An Algebra of Behavioural Types

António Ravara [*] Pedro Resende [*] Vasco T. Vasconcelos [†]

Abstract

We propose a process algebra, the Algebra of Behavioural Types. A type is a higher-order labelled transition system that characterises all possible life cycles of a concurrent object. States represent interfaces of objects; state transitions model the dynamic change of object interfaces. Moreover, a type provides an internal view of the objects that inhabit it: a synchronous one, since transitions correspond to message reception. To capture this internal view of objects we define a notion of bisimulation, strong on labels and weak on silent actions. We study several algebraic laws and obtain completeness results for finite types.

1 Introduction

In a name-passing calculus such as TyCO, processes denote the behaviour of a community of interacting objects, where each object has a location identified by a name [24]. The processes determine an assignment of types to names that reflects a discipline for communication. The usual types-as-records paradigm gives each name a static type that contains information about all the methods of the object, regardless of whether they are enabled or not [25]. However, concurrent objects can offer services non-uniformly, according to synchronisation constraints, making the availability of a service depend on the state of the system [17]. Objects with methods enabled or disabled according to their internal state are very common in object-oriented programming (e.g. a stack, a finite buffer, an ftp server, a bank account, an automatic cash machine). Non-uniform concurrent objects, and types able to cope with them, constitute the object of study of several authors [3, 10, 16, 17, 19, 21]. These types reflect a dependency of the interface of an object on its internal state, conveying information about dynamic properties of objects. The aim is to build type systems capable of ensuring not only the usual safety properties like subject-reduction, but also more complex properties, like the absence of some deadlocks.

Herein we propose an Algebra of Behavioural Types, ABT, where a type is basically a collection of enabled methods (an interface), and a behavioural type is dynamic in the sense that the execution of a method can change the type. Therefore, the type of an object is a partial representation of its behaviour modelled as a labelled transition system. 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 the gnome can detect refusals of methods. The action of unblocking an object, denoted by υ, cannot be directly observed by the gnome because it corresponds to an invocation of some method in another object. Thus, this action is similar to CCS's τ since it is hidden, but it is external rather than internal [13]. Therefore, we reach a set of laws different from Milner's τ-laws. ABT is similar to the Basic Parallel Processes, BPP, a fragment of CCS proposed by Christensen where communication is not present (parallel composition is simply a merge) [5].

[*]Department of Mathematics, Instituto Superior Técnico, Lisbon, Portugal.
Email: `{amar,pmr}@math.ist.utl.pt`

[†]Department of Informatics, Faculty of Sciences, University of Lisbon, Portugal.
Email: `vv@di.fc.ul.pt`

The purpose of this paper is the study of semantic foundations of types for non-uniform concurrent objects. In particular, we discuss on a notion of equivalence and study its algebraic properties. The use of ABT as syntactic types for a calculus of concurrent objects, with a static type system enjoying the subject reduction property, is already in progress [22].

We build ABT gradually. Section 2 presents finite types, with a non-deterministic labelled sum and a blocking operator, defines the operational semantics, an equivalence notion, and a complete axiomatisation for that equivalence. In Section 3 we add a parallel composition operator that is a merge of processes without communication, and show that an extension of the previous axiomatic system (with expansion laws and a saturation law) is still complete. Section 4 finally presents the full algebra, with dynamic types obtained by adding a constructor for recursion. The axiom system has two recursion laws that we prove sound, and we conjecture that it is also complete.

2 Non-Deterministic Finite Types

We start by presenting an algebra of non-deterministic sequential finite types. The basic term is an *object type*, a finite collection of methods denoted by labelled sums standing for an interface of an object. As we allow the same label to appear more than once (possibly with different continuations under the prefix), the sum is non-deterministic. This fact makes possible the definition of an expansion law later on when we introduce a parallel composition operator.

We construct unavailable, i.e., *blocked* object types by means of a blocking operator – denoted by υ – that prefixes object types. A sum of blocked object types is a sum of possible types of an object, depending on the state of the system. Hence, the release of the blocked type is an *inter-object choice* that makes available one of the types in the sum.

Syntax. Consider a countable set of *method names* l, m, possibly subscripted.

Definition 2.1. The grammar defines the set $\mathcal{T}_{\mathrm{sf}}$ of *sequential finite types.*

$$\alpha \quad ::= \quad \sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i \quad | \quad \sum_{i\in I} \upsilon.\alpha_i$$

where I is a finite indexing set, and each $\widetilde{\alpha}_i$ is a finite sequence of types.

We call a term such as $l(\widetilde{\alpha}).\alpha$ a *method type.* The label l in the prefix stands for the name of the method, which has parameters of type $\widetilde{\alpha}$; the type α prescribes the behaviour of the object after the execution of the method with parameters of type $\widetilde{\alpha}$. The term $\upsilon.l(\widetilde{\alpha}).\alpha$ is a *blocked type*, the type of an unavailable method.

The only type composition operator of the algebra is the sum, '$\sum$', which has two uses:

1. it puts various method types together to form the type of an object that offers the corresponding collection of methods: the *labelled sum* $\sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i$;
2. associates various blocked types in the (blocked) sum $\sum_{i\in I} \upsilon.\alpha_i$; after being released the object behaves according to one of the types α_i.

Notation 2.2.

1. We define the following abbreviations:
   (a) $\mathbf{0}$ denotes the empty type (sum with empty indexing set), we omit the sum symbol if the indexing set is singular, and we use the plus ('$+$') to denote binary sums of types;
   (b) $l(\widetilde{\alpha})$ denotes $l(\widetilde{\alpha}).\mathbf{0}$, and l denotes $l()$.
2. We write $\alpha \equiv \beta$ when the types α and β are *syntactically identical.*

Operational semantics. We define a structural operational semantics for finite types via a labelled transition relation on types.

Definition 2.3. The grammar defines the set of *labels*: $\pi \ ::= \ \upsilon \mid l(\widetilde{\alpha})$.

The label υ denotes a silent transition that releases a blocked object; a label $l(\widetilde{\alpha})$ denotes a transition corresponding to the invocation of method l with actual parameters of types $\widetilde{\alpha}$.

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

$$\textsc{Act} \quad \sum_{i\in I} \pi_i.\alpha_i \xrightarrow{\pi_j} \alpha_j \quad (j\in I)$$

Act is in fact an axiom-schema that captures two cases:

1. the basic transition is the invocation of a method with name l and parameters of types $\widetilde{\alpha}$, yielding the type of the object in the method's body;
2. a silent transition υ (unblocking) releases a blocked type.

Terminology. Some terminology regarding the transition relation will make some proofs clearer.

1. If $\alpha \xrightarrow{l(\widetilde{\alpha})} \alpha'$ or $\alpha \xrightarrow{\upsilon} \alpha'$ then α' is a *derivative* of α. In the first case it is an l-derivative and the second an υ-derivative;
2. types that are not of the form $\sum_{i\in I} \upsilon.\alpha_i$ are *unblocked*. Furthermore, α is *strictly blocked* (respectively *strictly unblocked*) if $I\neq\emptyset$ (respectively if α has an l-derivative).

Notation 2.5. 1. Let $\Longrightarrow$ denote $\xrightarrow{\upsilon}{}^*$;

2. in a labelled sum $\alpha \equiv \sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i$ the set $\{l_i(\widetilde{\alpha}_i)\}_{i\in I}$ is the *interface* of the object, denoted by $\mathrm{int}(\alpha)$. Note that the interface of a blocked type is empty.

Equivalence notion. We want two types to be equivalent if they have the same interface and if after each transition they continue to be equivalent, in a bisimulation style. Furthermore, from the point of view of each type, transitions of other types can be regarded as hidden transitions, which would suggest weak bisimulation as the right notion of equivalence for our types, with υ playing the role of Milner's τ, but representing external interaction rather than internal. However, we want 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 $\upsilon.l$ to be different from l, since all the internal observer can see is that the object is blocked, and after being released it can eventually execute the method l. This discards weak bisimulation as a candidate for type equivalence. However, we want $\upsilon.l$ and $\upsilon.\upsilon.l$ to be equivalent because the number of unblockings cannot be counted from within the object as they correspond to transitions on other objects, which discards strong bisimulation [13] and progressing bisimulation [15]. We also want to distinguish $l.\upsilon.m$ from $l.m$ on the grounds that for the latter a blocking after l cannot be observed, and thus observational congruence [13] and rooted bisimulation [1] are unsuitable. Also, notice that in a sense all the above mentioned equivalences, with the exception of weak bisimulation, are stronger than necessary because they are congruences with respect to binary sums, as in CCS [13], whereas in this paper we stick to prefixed sums.

These considerations lead to the choice of a notion of equivalence that we call *label-strong bisimulation*, or *lsb*. It is a higher-order strong bisimulation on labels and a weak bisimulation on unblockings. We require that if α and β are bisimilar then:

1. if α offers a particular method then also β offers that method, and the parameters and the bodies of the methods are pairwise bisimilar;
2. if α offers a hidden transition then β can offer zero or more hidden transitions.

Definition 2.6 (Bisimilarity on types).

1. A symmetric binary relation $R \subseteq \mathcal{T}_{\text{sf}} \times \mathcal{T}_{\text{sf}}$ is a *label-strong bisimulation* if whenever $\alpha R \beta$
   (a) $\alpha \xrightarrow{l(\widetilde{\alpha})} \alpha'$ implies $\exists \beta', \widetilde{\beta}$ ($\beta \xrightarrow{l(\widetilde{\beta})} \beta'$ and $\alpha' \widetilde{\alpha} R \beta' \widetilde{\beta}$);[1]
   (b) $\alpha \xrightarrow{\upsilon} \alpha'$ implies $\exists \beta'$ ($\beta \Longrightarrow \beta'$ and $\alpha' R \beta'$);
2. Two types α and β are *label-strong bisimilar*, or simply *bisimilar*, and we write $\alpha{\approx}\beta$, if there is a label-strong bisimulation R such that $\alpha R \beta$.

The usual properties of bisimilarities hold, namely $\approx$ is an equivalence relation and a fixed point, in the sense that $\alpha{\approx}\beta$ holds if, and only if,

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

Also, it is simple to verify that $\alpha{\approx}\beta$ holds if, and only if,

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

Two types are not bisimilar if they have different interfaces or after some matching transitions there derivatives have different interfaces.

The sum of method types and the sum of blocked types preserve bisimilarity. This result is simple to verify, since the sum is guarded.

Proposition 2.1. Let $\widetilde{\alpha}_i \approx \widetilde{\beta}_i$ and $\alpha_i \approx \beta_i$ for all i in some indexing set I.

1. $\sum_{i \in I} l_i(\widetilde{\alpha}_i).\alpha_i \approx \sum_{i \in I} l_i(\widetilde{\beta}_i).\beta_i$, and
2. $\sum_{i \in I} \upsilon.\alpha_i \approx \sum_{i \in I} \upsilon.\beta_i$.

[Briefly, $\approx$ is a (higher-order) congruence relation with respect to prefixing and summation.]

Proof. Proving that labelled sums and blocked sums preserve $\approx$ is a direct application of Definition 2.6, and the fixed point property of $\approx$. □

Algebraic characterisation. We present an axiomatisation of the equivalence notion and show that it is sound and complete. The proof technique for completeness is standard: a definition of a normal form for the types, a lemma ensuring that for all types their exists an equivalent normal form, and finally the completeness theorem says that for all pairs of equivalent normal forms their exists a derivation of there equality using the rules of the axiomatic system.

Prop/Definition 2.7. The Axiom System $\mathcal{A}_{\text{sf}}$.

Commutativity $\ldots + \pi_1.\alpha_1 + \pi_2.\alpha_2 + \ldots \approx \ldots + \pi_2.\alpha_2 + \pi_1.\alpha_1 + \ldots$;

Idempotence $\pi.\alpha + \pi.\alpha + \ldots \approx \pi.\alpha + \ldots$;

υ-law $\upsilon. \sum_{i \in I} \upsilon.\alpha_i \approx \sum_{i \in I} \upsilon.\alpha_i$.

Proof. Straightforward. □

Remark 2.8.

1. We write $\vdash \alpha{=}\beta$ when we can prove $\alpha{\approx}\beta$ using the laws above and the usual rules of equational logic, together with the additional one: if $\vdash \alpha{=}\beta$ and $\vdash \widetilde{\alpha}{=}\widetilde{\beta}$ then $\vdash l(\widetilde{\alpha}).\alpha{=}l(\widetilde{\beta}).\beta$.
2. An interesting instance of the υ-law is $\alpha{\approx}\mathbf{0}$ if α is a blocked sum with all its derivatives being also blocked sums (i.e., α is a tree where all branches have label υ).

[1]Let $\alpha' \widetilde{\alpha} R \beta' \widetilde{\beta}$ denote $\alpha' R \beta', \alpha_1 R \beta_1, \ldots$, and $\alpha_n R \beta_n$, where n is the length of the sequences $\widetilde{\alpha}$ and $\widetilde{\beta}$.

3. One can easily recognise particular instances of the τ-laws of CCS that hold in this setting. For example, the υ-law corresponds to an instance of the τ-law $\alpha.\tau.P{=}\alpha.P$, where $\alpha{\equiv}\tau$, and it is possible to derive instances of the second and third τ-laws, namely:
   (a) $\tau.P + \tau.\tau.P = \tau.\tau.P$, since $\tau.\tau.P = \tau.P$;
   (b) $\tau.(\tau.P + \tau.Q) = \tau.(\tau.P + \tau.Q) + \tau.Q$;
   use first the υ-law, then idempotence, and then again the υ-law.

   However, the laws do not hold in general; for instance, in this setting the third one does not hold: $l.(\upsilon.\alpha{+}\upsilon.\beta){\neq}l.(\upsilon.\alpha{+}\upsilon.\beta){+}l.\beta$.

Definition 2.9 (Normal forms of sequential types).

1. A type α is *saturated* if $\alpha \overset{\upsilon}{\Longrightarrow} \alpha'$ implies $\alpha \xrightarrow{\upsilon} \alpha'$.
2. A type α is a *normal form* if it is saturated and furthermore one of the following conditions holds:
   (a) $\alpha \equiv \mathbf{0}$;
   (b) $\alpha \equiv \sum_{i\in I} \upsilon.\alpha_i$ and each α_i is a normal form;
   (c) $\alpha \equiv \sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i$, where each component of each sequence $\alpha_i\widetilde{\alpha}_i$ is a normal form.

Definition 2.10 (Depth of a Type).
The depth of a type is inductively defined as follows:

$$\begin{aligned}
\mathrm{depth}(\mathbf{0}) &= 0,\\
\mathrm{depth}(\textstyle\sum_{i\in I} \upsilon.\alpha) &= 1 + \max\{\mathrm{depth}(\alpha_i) \mid i \in I\}, \text{ and}\\
\mathrm{depth}(\textstyle\sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i) &= 1 + \max\{\mathrm{depth}(\widetilde{\alpha}_i) + \mathrm{depth}(\alpha_i) \mid i \in I\},
\end{aligned}$$

where $\mathrm{depth}(\widetilde{\alpha}_i) = \max\{\mathrm{depth}(\beta) \mid \beta \in \widetilde{\alpha}_i\}$.

We prove now that all types have equivalent normal forms, a crucial lemma to achieve completeness.

Lemma 2.2 (Normal Forms).
For all types α there exists a normal form α' such that $\vdash \alpha{=}\alpha'$, with $\mathrm{depth}(\alpha'){\leq}\,\mathrm{depth}(\alpha)$.

Proof. By induction on the depth of α.
If $\alpha{\equiv}\mathbf{0}$ then it is a normal form by definition. Otherwise, if $\alpha \equiv \sum_{i\in I} \pi_i.\alpha_i$ then, by induction hypothesis, for each α_i there exists a normal form α_i' such that $\vdash \alpha_i{=}\alpha_i'$, with $\mathrm{depth}(\alpha_i'){\leq}\,\mathrm{depth}(\alpha_i)$. The prefix can be of two forms:

1. $\pi_i \equiv l_i(\widetilde{\alpha}_i)$, for all i. Since the types $\widetilde{\alpha}_i$ also have normal forms $\widetilde{\alpha}_i'$, and as clearly, $\mathrm{depth}(\sum_{i\in I} l_i(\widetilde{\alpha}_i').\alpha_i') \leq \mathrm{depth}(\alpha)$, we conclude that
   $\vdash \alpha = \sum_{i\in I} l_i(\widetilde{\alpha}_i').\alpha_i'$.
2. $\pi_i \equiv \upsilon$, for all i. We have to guarantee saturation. For each i such that $\alpha_i' \xrightarrow{\upsilon}$, there is a $J_i{\neq}\emptyset$ such that $\alpha_i' \equiv \sum_{j\in J_i} \upsilon.\alpha_j$ and, for each $j{\in}J_i$ we have $\sum_{i\in I} \upsilon.\alpha_i' \xrightarrow{\upsilon.\upsilon} \alpha_j$. By idempotence and the υ-law, we have
   $\vdash \sum_{i\in I} \upsilon.\alpha_i' = \sum_{i\in I} \upsilon.\alpha_i' + \upsilon.\alpha_j$. We can add $\upsilon.\alpha_j$ to $\sum_{i\in I} \upsilon.\alpha_i'$ for all such α_j and thus obtain a normal form of α whose depth clearly equals that of $\sum_{i\in I} \upsilon.\alpha_i'$.

□

We are now in a position to prove the main result of this section: the completeness of the axiomatisation with respect to the equivalence notion.

Theorem 2.3. $\alpha \approx \beta$ if, and only if, $\vdash \alpha{=}\beta$.

Proof. The 'if' direction (*soundness*) is a consequence of Proposition 2.1 and of Prop/Definition 2.7. In particular, Proposition 2.1 ensures that the inference rule of Remark 2.8 is sound. The 'only-if' direction (*completeness*) is done by induction on the sum of the depths of the types α and β (assumed to be normal forms, by Lemma 2.2). □

3 Concurrent Finite Types

Now we extend the algebra of non-deterministic sequential finite types with concurrent types, via a parallel composition operator. Since the algebra does not have communication this operator is simply a merge of types.

Syntax. Take the set of method names assumed in the previous section.

Definition 3.1. The grammar defines the set $\mathcal{T}_{\mathrm{f}}$ of *concurrent finite types.*

$$\alpha \quad ::= \quad \sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i \quad | \quad \sum_{i\in I} \upsilon.\alpha_i \quad | \quad \alpha \parallel \alpha$$

where I is a finite indexing set, and each $\widetilde{\alpha}_i$ 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). Consider that the prefixes bind tighter than the parallel constructor, i.e., $l.m \parallel n$ is $(l.m) \parallel n$.

Notice that unblocked types are now not only labelled sums, but also parallel compositions involving (at least) a labelled sum. Therefore, if α is an unblocked type, its interface is the union of the interfaces of the labelled sums that are elements of the parallel composition.

Definition 3.2 (Interface of a Type).
The interface of a type is inductively defined by the following rules:

$$\begin{array}{rcl} \mathrm{int}(\sum_{i\in I} \upsilon.\alpha) & = & \emptyset \\ \mathrm{int}(\sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i) & = & \{l_i(\widetilde{\alpha}_i)\}_{i\in I} \\ \mathrm{int}(\alpha_1 \parallel \alpha_2) & = & \mathrm{int}(\alpha_1) \cup \mathrm{int}(\alpha_2) \end{array}$$

Definition 3.3 (Labelled transition relation).
The axiom schema of Definition 2.4 together with the two rules below inductively define the labelled transition relation of the algebra of concurrent finite types.

$$\textsc{Rpar} \quad \frac{\alpha \xrightarrow{\pi} \alpha'}{\alpha \parallel \beta \xrightarrow{\pi} \alpha' \parallel \beta} \qquad\qquad \textsc{Lpar} \quad \frac{\alpha \xrightarrow{\pi} \alpha'}{\beta \parallel \alpha \xrightarrow{\pi} \beta \parallel \alpha'}$$

Proposition 3.1. The parallel composition operator preserves the bisimilarity relation (cf. Definition 2.6).

Proof. It is easy to see that the relation $\{(\alpha \parallel \beta, \alpha' \parallel \beta') \mid \alpha{\approx}\alpha' \text{ and } \beta{\approx}\beta'\}$ is a bisimulation, and thus, $\parallel$ preserves $\approx$. □

Algebraic characterisation. We extend the axiom system $\mathcal{A}_{\mathrm{sf}}$ with laws regarding the parallel composition operator, and show that the resulting axiom system is sound and complete. Notice that we have two expansion laws, since the syntax of finite types does not allow mixing labels and υ in sums, e.g., as in $l.\alpha{+}\upsilon.\beta$. Moreover, we have an extra υ-law which allows us to saturate blocked parallel types that do not expand.

Prop/Definition 3.4. The Axiom System $\mathcal{A}_\mathrm{f}$.
The axiom system $\mathcal{A}_\mathrm{f}$ is composed by the laws of Prop/Definition 2.7 together with the following laws[2].

CM $\langle \mathcal{T}_\mathrm{f}/{\approx}, \|, \mathbf{0}\rangle$ is a commutative monoid;

EXP1 $(\sum_{i\in I} \upsilon.\alpha_i) \parallel (\sum_{j\in J} \upsilon.\beta_j) \approx$

$\sum_{i\in I} \upsilon.(\alpha_i \parallel \sum_{j\in J} \upsilon.\beta_j) + \sum_{j\in J} \upsilon.(\sum_{i\in I} \upsilon.\alpha_i \parallel \beta_j)$;

EXP2 $(\sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i) \parallel (\sum_{j\in J} m_j(\widetilde{\beta}_j).\beta_j) \approx$

$\sum_{i\in I} l_i(\widetilde{\alpha}_i).(\alpha_i \parallel \sum_{j\in J} m_j(\widetilde{\beta}_j).\beta_j) + \sum_{j\in J} m_j(\widetilde{\beta}_j).(\sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i \parallel \beta_j)$;

U1 $\upsilon.\sum_{i\in I} \upsilon.\alpha_i \approx \sum_{i\in I} \upsilon.\alpha_i$;

U2 $\upsilon.(\sum_{i\in I} \upsilon.\alpha_i \parallel \sum_{j\in J} m_j(\widetilde{\beta}_j).\beta_j) \approx$

$\upsilon.(\sum_{i\in I} \upsilon.\alpha_i \parallel \sum_{j\in J} m_j(\widetilde{\beta}_j).\beta_j) + \upsilon.(\alpha_k \parallel \sum_{j\in J} m_j(\widetilde{\beta}_j).\beta_j)$, with $k{\in}I$.

Proof. Straightforward. □

To prove the completeness of the axiom system with respect to the equivalence notion we have three alternatives:

1. allow arbitrary labelled sums, i.e., mixing labels and υ in sums, and having a single expansion law; the proof of completeness is standard;
2. add a new inference rule in the equational logic:
   if $\vdash \alpha_1\|\alpha_2{=}\beta_1\|\beta_2$ and $\vdash \widetilde{\alpha}{=}\widetilde{\beta}$ then $\vdash l(\widetilde{\alpha}).\alpha_1\|\alpha_2{=}l(\widetilde{\beta}).\beta_1\|\beta_2$;
3. use only induction.

The first alternative is, somehow, unnatural, since labelled sums represent interfaces of objects, and an arbitrary labelled sum does not represent an interface. Hence, we do not follow this path. Notice that, for the last two alternatives, normal forms include a parallel composition, as there is no expansion law for the parallel composition of a labelled sum and a blocked type; thus the proofs of the normal form lemma and of the completeness theorem are different from those for CCS. We proceed now according to the second alternative.

Definition 3.5 (Normal forms of concurrent types).
A type α is a *normal form* if it is saturated (cf. Definition 2.9), and furthermore, one of the following conditions holds:

1. $\alpha \equiv \mathbf{0}$;
2. $\alpha \equiv \sum_{i\in I} \upsilon.\alpha_i$ and each α_i is a normal form;
3. $\alpha \equiv \sum_{i\in I} l_i(\tilde{\alpha}_i).\alpha_i$ and each component of each $\alpha_i\tilde{\alpha}_i$ is a normal form;
4. $\alpha \equiv \alpha_1 \parallel \alpha_2$, where α_1 and α_2 are respectively as in 2 and 3 above, with $\alpha_1, \alpha_2 {\not\approx} \mathbf{0}$ and $I, J{\neq}\emptyset$.

Definition 3.6 (Depth of a Type).
The depth of a process is inductively defined by the rules of Definition 2.10, together with the rule $\mathrm{depth}(\alpha \parallel \beta) = \mathrm{depth}(\alpha) + \mathrm{depth}(\beta)$.

Lemma 3.2 (Normal Forms).
For all object types α there exists a normal form α' such that $\vdash \alpha{=}\alpha'$, with $\mathrm{depth}(\alpha'){\leq}\,\mathrm{depth}(\alpha)$.

Proof. By induction on the depth of α. The proof is similar to the one of Proposition 2.2; see the appendix for details. □

A non-trivial consequence of the normal form lemma is the following result.

[2]Notice that the υ-law is now law U1.

Lemma 3.3. If $\alpha \equiv \sum_{i\in I} \upsilon.\alpha_i$ is a normal form then no α_i is a parallel composition.

Proof. Assume that $\alpha_1 \equiv \sum_{j\in J_i} \upsilon.\alpha_j \parallel \sum_{k\in K_i} l_k(\tilde{\alpha}_k).\alpha_k$, which is a normal form. By law U2, $\vdash \alpha = \alpha + \upsilon.(\alpha_j \parallel \sum_{k\in K_i} l_k(\tilde{\alpha}_k).\alpha_k)$, for some $j \in J_i$, and α_j is an υ-sum since α is a normal form. Applying repeatedly law U2, and as the types are finite, we conclude that $\sum_{j\in J_i} \upsilon.\alpha_j \approx \mathbf{0}$, what is an absurd, since α is a normal form. Whence, no derivative of α can be a parallel composition. □

Theorem 3.4. $\alpha \approx \beta$ if, and only if, $\vdash \alpha{=}\beta$.

Proof. The 'if' direction (*soundness*) is a consequence of Proposition 2.1 and of Prop/Definition 3.4. The 'only-if' direction (*completeness*) is done by induction on the sum of the depths of the types α and β (assumed to be normal forms, by Lemma 3.2); see the Appendix A for details. □

To be able to prove the completeness result by a simple induction, the third of the alternatives discussed above, we need the following auxiliary lemmas. However, Lemmas 3.6 and 3.7 are still conjectures. See the second appendix for a detailed discussion.

Lemma 3.5. Let α, β and γ be types, α a labelled sum and γ blocked, such that $\alpha \approx \beta \| \gamma$. Then $\gamma \approx \mathbf{0}$.

Proof. By induction on the depth of α. □

Lemma 3.6. Let α, β and γ be normal forms such that $\alpha \parallel \beta$ and $\alpha \parallel \gamma$ are also normal forms. Then $\alpha \parallel \beta \approx \alpha \parallel \gamma$ implies $\beta \approx \gamma$.

Lemma 3.7. Consider types α and β normal forms, and let $\alpha \equiv \alpha_1 \parallel \alpha_2$ and $\beta \equiv \beta_1 \parallel \beta_2$, with α_1 and β_1 labelled sums, and α_2 and β_2 blocked.
Then $\alpha \approx \beta$ if, and only if, $\alpha_1 \approx \beta_1$ and $\alpha_2 \approx \beta_2$.

Theorem 3.8. $\alpha \approx \beta$ if, and only if, $\vdash \alpha{=}\beta$.

Proof. The 'if' direction (*soundness*) is a consequence of Proposition 2.1 and of Prop/Definition 3.4. The 'only-if' direction (*completeness*) is done by induction on the sum of the depths of the types α and β (assumed to be normal forms, by Lemma 3.2), using Lemmas 3.5 and 3.7; see Appendix B for details. □

4 Dynamic Types

We finally present the Algebra of Behavioural Types. We obtain it by extending the algebra of concurrent finite types with a recursive operator μ to denote infinite types, which enables us to characterise the behaviour of persistent objects.

Syntax. Assume a countable set of variables, denoted by t and possibly subscripted or primed, disjoint from the set of method names considered in the previous sections.

Definition 4.1. The grammar below defines the set $\mathcal{T}$ of *behavioural types.*

$$\alpha \quad ::= \quad \sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i \quad | \quad \sum_{i\in I} \upsilon.\alpha_i \quad | \quad \alpha\|\alpha \quad | \quad t \quad | \quad \mu t.\alpha$$

where I is a finite indexing set, and $\widetilde{\alpha}$ is a finite sequence of types.

Definition 4.2.

1. An occurrence of the variable t in the part α of the type $\mu t.\alpha$ is *bound*; otherwise the occurrence of t is *free*. Moreover, $\mathrm{fv}(\alpha)$ is the set of variables that occur free in α.
2. The type $\alpha[\beta/t]$ denotes the substitution in α of the free occurrences of t by β.

Notation 4.3. For simplicity we sometimes write $t{=}\alpha$ instead of $\mu t.\alpha$.

Operational semantics. Assume the set of labels defined in Definition 2.4. A labelled transition relation defines the operational semantics of the algebra. It is obtained from the one of the previous section by adding a new rule:

$$\text{REC} \quad \frac{\alpha[\mu t.\alpha/t] \xrightarrow{\pi} \alpha'}{\mu t.\alpha \xrightarrow{\pi} \alpha'}$$

It is simple to verify, using standard techniques, that *lsb* is still a congruence, i.e., the recursive operator preserves label-strong bisimulation.

Algebraic laws. We present an axiomatization of the equivalence notion, adding two recursion rules to the previous axiom system, and show its soundness.

DEFINITION 4.4. A free variable t is *guarded* in α if all its occurrences are within some label-prefixed subexpression of α. We say α is *guarded* if all its free variables are guarded.

EXAMPLE 4.5. The variable t is guarded in $l(t).\alpha$, $l(\widetilde{\alpha}).t$, and in $l(v.t).\mathbf{0}$, but not in $v.t$.

PROP/DEFINITION 4.6. The Axiom System $\mathcal{A}$.
The axiom system $\mathcal{A}$ is composed by the laws of Prop/Definition 3.4 together with the following recursion laws.

R1 $\mu t.\alpha \approx \alpha[\mu t.\alpha/t]$;

R2 if $\beta \approx \alpha[\beta/t]$ then $\beta \approx \mu t.\alpha$, provided that α is guarded.

To prove the soundness of the axiom system $\mathcal{A}$ we have to ensure that equations have unique solutions, i.e., if β is guarded, $\alpha_1 \approx \beta[\alpha_1/t]$, and $\alpha_2 \approx \beta[\alpha_2/t]$ then $\alpha_1 \approx \alpha_2$. Law R2 of Prop/Definition 4.6 is a corollary of this result and of law R1 (which has a straightforward proof). The proof follows a method analogous to the one used for CCS, and it can be found in the third appendix.

5 Further Issues

On Bisimulation. An alternative notion of bisimulation is below.

DEFINITION 5.1.

1. A symmetric binary relation $R \subseteq \mathcal{T} \times \mathcal{T}$ is a *label-semi-strong bisimulation*, (*lssb*) if whenever $\alpha R \beta$ then
   (a) $\alpha \xrightarrow{l(\widetilde{\alpha})} \alpha'$ implies $\exists \beta', \widetilde{\beta}, \gamma$ ($\beta \xrightarrow{l(\widetilde{\beta})} \gamma \Longrightarrow \beta'$ and $\alpha' R \beta'$ and $\widetilde{\alpha} R \widetilde{\beta}$);
   (b) $\alpha \xrightarrow{v} \alpha'$ implies $\exists \beta'$ ($\beta \Longrightarrow \beta'$ and $\alpha' R \beta'$);
2. Two types α and β are *label-semi-strong bisimilar*, or simply *lssb*, and we write $\alpha \approx_s \beta$, if there is a label-semi-strong bisimulation R such that $\alpha R \beta$.

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 previous definition hold with R replaced by $\approx_s$. Furthermore, *lssb* is a congruence relation.

This notion differs from *lsb* by allowing unblockings after label execution (condition 1(a)). For deterministic finite types the two notions coincide, as we previously shown [20]. However, as we discussed in that paper, the notions do not coincide in more general transition systems, namely in non-deterministic ones.

Example 5.2 (Comparing *lsb* and *lssb*).
Take the types $l.v.l$ and $l.v.l{+}l.l$;

$$\begin{tikzcd}
\odot \arrow[d, "l"'] & & \odot \arrow[d, "l"'] \arrow[dr, "l"] & \\
\bullet \arrow[d, "v"'] & \approx_s & \bullet \arrow[d, "v"'] & \bullet \arrow[d, "l"] \\
\bullet \arrow[d, "l"'] & \not\approx & \bullet \arrow[d, "l"'] & \bullet \\
\bullet & & \bullet &
\end{tikzcd}$$

In $l.v.l$, the second l is only observable after the occurrence of the unblocking, which corresponds to the execution of some action in another object. There is a causal dependency between the first l, the action corresponding to the unblocking, and the second l. If the law $l.v.l{=}l.v.l{+}l.l$ holds for some equivalence notion then the notion does not capture causality between action execution in different objects, and thus, is a *local* notion, whereas a notion that distinguishes the types in the law is *global* (with respect to the community of objects).

We adopt *lsb* as the "right" notion of bisimulation, for it is global, and it is technically simpler. Furthermore, it is finer than *lssb*. Notice that progressing bisimulation (*pb*) is, in this setting, finer than the previous two since it distinguishes, e.g., $l.v.m$ from $l.v.v.m$ (thus law U1 is not valid for *pb*). However, if we define our notions of bisimulation in CCS they are incomparable to *pb* as, e.g., $P{+}\tau.P \neq \tau.P$ (it is easy to show that $a{+}\tau.a \neq \tau.a$).

On Completeness for infinite types. Having proved the uniqueness of solutions of equations, we conjecture that the axiom system $\mathcal{A}$ of Prop/Definition 4.6 is complete for finite state types and guarded recursion. Once again the proof is inspired by the one done by Milner for image-finite CCS [14].

Completeness for infinite state types is a considerably more difficult problem. One cannot hope for completeness of axiomatizations of equivalence notions in CCS for the calculus is computationally complete. Since the full computational power comes from the substitution mechanisms (communication in this case), in calculi without communication it still makes sense to look for completeness. The study of image-infinite (or infinite-state) systems is a lively area of concurrency theory, with several important results established [4, 8, 12]. We focus our attention in two process algebras: BPA and BPP. BPA is the class of Basic Process Algebra of Bergstra and Klop [2], corresponding to the transition systems associated with Greibach Normal Form (GNF) context-free grammars, in which only left-most derivations are allowed. BPP is the class of Basic Parallel Processes of Christensen [5], which is the parallel counterpart of BPA but with arbitrary derivations. Strong bisimilarity is decidable for BPA [9] and BPP [6, 7]. However there is still no such result for weak bisimilarity on full BPA and BPP, although the result is already established for the totally normed subclasses [11], and a possible decision procedure for full BPP is NP-hard [23]. Nevertheless, we are looking for completeness, since decidability is stronger than what we need: the existence of a proof for each equation suffices.

On Subtyping. The equivalence *lsb* is a symmetric simulation; the simulation itself is a *subtyping* relation, $\leq$, i.e., it is a partial order (reflexive, transitive, and anti-symmetric – any two types which simulate each other are equivalent). Thus, if α' simulates α, $\alpha' \leq \alpha$, α' is also a subtype of α. The simulation is a *subtyping* relation, and the partially ordered set $(T, \leq)$ is a complete lattice.

This is a semantic notion of subtyping. A syntactic notion comes from extra rules of the type system, as for instance, in [18]. It is interesting to define those rules and study the relationship among both notions, and we expect them to coincide.

6 Concluding remarks

The approach of behavioural types in other works is done by using an existing process algebra as types for a language or calculus. We took the inverse approach: we first define adequate types for our setting – non-uniform objects – and then show that the types are a process algebra with convenient properties.

The Algebra of Behavioural Types is a process algebra in the style of CCS. It is very similar to its proper subclass BPP; in particular, communication is not present. There are two major differences:

1. the actions have terms of the process algebra as parameters;
2. the silent action does not denote internal activity in the process, but rather captures external activity, i.e., action execution in other processes.

The nature of the silent action induces an original equivalence notion, different from all other equivalences known for process algebras. Naturally the set of axioms that characterises the equivalence notion is also original. However, the proof techniques are basically the same, but some crucial proofs are simpler. The interesting aspect is that normal forms include a parallel composition, as there is no expansion law for the parallel composition of a labelled sum and a blocked type. Thus the proof of the normal form lemma and of the completeness theorem are different from those for CCS.

Some of the ideas presented in this paper, namely regarding external silent actions and label-(semi-)strong bisimulation, appeared first in [20], where however the algebra was much less tractable (e.g., with non-associative sums) and no completeness results were obtained.

We are currently using ABT to type non-uniform concurrent objects in TyCO, where we formalise a notion of process with a communication error that copes with non-uniform service availability [22]. The type system assigns terms of the type algebra to processes, and ensures that typable processes are not locally deadlocked, and do not run into errors.

We believe that ABT can be use to type other concurrent calculi with extensions for objects, but also to type calculi with localities. To fully use its expressiveness one can define in its favourite calculus functionalities like a method update that changes the type of the method, object extension adding methods, distributed objects without uniqueness of objects' identifiers, and non-uniform objects.

Acknowledgements

Special thanks to everyone in the Mobility Group of INRIA at Sophia-Antipolis, for very fruitful discussions and careful reading of parts of this document.

This work was partially supported by the PRAXIS XXI Program and FCT, as well as by PRAXIS XXI Projects PCEX/P/MAT/46/96 ACL and 2/2.1/TIT/1658/95 LogComp, and ESPRIT IV Working Groups 22704 ASPIRE and 23531 FIREworks.

References

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

[2] Jan A. Bergstra, and Jan W. Klop. Algebra of Communicating Processes with Abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

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

[4] Olaf Burkart, and Javier Esparza. More infinite results. In *Bulletin of the European Association for Theorectical Computer Science*, volume 62, pages 138–159. 1997.

[5] Søren Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis ECS-LFCS-93-278, Departament of Computer Science, University of edinburgh, U. K., 1993.

[6] Søren Christensen, Yoram Hirshfeld, and Faron Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *Logic in Computer Science Conference*, IEEE, Computer Society Press, 1993.

[7] Søren Christensen, Yoram Hirshfeld, and Faron Moller. Bisimulation equivalence is decidable for basic parallel processes. In *4th International Conference on Concurrency Theory*, volume LNCS 715, pages 143–157. Springer-Verlag, 1993.

[8] Søren Christensen, and Hans Hüttel. Decidability issues for infinite-state processes – a survey. In *Bulletin of the European Association for Theorectical Computer Science*, volume 51, pages 156–166. 1993.

[9] Søren Christensen, Hans Hüttel, and Colin Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121(2):143–148, 1995.

[10] Jean-Louis Colaço, Mark Pantel, and Patrick Sallé. A set constraint-based analyses of actors. In *2nd IFIP Conference on Formal Methods for Open Object-based Distributed Systems*, 1997.

[11] Yoram Hirshfeld. Bisimulation trees and the decidability of weak bisimulations. In *1st International Workshop on the Verification of Infinite-State Systems*, volume ENTCS 5. Elsevier Science Publishers, 1997

[12] Faron Moller. Infinite results. In *7th International Conference on Concurrency Theory*, volume LNCS 1119, pages 195–216. Springer-Verlag 1996.

[13] Robin Milner. *Communication and Concurrency.* C. A. R. Hoare Series Editor—Prentice-Hall, 1989.

[14] Robin Milner. A complete axiomatisation for observational congruence of finite-state behaviors. *Information and Computation*, 81(2):227–247, 1989.

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

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

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

[18] Benjamin Pierce and Davide Sangiorgi. Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science*, 6 (5):409–454, 1996.

[19] Franz Puntigam. Coordination requirements expressed in types for active objects In *11th European Conference on Object-Oriented Programming*, volume LNCS 1241, pages 367–388. Springer-Verlag, 1997.

[20] António Ravara, Pedro Resende, and Vasco T. Vasconcelos. Towards an algebra of dynamic object types. In *ICALP'98 Workshop on Semantics of Objects as Processes*. BRICS Note Series NS–98–05, 1998.

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

[22] António Ravara and Vasco T. Vasconcelos. Typing non-uniform concurrent objects. Research report, Department of Mathematics, Instituto Superior Técnico, Av. Rovisco Pais 1096 Lisboa, Portugal, 1999.

[23] Jitka Stříbrná. Hardness results for weak bisimilarity of simple process algebras. In *MFCS'98 Workshop on Concurrency*, volume ENTCS 18. Elsevier Science Publishers, 1998

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

[25] Vasco T. Vasconcelos and Mario 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.

A Proof of Theorem 3.4

Proof. The 'if' direction (soundness) is a consequence of Proposition 2.1 and of Prop/Definition 3.4. The 'only-if' direction is done by induction on the sum of the depths of the types α and β (assumed to be normal forms, by Lemma 3.2). Taking into account the proof of Theorem 2.3 we only have one case to consider.

Case $\alpha{\equiv}(\sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i) \parallel (\sum_{j\in J} \upsilon.\alpha_j)$ and $\beta{\equiv}(\sum_{k\in K} m_k(\widetilde{\beta}_k).\beta_k) \parallel (\sum_{l\in L} \upsilon.\beta_l)$;

If $I{\neq}\emptyset$ then obviously also $I{\neq}\emptyset$ as $\alpha{\approx}\beta$. If $J{\neq}\emptyset$ then, by Lemma 3.5, also $L{\neq}\emptyset$. Thus, consider $I, J, K, L{\neq}\emptyset$, and consider the blocked sums not equivalent to $\mathbf{0}$. If $\alpha \xrightarrow{l_i(\widetilde{\alpha}_i)} \alpha_i \| \sum_{j\in J} \upsilon.\alpha_j$, and since, by hypothesis we have $\alpha{\approx}\beta$, then there exists a $k{\in}K$ such that $l_i{=}m_k$ and $\beta \xrightarrow{m_k(\widetilde{\beta}_k)} \beta_k \| \sum_{l\in L} \upsilon.\beta_l$, with $\widetilde{\alpha}_i{\approx}\widetilde{\beta}_k$ and $\alpha_i \| \sum_{j\in J} \upsilon.\alpha_j \approx \beta_k \| \sum_{l\in L} \upsilon.\beta_l$. Now, α_i is either a labelled sum or a blocked sum. In the first case, the result follows by induction. In the second case we apply EXP1 and the proof is similar to the corresponding case in the proof of Theorem 2.3, using the inference rule. If $\alpha \xrightarrow{\upsilon} \sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i) \parallel \alpha_i$ then we proceed similarly to the previous case, using the corresponding case in the proof of Theorem 2.3. $\square$

B Proofs of Section 3

Lemma B.1 (Normal Forms).
For all object types α there exists a normal form α' such that $\vdash \alpha{=}\alpha'$, with $\mathit{depth}(\alpha'){\leq}\mathit{depth}(\alpha)$.

Proof. By induction on the depth of α. The proof is similar to the one of Proposition 2.2, and we omit the common cases. Hence, we only have two cases to consider.

Case $\alpha \equiv \sum_{i\in I} \upsilon.\alpha_i$; by induction hypothesis, for each α_i there exists a normal form α'_i such that $\vdash \alpha_i{=}\alpha'_i$, with $\mathit{depth}(\alpha'_i){\leq}\mathit{depth}(\alpha_i)$. The interesting case now is when $\alpha'_i \equiv (\sum_{j\in J_i} \upsilon.\alpha_j) \parallel \beta$, with $\beta \equiv \sum_{k\in K_i} l_k(\widetilde{\beta}_k).\beta_k$ and $J_i, K_i{\neq}\emptyset$; then we have $\sum_{i\in I} \upsilon.\alpha'_i \xrightarrow{\upsilon.\upsilon} \alpha_j \parallel \beta$, and, by U2, $\vdash \sum_{i\in I} \upsilon.\alpha'_i = \sum_{i\in I} \upsilon.\alpha'_i + \upsilon.(\alpha_j \parallel \beta)$.

Now $(\alpha_j \parallel \beta)$ may not be a normal form (e.g., if $\alpha_j{\equiv}\mathbf{0}$), but $\mathit{depth}(\alpha_j \parallel \beta){<}\mathit{depth}(\alpha)$, and thus by induction hypothesis there is a normal form γ such that $\vdash \alpha_j \parallel \beta{=}\gamma$; hence $\vdash \sum_{i\in I} \upsilon.\alpha'_i = \sum_{i\in I} \upsilon.\alpha'_i{+}\upsilon.\gamma$. We still have to saturate $\upsilon.\gamma$. If it is a blocked sum, using U1 we obtain a saturated form γ'; if it is a parallel composition we use U2 to obtain γ'. So, $\vdash \sum_{i\in I} \upsilon.\alpha'_i = \sum_{i\in I} \upsilon.\alpha'_i{+}\gamma'$. Applying repeatedly U2 in the same way to all α'_i of this form, and applying U1 as in the proof of Theorem 2.3, we attain a normal form for α whose depth does not exceed that of $\sum_{i\in I} \upsilon.\alpha'_i$.

Finally, if $\alpha{\equiv}\gamma_1 \parallel \gamma_2$ we use the commutative monoid laws and the expansion laws to rewrite α either as a blocked sum, a labelled-prefixed sum, or a parallel composition of the previous two with $I, J{\neq}\emptyset$ and without increasing the depth of the types. The first two cases are treated as above, and in the parallel composition case we apply the same reasoning to each sum separately, obtaining $\vdash \alpha{=}\alpha_1 \parallel \alpha_2$, where α_1 and α_2 are respectively a blocked sum and a labelled-prefixed sum, both normal forms. If $\vdash \alpha_1{=}\mathbf{0}$ then $\vdash \alpha_1 \parallel \alpha_2{=}\alpha_2$; otherwise, $\alpha_1 \parallel \alpha_2$ is a normal form. Moreover, notice that $\mathit{depth}(\alpha_1){\leq}\mathit{depth}(\sum_{i\in I} \upsilon.\alpha_i)$ and $\mathit{depth}(\alpha_2){\leq}\mathit{depth}(\sum_{j\in J} l_j(\widetilde{\beta}_j).\beta_j)$. Whence $\mathit{depth}(\alpha_1 \parallel \alpha_2){\leq}\mathit{depth}(\alpha)$. $\square$

Lemma B.2. Let α, β and γ be types, α a labelled sum and γ blocked, such that $\alpha \approx \beta \| \gamma$. Then $\gamma{\approx}\mathbf{0}$.

Proof. If β is blocked then clearly $\alpha{\approx}\mathbf{0}$, whence $\gamma{\approx}\mathbf{0}$. For the remainder of this proof let us assume that β is unblocked. The proof follows by induction on depth(α). If depth(α)=1 then $\alpha{\approx}\beta$ and thus $\gamma{\approx}\mathbf{0}$. For the induction step we assume $\gamma{\not\approx}\mathbf{0}$ and we derive a contradiction, as follows. If $\gamma{\not\approx}\mathbf{0}$ then $\gamma \Longrightarrow \gamma' \xrightarrow{l(\widetilde{\gamma})} \gamma''$ for some γ', l, $\widetilde{\gamma}$, and γ''. Since α is unblocked and $\alpha \approx \beta \| \gamma$, we have $\alpha \approx \beta \| \gamma'$ and $\alpha' \approx \beta \| \gamma''$ for some l-derivative α' of α, where we are assuming that β is strictly unblocked and thus so is α'. Furthermore, since $\alpha \approx \beta \| \gamma$ and γ is blocked, there is some l-derivative β' of β such that $\alpha' \approx \beta' \| \gamma$, where β' is strictly unblocked because α' is and γ is blocked. Finally, depth(α')$<$ depth(α), and thus by induction we conclude $\gamma \approx \mathbf{0}$, a contradiction. $\square$

CONJECTURE B.3. Let α, β, and γ be normal forms such that $\alpha \parallel \beta$ and $\alpha \parallel \gamma$ are also normal forms. Then $\alpha \parallel \beta \approx \alpha \parallel \gamma$ implies $\beta \approx \gamma$.

Discussion. Since $\alpha \parallel \beta$ and $\alpha \parallel \gamma$ are both normal forms, by the previous lemma, either α is a blocked sum, and β and γ are both labelled sums, or vice-versa; moreover, $\alpha, \beta, \gamma \not\approx \mathbf{0}$. We proceed by induction ondepth(α)+ depth(β)+ depth(γ). The base cases are:

1. depth(α)+ depth(β)+ depth(γ) = 4 (α is a blocked sum with depth(α)=2, and β and γ are labelled sums of depth 1); the result is immediate;
2. depth(α)+ depth(β)+ depth(γ) = 5 (α is a labelled sum with depth(α)=1, and β and γ are blocked sums of depth 2); the result follows by the definition of the equivalence.

For the induction step we do a case analysis on the structure of the types:

1. case α is blocked (where all its derivatives are not parallel compositions, by Lemma 3.3); if $\alpha \xrightarrow{\upsilon} \alpha_i$ then $\exists \alpha_j \; \alpha \xrightarrow{\upsilon} \alpha_j$ and $\alpha_i \parallel \beta \approx \alpha_j \parallel \gamma$; if $\alpha_i \approx \mathbf{0}$ then, by the previous lemma, $\alpha_j \approx \mathbf{0}$ and the result follows immediatly; otherwise, if $\alpha_i \approx \alpha_j$ then the result follows by induction; the remaining case ($\alpha_i \not\approx \alpha_j$) should be impossible, but we still did not prove it;
2. case α is a labelled sum the proof is similar to the previous case.

$\square$

CONJECTURE B.4. Consider types α and β normal forms.
Let $\alpha \equiv \alpha_1 \parallel \alpha_2$, with $\alpha_1 \equiv \sum_{i \in I} l_i(\widetilde{\alpha}_i).\alpha_i$ and $\alpha_2 \equiv \sum_{j \in J} \upsilon.\alpha_j$, and let $\beta \equiv \beta_1 \parallel \beta_2$, with $\beta_1 \equiv \sum_{k \in K} m_k(\widetilde{\beta}_k).\beta_k$ and $\beta_2 \equiv \sum_{l \in L} \upsilon.\beta_l$.
$\alpha \approx \beta$ if, and only if, $\alpha_1 \approx \beta_1$ and $\alpha_2 \approx \beta_2$.

Discussion. The 'if' direction is a consequence of Proposition 2.1. For the 'only-if' direction notice first that if $\alpha_2 \approx \mathbf{0}$ (or β_2), the result follows by the Lemma B.2. Notice that if we consider $\alpha_1 \approx \beta_1$ but $\alpha_2 \not\approx \beta_2$ or vice-versa then by the contrapositive of the previous lemma we conclude an absurd; moreover, clearly $\alpha_1 \not\approx \beta_2$ and $\alpha_2 \not\approx \beta_1$. ¿From here the proof follows by induction on depth(α_1)+ depth(β_1). The base is depth(α_1)= depth(β_1)=0; we immediately obtain $\alpha_1 \approx \beta_1 \approx \mathbf{0}$ and thus, $\alpha_2 \approx \beta_2$. For the induction step assume depth(α_1)+ depth(β_1)$>$0. Then $\alpha_1 \parallel \alpha_2$ and $\beta_1 \parallel \beta_2$ are strictly unblocked (and in fact the sum of the depths is at least 2—if it is exactly 2 the result follows immediately, therefore we consider it bigger than 2).

So, as $\alpha \approx \beta$, there are l-derivatives $\alpha_i, \beta_k \not\approx \mathbf{0}$ of α_1, β_1 such that $\alpha_i \parallel \alpha_2 \approx \beta_k \parallel \beta_2$ holds; case α_i and β_k are bisimilar labelled sums, by induction we obtain $\alpha_2 \approx \beta_2$, and thus, by Conjecture B.3, also $\alpha_1 \approx \beta_1$; case α_i and β_k are bisimilar blocked sums, again by Conjecture B.3, $\alpha_2 \approx \beta_2$, and $\alpha_1 \approx \beta_1$.

It is also the case that there are υ-derivatives α_j, β_l of α_2, β_2, such that $\alpha_1 \parallel \alpha_j \approx \beta_1 \parallel \beta_l$ holds, with $\alpha_1 \not\approx \beta_1$ and $\alpha_2 \not\approx \beta_2$; the proof follows similarly to the previous case. $\square$

Notice that the previous conjecture is false in general, as the following examples show: $(\upsilon.l \parallel \upsilon.m) \parallel \upsilon.n \approx \upsilon.l \parallel (\upsilon.m \parallel \upsilon.n)$, $(l \parallel m) \parallel n \approx l \parallel (m \parallel n)$, and $(l \parallel \upsilon.m) \parallel \upsilon.n \approx l \parallel (\upsilon.m \parallel \upsilon.n)$.

THEOREM B.5. $\alpha \approx \beta$ if, and only if, $\vdash \alpha{=}\beta$.

The following 'proof' is only a proof if Conjecture B.4 holds. We strongly believe that indeed it is the case.

Proof. The 'if' direction (soundness) is a consequence of Proposition 2.1 and of Prop/Definition 3.4. The 'only-if' direction (completeness) is done by induction on the sum of the depths of the types α and β (assumed to be normal forms, by Lemma B.1).

Taking into account the proof of Theorem 2.3 we only have one case to consider.

Case $\alpha{\equiv}(\sum_{i\in I} l_i(\widetilde{\alpha}_i).\alpha_i) \,\|\, (\sum_{j\in J} \upsilon.\alpha_j)$ and $\beta{\equiv}(\sum_{k\in K} m_k(\widetilde{\beta}_k).\beta_k) \,\|\, (\sum_{l\in L} \upsilon.\beta_l)$, where $I, K{\neq}\emptyset$. There are two cases to be considered:

1. if $J{=}\emptyset, L{\neq}\emptyset$ then the result follows by Lemma B.2;
2. if $J, L{\neq}\emptyset$ then the result follows by Conjecture B.4.

□

C Proofs of Section 4

Definition C.1. An *lsb up to* $\approx$ is a symmetric binary relation R on types such that, whenever $\alpha R \beta$ then

1. $\alpha \xrightarrow{l(\widetilde{\alpha})} \alpha'$ implies $\exists_{\widetilde{\beta},\beta'}(\beta \xrightarrow{l(\widetilde{\beta})} \beta'$ and $\alpha'\widetilde{\alpha} \approx R \approx \beta'\widetilde{\beta})$
2. $\alpha \Longrightarrow \alpha'$ implies $\exists_{\beta'}(\beta \Longrightarrow \beta'$ and $\alpha' \approx R \approx \beta')$

Proposition C.1. Let R be an *lsb* up to $\approx$. Then,

1. $\approx R \approx$ is an *lsb*.
2. $R \subseteq \approx$.

Proof. Similar to the proof for weak bisimulation up to weak bisimilarity (see [13]). □

Remark C.2. With $\xrightarrow{\upsilon}$ instead of $\Longrightarrow$ in the second condition of Definition C.1 the previous proposition would not hold, as the example $R = \{(\upsilon.\upsilon.a, \upsilon.\upsilon.\mathbf{0})\}$ shows.

Notation C.3. Let Vars(α) denote the set of type variables of the type α, and let $\{\tilde{t}\}$ denote the set of the elements of the sequence $\tilde{t}$.

Lemma C.2. Let α be guarded with free variables in $\tilde{t}$.

1. If $\alpha(\widetilde{\beta}) \xrightarrow{l(\widetilde{\gamma})} \gamma$ then there exist α' and $\widetilde{\alpha}$ with free variables in $\tilde{t}$ such that $\gamma \equiv \alpha'(\widetilde{\beta})$, $\widetilde{\gamma} \equiv \widetilde{\alpha}(\widetilde{\beta})$ and, for all $\widetilde{\beta}'$, $\alpha(\widetilde{\beta}') \xrightarrow{l(\widetilde{\alpha}(\widetilde{\beta}'))} \alpha'(\widetilde{\beta}')$.
2. If $\alpha(\widetilde{\beta}) \xrightarrow{\upsilon} \gamma$ then there exists α' with free variables in $\tilde{t}$ such that $\gamma \equiv \alpha'(\widetilde{\beta})$ and, for all $\widetilde{\beta}'$, $\alpha(\widetilde{\beta}') \xrightarrow{\upsilon} \alpha'(\widetilde{\beta}')$. Furthermore, α' is guarded.

Proof. 1. Let $\alpha(\widetilde{\beta}) \xrightarrow{l(\widetilde{\gamma})} \gamma$. The proof is by induction on the maximum length of the derivation of the transition (i.e., by "transition induction"—see [13]). The base case is trivial.

Case $\alpha \equiv \sum_{j\in I} l_j(\widetilde{\alpha}_j).\alpha_j$. Then $\alpha(\widetilde{\beta})$ is $\sum_{j\in I} l_j(\widetilde{\alpha}_j(\widetilde{\beta})).\alpha_j(\widetilde{\beta})$, and thus $l(\widetilde{\gamma})$ is $l_j(\widetilde{\alpha}_j(\widetilde{\beta}))$ and γ is $\alpha_j(\widetilde{\beta})$ for some $j \in I$. The result follows from taking $\alpha' \equiv \alpha_j$ and $\widetilde{\alpha} \equiv \widetilde{\alpha}_j$.

Case $\alpha \equiv \alpha_1 \,\|\, \alpha_2$. Then $\alpha(\widetilde{\beta}) \equiv \alpha_1(\widetilde{\beta}) \,\|\, \alpha_2(\widetilde{\beta})$. There are two cases: either $\gamma \equiv \gamma_1 \,\|\, \alpha_2(\widetilde{\beta})$ with $\alpha_1(\widetilde{\beta}) \xrightarrow{l(\widetilde{\gamma})} \gamma_1$ or $\gamma \equiv \alpha_1(\widetilde{\beta}) \,\|\, \gamma_2$ with $\alpha_2(\widetilde{\beta}) \xrightarrow{l(\widetilde{\gamma})} \gamma_2$, by a shorter derivation. Without loss of generality, assume the first case. Since α is guarded so is α_1, and thus by induction hypothesis it follows that γ_1 is of the form $\alpha_1'(\widetilde{\beta})$ and $\widetilde{\gamma}$ is of the form $\widetilde{\alpha}_1(\widetilde{\beta})$. The result follows from taking $\alpha' \equiv \alpha_1' \,\|\, \alpha_2$ and $\widetilde{\alpha} \equiv \widetilde{\alpha}_1$.

Case $\alpha \equiv \mu t.\beta$. Then $\alpha(\widetilde{\beta}) \equiv \mu t.\beta(\widetilde{\beta})$, where the free variables of β are taken from t and $\tilde{t}$. The variables $\tilde{t}$ must be guarded in β, otherwise they would not be guarded in α. If $\mu t.\beta(\widetilde{\beta}) \xrightarrow{l(\widetilde{\gamma})} \gamma$ then $\beta[\mu t.\beta(\widetilde{\beta})/t] \xrightarrow{l(\widetilde{\gamma})} \gamma$, by a shorter derivation. But the variables of $\tilde{t}$ are guarded in $\beta[\mu t.\beta/t]$, and thus by induction hypothesis we conclude that γ is of the form $\alpha'(\widetilde{\beta})$ and $\widetilde{\gamma}$ of the form $\widetilde{\alpha}(\widetilde{\beta})$.

2. Let $\alpha(\widetilde{\beta}) \xrightarrow{\upsilon} \gamma$. The proof is as in the first case, except that we need not worry about parameters in prefixes. The main difference is that we must also prove that α' is guarded. The case $\alpha \equiv \mathbf{0}$ is trivial. Case $\alpha \equiv \sum_{j \in I} \upsilon.\alpha_j$ all the α_j must be guarded because α is, and thus α' is guarded.

   In the remaining cases the conclusion is a consequence of the induction hypothesis.

□

Lemma C.3. Let α be guarded with free variables in $\tilde{t}$. If $\alpha(\widetilde{\beta}) \Longrightarrow \gamma$ then there exists α' with free variables in $\tilde{t}$ such that $\gamma \equiv \alpha'(\widetilde{\beta})$ and, for all $\widetilde{\beta}'$, $\alpha(\widetilde{\beta}') \Longrightarrow \alpha'(\widetilde{\beta}')$.

Proof. Let $\alpha(\widetilde{\beta}) \Longrightarrow \gamma$ and let n be the actual number of υ's in the transition. The proof follows easily from the previous lemma, by induction on n. □

Now we simplify notation by writing, e.g. $\alpha(\widetilde{\beta})$ instead of $\alpha[\widetilde{\beta}/\tilde{t}]$. We are finally in a position to prove the main result for infinite types: the theorem of the uniqueness of solutions of equations.

Theorem C.4. Let $\widetilde{\beta}$ be guarded, $\widetilde{\alpha}_1 \approx \widetilde{\beta}(\widetilde{\alpha}_1)$ and $\widetilde{\alpha}_2 \approx \widetilde{\beta}(\widetilde{\alpha}_2)$. Then $\widetilde{\alpha}_1 \approx \widetilde{\alpha}_2$.

Proof. Let R be the following relation.

$$R = \{(\gamma(\widetilde{\alpha}_1), \gamma(\widetilde{\alpha}_2)) \mid \mathrm{Vars}(\gamma) \subseteq \tilde{t}\}$$

We will show that:

1. $\gamma(\widetilde{\alpha}_1) \xrightarrow{l(\widetilde{\delta}_1)} \alpha_1$ implies $\exists \alpha_2, \widetilde{\delta}_2(\gamma(\widetilde{\alpha}_2) \xrightarrow{l(\widetilde{\delta}_2)} \alpha_2$ and $\alpha_1(\widetilde{\delta}_1) \approx R \approx \alpha_2(\widetilde{\delta}_2))$;
2. $\gamma(\widetilde{\alpha}_1) \Longrightarrow \alpha_1$ implies $\exists \alpha_2(\gamma(\widetilde{\alpha}_2) \Longrightarrow \alpha_2$ and $\alpha_1 \approx R \approx \alpha_2)$.

By the results we have seen before, and by symmetry, this establishes that R is a label-strong bisimulation up to label-strong bisimilarity, and also that $\gamma(\widetilde{\alpha}_1) \approx \gamma(\widetilde{\alpha}_2)$ for all γ, which includes the cases $\alpha_{1i} \approx \alpha_{2i}$ $(\gamma \equiv t_i)$.

So let us prove 1. We have $\gamma(\widetilde{\alpha}_1) \approx \gamma(\widetilde{\beta}(\widetilde{\alpha}_1))\ R\ \gamma(\widetilde{\beta}(\widetilde{\alpha}_2)) \approx \gamma(\widetilde{\alpha}_2)$ because $\approx$ is a congruence and, by hypothesis, $\widetilde{\alpha}_1 \approx \widetilde{\beta}(\widetilde{\alpha}_1)$ and $\widetilde{\beta}(\widetilde{\alpha}_2) \approx \widetilde{\alpha}_2$. If $\gamma(\widetilde{\alpha}_1) \xrightarrow{l(\widetilde{\delta}_1)} \alpha_1$ then $\gamma(\widetilde{\beta}(\widetilde{\alpha}_1)) \xrightarrow{l(\widetilde{\delta}'_1)} \alpha'_1$, with $\widetilde{\delta}_1 \approx \widetilde{\delta}'_1$ and $\alpha_1 \approx \alpha'_1$. By Lemma C.2, there are $\widetilde{\gamma}$ and γ' such that $\widetilde{\delta}'_1 \equiv \widetilde{\gamma}(\widetilde{\alpha}_1)$, $\alpha'_1 \equiv \gamma'(\alpha_1)$ and $\gamma(\widetilde{\beta}(\widetilde{\alpha}_2)) \xrightarrow{l(\widetilde{\gamma}(\widetilde{\alpha}_2))} \alpha'_2 \equiv \gamma'(\widetilde{\alpha}_2)$, which implies $\widetilde{\gamma}(\widetilde{\alpha}_2) \xrightarrow{l(\widetilde{\delta}_2)} \alpha_2$, with $\widetilde{\gamma}(\widetilde{\alpha}_2) \approx \widetilde{\delta}_2$ and $\alpha'_2 \approx \alpha_2$.
Hence, $\widetilde{\delta}_1 \approx R \approx \widetilde{\delta}_2$ and $\alpha_1 \approx R \approx \alpha_2$.
We prove 2 by a similar reasoning, but using Lemma C.3, instead of Lemma C.2. □