Behavioural Types for a Calculus of Concurrent Objects*

António Ravara — Departamento de Matemática, Instituto Superior Técnico, Universidade Técnica de Lisboa

Vasco Vasconcelos — Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa

Abstract. We present a new type system for TyCO, a name-passing calculus of concurrent objects. The system captures dynamic aspects of objects' behaviours, namely non-uniform service availability of active objects. The notion of processes without errors is loosened, demanding only weak fairness in the treatment of messages.

1 Motivation

This paper proposes a type system for TyCO (TYped Concurrent Objects) [VT93], a name-passing calculus of concurrent objects. In a setting where (active) concurrent objects are characterized by *non-uniform service availability* [Nie95], a static "types-as-interfaces" approach is not suitable to capture dynamic aspects of objects' behaviours. We propose types as graphs (representing objects as state-transition systems), and demand weak fairness in the treatment of messages. The type system is able to type objects with a non-uniform service availability, while preserving the subject-reduction property.

A typical process not typable by "traditional" type systems [VH93, VT93, KY95, LW95] is a one-place buffer that only allows read operations when it is full, and write operations when it is empty.

$$\begin{aligned}\mathrm{Empty}(b) &= b \rhd [\, \mathit{write} : (u)\, \mathrm{Full}(b\, u)\,] \\ \mathrm{Full}(b\, u) &= b \rhd [\, \mathit{read} : (r)\, r \lhd \mathit{val} : [u] \mid \mathrm{Empty}(b)\,]\end{aligned}$$

The type systems mentioned above assign interface-like types to names. Therefore, name b should have a single interface, containing both methods' labels *write* and *read*, and thus the example presented can not be typed. Nevertheless, the behaviour of the process (alternating between *write* and *read* operations) is very clear. Furthermore, a process containing the redex $\mathrm{Empty}(b) \mid b \lhd \mathit{read}\colon [r]$ should not be considered an error, for the presence of a message $b \lhd \mathit{write}\colon [u]$ makes the reception of the *read* message possible.

The development of a type system able to type processes like the one above is the main motivation of this work. This paper is a short version of [RV97].

* This work was partially supported by JNICT PRAXIS XXI projects 2/2.1/MAT/46/94 Escola, 2/2.1/MAT/262/94 SitCalc and 2/2.1/TIT/1658/95 LogComp, and by the ESPRIT Working Groups 22704 ASPIRE and 23531 FIREworks.

2 The calculus of objects

TyCO is an object-oriented name-passing calculus with asynchronous communication between concurrent objects via labelled messages carrying names. The calculus is developed along the trends of well-known models of concurrency, such as the π-calculus [MPW92], the ν-calculus [HT91], and the actor model [Agh86].

Consider *names* $u, v, x, y \in \mathcal{N}$, *labels* $\mathsf{a}, \mathsf{b}, \mathsf{c} \in \mathcal{L}$, and *processes* $P, Q \in \mathcal{P}$. Let $\tilde{v}$ stand for a sequence of names, and $\tilde{x}$ for a sequence of pairwise distinct names.

Definition 1. The set $\mathcal{P}$ of *processes* is given by the following grammar.

$$P \ ::= \ x \rhd M \ \mid \ x \lhd m \ \mid \ P \mid Q \ \mid \ \nu x\, P \ \mid \ !x \rhd M \ \mid \ \mathbf{0}$$

where $M \stackrel{\text{def}}{=} \sum_{i \in \mathrm{I}} \mathsf{a}_i : (\tilde{x}_i)\, P_i$ for I a finite index set, and $m \stackrel{\text{def}}{=} \mathsf{a} : [\tilde{v}]$.

The basic processes are objects $x \rhd M$, located at some name x and composed of a finite collection M of labelled methods (with pairwise distinct labels), and asynchronous labelled messages $x \lhd \mathsf{a} : [\tilde{v}]$, targeted at some object's location x and selecting its method a with actual parameters $\tilde{v}$. Each method $\mathsf{a} : (\tilde{x})\, P$ is labelled by a distinct label a, has formal parameters $\tilde{x}$ and body P. The other constructors are the concurrent composition of processes, the restriction of the scope of a name to a process, the replication of objects, and the terminated process. We abbreviate a method $\mathsf{a} : ()\,\mathbf{0}$ to a, and a process $\nu x_1 \cdots \nu x_n\, P$ to $\nu \tilde{x}\, P$.

We impose one important restriction on processes: the formal parameters $\tilde{x}$ in a method are not allowed to be locations of objects in the body P.

An occurrence of a name x in a process P is *bound* if it is in a part of P with the form $\mathsf{a} : (\tilde{w}x\tilde{y})\, Q$ or $\nu x\, Q$; otherwise the occurrence of x is *free*. The set $fn(P)$ of the *free names* in a process P is defined accordingly, and so is *alpha-conversion*, denoted by $\equiv_\alpha$. The process $P[\tilde{v}/\tilde{x}]$ denotes the simultaneously substitution of the free occurrences of $\tilde{x}$ in P by $\tilde{v}$, defined only when $\tilde{x}$ and $\tilde{v}$ have the same length.

Definition 2. *Structural congruence* is the smallest congruence relation over processes generated by the following rules.

$$P \equiv Q \ \text{ if } P \equiv_\alpha Q \qquad P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$
$$\nu xy\, P \equiv \nu yx\, P \qquad \nu x\, \mathbf{0} \equiv \mathbf{0} \qquad \nu x\, P \mid Q \equiv \nu x\, (P \mid Q) \ \text{ if } x \notin fn(Q)$$

The result $M \bullet m$ of applying a communication m to a collection of methods M is the process $P[\tilde{v}/\tilde{x}]$ if m is of the form $\mathsf{a} : [\tilde{v}]$, and $\mathsf{a} : (\tilde{x})P$ is a method in M, and the substitution is defined.

Definition 3. *One-step reduction* $\rightarrow$ is the smallest relation over processes generated by the following rules.

$$\textsc{Com}\ \ x \rhd M \mid x \lhd m \rightarrow M \bullet m \qquad \textsc{Rep}\ \ !x \rhd M \mid x \lhd m \rightarrow\, !x \rhd M \mid M \bullet m$$
$$\textsc{Par}\ \ P \mid R \rightarrow Q \mid R \ \text{ if } P \rightarrow Q \qquad \textsc{Res}\ \ \nu x\, P \rightarrow \nu x\, Q \ \text{ if } P \rightarrow Q$$
$$\textsc{Str}\quad P' \rightarrow Q' \ \text{ if } P' \equiv P,\ \ P \rightarrow Q,\ \ Q \equiv Q'$$

Reduction $\twoheadrightarrow$ is the relation $\equiv \cup \rightarrow^+$, where $\rightarrow^+$ denotes the transitive closure of $\rightarrow$.

The new notion of process with error requires two further notions. A *context* $\mathbf{C}$ is the concurrent composition of messages and a constant $[]$ (called the *hole*). Filling the hole of a context $\mathbf{C}$ with a process P results in the process $\mathbf{C}[P]$. A process P has a (replicated) *x-redex* if $P \equiv \nu\tilde{x}\,(!x \rhd M \mid x \lhd m \mid Q)$. A process P has a *bad x-redex* if P has an x-redex and $M \bullet m$ is not defined.

Definition 4. A process P is an *error*, notation $P \in \text{Err}$, if

1. $\exists_{\mathbf{C}}\, \mathbf{C}[P] \twoheadrightarrow Q$, and Q has a bad x-redex, for some x not in $\mathbf{C}$, and
2. $\forall_{\mathbf{C}'}\, \mathbf{C}'[Q] \twoheadrightarrow R$, and R has a bad x-redex.

Errors are processes with bad redexes that persist throughout reduction. An occasional bad redex is not enough to make the process an error. So, we give messages a chance to find their target, and therefore, we say that this calculus has weak fairness in the treatment of messages.

Example. 1. $S \stackrel{\text{def}}{=} \text{Empty}(x) \mid x \lhd \mathit{read}\colon[u] \notin \text{Err}$ since, although the process S is a bad x-redex, we have $\mathbf{C}[S] \to \text{Full}(x\ v) \mid x \lhd \mathit{read}\colon[u]$ for $\mathbf{C} \stackrel{\text{def}}{=} x \lhd \mathit{write}\colon[v] \mid []$ containing no bad x-redexes;
2. $P \stackrel{\text{def}}{=} y \rhd [b : x \lhd \text{a}] \mid !x \rhd [c] \in \text{Err}$ since $\mathbf{C}[P] \to x \lhd \text{a} \mid !x \rhd [c]$ with $\mathbf{C} \stackrel{\text{def}}{=} y \lhd b \mid []$ and no context can undo the bad x-redex.

3 The type assignment system

Processes are implicitly typed: although no type information is present in processes, it can be inferred by a type system that assigns type to names and sets of name-type pairs (called typings) to processes.

A type is a *graph* whose nodes (states) can be interpreted as an object's interface and the arcs (transitions) as the invoked methods. The type of an object represents its possible life-cycles.

Definition 5. The set $\mathcal{T}$ of *types* is inductively defined as follows.

1. $\mathcal{D} \subseteq \mathcal{T}$, for $\mathcal{D}$ an initial algebra of some fixed data types;
2. $(V, I, A) \subseteq \mathcal{T}$, where V is a nonempty set of nodes, $I \subseteq V$ is a nonempty set of initial nodes, and $A \subseteq V \times (\mathcal{L} \times \mathcal{T}^*) \times V$ is a set of arcs labelled by $\mathcal{L} \times \mathcal{T}^*$. Graphs are directed and contain no isolated nodes. We further require that a graph with more than one initial node is the disjoint union of connected components, one for each initial node.

We use α, β, γ to denote types. For a given graph α, V_α denotes its set of nodes, I_α denotes its set of initial nodes, and A_α denotes its set of arcs; the label of an arc is denoted by t. Graphs are consider equal up to isomorphism on nodes. The union of graphs is a sum of behaviours. A graph that is the disjoint union of connected components represents a set of possible behaviours of an object, each behaviour represented by a connected subgraph.

Definition 6. The set T_α of *terminal nodes* of a graph α is the set $\{v \in V_\alpha \setminus I_\alpha \mid \nexists_{u\in V_\alpha} (v,t,u) \in A_\alpha, u \neq v\} \cup \{v \in V_\alpha \setminus I_\alpha \mid \exists_{u \in I_\alpha} (v,t,u) \in A_\alpha\}$.

Definition 7. The *union* $\alpha \uplus \beta$ of types α and β is the type γ such that

1. if α, β are graphs, then
$$\gamma \stackrel{\text{def}}{=} \begin{cases} (V_\alpha \cup V_\beta, I_\alpha \cup I_\beta, A_\alpha \cup A_\beta), & \text{if } V_\alpha \cap V_\beta = \emptyset \\ (V_\alpha \cup V_\beta, I_\alpha \setminus (V_\beta \setminus I_\beta) \cup I_\beta \setminus (V_\alpha \setminus I_\alpha), A_\alpha \cup A_\beta), & \text{otherwise} \end{cases}$$
2. if $\alpha, \beta \in \mathcal{D}$ and $\alpha = \beta$, then $\gamma \stackrel{\text{def}}{=} \alpha$;
3. the union is undefined otherwise.

A graph that is the product of two graphs represents the joint behaviour (parallel composition or interleaving) of two objects located at a same name.

Definition 8. The *interleaving* $\alpha \parallel \beta$ of graphs α and β is the graph γ such that

1. $V_\gamma \stackrel{\text{def}}{=} V_\alpha \times V_\beta$, and $I_\gamma \stackrel{\text{def}}{=} I_\alpha \times I_\beta$, and
2. $A_\gamma \stackrel{\text{def}}{=} \{(uv,t,u'v) \mid \forall_{(u,t,u')\in A_\alpha} \exists_{v\in V_\beta}\} \cup \{(uv,t,uv') \mid \forall_{(v,t,v')\in A_\beta} \exists_{u \in V_\alpha}\}$.

Types abstract from objects' concrete behaviour: two different objects with equivalent behaviours have the same type. The equivalence relation is a pair of binary relations over types, one over the nodes of the graph and the other over the types labelling the graph's arcs.

Definition 9. *Bisimilarity on types.*

1. A symmetric binary relation $\mathcal{R} \subseteq V_\alpha \times V_\beta$ is a *bisimulation on graphs* (over a binary relation $\mathcal{C}$ on types) if $\forall_{u\in V_\alpha} \forall_{v \in V_\beta}$ such that $u\mathcal{R}v$, if $(u, \mathrm{a} : \tilde{\alpha}, u') \in A_\alpha$ then $\exists_{(v,\mathrm{a}:\tilde{\beta},v')\in A_\beta}$ with $u'\mathcal{R}v'$ and $\tilde{\alpha}\mathcal{C}\tilde{\beta}$[2].
2. Two nodes $u \in V_\alpha$, $v \in V_\beta$ are *bisimilar over* $\mathcal{C}$, denoted by $u \sim_\mathcal{C} v$, if there is a bisimulation $\mathcal{R}$ over $\mathcal{C}$ such that $u\mathcal{R}v$.

Compatibility of types.

1. A symmetric binary relation $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$ is a *type compatibility*, if $\alpha\mathcal{C}\beta$ implies $\forall_{u\in I_\alpha} \exists_{v \in I_\beta}\, u \sim_\mathcal{C} v$[3] when α, β are graphs, or $\alpha = \beta$ otherwise;
2. Two types α and β are *compatible*, denoted by $\alpha \sim \beta$, if there exists a type compatibility relation $\mathcal{C}$ such that $\alpha\mathcal{C}\beta$.

The compatibility relation $\sim$ is the largest type compatibility; thus, two graph types are compatible if all their nodes are bisimilar over the compatibility relation. One can easily observe that $\mathcal{T}/\sim$ specifies a class of process *behaviours*.

To characterize how graphs evolve with the reduction of processes we need the notion of subgraphs.

Definition 10. 1. A *path* μ_α in a graph α is a chain of arcs in α of the form $(u_0, t_1, u_1), \ldots, (u_{n-1}, t_n, u_n)$, with $n \geq 1$. We write $\mu_\alpha^{u,v}$ to denote the path starting at node u and ending at node v.

---

[2] Let $\alpha_1 \cdots \alpha_k \mathcal{C} \beta_1 \cdots \beta_k \stackrel{\text{def}}{=} \alpha_1\mathcal{C}\beta_1 \wedge \cdots \wedge \alpha_k\mathcal{C}\beta_k$.

[3] This condition is enough to guaranty the bisimilarity of the graphs, since graphs do not have unreachable nodes.

2. A path $\mu_\alpha^{u,v}$ is *complete* if $u \in I_\alpha$, and $v \in T_\alpha$ or $v \in I_\alpha$ if $\exists_{w \in T_\alpha} (w, t, v) \in A_\alpha$.
3. For a path μ_α, a sequence of some of its arcs preserving the original ordering is called a *projection* of μ_α.
4. A graph α is a *subgraph* of a graph β, denoted by $\alpha \leq \beta$, if $\alpha = \beta$ or $\exists_{u \in I_\beta} (u, t, v) \in A_\beta$ with $v \in I_\alpha$ and each path of β starting in v is also a path of α.

Lemma 11. *If α, β, γ are graphs and $\alpha \leq \beta$, then $(\alpha \parallel \gamma) \leq (\beta \parallel \gamma)$.*

Proof. Directly from the definitions of $\parallel$ and $\leq$. $\square$

Types for replicated objects are obtained from a finite graph by means of a fix point operation.

Prop/Definition 12 . 1. $\alpha_0 \stackrel{\text{def}}{=} (V_\alpha \setminus T_\alpha, I_\alpha, A_{\alpha_0})$, where
$A_{\alpha_0} \stackrel{\text{def}}{=} \{(u, t, v) \mid (u, t, w) \in A_\alpha, \text{ and } v = w \text{ if } w \notin T_\alpha \text{ or } v = u \text{ otherwise}\}$.

2. $\mathcal{F}(\alpha_0) = \biguplus_{v \in V_{\alpha_0} \setminus I_{\alpha_0}} \alpha_0 \sigma_v \uplus \alpha_0$, and $\mathcal{F}(\alpha_{i+1}) = \biguplus_{v \in V_{\alpha_i} \setminus V_{\alpha_{i-1}}} \alpha_0 \sigma_v \uplus \alpha_i$ with $i \geq 1$, where σ_v is the substitution
$$\sigma_v \stackrel{\text{def}}{=} \begin{cases} I_{\alpha_0} \mapsto \{v\} \\ V_\alpha \setminus I_\alpha \mapsto \{w \mid \text{for each } u \in V_\alpha \setminus I_\alpha,\ w \text{ is fresh}\}. \end{cases}$$
3. The replication $repl(\alpha)$ of a finite graph α is the graph $fix(\mathcal{F}(\alpha_0))$.
4. One can easily see that $\mathcal{F}$ is a continuous function since it is increasing by definition, and it is monotonous since if $\alpha_0 \subseteq \beta_0$ then $\mathcal{F}(\alpha_0) \subseteq \biguplus_{v \in V_{\alpha_0} \setminus I_{\alpha_0}} \alpha_0 \sigma_v \uplus \beta_0 \subseteq \mathcal{F}(\beta_0)$ and, similarly, $\mathcal{F}(\alpha_i) \subseteq \mathcal{F}(\beta_i)$ for some $i \geq 1$.

The *name-usage-type* triple $x^* : \alpha$, with $* \in \{!, \downarrow, \uparrow\}$, is a formula denoting the assignment of type α to name x, location of an object ($\downarrow$), location of a replicated object (!), or destination of a message ($\uparrow$). A *typing* Γ is a finite set of name-usage-type triples that has at most two occurrences of the same name, one as an object (replicated or not) and a second as a message.

Example. 1. The message $x \lhd \mathrm{a} : [\tilde{u}]$ has a typing $\{x^\uparrow : (\{u_0, u_1\}, \{u_0\}, \{(u_0, \mathrm{a} : \tilde{\alpha}, u_1)\}), \tilde{u} : \tilde{\alpha}\}$. Graphically $\{x^\uparrow : \begin{tikzcd}[row sep=small] \odot \arrow[d, "a:\tilde{\alpha}"] \\ \bullet \end{tikzcd}, \tilde{u} : \tilde{\alpha}\}$, where $\odot$ denotes an initial node. The objects $x \rhd [\mathrm{a} : x \rhd [b]]$ and $x \rhd [\mathrm{a} + b]$ have respectively typings $\{x^\downarrow : \begin{tikzcd}[row sep=small] \odot \arrow[d, "a"] \\ \bullet \arrow[d, "b"'] \\ \bullet \end{tikzcd}\}$, and $\{x^\downarrow : \begin{tikzcd}[column sep=small] & \odot \arrow[dl, "a"'] \arrow[dr, "b"] & \\ \bullet & & \bullet \end{tikzcd}\}$.

2. For $x \rhd [\mathrm{a}], x \rhd [b]$, for $x \rhd [\mathrm{a} : x \rhd [b] + b : x \rhd [\mathrm{a}]]$, and for $!x \rhd [\mathrm{a} : x \rhd [b]]$, types $\begin{tikzcd}[column sep=small] & \odot \arrow[dl, "a"'] \arrow[dr, "b"] & \\ \bullet \arrow[dr, "b"'] & & \bullet \arrow[dl, "a"] \\ & \bullet & \end{tikzcd}$, $\begin{tikzcd}[column sep=small] & \odot \arrow[dl, "a"'] \arrow[dr, "b"] & \\ \bullet \arrow[d, "b"'] & & \bullet \arrow[d, "a"] \\ \bullet & & \bullet \end{tikzcd}$, and $\begin{tikzcd}[row sep=small] \odot \arrow[d, bend right, "b"'] \arrow[d, bend left, "a"] \\ \bullet \arrow[d, bend right, "b"'] \arrow[d, bend left, "a"] \\ \vdots \end{tikzcd}$ are assigned to $x^\downarrow$, $x^\downarrow$, and $x^!$, respectively.

Let $dom(\Gamma)$ be the set of the name-usage pairs in each triple of Γ; then $\Gamma \cdot x^* : \alpha$ denotes the union of Γ and $\{x^* : \alpha\}$, provided $x^* \notin dom(\Gamma)$. For $x^* : \alpha \in \Gamma$ let $\Gamma(x^*) \stackrel{\text{def}}{=} \alpha$; let $\Gamma \backslash x^*$ denote the typing Γ without the occurrences of formulas with x^*. Let $\Gamma \upharpoonright P$ denote the restriction of Γ to the free names in P, and let $\Gamma[z/x]$ denote the result of replacing in Γ occurrences of x by the fresh name z.

Definition 13. Two typings Γ and Δ are *compatible*, denoted by $\Gamma \asymp \Delta$, if $\Gamma(x^!)$ has a projection equal to some complete path of $\Delta(x^\uparrow)$.

Definition 14. The *union* $\Gamma \uplus \Delta$ of two compatible typings is the typing:

1. $\Gamma \cup \Delta$, if $dom(\Gamma) \cap dom(\Delta) = \emptyset$;
2. $(\Gamma \cdot x^* : \gamma) \uplus (\Delta \backslash x^*)$, if $\Gamma(x^*) = \alpha$, $\Delta(x^*) = \beta$, and $\gamma \stackrel{\text{def}}{=} \begin{cases} \alpha, & \text{if } \alpha \sim \beta \\ \alpha \uplus \beta, & \text{otherwise;} \end{cases}$
3. $\{x^* : \alpha, x^\uparrow : \beta\} \cup ((\Gamma \backslash x^*) \uplus (\Delta \backslash x^\uparrow))$, if $\Gamma(x^*) = \alpha$, $\Delta(x^\uparrow) = \beta$, and $* \in \{!, \downarrow\}$;
4. $\{x^! : \alpha\} \cup ((\Gamma \backslash x^!) \uplus (\Delta \backslash x^\downarrow))$, if $\Gamma(x^!) = \alpha$, $\Delta(x^\downarrow) = \beta$, and $\alpha = repl(\gamma)$, for some γ such that $\gamma \sim \beta$;
5. $\{x^! : \alpha, x^\downarrow : \beta\} \cup ((\Gamma \backslash x^!) \uplus (\Delta \backslash x^\downarrow))$, if $\Gamma(x^!) = \alpha$ and $\Delta(x^\downarrow) = \beta$.

Definition 15. The *interleaving* $\Gamma \parallel \Delta$ of two compatible typings is the typing:

1. $\Gamma \cup \Delta$, if $dom(\Gamma) \cap dom(\Delta) = \emptyset$;
2. $(\Gamma \cdot x^* : \gamma) \parallel (\Delta \backslash x^*)$, if $\Gamma(x^*) = \alpha$, $\Delta(x^*) = \beta$, and $\gamma \stackrel{\text{def}}{=} \begin{cases} \alpha \parallel \alpha, & \text{if } \alpha \sim \beta \\ \alpha \parallel \beta, & \text{otherwise;} \end{cases}$;
3. $\{x^* : \alpha, x^\uparrow : \beta\} \cup ((\Gamma \backslash x^*) \parallel (\Delta \backslash x^\uparrow))$, if $\Gamma(x^*) = \alpha$, $\Delta(x^\uparrow) = \beta$, and $* \in \{!, \downarrow\}$;
4. $\{x^! : \alpha\} \cup ((\Gamma \backslash x^!) \parallel (\Delta \backslash x^\downarrow))$, if $\Gamma(x^!) = \alpha$, $\Delta(x^\downarrow) = \beta$, and $\alpha = repl(\gamma)$, for some γ such that $\gamma \sim \beta$;
5. $\{x^! : \alpha, x^\downarrow : \beta\} \cup ((\Gamma \backslash x^!) \parallel (\Delta \backslash x^\downarrow))$, if $\Gamma(x^!) = \alpha$ and $\Delta(x^\downarrow) = \beta$.

In the two definitions above, the rules should always be tried in the order presented.

Definition 16. Let $\Delta \leq \Gamma$ if $dom(\Delta) \subseteq dom(\Gamma)$ and $\Delta(x^*) \leq \Gamma(x^*)$ for all $x^* \in dom(\Delta) \cap dom(\Gamma)$.

Lemma 17. *If $\Delta \leq \Gamma$ and $\Gamma \asymp \Lambda$, then $(\Delta \parallel \Lambda) \leq (\Gamma \parallel \Lambda)$.*

Proof. Follows from the definitions of $\parallel$ and $\leq$, and from lemma 11. □

Definition 18 Behavioural type system. The type assignment system is inductively defined by the following axioms and rules.

$$\text{MSG} \quad \{\tilde{u}^* : \tilde{\alpha}, x^\uparrow : (\{v_0, v_1\}, \{v_0\}, \{(v_0, \mathrm{a} : \tilde{\alpha}, v_1)\})\} \vdash x \lhd \mathrm{a} : [\tilde{u}] \quad (v_0 \neq v_1)$$

$$\text{OBJ} \ \frac{\Gamma_i \cdot \tilde{x}_i^! : \tilde{\alpha}_i \vdash P_i}{\biguplus_{i \in I} \Gamma_i \uplus \{x^\downarrow : \alpha\} \vdash \ x \rhd \sum_{i \in I} \mathrm{a}_i : (\tilde{x}_i)\, P_i} \ (1) \qquad \text{NIL} \ \ \emptyset \vdash \mathbf{0}$$

$$\text{REP} \ \frac{\Gamma \cdot x^\downarrow : \alpha \vdash x \rhd M}{\Gamma \cdot x^! : repl(\alpha) \vdash !x \rhd M} \qquad \text{RES} \ \frac{\Gamma \vdash P}{\Gamma \backslash x \vdash \nu x\, P}$$

$$\text{PAR}\ \frac{\Gamma \vdash P \quad \Delta \vdash Q}{\Gamma \parallel \Delta \vdash P, Q}\ (\Gamma \asymp \Delta) \qquad \text{WEAK}\ \frac{\Gamma \vdash P}{\Gamma \cdot \{x^{\uparrow} : \alpha\} \vdash P}$$

where, in rule OBJ,

(1) $(\asymp_{i \in I} \Gamma_i) \asymp \{x^{\downarrow} : \alpha\}$, and α is such that $I_\alpha \stackrel{\text{def}}{=} \{u\}$ for u a fresh node, and

$$A_\alpha = \bigcup_{i \in \mathrm{I}} \{(u, a_i : \tilde{\alpha}_i, w) \mid x^* \notin dom(\Gamma_i) \text{ and } w \text{ is fresh}\} \cup$$
$$\bigcup_{i \in \mathrm{I}} \{(u, a_i : \tilde{\alpha}_i, v) \mid x^* \in dom(\Gamma_i) \text{ and for each } v \in I_{\Gamma_i(x^*)}\} \cup \bigcup_{i \in \mathrm{I}} A_{\Gamma_i(x^*)},$$

for $* \in \{!, \downarrow\}$.

We say a process P is well typed if $\exists_\Gamma\, \Gamma \vdash P$. Important properties of the above system include typability of subterms, the substitution lemma (if $\Gamma \vdash P$ then $\Gamma[z/x] \vdash P[z/x]$) and the congruence lemma (if $\Gamma \vdash P$ and $P \equiv Q$, then $\Gamma \vdash Q$).

Theorem 19 Subject-reduction. *If $\Gamma \vdash P$ and $P \twoheadrightarrow Q$, then $\exists_\Delta\, \Delta \vdash Q$ and $\Delta \leq \Gamma$.*

Proof. By induction on $\twoheadrightarrow$. The non-trivial cases are when reduction ends with the PAR-rule and the COM-axiom (REP is similar to COM).

If reduction ends with the PAR-rule let $P \equiv P' \mid R$ and $Q \equiv Q' \mid R$; by typability of subterms $\exists_{\Gamma', \Lambda}\, \Gamma' \vdash P'$ and $\Lambda \vdash R$ with $\Gamma = \Gamma' \parallel \Lambda$, and by induction hypothesis $\exists_{\Delta'}\, \Delta' \vdash Q'$ and $\Delta' \leq \Gamma'$. The result follows by lemma 17.

If reduction ends with the COM-axiom let $P \equiv x \triangleright M \mid x \triangleleft m$. By the PAR-rule and by typability of subterms $\Gamma = \Gamma' \parallel \Delta'$ with $x^{\downarrow} : \alpha \in \Gamma'$ and $x^{\uparrow} : \beta \in \Delta'$; if $M \bullet m$ is undefined then $\Delta = \Gamma$ else by the PAR-rule and by the substitution lemma $\exists_\Delta\, \Delta \vdash M \bullet m$, and if $x^{\downarrow} \in dom(\Delta)$ then $\Delta(x^{\downarrow}) \leq \alpha$, and if $x^{\uparrow} \in dom(\Delta)$ then $\Delta(x^{\uparrow}) \leq \beta$; it follows that $\Delta \leq \Gamma$. □

Corollary 20. *If P is well-typed then $P \notin$* ERR.

Proof. Suppose P is well-typed and $P \in$ ERR. By definition of ERR, P has $!x \triangleright M$ and $x \triangleleft m$ as subterms, both typable by typability of subterms, with compatible types. Therefore, $M \bullet m$ is defined, and then it is not a persistent bad x-redex; we have reached an absurd, since, by hypothesis, $P \in$ ERR. □

The system enjoys the property of *uniqueness* of the types assigned to the free names in a process.

Proposition 21. *If $\Gamma \vdash P$ and $\Delta \vdash P$ then $\Gamma \upharpoonright P = \Delta \upharpoonright P$.*

Proof. By a case analysis of the rules defining the type system, noting that each rule defines one and only one typing for a process, up to renaming of nodes. □

4 Discussion

The present type system types all processes the previous system [VT93] does, except for those that do not conform to the restriction in section 2. The buffer-cell in section 1 constitutes an example of a process this system types the previous not. Nevertheless some "basic" mistakes (like typing q instead of w in process $x \triangleright [w] \mid x \triangleleft q)$, are no longer detected as error-processes.

The starting point for this work are the ideas of Nierstrasz on regular types for active objects. Puntigam also starts from Nierstrasz work, and uses terms of a process algebra (without name-passing) as types [Pun96]. His work is centered on subtyping, and not on type assignment systems. There is now a lot of work on types for mobile processes but, up to our knowledge, the only work in the context of mobile processes where types are graph seems to be Yoshida's [Yos96]. Her graphs give information about the deterministic behaviour of a process; our graphs are inspired on Milner's derivation trees [Mil89].

References

[Agh86] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* M.I.T. Press, 1986.

[HT91] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *ECOOP'91*, pages 141–162. Springer-Verlag LNCS 512, 1991.

[KY95] N. Kobayashi and A. Yonezawa. Towards foundations of concurrent object-oriented programming - types and language design. *Theory and Practice of Object Systems*, 1(4), 1995.

[LW95] X. Liu and D. Walker. A polymorphic type system for the polyadic π-calculus. In *Concur'95*, pages 103–116. Springer-Verlag LNCS 962, 1995.

[Mil89] R. Milner. *Communication and Concurrency.* C. A. R. Hoare Series Editor – Prentice-Hall Int., 1989.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, i e ii. *Information and Computation*, 100:1–77, 1992.

[Nie95] O. Nierstrasz. Regular types for active objects. In O. Nierstrasz and D. Tsichritzis, editors, *Object-Oriented Software Composition*, pages 99–121. Prentice Hall, 1995.

[Pun96] F. Puntigam. Types for a active objects based on trace semantics. In *FMOODS'96*, 1996.

[RV97] A. Ravara and V. Vasconcelos. Behavioural types for a calculus of concurrent objects. Technical report DM-IST 6/97, Department of Mathematics, Instituto Superior Técnico, 1096 Lisboa, Portugal, 1997. Available from ftp://ftp.cs.math.ist.utl.pt/pub/RavaraA/97-R-BEVTYP.ps.gz.

[VH93] V. Vasconcelos and K. Honda. Principal typing-schemes in a polyadic π-calculus. In *Concur'93*, pages 524–538. Springer-Verlag LNCS 715, 1993.

[VT93] V. Vasconcelos and M. Tokoro. A typing system for a calculus of objects. In *1st ISOTAS*, pages 460–474. Springer-Verlag LNCS 742, 1993.

[Yos96] N. Yoshida. Graph types for monadic mobile processes. In *16th FST/TCS*, pages 371–386. Springer-Verlag LNCS 1180, 1996.