I},v} t \doteq t & \\
\Sigma; p \models_{\mathbf{I},v} \neg\varphi & \text{if not } \Sigma; p \models_{\mathbf{I},v} \varphi \\
\Sigma; p \models_{\mathbf{I},v} \varphi \wedge \psi & \text{if } \Sigma; p \models_{\mathbf{I},v} \varphi \text{ and } \Sigma; p \models_{\mathbf{I},v} \psi \\
\Sigma; p \models_{\mathbf{I},v} \forall x : \tau\varphi & \text{if } \Sigma; p \models_{\mathbf{I},v} \varphi\{^x/_t\} \text{ for all } t \in \mathcal{T}_\tau(\Sigma) \\
\Sigma; p \models_{\mathbf{I},v} \langle S \rangle \varphi & \text{if } \exists (\Sigma; a) \in S \exists (\Sigma'; q) \in \mathcal{P}(\Sigma; p \xrightarrow{a} \Sigma'; q \text{ and } \Sigma'; q \models_{\mathbf{I},v} \varphi) \\
\Sigma; p \models_{\mathbf{I},v} X & \text{if } \Sigma; p \in v(X) \\
\Sigma; p \models_{\mathbf{I},v} \mu X.\varphi & \text{if } \forall S \subseteq \mathcal{P}(clos(S) \Rightarrow \Sigma; p \in S) \text{ where} \\
 & \quad clos(S) \text{ is } \forall (\Gamma; q) \in \mathcal{P}(\Gamma; q \models_{\mathbf{I},v[X \mapsto S]} \varphi \Rightarrow (\Gamma; q) \in S) \\
\Sigma; p \models_{\mathbf{I},v} [a] & \text{if } p \cong a|\surd \\
\Sigma; \nu x : \tau p \models_{\mathbf{I},v} \Sigma_x \varphi & \text{if } \Sigma, x : \tau; p \models_{\mathbf{I},v} \varphi \\
\Sigma; p \models_{\mathbf{I},v} \varphi \otimes \psi & \text{if } p \cong q|r \text{ and } \Sigma; q \models_{\mathbf{I},v} \varphi \text{ and } \Sigma; r \models_{\mathbf{I},v} \psi \\
\Sigma; \mathbf{0} \models_{\mathbf{I},v} \mathbf{1} &
\end{array}
$$

Fig. 3. Satisfaction of $\mu\mathcal{L}_\pi$ formulas.

3 A program logic for core-$\mathcal{L}_\pi$

The program logic $\mu\mathcal{L}_\pi$ is basically a many sorted first-order version of the propositional μ-calculus (see [5]) extended with operators for handling private names and spatial distribution. Formulas of $\mu\mathcal{L}_\pi$ are

$$\varphi ::= P(\tilde{t}) \mid t \doteq u \mid \neg\varphi \mid \varphi \wedge \varphi \mid \forall x : \tau\varphi \mid \langle S \rangle \varphi \mid X \mid \mu X.\varphi \mid \Sigma_{x:\tau}\varphi \mid \varphi \otimes \varphi \mid [t] \mid \mathbf{1}$$

The formulas expressing conjunction ($\varphi \wedge \psi$), properties $P(\tilde{t})$, equality ($t \doteq u$), possibility ($\langle S \rangle \varphi$), universal quantification ($\forall x : \tau\varphi$), negation ($\neg\varphi$) and least fixed point ($\mu X.\varphi$) are to be understood in the usual way. In a fixed point formula $\mu X.\varphi$, any occurrence of X must be under an even number of negations and under no $\otimes$. The remaining forms are particular to $\mu\mathcal{L}_\pi$ and deserve further comments. $\Sigma_x\varphi$ asserts a property of a state embedded into a context where the name x is private. The formula $\varphi \otimes \psi$ holds in every state that can be partitioned into a component satisfying φ and a component satisfying ψ. $\mathbf{1}$ is the unit for $\otimes$. Finally, $[t]$ is true of those states that essentially contain just the message t. Formulas are expected to be well-typed w.r.t. a signature Σ; for $\mu\mathcal{L}_\pi$ formulas φ, we assume a corresponding judgement form $\Sigma \vdash \varphi$ defined as expected. If such a judgement is valid, φ will be called a Σ-formula.

SEMANTICS. Truth of formulas of $\mu\mathcal{L}_\pi$ is taken with respect to a structure that consists in a labelled transition system for core-$\mathcal{L}_\pi$[1] together with a map $\mathbf{I}$ assigning to each signature Σ an interpretation $\mathbf{I}_\Sigma$. Such interpretations assign to each predicate symbol of type $(\tau_1, \ldots, \tau_n)o$ a subset of $\mathcal{T}_{\tau_1}(\Sigma) \times \cdots \times \mathcal{T}_{\tau_n}(\Sigma)$. $\mathbf{I}$ is required to be monotonic, that is $\Sigma \subseteq \Sigma'$ implies that, for every predicate

[1] In general, the reference LTS; sometimes a suitable restriction may be required, cf. previous remarks on relevance.

symbol P, $\mathbf{I}_\Sigma(P) \subseteq \mathbf{I}_{\Sigma'}(P)$. In Figure 3 we present the relation $- \models_{\mathbf{I},v} -$ of satisfaction between Σ-terms and well-typed Σ-formulas of $\mu\mathcal{L}_\pi$. The definition is parametric on the interpretation map $\mathbf{I}$ and also on a valuation v. This valuation assigns sets of well-typed terms to propositional variables X, and is needed to interpret the fixed point operator. When interpreting formulas without free propositional variables, the valuation is irrelevant and could be omitted.

Some specific comments about this definition are in order. First, core-$\mathcal{L}_\pi$ terms are taken modulo structural congruence $\cong$. In the clause for $\langle S\rangle\,\varphi$, S is a set of $\mathcal{L}_\pi$ actions. To follow usual notations we will write $\langle\cdot\rangle\,\varphi$ for $\langle S\rangle\,\varphi$ when S is the set of all actions. In the clause of $\mu X.\varphi$, the property $clos(S)$ holds of prefixed points of the mapping that sends sets of terms satisfying a property ψ to the set of terms satisfying the property $\varphi\{^X/_\psi\}$. The syntactic restriction on the occurrences of X in $\mu X.\varphi$ causes this mapping to be monotonic, therefore the clause for $\mu X.\varphi$ defines the least fixed point as the intersection of all prefixed points, cf. Knaster-Tarsky theorem. Other logical connectives (say, $\vee$, $\exists$, $\Rightarrow$) are defined in the usual way according to standard tarskian semantics. The following usual abbreviations will also be used: $[S]\varphi$ for $\neg\langle S\rangle\neg\varphi$ (necessity), $\nu X.\varphi$ for $\neg\mu X.\neg\varphi\{^X/_{\neg X}\}$ (greatest fixed point), $\bot$ for $\mu X.X$ (false) and $\top$ for $\neg\bot$ (true). We will also write $\{\varphi\}$ for $\Sigma_{\tilde{x}}\varphi$ when no x occurs free in φ, and introduce the following derived operators: $\langle[S]\rangle\,\varphi$ standing for $\langle S\rangle\top \wedge [S]\varphi$, $\mathbf{inv}_{[S]}\varphi$ standing for $\nu X.\varphi\wedge[S]X$ (φ holds along any S-path) and $\mathbf{ev}_{[S]}\varphi$ standing for $\mu X.\varphi\vee\langle S\rangle X$ (φ will eventually hold along some S-path) and $\mathbf{evs}_{[S]}\varphi$ for $\mu X.\varphi\vee(\langle S\rangle\top\wedge[S]X)$ (φ will eventually hold along any S-path); $\mathbf{inv}_{[\cdot]}\varphi$ will be abbreviated by $\mathbf{inv}\varphi$ and likewise for $\mathbf{ev}\varphi$ and $\mathbf{evs}$.

The use of an explicit signature is essential in the definition of satisfaction. For instance, note that $\Sigma_x\top \Rightarrow \forall_y\Sigma_x(\neg x \doteq y)$ is true in any interpretation. This formula asserts intuitively that if a state has some private name, then this name is distinct from every term in the current environment. The scoping behaviour of private names induces the need to consider the brackets $\{\varphi\}$ even when the private names do not ocurr in φ. For instance, $a : o; \nu b : o\ b[]a \models \{\langle\cdot\rangle\langle\uparrow a\rangle\top\}$ but $a : o; \nu b : o\ b[]a \not\models \langle\cdot\rangle\langle\uparrow a\rangle\top$.

4 A specification language for concurrent objects

The specification language we present here is an idiom of the program logic of Section 3. Essentially, we consider specifications of classes of objects that indicate besides the instance state variables, for each possible action A a set of dynamic axioms of the form $\varphi \Rightarrow \langle[A]\rangle\,\psi$, where φ (the pre-state formula) is a proposition over the object's state and ψ (the pos-state formula) is a proposition defining the new state of the object and other side effects induced by the execution of A. Possible side effects are the posting of action requests $i.m$ (for other objects) and of creation requests $\nu i : c(\tilde{a})$ (for a new object i of class $c(\tilde{a})$). Besides this components, a specification also comprises a post-state formula defining the initial state of objects of the class and a static theory (called so because it does not involve any dynamic modalities) defining the meaning of the non-logical

constants (predicate symbols) occurring elsewhere in the specification. At this stage, the reader may want to give a look at the sample specification in Fig. 4.

We assume given a set of basic types and a set of basic sorts (containing, at least *o* and *act*). The type $(act)o$ will be abbreviated by obj. A *system specification* is triple $(\Sigma, \mathcal{C}, \mathcal{S})$ where Σ is a signature, the set of class names $\mathcal{C}$ is a subset of Σ, and $\mathcal{S}$ is a mapping assigning to each $c \in \mathcal{C}$ a class specification $\mathcal{S}_c$ of type $\Sigma(c) = \tau_c$. A *parametric Σ-class specification* of type $(\tau_1, \ldots, \tau_n)o$ is then a five-tuple $(\mathcal{A}, \mathcal{L}, \mathcal{I}, \mathcal{R}, \mathcal{T})$ such that (1) $\mathcal{A}$ and $\mathcal{L}$ and Σ are pairwise disjoint signatures, (2) $\mathcal{A}$ is of the form $a_1 : \tau_1, \ldots, a_n : \tau_n$ (3) $\mathcal{I}$ is a post-state formula, $\mathcal{R}$ is a finite set of dynamic axioms and $\mathcal{T}$ is a finite sets of static axioms. Post-state formulas and dynamic and static axioms will be defined shortly.

PARAMETERS. Signature $\mathcal{A}$ lists the parameters of the class specification. If $\mathcal{A}$ is empty, the specification is non-parametric and therefore fully determined. Given a Σ-class specification $(\mathcal{A}, \mathcal{L}, \mathcal{I}, \mathcal{R}, \mathcal{T})$ of type $(\tau_1, \ldots, \tau_n)o$ and terms $t_i \in \mathcal{T}_{\tau_i}(\Sigma)$ we obtain some non-parametric concrete instance $(\emptyset, \mathcal{I}\{\bar{a}/\bar{\imath}\}, \mathcal{R}\{\bar{a}/\bar{\imath}\}, \mathcal{T}\{\bar{a}/\bar{\imath}\})$.

LOCAL SYMBOLS. Signature $\mathcal{L}$ splits in two components: $\mathcal{L}_{\text{attrs}}$, containing the declarations of the class instance's attributes and $\mathcal{L}_{\text{preds}}$, which assigns to each predicate symbol defined by the local static theory $\mathcal{T}$ a predicate type $(\bar{\gamma})o$.

DYNAMIC THEORY. Dynamic axioms have the form $\forall \bar{x} : \bar{\tau}(S \wedge \varphi \Rightarrow \langle [A] \rangle \psi)$ where the action A is a term of type *act*, S is a state formula, φ is a static formula, ψ is a post-state formula and $\bar{x}$ are some variables with free occurrences at least in S, φ or A - those that occur just in φ and ψ should have basic type. A formula of the form $S \wedge \varphi$ will be called a pre-state formula. These kinds of formulas are defined by

$$\begin{array}{ll} \text{(state formulas)} & S ::= \top \mid a = t \mid S \wedge S \\ \text{(static formulas)} & \varphi ::= P(\bar{t}) \mid \top \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \\ \text{(post-state formulas)} & \psi ::= S \mid \nu i : c(\bar{t})\psi \mid i.m \otimes \psi \end{array}$$

where $a \in \mathcal{L}_{\text{attrs}}$, $P \in \mathcal{L}_{\text{preds}}$ and $c \in C$. In the definition of post-state formulas, i is assumed of type obj and m is assumed of type *act*.

INITIAL. $\mathcal{I}$ is a post-state formula specifying the object's state at birth time.

STATIC THEORY. $\mathcal{T}$ is a set of static axioms of the form $\forall \bar{x} : \bar{\tau}(\varphi \Rightarrow P(\bar{t}))$ where φ is a static formula.

COMMENTS. All formulas in the class specification are well-typed w.r.t. the signature $\Sigma\mathcal{A}\mathcal{L}$, self:obj, except that self cannot occur in $\mathcal{T}$. To avoid useless inconsistencies, no more than a single occurrence of a particular attribute symbol is allowed in pre- and post-state formulas, and the state formulas in the antecedent of dynamic axiom should be pairwise exclusive.

SEMANTICS. Semantics of class specifications is given by translation into a $\mu\mathcal{L}_\pi$ formulas. To perform this, we must understand formulas in a specification as asserting properties of a prototypical object of the given class and bring this arbitrary object explicit. More precisely, if i is a variable of type obj and φ a formula in a class specification, the formula $i.\varphi$ asserts the property φ relativized to the object i. This relativization is essential also to enable global reasoning about systems composed of multiple interacting objects. Now, $i.\varphi$ is obtained

from φ as follows: replace self by i, rename predicate symbols P as $c.P$ [2] is the class name, replace actions A in dynamic axioms by $\downarrow i(A)$ and replace equations $a = t$ by $i.a = t$. Moreover, understand $\nu i : c(\tilde{t})\psi$ as $\Sigma_{i:\mathsf{obj}}([c(i,\tilde{t})] \otimes \psi)$ and $j.m$ as $[j(m)]$. Every expression of the form $i.a = t$ must also be translated to some formula of $\mu\mathcal{L}_\pi$. This translation depends on the actual representation of objects as agents of core-$\mathcal{L}_\pi$. Several possibilities arise in this point, we just require invariant validity of the following principles

$P1\ \forall t : \tau(i.a = t \Rightarrow [\cdot](\exists y : \tau i.a = y))$ (Persistence of Attributes)
$P2\ \forall t : \tau_t v : \tau_v(i.a = t \wedge i.a = v \Rightarrow v \doteq t)$ (Attribute values are functional)
$P3\ \neg(\exists z : \tau(i.a = z) \otimes \exists z : \tau(i.a = z) \otimes \top)$ (Unicity)

SPECIFICATION FORMULAS AND MODELS. Let Σ° be obtained from Σ by changing the types assigned to class names in C from $(\tilde{\tau})o$ to $(\mathsf{obj}, \tilde{\tau})o$ and $\mathcal{L}^\circ$ be obtained from $\mathcal{L}$ by renaming each predicate symbol in $\mathcal{L}_{preds}$ from P to $c.P$. Then, every Σ-class specification $C = (\mathcal{A}, \mathcal{L}, \mathcal{I}, \mathcal{R}, \mathcal{T})$ yields the $\Sigma^\circ\mathcal{A}\mathcal{L}^\circ_{preds}$-formula

$$C^*(i,\tilde{a}) = \exists\tilde{v}(\bigwedge_{a_l \in \mathcal{L}_{attrs}} i.a_l = v_l) \wedge (\bigwedge_{D_j \in \mathcal{R}} i.D_j) \wedge (\bigwedge_{P_k \in \mathcal{T}} c.P_k)$$

This formula expresses the full content of the specification and can be used to define a notion of model for a system specification.

Definition 1 *A model for a system specification $\Psi = (\Sigma, \mathcal{C}, \mathcal{S})$ is an interpretation map* $\mathbf{I}$ *and an assignment of a Σ°-agent $!d_c$ to each $c \in \mathcal{C}$ such that*

$$\textit{for all } c \in \mathcal{C}\ P_c \models_{\mathbf{I}} \langle !d_c\rangle\,(i.\mathcal{I} \wedge \mathbf{inv}\{\mathcal{S}^*_c(i,\tilde{v}) \otimes \top\})$$

where $P_c \equiv \Sigma^\circ\mathcal{L}^{C\circ}_{preds}, i : obj; c(i,\tilde{v})$ for some $\tilde{v}$, and satisfaction is taken w.r.t. some sound restriction of the reference LTS relevant to $\{Q | P_c \xrightarrow{*} Q$ for $c \in \mathcal{C}\}$.

5 From Specifications to Agents

In this section, we show how models for class specifications can be systematically constructed in a rather simple way. Basically, an object "located at" name i will be represented by an agent of the form $\nu s : \sigma(s(i,\tilde{v}) | \nu\tilde{c}p(\mathsf{R}\langle\mathcal{R}\rangle^i_s | \mathsf{T}\langle c.\mathcal{T}\rangle))$ where $\mathsf{R}\langle\mathcal{R}\rangle^i_s$ and $\mathsf{T}\langle c.\mathcal{T}\rangle$ are encodings of respectively the dynamic and static theory, and $s(i,\tilde{v})$ is a "record" of the attribute values. We first introduce a canonic form for dynamic axioms in which all of the object's attributes occur both in pre- and post -state formulas. The motivation is that this simplifies the description of the translation and renders explicit the intended frame condition.
CANONIC FORM. Let $\forall x(S_1 \wedge \varphi \Rightarrow \langle[a]\rangle\,\psi_{(S_2)})$ be a dynamic axiom where S_2 is the maximal state formula embedded into the post-state formula ψ. Then, its canonic form is $\forall\tilde{x}\tilde{y}_k(S_1 \wedge S_1' \wedge \varphi \Rightarrow \langle[a]\rangle\,\psi_{(S_2 \wedge S_2')})$ where S_1' is $a_1 = y_1 \wedge \cdots \wedge a_k = y_k$

[2] When φ is a static axiom, we also write $c.\varphi$ for $i.\varphi$

for those attributes a_i not occurring in S_1 and S_2' is $b_1 = v_1 \wedge \cdots \wedge b_l = v_l$ where $b_i = v_i$ is the (unique) equation in $S \wedge S_1'$ for an attribute b_i not occurring in S_2.

From now on, we fix a Σ-class specification $\mathcal{S} = (\mathcal{A}, \mathcal{L}, \mathcal{I}, \mathcal{R}, \mathcal{T})$ associated to $c : (\tau_1, \ldots, \tau_\ell)o$ in some system specification $(\Sigma, \mathcal{C}, \mathcal{S})$, and consider its associated specification formula $\mathcal{S}^*(i, \tilde{a}_\ell)$ for some i of type obj. W.l.o.g. we will take the canonic form of every axiom in $\mathcal{R}$.

STATIC FORMULAS AND AXIOMS. Are encoded by the mapping $\mathsf{T}\langle - \rangle$

$$\begin{array}{ll}
\mathsf{T}\langle \top \rangle & \mapsto \mathbf{0} \\
\mathsf{T}\langle \varphi_1 \wedge \varphi_2 \rangle & \mapsto [\mathsf{T}\langle \varphi_1 \rangle]\mathsf{T}\langle \varphi_2 \rangle \\
\mathsf{T}\langle \varphi_1 \vee \varphi_2 \rangle & \mapsto [\nu n : o(n|!n[\mathsf{T}\langle \varphi_1 \rangle]\mathbf{0}|!n[\mathsf{T}\langle \varphi_2 \rangle])]\mathbf{0} \\
\mathsf{T}\langle P(\tilde{t}) \rangle & \mapsto c.P(\tilde{t}) \\
\mathsf{T}\langle \forall \tilde{x} : \tilde{\tau}(\varphi \Rightarrow c.P(\tilde{t})) \rangle & \mapsto !\tilde{x} : \tilde{\tau} \triangleright P(\tilde{t})[\mathsf{T}\langle \varphi \rangle]\mathbf{0}
\end{array}$$

This translation assigns to the theory $\mathcal{T}$ a well-typed core-$\mathcal{L}_\pi$ Γ-term $\mathsf{T}\langle c.\mathcal{T} \rangle$ defined as $\mathsf{T}\langle c.d_1 \rangle \,|\, \cdots \,|\, \mathsf{T}\langle c.d_n \rangle$ where $\tilde{d_n}$ are the formulas in $\mathcal{T}$. We have

Proposition 1 *Let Ξ be a signature, $\mathcal{T}$ be a static theory and $i.\varphi$ a goal formula over Ξ. Then $\Xi; \mathcal{T} \vdash \varphi$ is provable in classical logic iff $\Xi; \mathsf{T}\langle \mathcal{T} \rangle \,|\, \mathsf{T}\langle i.\varphi \rangle \xrightarrow{*} \surd$.*

STATE AND POST-STATE FORMULAS. We consider now the encoding of post-state formulas; state formulas are but a special case of these. We first sequentialise in an arbitrary way $a_1, a_2, \ldots, a_m$ the attribute names defined in $\mathcal{L}_{attrs}$ and define the type $\sigma_\mathcal{S}$ as $(\mathsf{obj}, o, \mathcal{L}(a_1), \cdots, \mathcal{L}(a_m))o$. Let $\mathsf{P}\langle - \rangle_s^o$ be the translation map

$$\begin{array}{ll}
\mathsf{P}\langle a_1 = t_1 \wedge \cdots \wedge a_m = t_m \rangle_s^i & \mapsto s(i, t_1, \ldots, t_m) \\
\mathsf{P}\langle \nu j : C(\tilde{t})\psi \rangle_s^i & \mapsto \nu_{j:\mathsf{obj}}(C(j, \tilde{t})|\mathsf{P}\langle \psi \rangle_s^i) \\
\mathsf{P}\langle j.a \otimes \psi \rangle_s^i & \mapsto j(a)|\mathsf{P}\langle \psi \rangle_s^i
\end{array}$$

$\mathsf{P}\langle - \rangle_s^i$ is parametric on two variables i and s, respectively of type obj and σ_C. This encoding represents the state of an object as a term $s(i, \tilde{v})$, where i is the object name and $\tilde{v}$ are the current values of it's attributes. We now can interpret in $\mu\mathcal{L}_\pi$ attribute valuation formulas $i.a_k = t$ (where a_k is the k-th attribute in sequence defined above) by the formula $\exists \tilde{x} : \tilde{\tau}\Sigma_s[s(i, x_1, \ldots, x_{k_1}, t, x_{k+2}, \ldots, x_{m-1})]$ where the $\tilde{\tau}$ are the appropriate types and t occurs in the k-th argument position of s. We will see shortly that this interpretation fully satisfies principles P1-3.

Proposition 2 *Let φ be a post-state formula in a class specification and $\Xi \vdash i.\varphi$. Then $\Xi; \nu s : \sigma_\mathcal{S}(\mathsf{P}\langle \psi \rangle_s^i \,|\surd) \models_\mathbf{I} i.\psi$ is valid for every $\mathbf{I}$.*

DYNAMIC AXIOMS. Of the form $\phi = \forall \tilde{x} : \tilde{\tau}(S \wedge \varphi \Rightarrow \langle [a] \rangle \psi)$ are translated by the map $\mathsf{R}\langle - \rangle_s^i$ given by $\mathsf{R}\langle \phi \rangle_s^i = !\tilde{x} : \tilde{\tau} \triangleright \mathsf{P}\langle S \rangle_s^i : i(a)[\mathsf{T}\langle i.\varphi \rangle]\mathsf{P}\langle \psi \rangle_s^i$

CLASSES AND SYSTEMS. A Σ-class specification $\mathcal{S}$ is encoded into the Σ° agent

$$\mathsf{C}\langle \mathcal{S} \rangle = !i : \mathsf{obj}, \tilde{a}_\ell : \tilde{\tau}_\ell \triangleright c(i, \tilde{a})[]Ob_\mathcal{S}(i, \tilde{a}_\ell)$$

where $Ob_\mathcal{S}(i, \tilde{a}) \equiv \nu s : \sigma_\mathcal{S}(\mathsf{P}\langle \mathcal{I} \rangle_s^i \,|\nu \tilde{c:p} : \mathcal{L}_{preds}(\mathsf{R}\langle \mathcal{R} \rangle_s^i \,|\mathsf{T}\langle c.\mathcal{T} \rangle))$ is the representation of an object i of class C in a state satisfying $\mathcal{I}$. In general, an agent of

the form $\nu s : \sigma_{\mathcal{S}}(s(i,\tilde{v})|\nu c.p : \mathcal{L}_{preds}(\mathsf{R}\langle\mathcal{R}\rangle^i_s\,|\mathsf{T}\langle c.\mathcal{T}\rangle))$ will represent an object "located at" i. Finally, a system $\Psi = (\Sigma, \mathcal{C}, \mathcal{S})$ is encoded into the Σ° agent $\mathsf{S}\langle\Psi\rangle = |_{c\in\mathcal{C}}\mathsf{C}\langle\mathcal{S}_c\rangle$ and we will take $Def = \mathcal{C}$ when defining the open operational semantic of a system of objects using the LTS of Section 2. Some remarks are needed before stating and proving the correctness of above defined encoding.

COMPUTATIONS AND CONFIGURATIONS. A computation in system $\Psi = (\Sigma, \mathcal{C}, \mathcal{S})$ is a reduction sequence $I \to S_1 \to S_2 \cdots$ with I of the form $\Sigma^\circ; \mathsf{S}\langle\Psi\rangle\,|p$ and $I \models [c(i,\tilde{v})]$ for some class $c \in \mathcal{C}$. Every S_i must have the form $\Sigma'; \mathsf{S}\langle\Psi\rangle\,|\nu\tilde{x} : \tilde{\tau}(Ob_1|\cdots|Ob_n|m_1|\cdots|m_k)$ where the Ob_j are objects and the m_j are either creation requests or messages $i(a)$. Since the agent $\mathsf{S}\langle\Psi\rangle$ encoding Ψ is fixed once for all, we can abstract of its presence and consider just agents like $\Sigma'; \nu\tilde{x} : \tilde{\tau}(Ob_1|\cdots|Ob_n|m_1|\cdots|m_k)$, assuming that for transitions $\stackrel{\downarrow!d.C}{\to}$ the action $!d.C$ is always the encoding of some class specification in $\mathsf{S}\langle\Psi\rangle$. Moreover, any component of a system also has this particular structure, so we can focus our concern just on well-typed agents of this form, to be called *configurations.* The following characterises the kind of actions configurations can perform with and without cooperation of the environment in a system reduction.

Proposition 3 *Let C be a configuration of a system Ψ. If $C \stackrel{c}{\to} C'$ is a transition occurring inside the derivation of a reduction between system configurations, then c is of the forms $\tau, \uparrow i.m$, $\downarrow i.m$ or $\Downarrow p$ for some definition $p = C\langle\mathcal{S}\rangle$.*

Hence, for instance when proving a (safety) property like $I \Rightarrow \mathbf{inv}\varphi$ from $I \Rightarrow \varphi$ and $\varphi \Rightarrow [\cdot]\varphi$ we can soundly restrict the universe of actions. Now, note that if $\Sigma; p \stackrel{\uparrow i.a}{\to} Q$ and $\Sigma; p \stackrel{\downarrow i.b}{\to} Q'$ then there is no configuration r such that $\Sigma; r \stackrel{\downarrow i.a}{\to} r'$ and $\Sigma; p|r$ is also configuration, because an object i cannot be located both in p and r. Hence, the reference LTS can be refined to approximate relevance, by adding to rule $[s]$ the proviso: "if not ($a \equiv\uparrow i(a)$ and $q \stackrel{\downarrow i.b}{\to}$ for some b)". The transition relation obtained thus is easily seen to be a subset of the reference transition relation and can be proven (still) sound w.r.t. the fragment of system configurations. Therefore, in the sequel, we will always refer to this restricted transition relation. We can now state correctness of the translation.

Lemma 1 *Let $\mathcal{S} = (\mathcal{A}, \mathcal{L}, \mathcal{I}, \mathcal{R}, \mathcal{T})$ be a Σ-class specification of a class c. Let $\mathbf{I}$ be any interpretation such that $\mathbf{I}_\Xi(c.P) = \{(t_1, \ldots, t_n) \mid \Xi; c.\mathcal{T} \vdash c.P(\tilde{t})\}$. Then*

$$\Gamma, i : \mathsf{obj}; Ob_{\mathcal{S}}(i,\tilde{v}) \models_{\mathbf{I}} i.\mathcal{I} \wedge \mathbf{inv}\{\mathcal{S}^*_c(i,\tilde{v}) \otimes \top\}$$

where Γ is of the form $\Sigma^\circ\mathcal{L}^\circ\Sigma'$ for any Σ'.

An immediate consequence of Lemma 1 is that the translation just presented provides a model for system specifications.

Theorem 1 *Let $\Psi = (\Sigma, \mathcal{C}, \mathcal{S})$ be a system specification. Then*

$$\Sigma^\circ\mathcal{L}^{C\circ}_{preds}, i : obj; c(i,\tilde{v}) \models_{\mathbf{I}} \langle C\langle\mathcal{S}_c\rangle\rangle\,(i.\mathcal{I} \wedge \mathbf{inv}\{\mathcal{S}^*_c(i,\tilde{v}) \otimes \top\})$$

for every $c \in \mathcal{C}$ and $\tilde{v}$ of the appropriate types, $\mathbf{I}$ assigns to each signature Σ the interpretation $\mathbf{I}_\Xi(c.P) = \{(v_1, \ldots, v_{k_P}) \mid \Xi; c.\mathcal{T} \vdash c.P(\tilde{v})\}$ and satisfaction is taken w.r.t. the relevant LTS for system configurations.

6 Some General Properties of Systems

In this section, we state several properties found useful to reason about systems and that can be proven valid in any configuration. We start by stating correctness of the interpretation of attribute valuations w.r.t. principles P1-3.

Proposition 4 *For every configuration P of a system $\Psi = (\Sigma, \mathcal{C}, \mathcal{S})$, and every class $c \in \mathcal{C}$ such that $a \in \mathcal{L}^{\mathcal{C}}_{attrs}$ we have $P \models P1 \wedge P2 \wedge P3$.*

We now introduce the following properties R1 and R2 that are true of all of $\mathcal{P}$.
R1. PRIVACY. $\Sigma_{x:\tau}\varphi \Rightarrow \forall_{y:\tau}\Sigma_{x:\tau}(\varphi \wedge \neg x \doteq y)$. If a state has some private name, then this name is distinct from every term in the current environment.
R2. TELESCOPING. $\Sigma_{\tilde{v}:\tilde{\tau}}(\mathbf{ev}_{[S]}\varphi \wedge \mathbf{inv}[U]\bot) \Rightarrow \mathbf{ev}\Sigma_{\tilde{v}:\tilde{\tau}}\varphi$ where S is the set of actions without free occurrences of $v \in \tilde{v}$ and U is the set of actions extruding some v. Thus, we can move a restriction "forward" in a computation if the internal context does not extrude any of the restricted names.

The remaining properties are specific to object system configurations. We assume that i, j are generic object names.
R3. LOCALITY. $(\neg i \doteq j) \wedge i.\varphi \otimes \top \Rightarrow [j.m](i.\varphi \otimes \top)$ for every pre-state formula φ. Attribute values of an object can only be changed by the object's own actions.
R4. GROUPING. $(\neg i \doteq j) \wedge (i.\varphi \otimes \top) \wedge (j.\psi \otimes \top) \Rightarrow (i.\varphi \otimes j.\psi \otimes \top)$ allows the concatenation of assertions about disjoint components of a configuration.
R5. MERGING. $(i.\varphi \otimes \top) \wedge (i.\psi \otimes \top) \Rightarrow ((i.\varphi \wedge i.\psi) \otimes \top)$ allows the merging of assertions about the same component of a configuration.
R6. RELEVANCE I. $\langle\downarrow i.a\rangle \top \Rightarrow [\uparrow i.m]\bot$ expresses that if some object is located in the system then no messages directed to it could be sent to the environment.
R7. RELEVANCE II. $S^*_c(i, \tilde{a}) \otimes \varphi \Rightarrow S^*_c(i, \tilde{a}) \otimes (\varphi \wedge [\downarrow i.m]\bot)$ Messages directed to an object i cannot be received by the internal context surrounding i.
R8. SUPPORT. $\forall \tilde{x} : \tilde{\tau}(\langle i.a\rangle \top \Rightarrow i.\varphi)$ given that $\forall \tilde{x} : \tilde{\tau}(i.\varphi \Rightarrow \langle [i.a]\rangle i.\psi)$ is the single dynamic axiom for the action a in $S^*_c(\tilde{i}, \tilde{a})$.
R9. ACT. $([a] \otimes \mathbf{inv}(\top \otimes (\mathbf{evs}\varphi) \wedge (\varphi \Rightarrow \langle\downarrow a\rangle \psi))) \Rightarrow \mathbf{ev}_{[S]}\{\psi \otimes \top\}$ where S is any action set containing τ. Together with a fairness assumption, this ensures that every request for a message that is never forever disabled will be attended.
R10. CREATION. For each class c of type $(\tau_1, \ldots, \tau_n)o$, the principle $c(i, \tilde{a}) \otimes [i.m_1] \otimes \cdots \otimes [i.m_k] \otimes \top) \Rightarrow \mathbf{ev}_{[S]}(\mathcal{I} \wedge \mathcal{S}^*_c(i, \mathbf{a}) \otimes [i.m_1] \otimes \cdots \otimes [i.m_k] \otimes \top$ where S can be anything not excluding $\downarrow d$ with d is $\mathsf{C}\langle\mathcal{S}_c\rangle$ for some $c \in \mathcal{C}$. Every request for a new object is fulfilled. We now bring explicit a fairness assumption.

Definition 2 *A computation $P_1 \xrightarrow{a_1} P_2 \xrightarrow{a_2} \cdots$ is fair if whenever $P_i \cong \nu\tilde{x} : \tilde{\tau}(b|Q)$ and $Q \models \mathbf{inv}\ \mathbf{evs}\langle\downarrow b\rangle \top$ then for some $n \geq i$, $P_n \cong \nu\tilde{y} : \tilde{\gamma}(b|Q')$, $Q' \xrightarrow{\downarrow b} Q''$, $P_{n+1} \cong \nu\tilde{y} : \tilde{\gamma}\ Q''$ and $P_n \xrightarrow{\tau} P_{n+1}$.*

In a fair computation, a message request for an action that is only sometimes disabled will necessarily result in its execution. The important remark to make is that the weak eventualities asserted in the creation and synchronisation axioms above express in fact *strong eventualities*, if we restrict our concern just to fair computations. For instance, we can prove that if $P \models_{\mathbf{I}} ([a] \otimes \mathbf{inv}(\top \otimes (\mathbf{evs}\varphi) \wedge$

Ser() is
attrs
$req : li$
ch : **obj**
initial
$\nu n : N(req = nil \wedge ch = n)$
dynamic
$\forall r, a :$ **obj** $ch = r \Rightarrow \langle [open(a)] \rangle$
$\quad \nu t : Thr(r, s) a.ans(t)$
$\forall s : li\ r :$ **obj**$, l : li\ ch = r \wedge req = s$
$\quad \Rightarrow \langle [done(r, l)] \rangle\ req = l \bullet s$
$\forall l : lli\ a :$ **obj** $req = l \Rightarrow \langle [dump(a)] \rangle$
$\quad req = nil \otimes a.sl(l)$
end

N() is
end

Thr(s:obj,r:obj) is
attrs
up : **bool**
$sl : li$
initial
$up = \mathbf{T} \wedge sl = nil$
dynamic
$\forall l : li, x : i\ up = \mathbf{T} \wedge sl = l \Rightarrow$
$\quad \langle [ins(x)] \rangle\ sl = x \bullet l$
$\forall lu : li, x : i\ up = \mathbf{T} \wedge sl = l$
$\quad \wedge rem(l, x, u) \Rightarrow$
$\quad \langle [del(x)] \rangle\ sl = u$
$\forall l : li\ up = \mathbf{T} \wedge sl = l \Rightarrow$
$\quad \langle [quit] \rangle\ up = \mathbf{F} \otimes s.done(r, l)$
static
$\forall x : i, l : li\ rem(x, x \bullet l, l)$
$\forall xy : i, lr : li\ rem(x, y \bullet l, x \bullet r)$
$\quad \Leftarrow rem(x, l, r)$
end

Fig. 4. A sample specification

$(\varphi \Rightarrow \langle a \rangle\, \psi))$ then for all fair reduction paths $P \equiv P_1 \xrightarrow{a_1} P_2 \xrightarrow{a_2} \cdots$ there is P_i such that $P_i \models \{\psi \otimes \top\}$.

7 Reasoning about Systems

We now present a toy specification of a commercial server usable for instance at a bookseller network site, and prove a couple of very simple properties about it with the program logic. Servers can perform three types of actions: $open(a)$, that creates a new thread whose identity is returned to the user a by means of a message $a.ans(t)$, $dump(a)$ that returns to a the current list of orders, clearing it afterwards, and $done(r, l)$, explained shortly. Each thread can execute insertions $ins(i)$ and deletions $del(i)$ of items i in the current shopping list at the user command, until *quit* is invoked. After this happens, the thread yields the collected shopping list (the order) to the server via a $done(r, l)$ request, committing itself to no further action. Obviously, only $s.done(r, l)$ requests originated by threads should be served by the server. This is achieved in the proposed specification by the use of a private communication channel kept in the ch attribute of servers. The specification is presented in Figure 4, in a sugared syntax not hard to relate to the definitions in Section 4.

So, let $I = \Sigma_{n:\mathsf{obj}}(s.req = nil \wedge s.ch = n) \wedge \mathbf{inv}\{Ser^*(s) \otimes \top\}$ characterise the initial state of a system. We now prove that $I \Rightarrow \mathbf{inv}\ \forall r : \mathsf{obj}, l : li[s.done(r, l)]\bot$ that is, the privacy assumption just mentioned. To that end, we first show the safety property $I \Rightarrow \mathbf{inv}P$ where $P = \Sigma_{n:\mathsf{obj}}(s.ch = n \otimes \top) \wedge \mathbf{inv}\{Ser^*(s) \otimes \top\}$. Since $I \Rightarrow P$, we first argue informally that $P \Rightarrow [a]P$ for every action a. Since

attribute $s.ch$ is never mentioned in s post-formulas, its value n never changes by the frame condition. So we must just prove that such private value is never extruded. The only possibility is in an action $\uparrow(n : \mathrm{obj})s.done(n,l)$. But since say $P \Rightarrow \langle\downarrow(u : \mathrm{obj})s.open(u)\rangle\top$ then $P \Rightarrow [\uparrow(n : \mathrm{obj})s.done(n,l)]\bot$ by R6, and therefore $P \Rightarrow [\uparrow(n)s.done(n,l)]P$. Thus $I \Rightarrow \mathbf{inv}P$. We now prove in detail that $P \Rightarrow \forall RL[s.done(R,L)]\bot$.

1. $\Sigma_n((Ser^*(s) \wedge s.ch = n) \otimes \top)$, hypothesis and R5. By def of $Ser^*(s)$ and R8
2. $\Sigma_n(((\forall s'r'l's.ch = r' \wedge s.req = s' \Leftarrow \langle s.done(r',l')\rangle\top) \wedge s.ch = n) \otimes \top)$
4. $\Sigma_n(((\forall r'l'\neg s.ch = r' \Rightarrow [s.done(r',l')]\bot) \wedge s.ch = n) \otimes \top)$, by pure logic.
5. $\Sigma_n((\forall r'l'\neg n \doteq r' \Rightarrow [s.done(r',l')]\bot) \otimes \top)$, by P2.
6. $\exists RL\,\langle s.done(R,L)\rangle\top$, hypothesis.
7. $\exists RL(\langle s.done(R,L)\rangle\top \wedge \Sigma_n((\forall r'l'\neg n \doteq r' \Rightarrow [s.done(r',l')]\bot) \otimes \top)$, by 5,6.
8. $\exists RL(\Sigma_n((\forall r'l'\neg n \doteq r' \Rightarrow [s.done(r',l')]\bot \wedge \langle s.done(R,L)\rangle\top \wedge \neg R \doteq n) \otimes \top)$, by R1 and noting that n is not free in $s.done(R,L)$.
9. $\exists RL(\Sigma_n(([s.done(R,L)]\bot \wedge \langle s.done(R,L)\rangle\top) \otimes \top)$, by pure logic.
10. $\forall RL[s.done(R,L)]\bot$, by 9,6 contradiction.
11. $P \Rightarrow \forall RL[s.done(R,L)]\bot$, by 1 discharge, and we are done.

We now consider the following system-wide property φ stating that, in certain conditions, after a thread executes a *quit* action, it should disable further actions and it's parent server must eventually acquire the shopping list produced by it. Then φ is

$\Sigma_n(Ser^*(s) \otimes (t.sl = l \wedge Thr^*(t,s,n)) \otimes \top) \Rightarrow [t.quit](\mathbf{ev}\{\exists u(s.reqs = l \bullet u) \otimes \top\} \wedge \mathbf{inv}[S]\bot)$

where S is the set of all actions by t. We now sketch the proof of φ, highlighting just the main steps. Assume $\Sigma_n(Ser^*(s) \otimes (t.sl = l \wedge Thr^*(t,s,n)) \otimes \top)$. This implies $\Sigma_n(Ser^*(s) \otimes (t.sl = l \wedge Thr^*(t,s,n) \wedge t.T_3(s,n)) \otimes \top)$, where $T_3(s,n)$ is the dynamic axiom for *quit* in the specification for threads. Now, $[t.quit]B$ where $B \equiv \Sigma_n(Ser^*(s) \otimes (t.up = \mathsf{F} \wedge Thr^*(t,s,n) \otimes s.done(n,l))$, using $t.T_3(s,n)$, and noting that n is not free in $t.quit$. Note that $B \Rightarrow \mathbf{inv}[S]\bot$, because $B \Rightarrow \mathbf{inv}(t.up = \mathsf{F} \otimes \top)$. On the other hand, $B \Rightarrow \Sigma_n(s.done(n,l) \otimes \mathbf{inv}\{Ser^*(s) \otimes \top\})$. Now, $s.done(n,l) \otimes \mathbf{inv}\{Ser^*(s) \otimes \top\}$ implies $s.done(n,l) \otimes \mathbf{inv}(\mathbf{evs}\exists r(s.ch = n \wedge s.reqs = r) \wedge \exists r(s.ch = n \wedge s.reqs = r) \Rightarrow \langle[s.done(n,l)]\rangle\exists r(s.reqs = l \bullet r) \otimes \top)$, and this last formula implies $\mathbf{ev}_{[U_n]}\{(\exists r(s.reqs = l \bullet r) \otimes \top)\}$ by R9, where U_n is the set of actions without free occurrences of n. Since $\Sigma_n\mathbf{ev}_{[U_n]}\phi \Rightarrow \mathbf{ev}\Sigma_n\phi$, by R2, we conclude φ.

8 Conclusions and Further Work

We presented a declarative executable specification language and program logic for concurrent objects providing a unified framework for specifying and reasoning about systems, thus demonstrating the usefulness of the $\mathcal{L}_\pi$ language as a meta-language for structuring the definition of operational semantics and program verification logics of languages with concurrent and logic features. An aspect not covered in the present paper was object types, we expect also types to be accommodated in this framework. The proposed model can be extended

in several directions (for instance, modelling and reasoning about inheritance, transactions, or mobile code). An interesting observation about $\mu\mathcal{L}_\pi$ is that the monoidal structure induced by $\otimes, \mathbf{1}$ together with $\Rightarrow$ and $\Sigma_{x:\tau}$ induces an Action Structure [9]. This fact may be explored in the definition of a more abstract characterisation of a verification logic for global/local properties of modular systems in the presence of private names.

Acknowledgements To the anonymous referees for their useful comments and Project ESCOLA PRAXIS/2/2.1/MAT/46/94 for partially supporting this work.

References

1. L. Caires. A language for the logical specification of processes and relations. In Michael Hanus, editor, *Proceedings of the Algebraic and Logic Programming International Conference*, number 1139 in LNCS, pages 150–164, 1996.
2. L. Caires. A language for the logical specification of processes and relations. Technical Report 6.96, Universidade Nova de Lisboa, DI/FCT, 1996. http://www-ctp.di.fct.unl.pt/~lcaires/writings/lpi6.96.ps.gz.
3. L. Caires and L. Monteiro. Proof net semantics of proof search computation. In K. Meinke and M. Hanus, editors, *Proceedings of the Algebraic and Logic Programming International Conference*, number 1298 in LNCS, pages 194–208, 1997.
4. J. Fiadeiro and T. Maibaum. Temporal theories as modularisation units for concurrent system specification. *Formal Aspects of Computing*, 4(3):239–272, 1992.
5. D. Kozen. Results on the propositional μ-calculus. *TCS*, 27(3):333–354, 1983.
6. D. Miller. A survey of linear logic programming. *Computational Logic*, 2(2), 1995.
7. D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proof as a foundation for logic programming. *Ann. of Pure and App. Logic*, (51):125–157, 1991.
8. R. Milner. Functions as processes. *Math. Struc. in Computer Sciences*, 2(2):119–141, 1992.
9. R. Milner. Calculi for interaction. *Acta Informatica*, 33(8):707–737, 1996.
10. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.
11. O. Nierstrasz, J-G. Schneider, and M. Lumpe. Formalizing composable software systems – A research agenda. In *Proceedings 1st IFIP Workshop on Formal Methods for Open Object-based Distributed Systems FMOODS'96*, pages 271–282. Chapmann and Hall, 1996.
12. B. Pierce and D. Turner. Concurrent objects in a process calculus. In Takayasu Ito and Akinori Yonezawa, editors, *Theory and Practice of Parallel Programming (TPPP), Sendai, Japan (Nov. 1994)*, number 907 in Lecture Notes in Computer Science, pages 187–215. Springer-Verlag, April 1995.
13. A. Pnueli. The temporal semantics of concurrent programs. In G. Kahn, editor, *Semantics of Concurrent Computations*, volume 70 of *LNCS*, pages 1–20, Evian, France, July 1979. Springer-Verlag , Berlin, Germany.
14. D. Sangiorgi. An interpretation of typed objects into the typed π-calculus. Technical report, INRIA Technical Report RR-3000, 1996.
15. A. Sernadas, C. Sernadas, and J. Costa. Object specification logic. *Journal of Logic and Computation*, 5(5):603–630, October 1995.
16. D. Walker. Objects in the π-calculus. *Journal of Information and Computation*, 116(2):253–271, 1995.