Local π-Calculus at Work: Mobile Objects as Mobile Processes[1]

Massimo Merro[2]
COGS, University of Sussex, United Kingdom

Josva Kleist[3]
INRIA, Sophia-Antipolis, France

Uwe Nestmann
EPF, Lausanne, Switzerland

February 7, 2001

[1]An extended abstract has appeared in *Proceedings of IFIP TCS 2000*, volume 1872 of *Lecture Notes in Computer Science*. Springer Verlag, August 2000.
[2]Partly supported by Marie Curie fellowhip, EU-TMR, No. ERBFMBICT983504.
[3]Partly supported by Danish National Research Foundation grant SNF-28808

Abstract

Obliq is a lexically-scoped, distributed, object-based programming language. In Obliq, the *migration* of an object is proposed as creating a clone of the object at the target site, whereafter the original object is turned into an alias for the clone. Obliq has only an informal semantics, so there is no proof that this style of migration is safe, i.e., transparent to object clients. In previous work, we introduced Øjeblik, an abstraction of Obliq, where, by lexical scoping, sites have been abstracted away. We used Øjeblik in order to exhibit how the semantics behind Obliq's implementation renders migration unsafe. We also suggested a modified semantics that we conjectured instead to be safe. In this paper, we rewrite our modified semantics of Øjeblik in terms of π-*calculus*, and we use it to formally prove the correctness of *object surrogation*, the abstraction of object migration in Øjeblik.

1 Introduction

The work presented in this paper is in line with the research activity to use the π-calculus as a toolbox for reasoning about object-based programming languages. Former works on the semantics of objects as processes showed the value of this approach: while [Wal95, HK96, San98, KS98] focused on just providing formal semantics to object-oriented languages and language features, the work of others [PW98, San99b] has been driven by a specific programming problem. Our work tackles a problem in Cardelli's *lexically-scoped distributed* programming language *Obliq* [Car95]. Cardelli proposed to derive *object migration* from two other primitives, *cloning* and *aliasing*, by performing one after the other. In Obliq, immutable values can be freely copied from site to site, whereas mutable values are stationary. Only references to mutable values may be transmitted between different sites. Accordingly, since objects are mutable, the migration of an object does not physically move the object, but instead creates a *clone* of the object at the target site and then turns the original local) object into an *alias*—sometimes called a *proxy*—for the new remote) object.

1.1 Previous work

When is object migration correct? In concurrent and distributed programs, it is important that certain state changes, in parts of the running system, may happen transparently from the point of view of the rest of the system. Ensuring that the implementation of an

Aliasing Semantics In [NHKM00], we gave several proposals of configuration-style semantics for Øjeblik. One of them fits the Obliq implementation [Car94, Car95], but does not guarantee the correctness of object surrogation as defined above. This was formally shown by exhibiting Øjeblik contexts that are able to distinguish the terms $a.\mathsf{ping}$ and $a.\mathsf{surrogate}$.

Channels:	$c \in \mathbf{C}$	*Values*		
Keys:	$k \in \mathbf{K}$	v	$::=$ x	variable
Names:	$\in \mathbf{N}$		$\mid$ ℓ_v	variant
n	$::=$ $c \mid k$		$\mid$ $\langle v_1 .. v_n \rangle$	tuple
Auxiliary:	$u \in \mathbf{U}$	*Types*		
Variables:	$\in \mathbf{X}$	T	$::=$ $\mathbf{C}\ T)$	channel type
x	$::=$ $n \mid u$		$\mid$ $\mathbf{K}$	key type
			$\mid$ $[\ell_1{:}T_1 ; \ldots ; \ell_m{:}T_m\,]$	variant type
Labels	$\in \mathbf{L}$		$\mid$ $\langle T_1 .. T_m \rangle$	tuple type
$\ell, \ell_1, \ell_2, \ldots$			$\mid$ X	type variable
			$\mid$ $\mu X.\, T$	recursive type

Processes

P	$::=$ $\mathbf{0}$		nil process
	$\mid$ $c\ x).P$		single *input*
	$\mid$ $\overline{c}v$		output
	$\mid$ $P_1 \mid P_2$		parallel
	$\mid$ $\boldsymbol{\nu} n{:}T)\, P$		restriction
	$\mid$ $\mathbf{!}\, c\ x).P$		replicated *input*
	$\mid$ if $[k{=}k_1]$ then P_1 elif $[k{=}k_2]$ then P_2 else P_3		key testing
	$\mid$ case v of $\ell_1\_\ x_1){:}P_1 ; \ldots ; \ell_m\_\ x_m){:}P_m$		variant *destructor*
	$\mid$ let $x_1 .. x_m\ ) = v$ in P		tuple *destructor*
	$\mid$ wrong		run time error

The *locality constraint* requires that in *single* and *replicated*) *inputs* and in *variant* and *tuple*) *destructors* the bound names $x, x_1, \ldots, x_m$ must not be used in free input position within the respective scope $P, P_1, \ldots, P_m$.

Table 1: The Calculus Lπ^+

reasoning about, concurrent object-oriented languages. In particular, we can easily guarantee the uniqueness of object identities—a fundamental feature of objects: in object-oriented languages, the name of an object may be transmitted; the recipient may use that name to access the methods of the object, but it cannot create a new object with the same name. When representing objects in the π-calculus, this translates directly into the constraint that the process receiving an object name may only use it in output actions—a guarantee in our setting.

2.1 Terms and Types

In Table 1, we introduce the calculus Lπ^+, a typed version of polyadic Lπ with: i) labelled values ℓ_v, called *variants* [San98], with case analysis; ii) tuple values $\langle v_1 .. v_n \rangle$, with pattern matching, iii) constants k, called *keys*, with equality; iv) a wrong construct to model run-time typing errors.

We introduce a few syntactic categories: the set $\mathbf{X}$ of *variables* includes the set $\mathbf{N}$ of *names* constants and variables) consisting of the two disjoint sets $\mathbf{C}$ of *channels* and $\mathbf{K}$ of *keys*. The auxiliary variables in the set $\mathbf{U}$ are variables for complex values. $\mathbf{L}$ is the set of *labels*. In addition to the metavariables mentioned in the grammar, we let s, p, q, r, m, t range over channels, y over variables, w over values, Q over processes, and i, j, d, h, m over tuple, variant, or other indices. We abbreviate $\ell\_\langle\rangle$ and $\ell\_\ )$ as ℓ, as well as $\overline{q}\langle\rangle$ and $q\ ).P$ as $\overline{q}$ and $q.P$, respectively, while $\widetilde{v}$ denotes

a sequence $v_1 \,..\, v_m$.

Restriction, both inputs, and both destructors are *binders* for the names $x, x_1, \ldots, x_m$ in the respective scopes $P, P_1, \ldots, P_m$. We assume the usual definitions of free and bound occurrences of names, based on these binders; the inductively defined functions fn P) and bn P) denote those of process P. Similarly, fc P) and bc P) denote the free and bound channels of process P. Moreover, n P)=fn P)$\cup$bn P) and c P)=fc P)$\cup$bc P). *Substitutions*, ranged over by σ, are type-preserving functions from variables to values types are introduced below). For an expression e, $e\sigma$ is the result of applying σ to e, with the usual renaming to avoid captures. *Relabellings*, ranged over by ρ, permit replacing a label ℓ with another label ℓ'. We denote such a relabelling with $[^{\ell'}/_{\ell}]$. The application of a relabelling to a term is defined thus:

- $\ell\_$

$$\textsc{Inp})\ \frac{-}{c\ x).P \xrightarrow{c\,v} P\{^{v}/_{x}\}} \qquad \textsc{Rep})\ \frac{-}{!c\ x).P \xrightarrow{c\,v} P\{^{v}/_{x}\} \mid !c\ x).P}$$

$$\textsc{Out})\ \frac{-}{\overline{c}v \xrightarrow{\overline{c}v} \mathbf{0}} \qquad \textsc{Open})\ \frac{P \xrightarrow{(\boldsymbol{\nu}\widetilde{q}:\widetilde{T})\,\overline{c}v} P' \quad n \in \mathrm{n}\ v) \backslash \{\widetilde{q}, c\}}{\boldsymbol{\nu}n{:}T)\ P \xrightarrow{(\boldsymbol{\nu}n:T,\widetilde{q}:\widetilde{T})\,\overline{c}v} P'}$$

$$\textsc{Com})\ \frac{P_1 \xrightarrow{(\boldsymbol{\nu}\widetilde{q}:\widetilde{T})\,\overline{c}v} P_1' \qquad P_2 \xrightarrow{c\,v} P_2' \qquad \widetilde{q} \cap \mathrm{fn}\ P_2) = \emptyset}{P_1 \mid P_2 \xrightarrow{\tau} \boldsymbol{\nu}\widetilde{q}{:}\widetilde{T})\ P_1' \mid P_2')}$$

$$\textsc{Par})\ \frac{P_1 \xrightarrow{\mu} P_1' \qquad \mathrm{bn}\ \mu) \cap \mathrm{fn}\ P_2) = \emptyset}{P_1 \mid P_2 \xrightarrow{\mu} P_1' \mid P_2}$$

$$\textsc{Res})\ \frac{P \xrightarrow{\mu} P' \qquad n \notin \mathrm{n}\ \mu)}{\boldsymbol{\nu}n{:}T)\ P \xrightarrow{\mu} \boldsymbol{\nu}n{:}T)\ P'}$$

$$\textsc{Test-1})\ \frac{P_1 \xrightarrow{\mu} P_1' \qquad k_1 = k}{\mathsf{if}\ [k{=}k_1]\ \mathsf{then}\ P_1\ \mathsf{elif}\ [k{=}k_2]\ \mathsf{then}\ P_2\ \mathsf{else}\ P_3 \xrightarrow{\mu} P_1'}$$

$$\textsc{Test-2})\ \frac{P_2 \xrightarrow{\mu} P_2' \qquad k_1 \neq k = k_2}{\mathsf{if}\ [k{=}k_1]\ \mathsf{then}\ P_1\ \mathsf{elif}\ [k{=}k_2]\ \mathsf{then}\ P_2\ \mathsf{else}\ P_3 \xrightarrow{\mu} P_2'}$$

$$\textsc{Test-3})\ \frac{P_3 \xrightarrow{\mu} P_3' \qquad k_1 \neq k \neq k_2}{\mathsf{if}\ [k{=}k_1]\ \mathsf{then}\ P_1\ \mathsf{elif}\ [k{=}k_2]\ \mathsf{then}\ P_2\ \mathsf{else}\ P_3 \xrightarrow{\mu} P_3'}$$

$$\textsc{Case})\ \frac{P_j\{^{v}/_{x_j}\} \xrightarrow{\mu} Q \qquad j \in 1..m}{\mathsf{case}\ \ell_j\_v\ \mathsf{of}\ \ell_1\_\ x_1){:}P_1\ ;\ \ldots\ ;\ \ell_m\_\ x_m){:}P_m \xrightarrow{\mu} Q}$$

$$\textsc{Let})\qquad P\{^{v_1\,..\,v_m}/_{x_1\,..\,x_m}$$

The proof of the above result is standard see

Definition 2. (Typed bisimilarity) Typed bisimilarity, *is the largest typed relation* $\mathcal{S}$ *such that* $\Delta; P; Q) \in \mathcal{S}$ *implies:*

1. *If* $P \xrightarrow{\tau} P'$, *then there exists* Q' *s.t.* $Q \Rightarrow Q'$ *and* $\Delta; P'; Q') \in \mathcal{S}$.
2. *If* $P \xrightarrow{(\boldsymbol{\nu}\widetilde{n}:\widetilde{T})\,\overline{c}v} P'$, *with* $\widetilde{n} \cap \mathrm{fn}\ Q) = \emptyset$, *then there exists* Q' *such that* $Q \xRightarrow{(\boldsymbol{\nu}\widetilde{n}:\widetilde{T})\,\overline{c}v} Q'$ *and* $\Delta, \widetilde{n}{:}\widetilde{T}); P'; Q') \in \mathcal{S}$.
3. *If*

   (i) Γ *is a closed extension of* Δ,
   (ii) $\Gamma \vdash c{:}\mathbf{C}\ T)$ *and* $\Gamma \vdash v{:}T$,
   (iii) $P \xrightarrow{cv} P'$, *with* $\mathrm{fc}\ v) \cap \mathrm{fc}\ P \mid Q) = \emptyset$,

   then there exists Q' *such that:*

   (i) either $Q \xRightarrow{cv}$

(i) Γ *is a closed extension of* Δ,
(ii)

a, b	$::= \mathbb{O}$	object
	$\mid a.\mathrm{l}\langle\, a_1 \,..\, a_n \,\rangle$	method invocation
	$\mid a.\mathrm{l} \Leftarrow m$	method update
	$\mid a.\mathsf{clone}$	shallow copy
	$\mid a.\mathsf{alias}\langle b \rangle$	object aliasing
	$\mid a.\mathsf{surrogate}$	object surrogation
	$\mid a.\mathsf{ping}$	object ping
	$\mid s, x, y, z$	variables
	$\mid \mathsf{let}\, x{:}A = a\, \mathsf{in}\, b$	local definition
	$\mid \mathsf{fork}\langle a \rangle$	thread creation
	$\mid \mathsf{join}\langle a \rangle$	thread destruction
$\mathbb{O}$	$::= [\mathrm{l}_j = m_j]_{j \in J}$	object record
m_j	$::= \varsigma\ s_j{:}A, \tilde{x}_j{:}\widetilde{B}_j) b_j$	method
A, B	$::= [\mathrm{l}_j{:}\widetilde{B}_j \to \widehat{B}_j]_{j \in J}$	object record type
	$\mid \mathrm{Thr}\ A)$	thread type

Table 4: Øjeblik Syntax and Types

we show that the relation

$$\mathcal{S} = \{\big(\overline{p}v \mid R\,,\ \ \boldsymbol{\nu} r{:}\mathbf{C}\ T))\ \ \overline{p}w \mid r \triangleright q) \mid R\big)\} \cup \dot{\cong}$$

is a barbed bisimulation up to $\equiv$. The requirements on the barbs are easily satisfied. As for the bisimulation game on silent moves, the only interesting case is when there is a communication along p, that is, when $R \xrightarrow{p(x)} R'$. In this case we get, up to structural equivalence, the pair of processes

$$\big(Q\{q/r\}\,,\ \ \boldsymbol{\nu} r{:}\mathbf{C}\ T))\ \ Q \mid r \triangleright q)\big)$$

where $Q = R'\{w/x\}$. By Lemma 2.14 we can conclude. □

3 Øjeblik: A Concurrent Object Calculus

In this section, we present Øjeblik [NHKM00], a typed abstraction of Obliq designed to study object migration. Øjeblik-expressions and Øjeblik-types are generated by the grammar in Table 4, where a ranges over *Øjeblik-terms*, l over *method labels*, m over *method bodies*, s, x, y, z over *variables*, $\mathbb{O}$ over *object records*, and A, B over types. The type language extends the one of the imperative object calculus [AC96] by thread types $\mathrm{Thr}\ A$). Pairs $\tilde{x}_j{:}\widetilde{B}_j$ denote sequences $x_{1_j}{:}B_{1_j} \,..\, x_{n_j}{:}B_{n_j}$. Function types $A \to B$ do only occur in object types $[\mathrm{l}_j{:}\widetilde{B}_j \to \widehat{B}_j]_{j \in J}$, so they are not first-class types. Yet, we sometimes abbreviate such object types by $[\mathrm{l}_j{:}A_j]_{j \in J}$ to clarify that a type is not a thread type. Typed terms are defined by adding type annotations to all binding occurrences of variables: in let-expressions and in method declarations.

For the sake of simplicity, compared to Obliq, in Øjeblik we omit ground values like numbers, booleans, strings, etc.), data operations, and procedures, we restrict field selection to method invocation, we restrict multiple cloning to single cloning, we omit flexibility of object attributes, we replace field aliasing with object aliasing, we omit explicit distribution, and we omit exceptions and advanced synchronisation, so we get a feasible, but still non-trivial language. As in Obliq, computation follows the call-by-value evaluation order. In particular, in the following, whenever we use a term a, we implicitly assume that we have first evaluated a to some actual value, i.e. in most cases to an object reference.

Objects

An object record $[\mathrm{l}_j{=}m_j]_{j\in J}$ is a finite collection of updatable named methods $\mathrm{l}_j{=}m_j$, for pairwise distinct labels l_j. In a method $\varsigma\ s, \tilde{x}$

Self-Infliction

The *current method* of a thread is the last method invoked in it that has not yet

$$\text{T-Var)}\ \frac{\Gamma\ x) = A}{\Gamma \vdash x{:}A} \qquad \text{T-Let)}\ \frac{\Gamma \vdash a{:}A \qquad \Gamma, x{:}A \vdash b{:}B}{\Gamma \vdash \mathsf{let}\, x{:}A = a\, \mathsf{in}\, b : B}$$

$$\text{T-Fork)}\ \frac{\Gamma \vdash a{:}A}{\Gamma \vdash \mathsf{fork}\langle a\rangle : \mathrm{Thr}\ A)} \qquad \text{T-Join)}\ \frac{\Gamma \vdash a : \mathrm{Thr}\ A)}{\Gamma \vdash \mathsf{join}\langle a\rangle : A}$$

$$\text{T-Obj)}\ \frac{\forall j{\in}J \quad \Gamma, s_j{:}A, \tilde{x}_j{:}\widetilde{B}_j \vdash b_j{:}\widehat{B}_j \qquad A = [\mathrm{l}_j{:}\widetilde{B}_j{\to}\widehat{B}_j]_{j\in J}}{\Gamma \vdash [\mathrm{l}_j{=}\varsigma\ s_j{:}A, \tilde{x}_j{:}\widetilde{B}_j)b_j]_{j\in J} : A}$$

$$\text{T-Inv)}\ \frac{\Gamma \vdash a : [\mathrm{l}_j{:}\widetilde{B}_j{\to}\widehat{B}_j]_{j\in J} \qquad \Gamma \vdash \widetilde{b}_k{:}\widetilde{B}_k \qquad k{\in}J}{\Gamma \vdash a.\mathrm{l}_k\langle \widetilde{b}_k \rangle : \widehat{B}_k}$$

$$\text{T-Upd)}\ \frac{\Gamma \vdash a{:}A \qquad A = [\mathrm{l}_j{:}\widetilde{B}_j{\to}\widehat{B}_j]_{j\in J} \qquad \Gamma, s{:}A, \tilde{x}{:}\widetilde{B}_k \vdash b{:}\widehat{B}_k \qquad k{\in}J}{\Gamma \vdash a.\mathrm{l}_k{\Leftarrow}\varsigma\ s{:}A, \tilde{x}{:}\widetilde{B}_k)b : A}$$

$$\text{T-Ping)}\ \frac{\Gamma \vdash a{:}A \qquad A = [\mathrm{l}_j{:}A_j]_{j\in J}}{\Gamma \vdash a.\mathsf{ping} : A}$$

$$\text{T-Clo)}\ \frac{\Gamma \vdash a{:}A \qquad A = [\mathrm{l}_j{:}A_j]_{j\in J}}{\Gamma \vdash a.\mathsf{clone} : A}$$

$$\text{T-Ali)}\ \frac{\Gamma \vdash a, b{:}A \qquad A = [\mathrm{l}_j{:}A_j]_{j\in J}}{\Gamma \vdash a.\mathsf{alias}\langle b\rangle : A}$$

$$\text{T-Sur)}\ \frac{\Gamma \vdash a{:}A \qquad A = [\mathrm{l}_j{:}A_j]_{j\in J}}{\Gamma \vdash a.\mathsf{surrogate} : A}$$

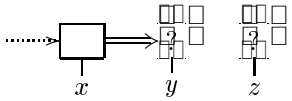
Table 5: Typing Rules for Øjeblik

semantics of alias nodes. We address the reader to [NHKM00, Mer00] for a full explanation about

j

$$
\begin{array}{l}
\mathsf{let}\ z = [\mathrm{l}=\textit{“bar”}]\ \mathsf{in} \\
\quad \mathsf{let}\ y = [\mathrm{l}=\textit{“foo”}]\ \mathsf{in} \\
\qquad \mathsf{let}\ x = [\mathrm{l}=\varsigma\ \ s, w)s.\mathsf{alias}\langle w\rangle\ ]\ \mathsf{in}\ x.\mathrm{l}\langle\, y\,\rangle; x.\mathrm{l}\langle\, z\,\rangle
\end{array}
$$

after it carried out the invocation $x.\mathrm{l}\langle\, y\,\rangle$, that is, when the object referred to by x has turned itself into an alias for y and then terminated its activity. We depict the situation as follows



where, in general, the node x may itself be referred to by other aliases, while y and z may be either an alias or an object record. In fact, the alias $x \quad y$ is *stable* in the very sense: no re-aliasing operation on x to another node will ever possibly take place

By calling $x.\mathrm{l}\langle x\rangle$, the aliasing operation $x.\mathsf{alias}\langle x\rangle$ is carried out giving rise to the cyclic alias chain $x \quad x$. As a consequence, the following external method call $x.\mathrm{k}$ will give rise to a diverging computation.

4.3 On forwarding requests within alias nodes

In this section, we describe the behaviour of single alias nodes in Øjeblik by addressing four crucial questions.

1. *What* is the current self of forwarded requests?
2. *Who* is in charge of sending the result of a forwarded external request?
3. *When* does the forwarding take place?
4. *Which* requests are forwarded and which requests fail in an alias node?

Our semantics behaves as follows:

What? Let a be an alias node forwarding requests to b, that is, $a \quad b$. Let c be a third object invoking a method of a. Then, when serving the external) request, the alias a simply forwards the request to b, and c *is still the current self.* Roughly speaking, it is as if c invokes directly a method of b. The self-inflicted case is trivial because then $a = c$.

Who? As above, let $a \quad b$ and c be a third object invoking a method of a. Since alias nodes simply forward requests unchanged, also *the transmission of the result of the request is delegated to* b. As a consequence: should the request in a have required a mutex, then the mutex can already be released once the request has been forwarded to b.

When? When addressed to stable alias nodes, incoming external requests do not have to wait until previously forwarded requests there can only be external ones in this case) have successfully signalled termination from their point of action. However, when addressed to unstable alias nodes, incoming external requests must wait for the termination of previous external and self-inflicted) requests.

Which? Protected external requests are supposed to fail only when addressed to non-aliased nodes, thus only in endpoints of alias chains.

- Method invocations as well as pings and surrogations) are always forwarded by transitivity to the endpoint of the chain, if it exists).
- Self-inflicted cloning and self-inflicted aliasing are performed at the alias node; external cloning and external aliasing are forwarded because they can possibly reach another node in the alias chain where they are self-inflicted and therefore executable.
- Self-inflicted update requests are forwarded. External update requests are forwarded because they may reach a non-aliased) object that serves them.

5 A translational semantics for Øjeblik

In this section we give a *translational semantics* of Øjeblik into Lπ^+ according to the informal semantics given in Sections 3 and 4. In addition to the syntax of Lπ^+ we use standard abbreviations for:

- *polyadic input* $a\ x_1 \ldots x_n).P \stackrel{\text{def}}{=} a\ y).\mathsf{let}\ \ x_1 \ldots x_n\ ) = y\ \mathsf{in}\ P$ where $y \notin \mathrm{fn}\ P)$. We will also write $\mathbf{C}\ T_1 \ldots T_n)$ instead of $\mathbf{C}\ \langle T_1 \ldots T_n \rangle)$ denoting the type of a channel carrying a tuple.
- *polyadic case destructor* $\ell\_\ x_1 \ldots x_n)\!:\!P \stackrel{\text{def}}{=} \ell\_\ y)\!:\!\mathsf{let}\ \ x_1 \ldots x_n\ ) = y\ \mathsf{in}\ P$, where $y \notin \mathrm{fn}\ P)$;

$$
\begin{array}{rcl}
[\![\, a.\mathsf{clone} \,]\!]_p^k & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q)\, \Big(\, [\![\, a \,]\!]_q^k \mid q(y, k') \,.\, \overline{y}\langle \mathsf{cln\_}p, k' \rangle \\
[\![\, a.\mathsf{alias}\langle b \rangle \,]\!]_p^k & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q_x\, q_y)\, \Big(\, [\![\, a \,]\!]_{q_y}^k \mid q_y(y, k_y).([\![\, b \,]\!]_{q_x}^{k_y} \;\mid\; q_x(x, k_x) \,.\, \overline{y}\langle \mathsf{ali\_}\langle x, p \rangle, k_x \rangle) \\
[\![\, a.\mathrm{l}_j \Leftarrow \varsigma(s, \tilde{x}) b \,]\!]_p^k & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q)\, \Big([\![\, a \,]\!]_q^k \mid q(y, k').(\boldsymbol{\nu} t)\, \big(\, !\, t(s, \widetilde{x}, r, k).[\![\, b \,]\!]_r^k \mid \overline{y}\langle \mathsf{upd}_j\_\langle t, p \rangle, k' \rangle \big)
\end{array}
$$

to become

$$[\![\,\mathbb{O}\,]\!]^{k}_{p} \stackrel{\text{def}}{=} \boldsymbol{\nu} s\widetilde{t})\,\Big(\,\overline{p}\langle s,k\rangle \mid \mathsf{newO}_{\mathbb{O}}\langle s,\widetilde{t}\rangle \mid \prod_{j\in J} !\,t_j\ s_j,\widetilde{x}_j,r,k').[\![\,b_j\,]\!]^{k'}_{r}\,\Big)$$

$$\mathsf{newO}_{\mathbb{O}}\langle s,\widetilde{t}\rangle \stackrel{\text{def}}{=} \boldsymbol{\nu} m_{\mathrm{e}} m_{\mathrm{i}} k_{\mathrm{e}} k_{\mathrm{i}})\,\Big(\,\overline{m_{\mathrm{e}}} \mid \mathsf{OM}_{\mathbb{O}}\langle s,m_{\mathrm{e}},m_{\mathrm{i}},k_{\mathrm{e}},k_{\mathrm{i}},\widetilde{t}\rangle\,\Big)$$

$$\mathsf{newA}_{\mathbb{O}}\langle s,s_{\mathrm{a}}\rangle \stackrel{\text{def}}{=} \boldsymbol{\nu} m_{\mathrm{e}} m_{\mathrm{i}} k_{\mathrm{e}} k_{\mathrm{i}})\,\Big(\,\overline{m_{\mathrm{e}}} \mid \mathsf{AM}_{\mathbb{O}}\langle s,m_{\mathrm{e}},m_{\mathrm{i}},k_{\mathrm{e}},k_{\mathrm{i}},s_{\mathrm{a}}\rangle\,\Big)$$

$$\mathsf{OM}_{\mathbb{O}}\langle s,\widetilde{m},k_{\mathrm{e}},k_{\mathrm{i}},\widetilde{t}\rangle \stackrel{\text{def}}{=} s\ l,k).\ \boldsymbol{\nu} k^{*})\,\Big($$

if $[k{=}k_{\mathrm{i}}]$ then

case l of cln_ r) : $\mathsf{OM}_{\mathbb{O}}\langle s,\widetilde{m},k_{\mathrm{e}},k^{*},\widetilde{t}\rangle \mid \ \boldsymbol{\nu} s^{*})\,(\,\overline{r}\langle s^{*},k^{*}\rangle \mid \mathsf{newO}_{\mathbb{O}}\langle s^{*},\widetilde{t}\rangle\,)$;

ali_ s_{a},r) : $\mathsf{AM}_{\mathbb{O}}\langle s,\widetilde{m},k_{\mathrm{e}},k^{*},s_{\mathrm{a}}\rangle \mid \overline{r}\langle s_{\mathrm{a}},k^{*}\rangle$;

upd_{j}

a

without affecting the state of the manager, so these transitions are completely ignored in Figure 1.

Serving external requests [k=k_{e}

to retrieve the value of a fork'ed term a, but we used it to send the result channel of the join'ing term, together with its current key—this is precisely represented in the translation of Thr A).

According to the translation of types, we can add type declarations in a straightforward way to all bindings in the translation of terms, as mentioned, although omitted, in Section 5.

Types witness the clean representation of Øjeblik terms as π-calculus terms.

Theorem 6.1 (Type Soundness) *Let $a \in \mathcal{L}$, let Γ be a type-environment, and let A be a type. Then $\Gamma \vdash a{:}A$ if and only if $[\![\Gamma]\!], p{:}\mathbf{R}\ [\![A]\!]), k{:}\mathbf{K} \vdash [\![a]\!]_p^k$ for names p and k.*

PROOF. The implication from left to right is proved using induction in the depth of the derivation of $\Gamma \vdash a{:}A$ with a case analysis of the last rule used. The implication from right to left is proved by induction in the structure of a. Details can be found in Appendix A.2. □

In addition to the initial correspondence of types in Øjeblik and their π-calculus counterparts, the preservation of types under reduction in the π-calculus provides us for free with preservation of Øjeblik types, thus witnessing the subject reduction theorem based on the operational

PROOF. By inspection of the encoding. If a manager is present, it must have been created at some point as described in the encoding, because initially, there is none. Upon creation, its name $s$some

where the keys mentioned in $\widetilde{v}$ of $\mathsf{PP}_{\mathbb{O}}\langle \ldots \rangle$ neither match k_{e} nor k_{i}. Notice that $\mathsf{newO}_{\mathbb{O}}\langle s, \widetilde{t} \rangle \equiv \boldsymbol{\nu} k_{\mathrm{i}})\, \mathsf{freeO}_{\mathbb{O}}\langle s, k_{\mathrm{i}}, \widetilde{t}, \emptyset \rangle$, and analogously for $\mathsf{newA}_{\mathbb{O}}\langle \ldots \rangle$.

Observation 4: An

$C[\cdot]$	$::= [\cdot]$	$\mid [\ \mathrm{l}_k{=}\varsigma\ s, \widetilde{x})C[\cdot]\ ,\ \mathrm{l}_{j\neq k}{=}m_{j\neq k}\ ]_{j\in J}$
	$\mid C[\cdot].\mathrm{l}\langle\, \tilde{a}\,\rangle$	$\mid a.\mathrm{l}\langle\, \tilde{a}, C[\cdot], \tilde{a}\,\rangle$
	$\mid C[\cdot].\mathrm{l}{\Leftarrow}m$	$\mid a.\mathrm{l}{\Leftarrow}\varsigma\ s, \widetilde{x})C[\cdot]$
	$\mid C[\cdot].\mathsf{alias}\langle b\rangle$	$\mid a.\mathsf{alias}\langle C[\cdot]\rangle$
	$\mid C[\cdot].\mathsf{clone}$	
	$\mid C[\cdot].\mathsf{surrogate}$	$\mid C[\cdot].\mathsf{ping}$
	$\mid \mathsf{let}\ x = C[\cdot]\ \mathsf{in}\ b$	$\mid \mathsf{let}\ x = a\ \mathsf{in}\ C[\cdot]$
	$\mid \mathsf{fork}\langle C[\cdot]\rangle$	$\mid \mathsf{join}\langle C[\cdot]\rangle$

Table 9: Øjeblik contexts

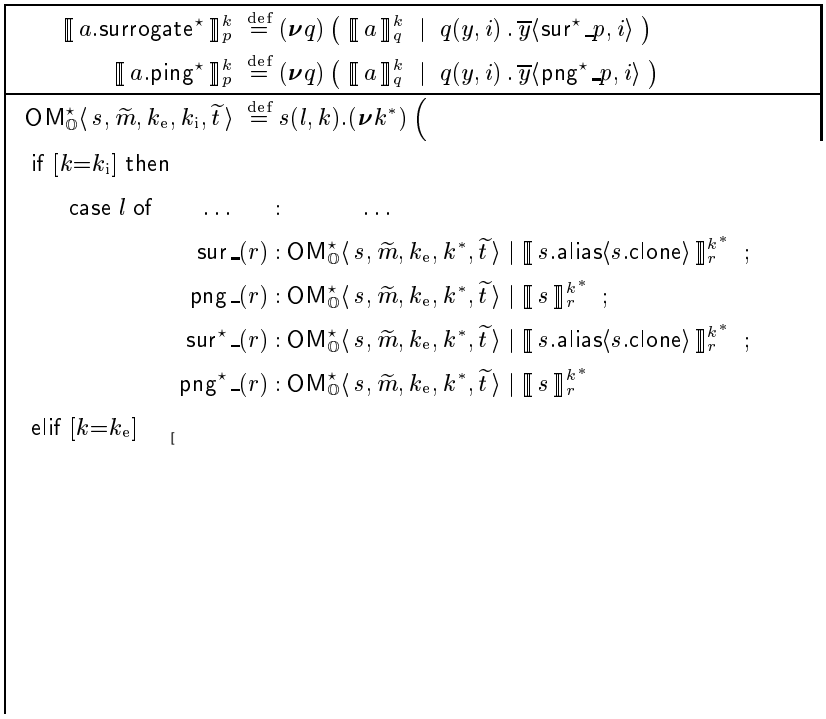
adds one unconditional step after reducing a) and that the notion of equivalence takes all Øjeblik contexts into account, Equation 1 can be reduced to the problem of surrogation on variables:

$$x \doteq x.\mathsf{surrogate} \qquad 2)$$

However, there is an inherent problem with Equation 2, which is exhibited by

7.2 On the absence of self-inflicted surrogation

One of the main observations in [NHKM00] was that the safety equation can not hold in full generality for Øjeblik-contexts, in which the operation $x.\mathsf{surrogate}$ could occur


$[\![\, a.\mathsf{surrogate}^\star \,]\!]_p^k \stackrel{\text{def}}{=} (\boldsymbol{\nu} q)\,(\, [\![\, a \,]\!]_q^k \mid q(y,i)\,.\,\overline{y}\langle \mathsf{sur}^\star\_p, i\rangle\,)$
$[\![\, a.\mathsf{ping}^\star \,]\!]_p^k \stackrel{\text{def}}{=} (\boldsymbol{\nu} q)\,(\, [\![\, a \,]\!]_q^k \mid q(y,i)\,.\,\overline{y}\langle \mathsf{png}^\star\_p, i\rangle\,)$
$\mathsf{OM}^\star_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k_{\mathrm{i}}, \widetilde{t}\,\rangle \stackrel{\text{def}}{=} s(l,k).(\boldsymbol{\nu} k^*)\,($
if $[k{=}k_{\mathrm{i}}]$ then
case l of ... : ...
$\mathsf{sur}\_(r) : \mathsf{OM}^\star_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k^*, \widetilde{t}\,\rangle \mid [\![\, s.\mathsf{alias}\langle s.\mathsf{clone}\rangle \,]\!]_r^{k^*}$;
$\mathsf{png}\_(r) : \mathsf{OM}^\star_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k^*, \widetilde{t}\,\rangle \mid [\![\, s \,]\!]_r^{k^*}$;
$\mathsf{sur}^\star\_(r) : \mathsf{OM}^\star_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k^*, \widetilde{t}\,\rangle \mid [\![\, s.\mathsf{alias}\langle s.\mathsf{clone}\rangle \,]\!]_r^{k^*}$;
$\mathsf{png}^\star\_(r) : \mathsf{OM}^\star_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k^*, \widetilde{t}\,\rangle \mid [\![\, s \,]\!]_r^{k^*}$
elif $[k{=}k_{\mathrm{e}}]$

compares the convergence behaviour of a tagged term and its untagged counterpart with respect to the tagged semantics. By definition, the tagged semantics treats tagged and untagged requests in exactly the same manner. □

Tagging helps us to detect all "requests arising from the hole".

Definition 7.4 (External Contexts) *Let x be a variable and $C[\cdot]$ an untagged Øjeblik context. Then, $C[\cdot]$ is called* external *for x.*surrogate*, if whenever*

$$[\![C[x.\mathsf{surrogate}^\star]]\!]_p^k \;\Rightarrow_{\equiv} E[\, \overline{s}\langle \mathsf{sur}^\star\_r, k\rangle \mid \mathsf{OM}^\star_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k_{\mathrm{i}}, \widetilde{t}\,\rangle\,]$$

it holds that $k \neq k_{\mathrm{i}}$.

We replay the definition using ping instead of surrogate. By definition of the semantics, an Øjeblik context $C[\cdot]$ is then external for $x.\mathsf{surrogate}$ if and only if it is external for $x.\mathsf{ping}$. For convenience, by abuse, we simply call $C[\cdot]$ to be *external for x.*

8 On the safety of surrogation

In this section, we prove that that

$$C[x.\mathsf{ping}]\Downarrow \text{ iff } C[x.\mathsf{surrogate}]\Downarrow$$

under the assumption that $C[\cdot]$ will never lead to self-inflicted

Lemma 8.3 proves that the alias manager

PROOF. By Lemma 7.3 our proof obligation is equivalent to:

$$[\![C[x.\mathsf{ping}^\star]]\!]_p^k \Downarrow_p \text{ iff } [\![C[$$

By the tagged counterpart of) Lemma 6.6 it holds that:

$$P_h \quad \equiv \quad \boldsymbol{\nu}\widetilde{z}_h)\,(\; M_h \mid \mathsf{surO}^{\star}_{\mathbb{O}}\langle\, s_h, q_h, k_h, \widetilde{t}_h, \widetilde{v}_h \,\rangle\;)$$

for some $\widetilde{z}_h$ and M_h. Now, we simulate the previous reduction sequence, which uses $\mathsf{sur}^{\star}$-requests, but now using $\mathsf{png}^{\star}$-requests and proceeding up to structural equivalence *and* barbed equivalence.

$D[\overline{y}\langle \mathsf{png}^{\star}\_q, j\rangle] =$

$$\begin{array}{llllllllllll}
Q_{1,1} & \to_{\mathrm{i}} & Q_{1,2} & \to_{\mathrm{i}} & \cdots & \to_{\mathrm{i}} & Q_{1,n_1} & \to_{\mathrm{s}} & Q_1 & \simeq_{\Gamma} & \widehat{Q}_1 & \equiv & Q_{2,1} \\
Q_{2,1} & \to_{\mathrm{i}} & Q_{2,2} & \to_{\mathrm{i}} & \cdots & \to_{\mathrm{i}} & Q_{2,n_2} & \to_{\mathrm{s}} & Q_2 & \simeq_{\Gamma} & \widehat{Q}_2 & \equiv & Q_{3,1} \\
\vdots & & \vdots & & & & \vdots & & \vdots & & \vdots & & \vdots \\
Q_{d,1} & \to_{\mathrm{i}} & Q_{d,2} & \to_{\mathrm{i}} & \cdots & \to_{\mathrm{i}} & Q_{d,n_d} & \to_{\mathrm{s}} & Q_d & \simeq_{\Gamma} & \widehat{Q}_d & \equiv & Q_{d+1,1} \\
Q_{d+1,1} & \to_{\mathrm{i}} & Q_{d+1,2} & \to_{\mathrm{i}} & \cdots & \to_{\mathrm{i}} & Q_{d+1,n_{d+1}} & \stackrel{\mathrm{def}}{=} & Q{\downarrow_p} & & & &
\end{array}$$

where:

$$Q_{h,g} \quad \stackrel{\mathrm{def}}{=} \quad P_{h,g}\big[{}^{\mathsf{png}^{\star}}\!/_{\mathsf{sur}^{\star}}\big]$$

The insignificant reduction steps $\to_{\mathrm{i}}$ exist because of Lemma 8.8. The significant reduction steps $Q_{h,n_h} \to_{\mathrm{s}} Q_h$ are analogous to

$$\text{T-Obj)}\ \frac{\forall j{\in}J \quad \Gamma, s_j{:}A, \tilde{x}_j{:}\widetilde{B}_j \vdash_A b_j{:}\widehat{B}_j \qquad A = [\mathrm{l}_j{:}\widetilde{B}_j{\to}\widehat{B}_j]_{j\in J}}{\Gamma \vdash_D [\mathrm{l}_j : \varsigma\ s_j{:}A, \tilde{x}_j{:}\widetilde{B}_j)b_j]_{j\in J} : A}$$

$$\text{T-Upd)}\ \frac{\Gamma \vdash_D a{:}A \qquad A = [\mathrm{l}_j{:}\widetilde{B}_j{\to}\widehat{B}_j]_{j\in J} \qquad \Gamma, s_k{:}A, \tilde{x}_k{:}\widetilde{B}_k \vdash_A b_k{:}\widehat{B}_k \qquad k{\in}J}{\Gamma \vdash_D a.\mathrm{l}_k{\Leftarrow}\varsigma\ s_k{:}A, \tilde{x}_k{:}\widetilde{B}}$$

semantics for Øjeblik, the question for some formal correspondence result among the semantics by translation and the direct semantics arises. On the other hand, one may ask to carry out the proofs on the direct semantics instead of employing some other lower-level formalism. However, we found it very natural and useful to develop two semantics at different abstraction levels hand-in-hand. In fact, most of the examples of unsafe surrogation were discovered by means of the π-calculus semantics, and only then "verified" in the direct semantics. Moreover, since we have developed both levels of semantics in lock-step, we have a good basis for formalizing their interrelation. Finally, in contrast to our abstract configuration-style semantics for closed terms only, the π-calculus provides indeed a very rich set of approved reasoning tools that make the life of a theorem prover much easier, as exemplified by Kleist and Sangiorgi [KS98], and also in this paper.

Other strands of future work are twofold. One is to continue to develop and exploit semantics for the Obliq-style of object migration, and to use our semantics also to prove other equations on Obliq-programs. For example, also equations like $\mathsf{join}\langle\mathsf{fork}\langle a\rangle\rangle{=}a$ do only hold under certain conditions inflicted by self-infliction. Another strand is to try to carry over our results to settings that are not based on the notion of serialization via self-infliction, but rather reentrant mutexes, as in Java.

Acknowledgements

We thank Luca Cardelli for several useful discussions on Obliq. We also thank Giuseppe Castagna, Rocco De Nicola, Joachim Parrow, and Davide Sangiorgi for comments on an early draft.

A Proofs

A.1 Proof of Lemma 2.14

PROOF. We show that the relation

$$\mathcal{S} = \{\big(Q\{p/q\}\,,\ \ \boldsymbol{\nu} q{:}\mathbf{C}\ T))\ \ Q \mid q \triangleright p)\big)\ :\ q \text{ in } Q \text{ only in output position}\}$$

is a barbed bisimulation up to structural equivalence.

- Let $Q\{p/q\} \xrightarrow{\tau} Q'\{p/q\}$. There are two cases.
  1. $Q \xrightarrow{\tau} Q'$. This case can be easily treated.
  2. Otherwise, since p and q are channels and they never appear in testing, this means that the τ-action is due to a communication along p. More precisely, Q must contain an occurrence of q in output subject position and an occurrence of p in input position which give rise to the communication. Up to structural equivalence, this implies that

     $$\boldsymbol{\nu} q{:}\mathbf{C}\ T))\ \ Q \mid q \triangleright p) \xrightarrow{\tau} \xrightarrow{\tau} \equiv\ \boldsymbol{\nu} q{:}\mathbf{C}\ T))\ \ Q' \mid q \triangleright p).$$

     As desired.
- Let $\boldsymbol{\nu} q{:}\mathbf{C}\ T))\ \ Q \mid q \triangleright p) \xrightarrow{\tau} R$ for some R. There are two cases.
  1. $R = \boldsymbol{\nu} q{:}\mathbf{C}\ T))\ \ Q' \mid q \triangleright p)$ since $Q \xrightarrow{\tau} Q'$. This case can be easily treated.
  2. The τ-action is due to some communication along q between Q and the link $q \triangleright p$. More precisely,

Before we start, let

$$A^*\ X) \text{ denote } \left[\begin{array}{lcc} \mathsf{cln} & : & \mathbf{R}\ X) \\ \mathsf{ali} & : & \langle X, \mathbf{R}\ X)\rangle \\ \mathsf{upd}_j & : & \langle \mathbf{C}\ X, \mathbf{M}\ \widetilde{B}_j{\rightarrow}\widehat{B}_j), \mathbf{K}), \mathbf{R}\ X)\rangle \\ \mathsf{inv}_j & : & \langle \mathbf{M}\ \widetilde{B}_j{\rightarrow}\widehat{B}_j)\rangle \\ \mathsf{sur} & : & \mathbf{R}\ X) \\ \mathsf{png} & : & \mathbf{R}\ X) \end{array}\right]_{j\in 1..n,}$$

with

and in order to type the object manager we must also have $K = J$ in order to have the same number of methods in the type and the object manger. The typing of the object manger also yields that we must have the types $T_j = \mathbf{C}\ [\![A]\!], [\![\widetilde{B}_j]\!], \mathbf{R}\ \widehat{B}_j), \mathbf{K})$. We are now

In state OM^{s}, a png request drives the system into state OM^{i}. In the case of method invocation a reduction along t_j may occur which allows the evaluation of the method body. At this point a number of self-inflicted requests may be served external requests are blocked because the external mutex m_{e} is no available). This part of the computation will not change the state. Notice that, by hypothesis, since we suppose that Z contain an object manager and non an alias manager, we exclude self-inflicted aliasing operations. When the last self-inflicted request is served, a reply $\overline{r^*}\langle\, o, k\,\rangle$ will appear unguarded. The confluent reduction along r^* will drive the computation to state OM^{i}. sur requests are treated similarly.

State OM^{i} can only evolve, by reducing along m_{i}, to state OM^{f}. □

A.4 Proof of Lemma 8.2

We show that there is a sequence of τ-actions such that:

$$\begin{array}{l}\mathsf{surO}_{\mathbb{O}}\langle\, s, r, k, \tilde{t}, \tilde{v}\,\rangle \Rightarrow_{\equiv} \\ \quad \boldsymbol{\nu} s^*)\,\big(\;\boldsymbol{\nu} k_{\mathrm{i}})\,\mathsf{freeA}_{\mathbb{O}}\langle\, s, k_{\mathrm{i}}, s^*, \tilde{v}\,\rangle \;\big|\; \mathsf{newO}_{\mathbb{O}}\langle\, s^*, \tilde{t}\,\rangle \;\big|\; \overline{r}\langle s^*, k\rangle\big).\end{array}$$

We prove that $\approx_{\Gamma;s}$ is insensitive to these particular τ-actions. To this end, we supply the two lemmas A.2 and A.3. We recall that $\mathsf{M}[\cdot]$ denote the call manager protocol as defined in Table 7.

Lemma A.2 *Let $\widetilde{n} := m_{\mathrm{e}}, m_{\mathrm{i}}, k_{\mathrm{e}}$, and $\widetilde{v} := v_1 \,..\, v_n$ with $v_j := \langle\, l_j, k_j\,\rangle$ for $j \in 1..n$, and*

$$\begin{array}{rcl} C_1 & := & \mathsf{M}[\;\boldsymbol{\nu} q)\;\; \overline{s}\langle \mathsf{cln\_}q, k^*\rangle \mid q\;\; x, k').\overline{s}\langle \mathsf{ali\_}\langle x, r^*\rangle, k'\rangle)\;] \\ C_2 & := & \mathsf{M}[\;\boldsymbol{\nu} q)\;\; \overline{q}\langle s^*, k^*\rangle \mid q\;\; x, k').\overline{s}\langle \mathsf{ali\_}\langle x, r^*\rangle, k'\rangle)\;] \\ P\langle\widetilde{v}\rangle & := & \boldsymbol{\nu}\widetilde{n}k^*)\,\big(\,\overline{m_{\mathrm{i}}}k \,\big|\, \mathsf{OM}_{\mathbb{O}}\langle\, s, \widetilde{n}, k^*, \widetilde{t}\,\rangle \,\big|\, \mathsf{PP}_{\mathbb{O}}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle \,\big|\, C_1\,\big) \\ & & \text{with } k^* \notin \mathrm{fn}\;\widetilde{v}) \\ Q\langle\widetilde{v}\rangle & := & \boldsymbol{\nu}\widetilde{n}k^*s^*)\,\big(\,\overline{m_{\mathrm{i}}}k \,\big|\, \mathsf{OM}_{\mathbb{O}}\langle\, s, \widetilde{n}, k^*, \widetilde{t}\,\rangle \,\big|\, \mathsf{newO}_{\mathbb{O}}\langle\, s^*, \widetilde{t}\,\rangle \,\big|\, \mathsf{PP}_{\mathbb{O}}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle \,\big|\, C_2\,\big) \\ & & \text{with } k^* \notin \mathrm{fn}\;\widetilde{v}) \\ \Gamma & \vdash & P\langle\widetilde{v}\rangle, Q\langle\widetilde{v}\rangle \;\text{ for some }\; \Gamma. \end{array}$$

Then, $P\langle\widetilde{v}\rangle \approx_{\Gamma;s} Q\langle\widetilde{v}\rangle$.

Proof. For simplicity, we omit the obligations on types in the coinductive definition of $\approx_{\Gamma;s}$. So, we prove that the relation:

$$\mathcal{S} = \{\; P\langle\widetilde{w}\rangle, Q\langle\widetilde{w}\rangle) : \widetilde{w} = w_1 \,..\, w_m \;\text{ with }\; w_j := \langle\, l_j, k_j\,\rangle,\; j \in 1..n\} \cup \mathcal{I}$$

where $\mathcal{I}$ is the identity relation, is a $\approx_{\Gamma;s}$-bisimulation up to $\equiv$.

The only channel which appear free in subject position in $P\langle\widetilde{w}\rangle$ and $Q\langle\widetilde{w}\rangle$ is s. Since both the external key k_{e} and the internal key k^* are restricted in $P\langle\widetilde{w}\rangle$ and $Q\langle\widetilde{w}\rangle$, an by well-typedness, the environment can send requests only of the form $\overline{s}\langle l, k\rangle$ with $k_{\mathrm{e}} \neq k \neq k^*$.

The process $P\langle\widetilde{w}\rangle$ can perform only two kinds of actions. Either i) an input action $s\langle l, k\rangle$ with $k_{\mathrm{e}} \neq k \neq k^*$), or ii) a silent move along s involving the self-inflicted cloning request contained in C_1. In case i), the pre-processing of the request creates the process $m_{\mathrm{e}}.\,\overline{s}\langle l, k_{\mathrm{e}}\rangle \mid \overline{m_{\mathrm{i}}}k)$ which can be added in $\mathsf{PP}_{\mathbb{O}}\langle\, s, \widetilde{n}, \widetilde{w}\,\rangle$ obtaining some $\mathsf{PP}_{\mathbb{O}}\langle\, s, \widetilde{n}, \widetilde{w'}\,\rangle$ with $\widetilde{w'} = \widetilde{w} \cup \langle l, k\rangle$. The process $Q\langle\widetilde{w}\rangle$ can perform the same action and the derivatives are again related by $\mathcal{S}$. In case ii), the process $Q\langle\widetilde{w}\rangle$ can mimic the τ-action by not performing any reduction at all. Up to structural equivalence, we get into the identity relation.

The process $Q\langle\widetilde{w}\rangle$ can only perform two kinds of actions. Either i) a input action $s\langle l, k\rangle$ with $k_{\mathrm{e}} \neq k \neq k^*$), and we reason as above, or ii) a silent move along the restricted channel q in C_2. In this case $P\langle\widetilde{w}\rangle$ can perform two silent actions, along s and q, getting, up to structural equivalence, into the identity relation. □

Lemma A.3 *Let $\widetilde{n} := m_{\mathrm{e}}, m_{\mathrm{i}}, k_{\mathrm{e}}$, and $\widetilde{v} := v_1 \,..\, v_n$ with $v_j := \langle\, l_j, k_j\,\rangle$ for $j \in 1..n$, and*

$$
\begin{array}{rcl}
C_3 & := & \mathsf{M}[\,\overline{s}\langle \mathsf{ali\_}\langle s^*, r^*\rangle, k^*\rangle\,] \\
C_4 & := & \mathsf{M}[\,\overline{r^*}\langle s^*, k^*\rangle\,] \\
P\langle \widetilde{v}\rangle & := & \boldsymbol{\nu}\widetilde{n}k^*s^*)\big(\,\overline{m_{\mathrm{i}}}k\big|\,\mathsf{OM}_{\mathbb{O}}\langle\, s, \widetilde{n}, k^*, \widetilde{t}\,\rangle \,\big|\, \mathsf{newO}_{\mathbb{O}}\langle\, s^*, \widetilde{t}\,\rangle \,\big|\, \mathsf{PP}_{\mathbb{O}}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle \,\big|\, C_3\,\big) \\
 & & \text{with } k^* \notin \text{fn } \widetilde{v}) \\
Q\langle \widetilde{v}\rangle & := & \boldsymbol{\nu}\widetilde{n}k^*s^*)\big(\,\overline{m_{\mathrm{i}}}k\big|\,\mathsf{AM}_{\mathbb{O}}\langle\, s, \widetilde{n}, k^*, s^*\,\rangle \,\big|\, \mathsf{newO}_{\mathbb{O}}\langle\, s^*, \widetilde{t}\,\rangle \,\big|\, \mathsf{PP}_{\mathbb{O}}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle \,\big|\, C_4\,\big) \\
 & & \text{with } k^* \notin \text{fn } \widetilde{v}). \\
\Gamma & \vdash & P\langle \widetilde{v}\rangle, Q\langle \widetilde{v}\rangle \text{ for some } \Gamma.
\end{array}
$$

Then, $P\langle \widetilde{v}\rangle \approx_{\Gamma;s} Q\langle \widetilde{v}\rangle$.

Proof. Similar to that of Lemma A.2. □

Proof of Lemma 8.2. As said above there is a sequence of τ-actions, such that:

$$
\mathsf{surO}_{\mathbb{O}}\langle\, s, r, k, \tilde{t}, \tilde{v}\,\rangle \Rightarrow_{\equiv} \boldsymbol{\nu}s^*)\,\big(\;\boldsymbol{\nu}k_{\mathrm{i}})\,\mathsf{freeA}_{\mathbb{O}}\langle\, s, k_{\mathrm{i}}, s^*, \tilde{v}\,\rangle \,\big|\, \mathsf{newO}_{\mathbb{O}}\langle\, s^*, \tilde{t}\,\rangle \,\big|\, \overline{r}\langle s^*, k\rangle\big).
$$

The above sequence consists of 7 silent steps. These τ-steps are of two kinds: i) confluent reductions along restricted channels of the form

$$
C[\;\boldsymbol{\nu}q)\;\; \overline{q}\langle \widetilde{v}\rangle \mid q\;\; \widetilde{x}).P)] \;\xrightarrow{\tau}_{\equiv} C[P\{{}^{\widetilde{v}}/_{\widetilde{x}}\}]
$$

where $q \notin \text{fn } P$), let us call these reductions of kind α; ii) reductions involving self-inflicted requests induced by the surrogation) of the form

$$
C[\;\boldsymbol{\nu}k^*)\;\; \mathsf{OM}_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k^*, \tilde{t}\,\rangle \mid \overline{s}\langle \mathsf{op\_}r^*, k^*\rangle)] \xrightarrow{\tau}
$$

where $k^* \notin \text{fn}\ \widetilde{v})$.

In the fifth τ-step we reduce the self-inflicted aliasing request contained in C_3. So, let us denote with C_4 the process $\mathsf{M}[\,\overline{r^*}\langle s^*, k^*\rangle\,]$. It holds that the process

$$\boldsymbol{\nu}\widetilde{n}k^*s^*)\big(\,\overline{m_{\mathrm{i}}}k\big|\,\mathsf{OM}_{\mathbb{O}}\langle\, s, \widetilde{n}, k^*, \widetilde{t}\,\rangle\,\big|\,\mathsf{newO}_{\mathbb{O}}\langle\, s^*, \widetilde{t}\,\rangle\,\big|\,\mathsf{PP}_{\mathbb{O}}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle\,\big|\,C_3\,\big)$$

reduces, up to structural equivalence, to

$$\boldsymbol{\nu}\widetilde{n}k^*s^*)\big(\,\overline{m_{\mathrm{i}}}k\big|\,\mathsf{AM}_{\mathbb{O}}\langle\, s, \widetilde{n}, k^*, s^*\,\rangle\,\big|\,\mathsf{newO}_{\mathbb{O}}\langle\, s^*, \widetilde{t}\,\rangle\,\big|\,\mathsf{PP}_{\mathbb{O}}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle\,\big|\,C_4\,\big)$$

where $k^* \notin \text{fn}\ \widetilde{v})$. By Lemma A.3 the relation $\approx_{\Gamma;s}$ is insensitive to this reduction.

The sixth and the seventh reductions are of kind α and involve channels r^* and m_{i}, respectively. Up to structural equivalence we get the desired process

$$\boldsymbol{\nu}s^*)\,(\ \boldsymbol{\nu}k_{\mathrm{i}})\,\mathsf{freeA}_{\mathbb{O}}\langle\, s, k_{\mathrm{i}}, s^*, \tilde{v}\,\rangle\,\big|\,\mathsf{newO}_{\mathbb{O}}\langle\, s^*, \tilde{t}\,\rangle\,\big|\,\overline{r}\langle s^*, k\rangle).$$

$\square$

A.5 Proof of Lemma 8.3

Lemma 8.3 proves that the aliased object manager appearing in Lemma 8.2 behaves as a forwarder.

As a first step we recall a well-known property of replicated input.

Lemma A.4 *Let $C[\cdot]$ be a π-calculus context where channel c does not appear either in input or in output object position. Then*

$$\boldsymbol{\nu}c)\ (\ !\,c\ x).P\ \big|\ C[\overline{c}v]\ )\approx_{\Gamma}\ \ \boldsymbol{\nu}c)\ (\ !\,c\ x).P\ \big|\ C[P\{^v\!/_x\}]\ )$$

Proof. By applying Milner's replications theorems [Mil93]. $\square$

Proof of Lemma 8.3. The obligations on types guarantee that values received along channel s are of the right type. This allows us to use polyadic input along s. By observing process $\boldsymbol{\nu}k_{\mathrm{i}})\,\mathsf{AM}_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k_{\mathrm{i}}, s^*\,\rangle$ we note that, since k_{i} is restricted and never extruded, the aliased object manager will never receive self-inflicted requests. By exhibiting the appropriate bisimulation, we can prove that such a process has the following functional behaviour.

$$\boldsymbol{\nu}k_{\mathrm{i}})\ \ \mathsf{AM}_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k_{\mathrm{i}}, s^*\,\rangle) \approx_{\Gamma} !\,s\ \ l, k).\mathsf{if}\ [k{=}k_{\mathrm{e}}]\ \ \mathsf{then}\ m_{\mathrm{i}}\ \ k).\ \overline{s^*}\langle l, k\rangle \mid \overline{m_{\mathrm{e}}})$$
$$\mathsf{else}\ \ m_{\mathrm{e}}.\ \overline{s}\langle l, k_{\mathrm{e}}\rangle \mid \overline{m_{\mathrm{i}}}k)$$

Since $\approx_{\Gamma}$ is preserved by parallel composition and restriction, we have that:

$$\boldsymbol{\nu}k_{\mathrm{i}})\ \ \mathsf{freeA}_{\mathbb{O}}\langle\, s, k_{\mathrm{i}}, s^*, \widetilde{v}\,\rangle)$$
$$\approx_{\Gamma}$$
$\boldsymbol{\nu}\widetilde{m}k_{\mathrm{e}})\,\big(\,\overline{m_{\mathrm{e}}}\ \mid\ !\,s\ \ l, k).\mathsf{if}\ [k{=}k_{\mathrm{e}}]$ then m_{i} self-inflicted)Tj55.80Td requests.)Tj42.96020Td By)Tj1

m_{e}.

$$\boldsymbol{\nu}\widetilde{m}k_{\mathrm{e}})\,\big(\,\overline{m_{\mathrm{e}}}\;\mid\;!s\;\;l,k).\mathsf{if}\;[k{=}k_{\mathrm{e}}]\;\;\mathsf{then}\;m_{\mathrm{i}}\;\;k).\;\overline{s^*}\langle l,k\rangle\mid\overline{m_{\mathrm{e}}})$$
$$\mathsf{else}\;\;\;m_{\mathrm{e}}.\;\overline{s}\langle l,k_{\mathrm{e}}\rangle\mid\overline{m_{\mathrm{i}}}k)$$
$$\mid\;\prod_{j\in 1..n}m_{\mathrm{e}}.\big(\overline{s}\langle l_j,k_{\mathrm{e}}\rangle\mid\overline{m_{\mathrm{i}}}k_j\big)\,\big)$$

$\approx_{\Gamma;s}$ by exhibiting the appropriate bisimulation)

$$\boldsymbol{\nu}\widetilde{m}s_{\mathrm{e}})\,\big(\,\overline{m_{\mathrm{e}}}\;\mid\;!s\;\;l,k).m_{\mathrm{e}}.\;\overline{s_{\mathrm{e}}}\langle l,k\rangle\mid\overline{m_{\mathrm{i}}}k)$$
$$\mid\;!s_{\mathrm{e}}\;\;l,k).m_{\mathrm{i}}\;\;k).\;\overline{s^*}\langle l,k\rangle\mid\overline{m_{\mathrm{e}}})$$
$$\mid\;\prod_{j\in 1..n}m_{\mathrm{e}}.\;\overline{s_{\mathrm{e}}}\langle l_j,k_j\rangle\mid\overline{m_{\mathrm{i}}}k_j)\,\big)$$

$\approx_{\Gamma}$ reductions on m_{i} are confluent)

$$\boldsymbol{\nu}m_{\mathrm{e}}s_{\mathrm{e}})\,\big(\,\overline{m_{\mathrm{e}}}\;\mid\;!s\;\;l,k).m_{\mathrm{e}}.\overline{s_{\mathrm{e}}}\langle l,k\rangle$$
$$\mid\;!s_{\mathrm{e}}\;\;l,k).\;\overline{s^*}\langle l,k\rangle\mid\overline{m_{\mathrm{e}}})$$
$$\mid\;\prod_{j\in 1..n}m_{\mathrm{e}}.\overline{s_{\mathrm{e}}}\langle l_j,k_j\rangle\,\big)$$

$\approx_{\Gamma}$ by Lemma A.4)

$$\boldsymbol{\nu}m_{\mathrm{e}}s_{\mathrm{e}})\,\big(\,\overline{m_{\mathrm{e}}}\;\mid\;!s\;\;l,k).m_{\mathrm{e}}.\big(\,\overline{s^*}\langle l,k\rangle\mid\overline{m_{\mathrm{e}}}\,\big)$$
$$\mid\;!s_{\mathrm{e}}\;\;l,k).\;\overline{s^*}\langle l,k\rangle\mid\overline{m_{\mathrm{e}}})$$
$$\mid\;\prod_{j\in 1..n}m_{\mathrm{e}}.\;\overline{s^*}\langle l_j,k_j\rangle\mid\overline{m_{\mathrm{e}}})\,\big)$$

$\approx_{\Gamma}$ by garbage collection on s_{e})

$$\boldsymbol{\nu}m_{\mathrm{e}})\,\big($$

We recall that $\approx_{\Gamma;s}$ is ground on channels. This means that we alway suppose to receive fresh channels, in particular, we never receive channels s and s^*.

As regards the left side, the only interesting transition is the input action along s. This action can be emulated by the

3. If $\boldsymbol{\nu}\widetilde{z})\ A \mid R) \xrightarrow{\tau} \boldsymbol{\nu}\widetilde{y})\ A' \mid R')$, where the τ-action is due to a communication along s between A and R recall that s can only appear in output in R), then we reason

References

[AC96] M. Abadi and L. Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer, 1996.

[ACS98] R. M. Amadio, I. Castellani and D. Sangiorgi. On Bisimulations for the Asynchronous π-Calculus. *Theoretical Computer Science*, 195(2):291–324, 1998. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 147–162.

[Bou92] G. Boudol. Asynchrony and the π-calculus (Note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.

[Car94] L. Cardelli. `obliq-std.exe` — Binaries for Windows NT. `http://www.luca.demon.co.uk/Obliq/Obliq.html`, 1994.

[Car95] L. Cardelli. A Language with Distributed Scope. *Computing Systems*, 8(1):27–59, 1995. Short version in *Proceedings of POPL '95*. A preliminary version appeared as Report 122, Digital Systems Research, June 1994.

[DF96] P. Di Blasio and K. Fisher. A Concurrent Object Calculus. In U. Montanari and V. Sassone, eds, *Proceedings of CONCUR '96*, volume 1119 of *LNCS*, pages 655–670. Springer, 1996. An extended version appeared as Stanford University Technical Note STAN-CS-TN-96-36, 1996.

[FG96] C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the Join-Calculus. In *Proceedings of POPL '96*, pages 372–385. ACM, Jan. 1996.

[GH98] A. D. Gordon and P. D. Hankin. A Concurrent Object Calculus: Reduction and Typing. In U. Nestmann and B. C. Pierce, eds, *Proceedings of HLCL '98*, volume 16.3 of *ENTCS*. Elsevier Science Publishers, 1998.

[GHL97] A. D. Gordon, P. D. Hankin and S. B. Lassen. Compilation and Equivalence of Imperative Objects. In S. Ramesh and G. Sivakumar, eds, *Proceedings of FSTTCS '97*, volume 1346 of *LNCS*, pages 74–87. Springer, Dec. 1997. Full version available as Technical Report 429, University of Cambridge Computer Laboratory, June 1997.

[HK96] H. Hüttel and J. Kleist. Objects as Mobile Processes. Research Series RS-96-38, BRICS, Oct. 1996. Presented at MFPS '96.

[HKMN99] H. Hüttel, J. Kleist, M. Merro and U. Nestmann. Migration = Cloning ; Aliasing (Preliminary Version). In *Informal Proceedings of the Sixth International Workshop on Foundations of Object-Oriented Languages (FOOL 6, San Antonio, Texas, USA)*. Sponsored by ACM/SIGPLAN, 1999.

[Hon92] K. Honda. Two bisimilarities for the ν-calculus. Technical Report 92-002, Keio University, 1992.

[HT91] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communication. In P. America, ed, *Proceedings of ECOOP '91*, volume 512 of *LNCS*, pages 133–147. Springer, July 1991.

[HY95] K. Honda and N. Yoshida. On Reduction-Based Process Semantics. *Theoretical Computer Science*, 152(2):437–486, 1995. An extract appeared in *Proceedings of FSTTCS '93*, LNCS 761.

[JLHB88] E. Jul, H. Levy, N. Hutchinson and A. Black. Fine-Grained Mobility in the Emerald System. *ACM Transactions of Computer Systems*, 6(1), Feb. 1988.

[KS98] J. Kleist and D. Sangiorgi. Imperative Objects and Mobile Processes. In D. Gries and W.-P. de Roever, eds, *Proceedings of PROCOMET '98*, pages 285–303. International Federation for Information Processing (IFIP), Chapman & Hall, 1998.

[Mer00] M. Merro. *Locality in the π-calculus and applications to distributed objects*. PhD thesis, Ecole des Mines, France, October 2000.

[Mil93] R. Milner. The Polyadic π-Calculus: A Tutorial. In F. L. Bauer, W. Brauer and H. Schwichtenberg, eds, *Logic and Algebra of Specification*, volume 94 of *Series F: Computer and System Sciences*. NATO Advanced Study Institute, Springer, 1993. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.

[Mor68] J.-H. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT, 1968.

[MS92] R. Milner and D. Sangiorgi. Barbed Bisimulation. In W. Kuich, ed, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[MS98] M. Merro and D. Sangiorgi. On Asynchrony in Name-Passing Calculi. In K. G. Larsen, S. Skyum and G. Winskel, eds, *Proceedings of ICALP '98*, volume 1443 of *LNCS*, pages 856–867. Springer, July 1998.

[NHKM00] U. Nestmann, H. Hüttel, J. Kleist and M. Merro. Aliasing Models for Mobile Objects. Accepted for *Journal of Information and Computation*. Available from `http://www.cs.auc.dk/research/FS/ojeblik/`. An extended abstract has appeared as Distinguished Paper in the *Proceedings of EUROPAR '99*, LNCS 1685, 2000.

[PS96] B. C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996. An extract appeared in *Proceedings of LICS '93*: 376–385.

[PW98] A. Philippou and D. Walker. On Transformations of Concurrent Object Programs. *Theoretical Computer Science*, 195(2):291–324, 1998. An extended abstract appeared in

[San98] D. Sangiorgi. An Interpretation of Typed Objects into Typed π-Calculus. *Information and Computation*, 143(1):34–73, 1998. Earlier version published as Rapport de Recherche RR-3000, INRIA Sophia-Antipolis, August 1996.

[San99a] D. Sangiorgi. The Name Discipline of Uniform Receptiveness. *Theoretical Computer Science*, 221(1–2):457–493, 1999. An abstract appeared in the *Proceedings of ICALP '97*, LNCS 1256, pages 303–313.

[San99b] D. Sangiorgi. The Typed π-Calculus at work: A Proof of Jones's Parallelisation Theorem on Concurrent Objects. *Theory and Practice of Object-Oriented Systems*, 5(1), 1999. An early version was included in the *Informal proceedings of FOOL 4*, January 1997.

[San00] D. Sangiorgi. Lazy Functions and Mobile Processes. In G. Plotkin, C. Stirling and M. Tofte, eds, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Foundations of Computing. MIT Press, May 2000. Available as INRIA Sophia-Antipolis Rapport de Recherche RR-2515.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001. To appear.

[VHB+97] P. Van Roy, S. Haridi, P. Brand, G. Smolka, M. Mehl and R. Scheidhauer. Mobile Objects in Distributed Oz. *ACM Transactions on Programming Languages and Systems*, 19(5):804–851, Sept. 1997.

[Wal95] D. Walker. Objects in the π-calculus. *Information and Computation*, 116(2):253–271, 1995.

Contents