Towards an Algebra of Dynamic Object Types

António Ravara * Pedro Resende * Vasco T. Vasconcelos †

Abstract

We propose an algebra of object types that characterises the semantics of concurrent objects in a process calculus setting where the communication is asynchronous. The types are non-uniform, and provide an internal (and synchronous) view of the objects that inhabit them. These ideas, along with the algebraic laws, are based on a notion of bisimulation that is unlike other notions in the literature.

1 Introduction

Non-uniform types for concurrent objects constitute the object of study of several authors [7, 2, 3, 6, 8]. The aim is to build type systems capable of ensuring more than the usual safety properties (such as subject-reduction); for instance, the absence of some deadlocks. These types reflect a dependency of the interface of an object upon its internal state, conveying information about some dynamic properties of objects.

In a process calculus setting such as TyCO [10], processes denote the behaviour of a community of interacting objects, where each object has a location identified by a name. Each process determines an assignment of types to names reflecting a discipline for communication. The usual types-as-records paradigm [11] gives each name a static type that contains information about all the methods of the object, regardless of whether they are enabled or not. Hereby we propose an algebra of object types, where each type is essentially a collection of enabled methods, and it is dynamic in the sense that the execution of a method can change this collection, i.e. the type. Therefore, the type of an object can also be seen as a partial representation of its behaviour.

We assume that objects communicate via asynchronous message-passing; nevertheless, types, as defined in this paper, essentially correspond to a notion of object behaviour as it would be perceived by an internal observer located within an object (the object's private gnome). This observer can see methods being invoked and it can detect whether the object is blocked, even though its methods may be internally enabled. Hence, this notion of behaviour is synchronous, as essentially the gnome can detect refusals of methods. The action of unblocking an object, denoted by υ, cannot be observed by the gnome because it corresponds to an invocation of some method in another object. Thus, this action is similar to Milner's τ [4], because it is hidden, but it is external, rather than an internal action.

In this paper we define a structural operational semantics for the algebra of object types, both for finite and infinite types. We also introduce two behavioural equivalences based on notions of bisimulation that characterise the referred aspects of an object, and show that the two coincide, at least in the case of finite types.

*Computer Science Section, Department of Mathematics, Instituto Superior Técnico. Lisbon, Portugal. Email: {amar,pmr}@math.ist.utl.pt

†Department of Computer Science, Faculty of Sciences, University of Lisbon, Portugal. Email: vv@di.fc.ul.pt

2 Algebra of Finite Object Types

We start by presenting the algebra of finite object types. Objects are records of methods, and we can represent unavailable (blocked) objects.

Assume a countable set of *method names* l, m, n, possibly subscripted or primed.

Definition 2.1. The set $\mathcal{O}$ of *finite types of objects* is given by the following grammar.

$$\alpha \quad ::= \quad \mathbf{0} \quad | \quad \sum_{i \in I} l_i(\widetilde{\alpha}_i).\alpha_i \quad | \quad \biguplus_{j \in J} \alpha_j$$

where I and J are non-empty finite indexing sets, with $\forall_{i,j \in I}\ i \neq j \Rightarrow l_i \neq l_j$, and, $\widetilde{\alpha}$ is a finite sequence of types.

We call a term such as $l(\widetilde{\alpha}).\alpha$ a *method type*. It corresponds to a method with name l and parameters of type $\widetilde{\alpha}$, which behaves as prescribed by α. The type composition operator sum ("$\sum$") puts various method types together to form a type of an object that offers the corresponding methods. The term $\mathbf{0}$ is the empty type. The disjoint union $\biguplus_{j \in J} \alpha_j$ is the type of a blocked object that will behave later according to one of the types α_j, after being released.

Notation 2.2.
1. We abbreviate the type $l(\widetilde{\alpha}).\mathbf{0}$ to $l(\widetilde{\alpha})$, and $l()$ to l.
2. We assume the following abbreviations:
   (a) $l(\widetilde{\alpha}).\alpha$ is $\sum_{i \in \{1\}} l_i(\widetilde{\alpha}_i).\alpha_i$;
   (b) $\uplus\alpha$ is $\biguplus_{i \in \{1\}} \alpha_i$;
   (c) $l_1(\widetilde{\alpha}_1).\alpha_1 + l_2(\widetilde{\alpha}_2).\alpha_2$ is $\sum_{i \in \{1,2\}} l_i(\widetilde{\alpha}_i).\alpha_i$;
   (d) $\alpha_1 \uplus \alpha_2$ is $\biguplus_{i \in \{1,2\}} \alpha_i$.

We define a structural operational semantics for the finite types of objects via a labelled transition relation.

Definition 2.3. The set of *labels* is given by the following grammar.

$$\pi \quad ::= \quad \upsilon \quad | \quad l(\widetilde{\alpha})$$

The label υ denotes a silent transition that releasees a blocked object; a label $l(\widetilde{\alpha})$ denotes a method invocation.

Definition 2.4. The *labelled transition relation* is inductively defined by the following rules.

$$\textsc{Act} \quad \sum_{i \in I} l_i(\widetilde{\alpha}_i).\alpha_i \xrightarrow{l_j(\widetilde{\alpha}_j)} \alpha_j \quad (j \in I) \qquad\qquad \textsc{Union} \quad \biguplus_{i \in I} \alpha_i \xrightarrow{\upsilon} \alpha_j \ (j \in I)$$

The axiom Act gives the basic transition: the invocation of a method with name l and parameters of type $\widetilde{\alpha}$ results in the type of the body of that method. The axiom Union captures a side effect: $\biguplus_{i \in I} \alpha_i$ is blocked, and one of its behaviours can only become available after an unblocking action occurs.

Notation 2.5. Let $\Longrightarrow$ denote $\xrightarrow{\upsilon}{}^*$, and $\overset{\upsilon}{\Longrightarrow}$ denote $\xrightarrow{\upsilon}{}^+$; furthermore, we write $\alpha \overset{\pi}{\not\longrightarrow}$ when $\alpha \xrightarrow{\pi}$ does not hold.

We want two object types to be equivalent if they have equivalent method types and if after each transition they continue to be equivalent, in a bisimulation style. Furthermore, from the point of view of each object type, transitions of other object types can be regarded as hidden transitions, which would suggest weak bisimulation as the right notion of equivalence for our object types, with υ playing the role of Milner's τ. However, we want our types to distinguish an object that immediately makes available a method from one that makes it available only after being unblocked by another object. This is because, although υ is supposed to be unobservable, we assume that from the point of view of an internal observer (the object's gnome) it is detectable that the object is blocked. Hence, we would expect $\uplus l$ to be different from l, but $\uplus l$ and $\uplus \uplus l$ to be equivalent; in both, all the internal observer can see is that the object is blocked, and after being released it can eventually execute the method l. We also want to distinguish $l. \uplus m$ from $l.m$, on the grounds that for the latter a blocking after l cannot be observed. This also discards Milner's observational congruence [4] and rooted bisimulation [1] as possible candidates for object equivalence. A notion such as progressing bisimulation [5] is however too strong because it would distinguish $\uplus\alpha$ from $\uplus \uplus \alpha$.

These considerations lead to the choice of a notion of equivalence that essentially strengthens weak bisimulation by requiring that if α and β are bisimilar and α offers a particular method then also β offers that method.

Definition 2.6. *Bisimilarity on object types.*

1. A symmetric binary relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{O}$ is an *object bisimulation* if whenever $\alpha \mathcal{R} \beta$ then
   (a) $\alpha \xrightarrow{l(\widetilde{\alpha})} \alpha'$ implies $\exists \beta', \widetilde{\beta}, \gamma$ ($\beta \xrightarrow{l(\widetilde{\beta})} \gamma \Longrightarrow \beta'$ and $\alpha' \mathcal{R} \beta'$ and $\widetilde{\alpha} \mathcal{R} \widetilde{\beta}$)[1];
   (b) $\alpha \xrightarrow{\upsilon} \alpha'$ implies $\exists \beta'$ ($\beta \Longrightarrow \beta'$ and $\alpha' \mathcal{R} \beta'$);
2. Two types α and β are *object-bisimilar*, or simply *bisimilar*, and we write $\alpha \approx_o \beta$, if there is an object bisimulation $\mathcal{R}$ such that $\alpha \mathcal{R} \beta$.

The usual properties of bisimilarities hold, namely $\approx_o$ is an equivalence relation, and $\alpha \approx_o \beta$ holds if and only if

1. $\alpha \xrightarrow{l(\widetilde{\alpha})} \alpha'$ implies $\exists \beta', \widetilde{\beta}, \gamma$ ($\beta \xrightarrow{l(\widetilde{\beta})} \gamma \Longrightarrow \beta'$ and $\alpha' \approx_o \beta'$ and $\widetilde{\alpha} \approx_o \widetilde{\beta}$);
2. $\alpha \xrightarrow{\upsilon} \alpha'$ implies $\exists \beta'$ ($\beta \Longrightarrow \beta'$ and $\alpha' \approx_o \beta'$);

We can strengthen this notion of equivalence even more by dropping another double arrow, as follows.

Definition 2.7. *Strict bisimilarity on object types.*

1. A symmetric binary relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{O}$ is a *strict object bisimulation* if whenever $\alpha \mathcal{R} \beta$ then
   (a) $\alpha \xrightarrow{l(\widetilde{\alpha})} \alpha'$ implies $\exists \beta', \widetilde{\beta}$ ($\beta \xrightarrow{l(\widetilde{\beta})} \beta'$ and $\alpha' \mathcal{R} \beta'$ and $\widetilde{\alpha} \mathcal{R} \widetilde{\beta}$);
   (b) $\alpha \xrightarrow{\upsilon} \alpha'$ implies $\exists \beta'$ ($\beta \Longrightarrow \beta'$ and $\alpha' \mathcal{R} \beta'$);
2. Two types α and β are *strictly bisimilar*, and we write $\alpha \approx_s \beta$, if there is a strict object bisimulation $\mathcal{R}$ such that $\alpha \mathcal{R} \beta$.

Again, $\approx_s$ is an equivalence relation and $\alpha \approx_s \beta$ holds if and only if conditions 1(a) and 1(b) of the above definition hold with $\mathcal{R}$ replaced by $\approx_s$. Although for an arbitrary labelled transition system the two bisimilarity relations do not coincide, on our particular system they do, as the following result shows. This provides a sense in which our definition of object type equivalence is robust.

[1] Let $(\alpha_1 \cdots \alpha_k) \mathcal{R} (\beta_1 \cdots \beta_k) \stackrel{\text{def}}{=} \alpha_1 \mathcal{R} \beta_1 \wedge \cdots \wedge \alpha_k \mathcal{R} \beta_k$.

THEOREM 2.8. Let $\alpha, \beta \in \mathcal{O}$. Then $\alpha \approx_o \beta$ if and only if $\alpha \approx_s \beta$.

Proof. The right to left implication is immediate. For the other implication we first remark that our labelled transition system satisfies the following conditions:

1. if $\alpha \xrightarrow{l(\widetilde{\alpha}_1)} \alpha_1$ and $\alpha \xrightarrow{l(\widetilde{\alpha}_2)} \alpha_2$ then $\alpha_1 = \alpha_2$ (label determinism);
2. no α can have both an v and an l transition, i.e., for all α either $\alpha \not\xrightarrow{v}$ or $\alpha \not\xrightarrow{l(\widetilde{\alpha})}$.

Let $\alpha \approx_o \beta$. We will show that

1. $\alpha \xrightarrow{l(\widetilde{\alpha})} \alpha'$ implies $\exists \beta', \widetilde{\beta}$ ($\beta \xrightarrow{l(\widetilde{\beta})} \beta'$ and $\alpha' \approx_o \beta'$ and $\widetilde{\alpha} \approx_o \widetilde{\beta}$),
2. $\alpha \xrightarrow{v} \alpha'$ implies $\exists \beta'$ ($\beta \Longrightarrow \beta'$ and $\alpha' \approx_o \beta'$);

that is, we will prove that $\approx_o$ is a strict object bisimulation, which will conclude the proof. The second condition is trivial, so let $\alpha \xrightarrow{l(\widetilde{\alpha})} \alpha'$. There exist $\widetilde{\beta}, \beta', \beta''$ such that $\beta \xrightarrow{l(\widetilde{\beta})} \beta' \Longrightarrow \beta''$, with $\widetilde{\alpha} \approx_o \widetilde{\beta}$ and $\alpha' \approx_o \beta''$. The condition $\beta \xrightarrow{l(\widetilde{\beta})} \beta'$ in turn implies, together with label determinism, that $\alpha' \Longrightarrow \alpha''$ with $\alpha'' \approx_o \beta'$, for some α''. All we have to do now is prove that $\alpha' \approx_o \beta'$. If $\alpha' = \alpha''$ or $\beta' = \beta''$ this is immediate. Otherwise we have the following situation:

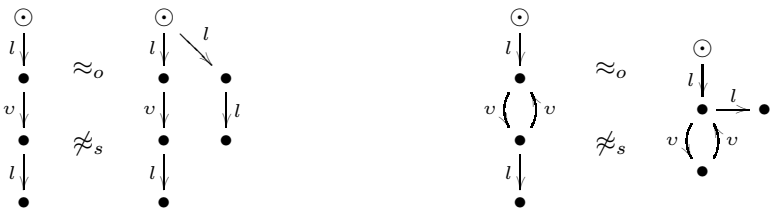
$$\begin{array}{ccc} \alpha' & \approx_o & \beta'' \\ {\scriptstyle v}\Downarrow & & \Uparrow{\scriptstyle v} \\ \alpha'' & \approx_o & \beta' \end{array}$$

In this case all the transitions from α' must be labelled with v; let then $\alpha' \xrightarrow{v} \alpha'''$; it follows that $\beta'' \Longrightarrow \beta'''$ for some β''' such that $\alpha''' \approx_o \beta'''$, hence also $\beta' \Longrightarrow \beta'''$. Similarly, β' can only do v, and if $\beta' \xrightarrow{v} \beta'''$ we can find α''' such that $\alpha' \Longrightarrow \alpha'''$ and $\alpha''' \approx_o \beta'''$, which concludes the proof. □

Therefore, in our type algebra the two equivalences coincide; henceforth we refer to both $\approx_o$ and $\approx_s$ as *object equivalence*, and write $\approx_o$.

The proof of the theorem relies on the fact that our transition system is deterministic on labels, and no state can have a transition labelled with l and another with v. If either of these conditions is violated the theorem no longer holds, as the following examples show:

EXAMPLE 2.9.



Naturally, the counterexamples above are not expressible in our language.

PROPOSITION 2.10. Object equivalence is a congruence relation.

Proof. Straightforward, since the sum is guarded. □

PROPOSITION 2.11 (ALGEBRAIC LAWS). 1. The operators "+" and "$\uplus$" are commutative; that is, for any permutation $\sigma : I \to I$ we have $\sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i \approx_o \sum_{i\in I} l_{\sigma(i)}(\widetilde{\alpha}_{\sigma(i)}).\alpha_{\sigma(i)}$, and $\uplus_{i\in I}\alpha_i \approx_o \uplus_{i\in I}\alpha_{\sigma(i)}$;

2. the operator "$\uplus$" enjoys a weak form of associativity, $\uplus(\uplus_{i\in I}\alpha_i) \approx_o \uplus_{i\in I}\alpha_i$;

3. if $\alpha_1 \uplus \cdots \uplus \alpha_n \xrightarrow{\upsilon}{}^{+} \beta$ then $\beta \uplus \alpha_1 \uplus \cdots \uplus \alpha_n \approx_o \alpha_1 \uplus \cdots \uplus \alpha_n$.

Proof. The respective bisimulations are straightforward. □

Notice that law 3 is not really algebraic, but rather it gives us an infinity of laws that express a form of idempotence. For instance, we have

$$\begin{array}{rcl} \alpha_1 \uplus (\alpha_1 \uplus \cdots \uplus \alpha_n) & \approx_o & \alpha_1 \uplus \cdots \uplus \alpha_n \\ \alpha_1 \uplus ((\alpha_1 \uplus \cdots \uplus \alpha_n) \uplus \beta_1 \uplus \cdots \uplus \beta_m) & \approx_o & (\alpha_1 \uplus \cdots \uplus \alpha_n) \uplus \beta_1 \uplus \cdots \uplus \beta_m \\ & \vdots & \end{array}$$

A more thorough discussion of the algebraic laws will appear in the full version of this report.

REMARK 2.12. One can easily recognise that what corresponds to the law $\tau.\tau.\alpha = \tau.\alpha$ of process algebra, namely $\uplus\uplus\alpha \approx_o \uplus\alpha$, holds in this setting, since it is an instance of the weak associativity law presented above. Notice however that other laws like $\tau.\alpha = \alpha$ and $a.\tau.\alpha = a.\alpha$ do not hold. Also, $\uplus$ is not associative; e.g., $l\uplus(m\uplus n) \not\approx_o l\uplus m\uplus n$, which means that although υ is unobservable, sometimes it can be indirectly counted.

3 The Algebra of Object Types

Now we briefly discuss how the algebra of object types can be extended to deal with multiple objects located at the same name (with a parallel composition operator) and infinite types (with recursion).

DEFINITION 3.1. Assume a countable set of variables, denoted by t, disjoint from the set of labels. The set $\mathcal{O}$ of *types of objects* is defined by the following grammar.

$$\alpha \quad ::= \quad \mathbf{0} \quad | \quad \sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i \quad | \quad \biguplus_{j\in J} \alpha_j \quad | \quad \alpha \parallel \alpha \quad | \quad t \quad | \quad \mu t.\alpha$$

where I and J are non-empty finite index sets, with $\forall_{i,j\in I}\ i \neq j \Rightarrow l_i \neq l_j$, and $\widetilde{\alpha}$ is a finite sequence of types.

The parallel composition ("$\parallel$") of types denotes the existence of several objects located at the same name in parallel (interpreted as different copies of the same object, possibly several in different states).

DEFINITION 3.2. The *labelled transition relation* is defined by the rules of Definition 2.4 together with the following rules.

$$\textsc{Rpar}\ \frac{\alpha \xrightarrow{\pi} \alpha'}{\alpha \parallel \beta \xrightarrow{\pi} \alpha' \parallel \beta} \qquad \textsc{Lpar}\ \frac{\beta \xrightarrow{\pi} \beta'}{\alpha \parallel \beta \xrightarrow{\pi} \alpha \parallel \beta'} \qquad \textsc{Rec}\ \frac{\alpha[\mu t.\alpha/t] \xrightarrow{\pi} \alpha'}{\mu t.\alpha \xrightarrow{\pi} \alpha'}$$

This transition system does not satisfy the two conditions that were used in proving the equality of the two notions of bisimilarity in the previous section, as the types $l\|l$ and $l\|\uplus l$ show. However, the counterexamples of Example 2.9 are still not expressible in our language; we are currently checking whether the two bisimilarities coincide in the presence of parallel composition and recursion. Apart from this, it is simple to verify that both $\approx_o$ and $\approx_s$ are congruences, and that standard algebraic laws hold. For instance, $\langle \mathcal{O}/\!\approx_{\{o,s\}}, \|, \mathbf{0}\rangle$ is a commutative monoid, and $\mu t.\alpha \approx_{\{o,s\}} \alpha[\mu t.\alpha/t]$.

4 Concluding remarks

We propose an algebraic treatment of non-uniform types for concurrent objects, with an operational semantics and a behavioural equivalence. A type characterises the semantics of an object in a concurrent setting with asynchronous message passing. It is an internal view of the object behaviour. Operationally, a type is a state transition system, where the basic transition is an object method execution. A silent (hidden) transition corresponds to the execution of a method of another object, and is not directly observable.

Further work includes the study of infinite processes and the search for a complete axiomatization of the algebra of object types. So far, candidate axiomatic systems tend to be cumbersome; we view this essentially as a consequence of the lack of associativity of $\uplus$. We also expect to apply the ideas described in this paper to the TyCO type system proposed in [9], and to relate the type algebra to a process calculus, for instance relating type equivalence to a process equivalence.

Acknowledgements

This work is partially supported by FCT, as well as by PRAXIS XXI Projects 2/2.1/MAT/262/94 SitCalc, 2/2.1/MAT/46/94 Escola, PCEX/P/MAT/46/96 ACL plus 2/2.1/TIT/1658/95 LogComp, and ESPRIT IV Working Groups 22704 ASPIRE and 23531 FIREworks.

References

[1] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51:129–176, 1987.

[2] G. Boudol. Typing the use of resources in a concurrent calculus. In *Asian Computing Science Conference*, volume LNCS 1345, pages 239–253. Springer-Verlag, 1997.

[3] J.-L. Colaço, M. Pantel, and P. Sallé. A set constraint-based analyses of actors. In *2nd IFIP conference on Formal Methods for Open Object-based Distributed Systems*, 1997.

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

[5] U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for CCS. *Fundamenta Informaticae*, 16 (2):171–199, 1992.

[6] E. Najm and A. Nimour. A calculus of object bindings. In *2nd IFIP conference on Formal Methods for Open Object-based Distributed Systems*, 1997.

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

[8] A. Ravara and V. Vasconcelos. Behavioural types for a calculus of concurrent objects. In *3rd International Euro-Par Conference*, volume LNCS 1300, pages 554–561. Springer-Verlag, 1997.

[9] A. Ravara and V. Vasconcelos. A type algebra for concurrent objects. Research report, Department of Mathematics, Instituto Superior Técnico, Av. Rovisco Pais 1096 Lisboa, Portugal, 1998.

[10] V. Vasconcelos. *A Process-Calculus Approach to Typed Concurrent Objects*. PhD thesis, Departament of Computer Science, Keio University, Japan, 1994.

[11] V. Vasconcelos and M. Tokoro. A typing system for a calculus of objects. In *1st International Symposium on Object Technologies for Advanced Software*, volume LNCS 742, pages 460–474. Springer-Verlag, 1993.