Math. Struct. in Comp. Science (1998), *vol.* 8, *pp.* 447–479. Printed in the United Kingdom
© 1998 Cambridge University Press

On the bisimulation proof method

DAVIDE SANGIORGI

INRIA-Sophia Antipolis, 2004 Rue des Lucioles, B.P. 93,
06902 Sophia Antipolis, France.
Email: davide.sangiorgi@inria.fr.

Received 19 June 1995; revised 3 November 1997

The most popular method for establishing *bisimilarities* among processes is to exhibit bisimulation relations. By definition, $\mathscr{R}$ is a bisimulation relation if $\mathscr{R}$ *progresses* to $\mathscr{R}$ itself, *i.e.*, pairs of processes in $\mathscr{R}$ can match each other's actions and their derivatives are again in $\mathscr{R}$.

We study generalisations of the method aimed at reducing the size of the relations to be exhibited and hence relieving the proof work needed to establish bisimilarity results. We allow a relation $\mathscr{R}$ to progress to a different relation $\mathscr{F}(\mathscr{R})$, where $\mathscr{F}$ is a function on relations. Functions that can be safely used in this way (*i.e.*, such that if $\mathscr{R}$ progresses to $\mathscr{F}(\mathscr{R})$, then $\mathscr{R}$ only includes pairs of bisimilar processes) are *sound*. We give a simple condition that ensures soundness. We show that the class of sound functions contains non-trivial functions and we study the closure properties of the class with respect to various important function constructors, like composition, union and iteration. These properties allow us to construct sophisticated sound functions – and hence sophisticated proof techniques for bisimilarity – from simpler ones.

The usefulness of our proof techniques is supported by various non-trivial examples drawn from the process algebras CCS and π-calculus. They include the proof of the unique solution of equations and the proof of a few properties of the replication operator. Among these, there is a novel result that justifies the adoption of a simple form of prefix-guarded replication as the only form of replication in the π-calculus.

1. Introduction

Bisimilarity has emerged among the most stable and mathematically natural concepts formulated in concurrency theory over the past decades. It is widely accepted as the finest (extensional) behavioural equivalence one would want to impose. Its robustness and elegance are evidenced by various characterisations, in terms of non-well-founded sets, domain theory, modal logic, final coalgebras, open maps (Aczel 1988; Abramsky 1991; Hennessy and Milner 1985; Rutten and Turi 1994; Joyal *et al.* 1994). Bisimilarity has also been advocated outside concurrency theory; for instance, co-induction principles based on bisimilarity have been proposed for reasoning about equality between elements of recursively defined domains and data types (Fiore 1993; Pitts 1994).

We first consider bisimilarity on standard labelled transition systems: transitions are of the form $P \xrightarrow{\mu} Q$, where P and Q are called *processes*, and label μ is drawn from some alphabet of *actions*. In such systems, bisimilarity, abbreviated $\sim$, is defined as the largest symmetric relation $\mathscr{R}$ on processes such that

$$\text{if } (P,Q) \in \mathscr{R} \text{ and } P \xrightarrow{\mu} P', \text{ there is } Q' \text{ such that } Q \xrightarrow{\mu} Q' \text{ and } (P',Q') \in \mathscr{R}. \qquad (*)$$

($\sim$ can also be viewed as the greatest fixed-point of a certain monotone function on relations, whose definition closely follows clause $(*)$.) A relation $\mathscr{R}$ which satisfies clause $(*)$, without necessarily being the largest such relation, is called a *bisimulation relation*. By definition of $\sim$, a bisimulation relation is contained in $\sim$, and hence it consists of only pairs of bisimilar processes. This immediately suggests a proof method for $\sim$ (which is also by far the most popular one): to demonstrate that $(P,Q) \in \sim$ holds, find a bisimulation relation containing the pair (P,Q).

Note that in clause $(*)$, the same relation $\mathscr{R}$ is mentioned in the hypothesis and in the thesis. In other words, when we check the bisimilarity clause on a pair (P,Q), all needed pairs of derivatives, like (P',Q'), must be present in $\mathscr{R}$. We cannot discard any such pair of derivatives from $\mathscr{R}$, or even 'manipulate' its process components. In this way, a bisimulation relation often contains many pairs strongly related with each other, in the sense that, at least, the bisimilarity between the processes in some of these pairs implies that between the processes in other pairs. (For instance, in a process algebra a bisimulation relation might contain pairs of processes obtainable from other pairs through application of algebraic laws for $\sim$, or obtainable as combinations of other pairs and of the operators of the language.) These redundancies can make both the definition and the verification of a bisimulation relation annoyingly heavy and tedious: it is difficult at the beginning to guess all pairs that are needed; and clause $(*)$ must be checked on all pairs introduced.

As an example, let P be a non-deadlocked process from a CCS-like language, and $!P$ the process defined by $!P \stackrel{\text{def}}{=} P \mid !P$. Process $!P$ represents the *replication* of P, that is, a countable number of copies of P in parallel. (In certain process algebras, *e.g.*, the π-calculus, replication is the only form of recursion allowed, since it gives enough expressive power and enjoys interesting algebraic properties – see Section 6.) A property that we naturally expect to hold is that duplication of replication has no behavioural effect, that is, $!P \mid !P \sim !P$. To prove this, we would like to use the *singleton* relation

$$\mathscr{R} \stackrel{\text{def}}{=} \{(!P \mid !P \,,\, !P)\}.$$

But $\mathscr{R}$ is easily seen not to be a bisimulation relation. If we add pairs of processes to $\mathscr{R}$ in order to make it into a bisimulation relation, we might find that the simplest solution is to take the *infinite* relation

$$\mathscr{R}' \stackrel{\text{def}}{=} \{(Q_1,Q_2) \;:\; \text{for some } R. \quad Q_1 \sim R \mid !P \mid !P \quad \text{and} \quad Q_2 \sim R \mid !P\}.$$

The size augmentation in passing from $\mathscr{R}$ to $\mathscr{R}'$ is rather discouraging. But it does somehow seem unnecessary, for the bisimilarity between the two processes in $\mathscr{R}$ already implies that between the processes of all pairs of $\mathscr{R}'$.

The study reported in this paper aims at relieving the work involved with the bisimulation proof method. To anticipate, when applied to the previous example, our proof techniques allow us to prove the property $!P \mid !P \sim !P$ simply using the singleton $\mathscr{R}$.

We generalise the bisimulation proof method by relaxing the bare recursion in (∗). First, we introduce the notion of *progression*: a symmetric relation $\mathscr{R}$ progresses to a relation $\mathscr{S}$, which is abbreviated to $\mathscr{R} \rightarrowtail \mathscr{S}$, if

$$(P, Q) \in \mathscr{R} \text{ and } P \xrightarrow{\mu} P' \text{ imply that there is } Q' \text{ such that } Q \xrightarrow{\mu} Q' \text{ and } (P', Q') \in \mathscr{S}.$$

(Therefore, a relation $\mathscr{R}$ is a bisimulation relation iff $\mathscr{R} \rightarrowtail \mathscr{R}$ holds.) We examine progressions of the form $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R})$, where $\mathscr{F}$ is a function from relations to relations. We are interested in functions $\mathscr{F}$ that are *sound* with respect to $\sim$, that is, such that $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R})$ implies $\mathscr{R} \subseteq \sim$. Questions we shall ask ourselves are: Which conditions ensure soundness of functions? Which interesting functions are sound? Which interesting properties are satisfied by the class of sound functions?

We show that a simple functorial-like condition, called *respectfulness*, guarantees the soundness of a function $\mathscr{F}$ on relations. This condition requires that if $\mathscr{R} \subseteq \mathscr{S}$ and $\mathscr{R} \rightarrowtail \mathscr{S}$ hold, then $\mathscr{F}(\mathscr{R}) \subseteq \mathscr{F}(\mathscr{S})$ and $\mathscr{F}(\mathscr{R}) \rightarrowtail \mathscr{F}(\mathscr{S})$ must hold too. A very useful property of the class of respectful functions is that it is closed under important function constructors like composition, union and iteration. Consequently, it suffices to define a few primitive respectful functions: more complex functions can then be derived via combinations of the primitive ones, and the soundness of the former follows from that of the latter.

Among our primitive functions there will be the identity function and the constant-to-$\sim$ function, which maps every relation onto $\sim$. Another primitive function worth mentioning is a function $\mathscr{C}$ that gives us the closure of a relation $\mathscr{R}$ under contexts; that is, $\mathscr{R} \rightarrowtail \mathscr{C}(\mathscr{R})$ holds if $(P, Q) \in \mathscr{R}$ and $P \xrightarrow{\mu} P'$ imply that

$$\begin{aligned} &\text{there are processes } P'', Q'' \text{ and a context } C \text{ such that} \\ &P' = C[P''],\ Q \xrightarrow{\mu} C[Q''] \text{ and } (P'', Q'') \in \mathscr{R}. \end{aligned} \qquad (**)$$

Function $\mathscr{C}$ yields an 'up-to context' technique by which a common context in the derivatives of two processes can be cancelled. We show that, in the case in which the transition relation among processes is defined structurally on the operators of the language, certain conditions on the form of the transition rules ensure the respectfulness of $\mathscr{C}$. These conditions are met in familiar process algebras like ACP (Bergstra and Klop 1984) and CCS (Milner 1989).

Examples of respectful functions easily derivable from our primitive ones are: the function that returns the transitive closure of a relation; the function that returns the closure of a relation under polyadic contexts (*i.e.*, contexts that might have more than one hole); the function mapping a relation $\mathscr{R}$ onto $\sim\mathscr{R}\sim$, where $\sim\mathscr{R}\sim$ is the composition of the three relations (this function gives us Milner's *bisimulation up-to* $\sim$ technique (Milner 1989) – in our setting, it is recovered as a combination of the identity and constant-to-$\sim$ functions). Again, more sophisticated functions (and hence proof techniques for $\sim$) can in turn be derived from these ones; some of them will be described (and used) in later sections.

A large part of the paper is devoted to applications of our proof techniques. For this, we have chosen CCS and the π-calculus. CCS is perhaps the most studied process algebra. The π-calculus is a process algebra that originates from CCS and permits a natural modelling of systems with dynamic reconfiguration of their communication topology. We show that our techniques yield simpler proofs of some standard theorems of CCS and π-calculus. Examples are the unique solution of equations and the distributivity properties of private replications. We also apply our techniques to derive a new normalisation result for the π-calculus, asserting that every replication $!P$ can be rewritten in terms of normal replications $!\alpha.P$, where α is a prefix of the language. Normal replications are easier to deal with. For instance, they enjoy simpler algebraic laws and are easier to implement.

Further applications of the techniques can be found in the proof of the main results in Sangiorgi (1996) and Boreale and Sangiorgi (1995) – namely the full abstraction of certain semantics of true-concurrent behavioural equivalences in the π-calculus and in Sangiorgi (1995) – namely the characterisation of the equivalence induced on lambda-terms by Milner's encoding of the (lazy) lambda-calculus into the π-calculus.

Our interest in the π-calculus is motivated by (in addition to its relevance as a process algebra) certain peculiarities of its transition system, which deviates from a standard system, like the one for CCS, in some important aspects: first, the π-calculus is a special case of a value-passing calculus, and hence the labels of its transitions may have more than one component; and second, π-calculus transition rules utilise alpha conversion and substitution on names ('name' is synonymous with 'channel'). These features have to be taken into account in the definition of bisimilarity and, among other things, may separate bisimilarity and its induced congruence. The separation affects, for instance, the definition of the function $\mathscr{C}$ (closure under contexts): for the use of clause $(**)$ it is fundamental that bisimilarity be a congruence, since then, intuitively, P'' bisimilar with Q'' implies $C[P'']$ bisimilar with $C[Q'']$. If this is not the case, appropriate constraints have to be added in $(**)$ on the form of context C or on the relationship between processes P'' and Q''. The peculiarities of π-calculus transition system also suggest other primitive respectful functions. One is a function that allows us to apply injective substitutions on names to the derivatives of two processes. This function yields a form of 'up-to injective substitution' technique, which is very handy when dealing with universally-quantified substitutions on names, which are common in the π-calculus.

Related work

Some of the proof techniques described in the paper, or special cases of them, have already appeared in the literature, but we should stress that there has never been a systematic study of the topic. For instance, we feel that we lacked the capability of combining simpler proof techniques into more powerful ones, which is made possible by the theory developed in this paper.

We have already mentioned Milner's *bisimulation up-to* $\sim$ technique (Milner 1989), in which the closure of a bisimulation relation is achieved up to bisimilarity itself. The portability of this technique onto *weak bisimilarities* (where a special action, called the *silent action*, is distinguished from the others and partially ignored in the bisimilarity clause) has been studied by Milner and Sangiorgi (Sangiorgi and Milner 1992).

Two special cases of the up-to-context technique had been put forward previously. In Caucal (1990), Caucal defines a notion of *self-bisimulation* in the setting of BPA processes (they can be viewed as the processes generated by a context-free grammar), which allows him to eliminate common prefixes and suffixes in the derivatives of two processes. Self-bisimulations have been used in Caucal (1990), as well as in a number of other papers (*e.g.*, Christensen *et al.* (1995) and Hirshfeld *et al.* (1996)), to establish decidability results for the classes of BPA and BPP processes (roughly, the latter differ from the former in that the composition operator is commutative). Another form of up-to-context technique is Milner, Parrow and Walker's *bisimulation up-to restriction* (Milner *et al.* 1992), in which common outermost restrictions in the derivatives of two processes can be discarded.

Finally, the up-to injective substitution technique for the π-calculus is also considered, or mentioned, by Boreale and De Nicola (Boreale and De Nicola 1995), and Milner, Parrow and Walker (Milner *et al.* 1992).

Structure of the paper

In Section 2 we develop the theory of progressions, sound functions and respectful functions. In Section 3 we present the process algebra CCS, and apply to it our proof techniques based on respectful functions. In Section 4 we present the syntax and the operational semantics of the π-calculus. In Section 5 we examine how to transport the theory of sound and respectful functions onto the non-standard transition system of the π-calculus; we also introduce a new primitive respectful function, which allows us to work up to injective substitution on names. In Section 6 we apply the theory of the previous section to reasoning about bisimilarity among π-calculus processes. Finally, in Section 7 we report some conclusions and possible directions for future work.

2. Progressions and respectful functions

The results in this section hold for any transition system $(Pr, Act, \longrightarrow)$ with domain Pr, set of *actions* (or *labels*) Act and transition relation $\longrightarrow \subseteq Pr \times Act \times Pr$. We use P, Q and R to range over Pr and call them *processes*; μ and λ range over Act. We write $P \xrightarrow{\mu} Q$ when $(P, \mu, Q) \in \longrightarrow$, to be interpreted as '$P$ may become Q by performing an action μ'.

We let $\mathscr{R}$ and $\mathscr{S}$ range over binary relations on processes, that is, if $\wp$ denotes the powerset construct, then $\mathscr{R}$ and $\mathscr{S}$ are elements of $\wp(Pr \times Pr)$. The union of relations $\mathscr{R}$ and $\mathscr{S}$ is $\mathscr{R} \cup \mathscr{S}$, and their composition is $\mathscr{R}\mathscr{S}$ (that is, $(P, P') \in \mathscr{R}\mathscr{S}$ holds if for some P'', both $(P, P'') \in \mathscr{R}$ and $(P'', P') \in \mathscr{S}$ hold). We often use the infix notation for relations; hence $P \mathrel{\mathscr{R}} Q$ means $(P, Q) \in \mathscr{R}$. We use letters I and J for countable indexing sets in unions and sums.

Definition 2.1. (Progression) Given two relations $\mathscr{R}$ and $\mathscr{S}$, we say that $\mathscr{R}$ *progresses to* $\mathscr{S}$, written $\mathscr{R} \rightarrowtail \mathscr{S}$, if $P \mathrel{\mathscr{R}} Q$ implies:

1 whenever $P \xrightarrow{\mu} P'$, there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \mathrel{\mathscr{S}} Q'$;
2 the converse, that is, whenever $Q \xrightarrow{\mu} Q'$, there is P' such that $P \xrightarrow{\mu} P'$ and $P' \mathrel{\mathscr{S}} Q'$.

When $\mathscr{R}$ and $\mathscr{S}$ coincide, the above clauses are the ordinary ones of the definition of a bisimulation relation.

Definition 2.2. $\mathscr{R}$ is a *bisimulation relation* if $\mathscr{R}$ progresses to itself, that is, $\mathscr{R} \rightarrowtail \mathscr{R}$ holds.

Definition 2.3. Two processes P and Q are *bisimilar*, written $P \sim Q$, if $P \mathrel{\mathscr{R}} Q$ holds, for some bisimulation relation $\mathscr{R}$.

Therefore, if $\mathscr{R}$ progresses to itself, $\mathscr{R}$ is made of pairs of bisimilar processes. This is the basis of the standard method for proving the bisimilarity between two processes: find a relation $\mathscr{R}$ that progresses to itself and that includes the pair of given processes.

However, self-progressions $\mathscr{R} \rightarrowtail \mathscr{R}$ are special cases of progressions, but not the only ones by which process bisimilarities can be inferred. In this paper, we look for general conditions on progressions that guarantee this property. As we shall see, the flexibility so gained will allow us to work with relations often much smaller than those needed to exhibit self-progressions.

We shall consider progressions of the form $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R})$, where $\mathscr{F}$ is a function on relations, that is, a function from $\wp(Pr \times Pr)$ to $\wp(Pr \times Pr)$. We call these *first-order functions* just *functions* for short. Below, $\mathscr{F}$ and $\mathscr{G}$ range over such functions.

Definition 2.4. (Soundness) A function $\mathscr{F}$ is *sound* if, for any $\mathscr{R}$, $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R})$ implies $\mathscr{R} \subseteq \sim$.

Not all functions are sound. An example is the function that maps every relation to the universal relation $Pr \times Pr$. We wish to determine a class of sound functions for which membership is easy to check, that includes interesting functions and that satisfies interesting properties. We propose the class of *respectful* functions.

Definition 2.5. (Respectfulness) A function $\mathscr{F}$ is *respectful* if whenever $\mathscr{R} \subseteq \mathscr{S}$ and $\mathscr{R} \rightarrowtail \mathscr{S}$ holds, then $\mathscr{F}(\mathscr{R}) \subseteq \mathscr{F}(\mathscr{S})$ and $\mathscr{F}(\mathscr{R}) \rightarrowtail \mathscr{F}(\mathscr{S})$ also holds.

Remark 2.6. If we replaced the respectfulness requirement by two separate ones, namely

(a) $\mathscr{R} \subseteq \mathscr{S}$ implies $\mathscr{F}(\mathscr{R}) \subseteq \mathscr{F}(\mathscr{S})$, and
(b) $\mathscr{R} \rightarrowtail \mathscr{S}$ implies $\mathscr{F}(\mathscr{R}) \rightarrowtail \mathscr{F}(\mathscr{S})$,

we would get a stronger definition (*i.e.*, a stronger condition on $\mathscr{F}$) that would not capture important sound functions, such as the function $\mathscr{C}$ for the closure under contexts (Section 2.1).

Remark 2.7. Bisimilarity can also be presented as the greatest fixed-point of a certain monotone function on relations (Milner 1989, Section 4.6), for which the bisimulation relations represent the post-fixed-points. Progressions and respectful functions can then be defined in terms of this fixed-point machinery. We preferred the more operational Definitions 2.1 and 2.5 because they are simpler to use – for the same reason that it is easier to establish that a relation is a bisimulation relation from Definition 2.2 rather than as a post-fixed-point. See the concluding section for more comments on fixed-points and co-induction.

We show that any respectful function is sound. First, we need two lemmas.

Lemma 2.8. Let $\mathscr{R} \stackrel{\text{def}}{=} \bigcup_{i \in I} \mathscr{R}_i$ and suppose for all $i \in I$ there is $j \in I$ such that $\mathscr{R}_i \rightarrowtail \mathscr{R}_j$ holds. Then $\mathscr{R}$ is a bisimulation relation.

Lemma 2.9.

1 If, for some $i \in I$, $\mathscr{S} \rightarrowtail \mathscr{R}_i$, then also $\mathscr{S} \rightarrowtail \left(\bigcup_{i \in I} \mathscr{R}_i\right)$.
2 If, for all $i \in I$, $\mathscr{R}_i \rightarrowtail \mathscr{S}$, then also $\left(\bigcup_{i \in I} \mathscr{R}_i\right) \rightarrowtail \mathscr{S}$.

Corollary 2.10. If for all $i \in I$ there is $j \in J$ such that $\mathscr{R}_i \rightarrowtail \mathscr{S}_j$ holds, then also $\left(\bigcup_{i \in I} \mathscr{R}_i\right) \rightarrowtail \left(\bigcup_{j \in J} \mathscr{S}_j\right)$.

Theorem 2.11. (Soundness of respectful functions) If $\mathscr{F}$ is respectful, then $\mathscr{F}$ is sound.

Proof. We have to show that if $\mathscr{F}$ is respectful and $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R})$ holds, then $\mathscr{R} \subseteq \sim$. Consider the following inductively-defined sequence of relations $\{\mathscr{R}_n \ : \ n \geq 0\}$:

$$\begin{aligned} \mathscr{R}_0 &\overset{\text{def}}{=} \mathscr{R}, \\ \mathscr{R}_{n+1} &\overset{\text{def}}{=} \mathscr{F}(\mathscr{R}_n) \cup \mathscr{R}_n . \end{aligned}$$

Fact: For all $n \geq 0$,

1 $\mathscr{R}_n \subseteq \mathscr{R}_{n+1}$;

2 $\mathscr{R}_n \rightarrowtail \mathscr{R}_{n+1}$.

Proof of the fact: (1) is by definition of $\mathscr{R}_{n+1}$. For (2), we proceed by induction on n. If $n = 0$, then $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R}) \cup \mathscr{R}$ follows from the hypothesis $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R})$ and Lemma 2.9(1). Suppose $n > 0$. By definition of $\mathscr{R}_n$ and $\mathscr{R}_{n+1}$, we have to show that

$$\left(\mathscr{F}(\mathscr{R}_{n-1}) \cup \mathscr{R}_{n-1}\right) \rightarrowtail \left(\mathscr{F}(\mathscr{R}_n) \cup \mathscr{R}_n\right). \tag{1}$$

Since $\mathscr{R}_{n-1} \subseteq \mathscr{R}_n$ and, by induction, $\mathscr{R}_{n-1} \rightarrowtail \mathscr{R}_n$, from the respectfulness of $\mathscr{F}$ we infer that $\mathscr{F}(\mathscr{R}_{n-1}) \rightarrowtail \mathscr{F}(\mathscr{R}_n)$. By Corollary 2.10, this and $\mathscr{R}_{n-1} \rightarrowtail \mathscr{R}_n$ prove (1).

We can now conclude the proof of the theorem. Since for all n, $\mathscr{R}_n \rightarrowtail \mathscr{R}_{n+1}$, by Lemma 2.8, $\bigcup_n \mathscr{R}_n$ is a bisimulation relation and hence is contained in $\sim$. This is enough because $\mathscr{R}$ is contained in $\bigcup_n \mathscr{R}_n$. $\square$

Remark 2.12. The proof of Theorem 2.11 carries over also with a weaker definition of respectfulness, namely

'whenever $\mathscr{R} \subseteq \mathscr{S}$ and $\mathscr{R} \rightarrowtail \mathscr{S}$ hold, then $\mathscr{F}(\mathscr{R}) \rightarrowtail \mathscr{F}(\mathscr{S})$ holds too'.

However, in this way we would lose some important properties of the class of respectful functions, for instance their closure under composition (Lemma 2.14).

Theorem 2.11 shows that a respectful first-order function yields a sound proof technique for bisimilarity. We can push further and look for ways of combining respectful functions in which respectfulness is preserved.

We call a function that takes first-order functions as arguments and yields back another first-order function as a result, a *second-order function* or, briefly, a *constructor*. A constructor is *respectful* if whenever its first-order function arguments are respectful, then also the first-order function result is respectful. This hierarchy of functions could be continued, by defining respectful third-order functions, respectful fourth-order functions and so on... . We stop at second order because it will be enough for our purposes.

We shall present a few primitive functions and constructors, and prove that they are respectful. They are rather simple, but give rise to interesting compounds, whose respectfulness—and hence soundness—comes then for free.

The following are two simple primitive respectful functions:

$$\begin{aligned}\mathscr{I}(\mathscr{R}) &\stackrel{\text{def}}{=} \mathscr{R}\\ \mathscr{U}(\mathscr{R}) &\stackrel{\text{def}}{=} \sim\end{aligned}$$

$\mathscr{I}$ is the identity function. $\mathscr{U}$ is the constant-to-$\sim$ function, mapping every relation onto the bisimilarity relation. Later we shall introduce two further primitive respectful functions. Roughly, one is a function that returns the closure of a relation under contexts (Section 2.1); the other is a function that allows us to manipulate a relation using injective substitutions on names (this will be introduced when dealing with the π-calculus, in Section 5).

The primitive constructors we consider are *composition* ($\circ$), *union* ($\cup$) and chaining ($\frown$), which are defined as follows:

$$\begin{aligned}(\mathscr{G}\circ\mathscr{F})(\mathscr{R}) &\stackrel{\text{def}}{=} \mathscr{G}(\mathscr{F}(\mathscr{R}))\\ (\bigcup_{i\in I}\mathscr{F}_i)(\mathscr{R}) &\stackrel{\text{def}}{=} \bigcup_{i\in I}(\mathscr{F}_i(\mathscr{R}))\\ (\mathscr{G}\frown\mathscr{F})(\mathscr{R}) &\stackrel{\text{def}}{=} \mathscr{G}(\mathscr{R})\;\mathscr{F}(\mathscr{R}) =\\ &\quad\{(P,P') \;:\; \text{for some } P'',(P,P'')\in\mathscr{G}(\mathscr{R}) \text{ and } (P'',P')\in\mathscr{F}(\mathscr{R})\}.\end{aligned}$$

(Note that, formally, for arity reasons, there is a different union operator for all $n \in \{0,1,\ldots,\omega\}$.) Before proving the respectfulness of these primitive functions and constructors, let us see what we can derive from combinations of them. Examples of derived functions are

$$\begin{aligned}\text{for } n>0,\quad \mathscr{D}_n &\stackrel{\text{def}}{=} \mathscr{I}\frown\ldots\frown\mathscr{I}, \quad n \text{ times}\\ \mathscr{B} &\stackrel{\text{def}}{=} \mathscr{U}\frown\mathscr{I}\frown\mathscr{U}\\ \mathscr{T} &\stackrel{\text{def}}{=} \bigcup_{n>0}\mathscr{D}_n\,.\end{aligned}$$

Function $\mathscr{D}_n$ takes a function $\mathscr{R}$ and makes the composition of $\mathscr{R}$ with itself n times. Function $\mathscr{B}$ represents the classical *bisimulation up-to* $\sim$, as in Milner's book (Milner 1989) (where the proof of the soundness of $\mathscr{B}$ is by checking that $\mathscr{R}\subseteq\mathscr{B}(\mathscr{R})$ and that $\mathscr{B}(\mathscr{R})$ is a bisimulation relation). Function $\mathscr{T}$ returns the transitive closure of a relation. The plain definitions of these functions are

$$\begin{aligned}\mathscr{D}_n(\mathscr{R}) &\stackrel{\text{def}}{=} \{(P,P') \;:\; \text{for some } P_1,\ldots,P_{n+1} \text{ with } P=P_1 \text{ and } P_{n+1}=P',\\ &\qquad\qquad\qquad \text{it holds that } P_i\mathscr{R}P_{i+1} \text{ for all } 1\le i\le n\}\\ \mathscr{B}(\mathscr{R}) &\stackrel{\text{def}}{=} \sim\mathscr{R}\sim\\ \mathscr{T}(\mathscr{R}) &\stackrel{\text{def}}{=} \{(P,P') \;:\; \text{for some } n>0 \text{ and processes } P_1,\ldots,P_{n+1}\\ &\qquad\qquad\qquad \text{with } P=P_1 \text{ and } P'=P_{n+1}\\ &\qquad\qquad\qquad \text{it holds that } P_i\;\mathscr{R}\;P_{i+1} \text{ for all } 1\le i\le n\}.\end{aligned}$$

Examples of derived constructors are exponentiation and iteration, defined using composition and union as follows:

$$\begin{aligned}\mathscr{F}^n(\mathscr{R}) &\stackrel{\text{def}}{=} \mathscr{F}((\ldots(\mathscr{F}(\mathscr{R}))\ldots)), \quad n \text{ times}\\ \mathscr{F}^*(\mathscr{R}) &\stackrel{\text{def}}{=} \bigcup_n\mathscr{F}^n(\mathscr{R})\,.\end{aligned}$$

We now come to the proof of the respectfulness of the primitive functions and constructors introduced above.

Lemma 2.13. (Identity and constant-to-$\sim$ functions) The identity function $\mathscr{I}$ and the constant-to-$\sim$ function $\mathscr{U}$ are respectful.

Lemma 2.14. (Composition, union, chaining) Composition, union and chaining are respectful constructors.

Proof. We only show the proof for chaining. Suppose $\mathscr{F}$ and $\mathscr{G}$ are respectful. We check that $\mathscr{G}{}^\frown\mathscr{F}$ is respectful also. Suppose $\mathscr{R} \subseteq \mathscr{S}$ and $\mathscr{R} \rightarrowtail \mathscr{S}$. Then $\mathscr{F}(\mathscr{R}) \subseteq \mathscr{F}(\mathscr{S})$ and $\mathscr{G}(\mathscr{R}) \subseteq \mathscr{G}(\mathscr{S})$, which gives $(\mathscr{G}{}^\frown\mathscr{F})(\mathscr{R}) \subseteq (\mathscr{G}{}^\frown\mathscr{F})(\mathscr{S})$. We also have to check that $(\mathscr{G}{}^\frown\mathscr{F})(\mathscr{R}) \rightarrowtail (\mathscr{G}{}^\frown\mathscr{F})(\mathscr{S})$. Take $(P,P') \in (\mathscr{G}{}^\frown\mathscr{F})(\mathscr{R})$ with $P \xrightarrow{\mu} P_1$. Since $(P,P') \in (\mathscr{G}{}^\frown\mathscr{F})(\mathscr{R})$, there is P'' such that $(P,P'') \in \mathscr{G}(\mathscr{R})$ and $(P'',P') \in \mathscr{F}(\mathscr{S})$. Moreover, since by respectfulness of $\mathscr{G}$ and $\mathscr{F}$ it holds that $\mathscr{G}(\mathscr{R}) \rightarrowtail \mathscr{G}(\mathscr{S})$ and $\mathscr{F}(\mathscr{R}) \rightarrowtail \mathscr{F}(\mathscr{S})$, for some P_1'' and P_1' the following diagram commutes:

$$\begin{tikzcd}
P \arrow[d, "\mu"'] \arrow[r, phantom, "\mathscr{G}(\mathscr{R})"] & P'' \arrow[d, "\mu"'] \arrow[r, phantom, "\mathscr{F}(\mathscr{R})"] & P' \arrow[d, "\mu"'] \\
P_1 \arrow[r, phantom, "\mathscr{G}(\mathscr{S})"] & P_1'' \arrow[r, phantom, "\mathscr{F}(\mathscr{S})"] & P_1'
\end{tikzcd}$$

This shows that $(P_1,P_1') \in (\mathscr{G}{}^\frown\mathscr{F})(\mathscr{S})$. In a symmetric way, one can show that if $P' \xrightarrow{\mu} P_1'$, then there is P_1 such that $P \xrightarrow{\mu} P_1$ and $(P_1,P_1') \in (\mathscr{G}{}^\frown\mathscr{F})(\mathscr{S})$. We conclude that $(\mathscr{G}{}^\frown\mathscr{F})(\mathscr{R}) \rightarrowtail (\mathscr{G}{}^\frown\mathscr{F})(\mathscr{S})$. $\square$

We saw that functions $\mathscr{B}$, $\mathscr{D}_n$ and $\mathscr{T}$, and constructors $\mathscr{F}^n$ and $\mathscr{F}^*$ are definable in terms of the primitive functions $\mathscr{I}$ and $\mathscr{U}$, and of the primitive constructors composition, chaining and union. Therefore, as a consequence of Lemmas 2.13–2.14, these derived functions and constructors are respectful.

2.1. *Closure of a relation under contexts*

We now consider the case (which is standard in process algebras) in which the class of processes is defined as the term algebra generated by some signature.

We work with one-sorted signatures Σ. We call the (possibly infinite) set of symbols in Σ the *operators of the language*. Each operator has a fixed arity $n \geq 0$. If the arity of the operator is 0, we call it a *constant operator*, if it is $n > 0$ we call it a *functional* operator. The *term algebra over* signature Σ, written Pr_Σ, is the least set of strings that satisfy:

— If f is an operator in Σ with arity 0, then f is in Pr_Σ.

— If f is an operator in Σ with arity $n > 0$, and $t_1,\dots,t_n$ are already in Pr_Σ, then $f(t_1,\dots,t_n)$ is in Pr_Σ.

Thus, having a signature Σ, the process language is Pr_Σ and a process is an element of Pr_Σ.

We shall also be interested in extensions of a signature Σ with constant operators. If $\mathscr{X}$ is a set of symbols not in Σ, then $\Sigma(\mathscr{X})$ is the signature that has all operators in Σ as before, and in addition each symbol in $\mathscr{X}$ is an operator in $\Sigma(\mathscr{X})$ with arity 0. We write $Pr_\Sigma(\mathscr{X})$ for the term algebra over $\Sigma(\mathscr{X})$.

2.1.1. *Closure under faithful contexts* Let Σ be a signature and $[\cdot]$ be a symbol not in Σ, called *hole*. A Σ-*context* is an element of $Pr_\Sigma([\cdot])$ with at most one occurrence of the hole $[\cdot]$ in it. We use C to range over Σ-contexts. If C is a Σ-context and $P \in Pr_\Sigma$, then $C[P] \in Pr_\Sigma$ is the process obtained from C by filling the hole $[\cdot]$ with P. We utilise contexts to define a function $\mathscr{C}_\Sigma$ on process relations that makes the closure of a relation $\mathscr{R}$ under a certain class of contexts. Function $\mathscr{C}_\Sigma$ will be one of our most useful primitive respectful functions.

$$\mathscr{C}_\Sigma(\mathscr{R}) \stackrel{\text{def}}{=} \bigcup_{C \text{ faithful}} \{(C[P], C[Q]) \ : \ (P, Q) \in \mathscr{R}\}. \tag{2}$$

Before saying what a faithful context is, note that in the definition of $\mathscr{C}_\Sigma$ the contexts used may have at most one occurrence of a unique hole $[\cdot]$. More sophisticated closures, involving contexts that may contain different holes, and each of them an arbitrary number of times, can be recovered as a combination of function $\mathscr{C}_\Sigma$ and other respectful functions of the previous section (see Lemma 3.2). Choosing a simple function $\mathscr{C}_\Sigma$ makes the proof of its soundness simple too.

Definition 2.15. A set Cont of Σ-contexts is a *faithful context-set* if for all $C \in$ Cont and $P \in Pr_\Sigma$, whenever $C[P] \xrightarrow{\mu} R$, there exist $C' \in$ Cont such that either

(a) $R = C'[P]$ and, for all Q, it holds that $C[Q] \xrightarrow{\mu} C'[Q]$, or

(b) there are $P' \in Pr_\Sigma$ and $\lambda \in Act$ such that $P \xrightarrow{\lambda} P'$ and $R = C'[P']$ and, moreover, for all $Q, Q' \in Pr_\Sigma$ such that $Q \xrightarrow{\lambda} Q'$, it holds that $C[Q] \xrightarrow{\mu} C'[Q']$.

A Σ-context C is *faithful* if $C \in$ Cont, for some faithful context-set Cont.

Remark 2.16. The use of Definition 2.15 is facilitated if Clauses (a) and (b) are merged. Thus, if $P \xrightarrow{\widehat{\lambda}} Q$ means '$P = Q$ or $P \xrightarrow{\lambda} Q$', then (a) and (b) can be rewritten as follows:

— there are $P' \in Pr_\Sigma$ and $\widehat{\lambda}$ such that $P \xrightarrow{\widehat{\lambda}} P'$ and $R = C'[P']$ and, moreover, for all $Q, Q' \in Pr_\Sigma$ such that $Q \xrightarrow{\widehat{\lambda}} Q'$ it holds that $C[Q] \xrightarrow{\mu} C'[Q']$.

The class of faithful contexts is usually very large. In familiar process algebras, such as ACP and CCS, all contexts are faithful (we shall prove this for CCS in Section 3.2). Indeed, faithful contexts correspond to Larsen and Liu's *1-to-1 contexts* (Larsen and Liu 1991) (1-to-1 meaning that these contexts have exactly one hole and that they produce one action at a time).

Lemma 2.17. (Closure under contexts) The function $\mathscr{C}$ is respectful.

Proof. Suppose $\mathscr{R} \subseteq \mathscr{S}$ and $\mathscr{R} \rightarrowtail \mathscr{S}$. Clearly, $\mathscr{C}(\mathscr{R}) \subseteq \mathscr{C}(\mathscr{S})$ also. Thus, we only have to prove $\mathscr{C}(\mathscr{R}) \rightarrowtail \mathscr{C}(\mathscr{S})$. For this, we have to show that if $P \mathrel{\mathscr{R}} Q$ holds, C is a faithful context and $C[P] \xrightarrow{\mu} P''$, then there are P', Q' and a faithful context C' such that $P'' = C'[P']$, $Q \xrightarrow{\mu} C'[Q']$ and $P' \mathscr{S} Q'$. By definition of faithfulness, if $C[P] \xrightarrow{\mu} P''$, then for some process P', faithful context C' and (possibly empty) action $\widehat{\lambda}$, we have $P \xrightarrow{\widehat{\lambda}} P'$

and $P'' = C'[P']$. Since $\mathscr{R} \rightarrowtail \mathscr{S}$ and $\mathscr{R} \subseteq \mathscr{S}$, for some Q' the diagram

$$\begin{tikzcd} P \arrow[d, "\widehat{\lambda}"'] \arrow[r, phantom, "\mathscr{R}"] & Q \arrow[d, "\widehat{\lambda}"'] \\ P' \arrow[r, phantom, "\mathscr{S}"] & Q' \end{tikzcd}$$

commutes. (Note that the hypothesis $\mathscr{R} \subseteq \mathscr{S}$ is needed for the case in which $\widehat{\lambda}$ is empty, when $P' = P$ and $Q' = Q$). Again by definition of faithfulness, we have $C[Q] \xrightarrow{\mu} C'[Q']$. This proves that the diagram

$$\begin{tikzcd} C[P] \arrow[d, "\mu"'] \arrow[r, phantom, "\mathscr{C}(\mathscr{R})"] & C[Q] \arrow[d, "\mu"'] \\ C'[P'] \arrow[r, phantom, "\mathscr{C}(\mathscr{S})"] & C'[Q'] \end{tikzcd}$$

commutes, and concludes the proof. $\square$

2.1.2. *The De Simone format for the transition rules* The transition relation for the processes of the language generated by a signature Σ can be defined structurally (Plotkin 1981), assigning a set of *transition rules* to each symbol in Σ. In some cases, it suffices to look at the format of such transition rules to know that the contexts of the language are faithful. We show that this is indeed the case for the rules in *unary De Simone format over* Σ, which we will often just call *De Simone format*. It is a simplified version of the format introduced by De Simone (De Simone 1985) (the main restriction is that only one action at a time is observable). In Rule (3) below, X_r, $1 \leq r \leq n$, and Y_j, $j \in J$, are metavariables, which are instantiated with processes when the rule is applied.

Definition 2.18. (Unary De Simone format) A transition rule

$$\frac{X_j \xrightarrow{\lambda_j} Y_j \ (j \in J)}{f(X_1, \ldots, X_n) \xrightarrow{\mu} t} \tag{3}$$

is in *unary De Simone format over* Σ if:

— n is the arity of f in Σ.
— $J \subseteq \{1, \ldots, n\}$.
— X_r, $1 \leq r \leq n$, and Y_j, $j \in J$, are distinct variables.
— t is a term in $Pr_\Sigma(X'_1, \ldots, X'_n)$, where for all $1 \leq r \leq n$, each X'_r occurs at most once in t, and $X'_r = Y_r$ if $r \in J$, $X'_r = X_r$ otherwise.

We show that all contexts of a language whose functional operators have transition rules in De Simone format are faithful. Actually, we shall be a little more general, and first consider the case in which only a *subset* of the functional operators have transition rules in De Simone format; in this case we can prove the faithfulness of only a subset of the contexts.

Definition 2.19. ((Σ, Σ')-contexts) Take signatures Σ and Σ' with $\Sigma' \subseteq \Sigma$. Suppose the meaning of each symbol in Σ' is given using a set of transition rules in unary De Simone format over Σ'. Then we say that a Σ-context C is a *(Σ, Σ')-context* if

1 $C \in Pr_\Sigma$ (that is, C is a process), or
2 $C = [\cdot]$, or
3 $C = f(P_1, \dots, P_{i-1}, C', P_{i+1}, \dots, P_n)$, where
 — $f \in \Sigma'$,
 — n is the arity of f,
 — $1 \leq i \leq n$,
 — $P_r \in Pr_\Sigma$ for $r \in \{1, \dots, n\} - \{i\}$,
 — C' is a (Σ, Σ')-context.

The above inductive definition first asserts that all functional operators in Σ' have transition rules in unary De Simone format over Σ' (that is, are definable within Σ'); then a Σ-context C is a (Σ, Σ')-context if all functional symbols above the hole of C are in Σ'.

Proposition 2.20. For any Σ and Σ', all (Σ, Σ')-contexts are faithful.

Proof. We show that the class of (Σ, Σ')-contexts is a faithful context-set. We consider a context C in such a class and verify the requirement in Definition 2.15 proceeding by induction on the structure of C. The basic case, when $C \in Pr_\Sigma$ or $\Sigma = [\cdot]$, is trivial.

In the inductive case, we have $C = f(R_1, \dots, R_{i-1}, C', R_{i+1}, \dots, R_n)$, for $f \in \Sigma'$ and $C[P] = f(R_1, \dots, R_{i-1}, C'[P], R_{i+1}, \dots, R_n)$. The last step of the derivation of $C[P] \xrightarrow{\mu} R$ uses a rule in unary De Simone format, like Rule (3). Supposing i is in the set J named in Rule (3) (the case where it is not is simpler), we can write this last step thus:

$$\frac{R_j \xrightarrow{\mu_j} T_j \ (j \in J - \{i\}), \qquad C'[P] \xrightarrow{\mu'} R'}{f(R_1, \dots, R_{i-1}, C'[P], R_{i+1}, \dots, R_n) \xrightarrow{\mu} R = C''[R']} \tag{4}$$

Context C'' is a (Σ, Σ')-context: since $f \in \Sigma'$, by definition of (Σ, Σ')-context, each transition rule for f is in De Simone format over Σ', and hence all functional operators above the hole of C'' are in Σ'.

By induction, from $C'[P] \xrightarrow{\mu'} R'$ we infer that there is $\widehat{\lambda}$, P' and a (Σ, Σ')-context D' such that

$$P \xrightarrow{\widehat{\lambda}} P' \text{ and } R' = D'[P'] \,, \tag{5}$$

and, moreover, for all $Q, Q' \in Pr_\Sigma$ with $Q \xrightarrow{\widehat{\lambda}} Q'$, we also have

$$C'[Q] \xrightarrow{\mu'} D'[Q'] \,.$$

From (4) and (5), we get that $R = C''[D'[P']] = D[P']$, for some (Σ, Σ')-context D. Moreover, from (4), but with $C'[Q] \xrightarrow{\mu'} D'[Q']$ in place of $C'[P] \xrightarrow{\mu'} R'$, we infer

$$f(R_1, \dots, R_{i-1}, C'[Q], R_{i+1}, \dots, R_n) \xrightarrow{\mu} C''[D'[Q']] = D[Q'] \,.$$

Summarising, we have found that if $C[P] \xrightarrow{\mu} R$, then there are P', $\widehat{\lambda}$ and a (Σ, Σ')-context D such that $P \xrightarrow{\widehat{\lambda}} P'$, $R = D[P']$ and for all $Q, Q' \in Pr_\Sigma$ with $Q \xrightarrow{\widehat{\lambda}} Q'$, also $C[Q] \xrightarrow{\mu} D[Q']$. This concludes the proof. □

Corollary 2.21. Consider the process language over a signature Σ in which the meaning of all functional symbols in Σ is given using a set of rules in unary De Simone format over Σ. Then all Σ-contexts are faithful.

Proof. With the hypothesis in the corollary, the (Σ, Σ')-contexts are precisely the Σ-contexts. Then the result follows from Proposition 2.20. □

Corollary 2.21 applies to well-know process algebras like CCS (see Lemma 3.1) and ACP. The De Simone format excludes, for instance, operators that, in order to release some action, may require the release of a *sequence* of actions – as opposed to *one* action – from some of their arguments (*i.e.*, using the terminology in Groote and Vaandrager (1992), these operators have lookahead greater than one), or operators defined with rules with negative premises, where the requirement on some of the arguments is that they *cannot* perform certain actions (Bloom *et al.* 1995; Groote 1990). Also, the format does not capture value-passing process algebras, where actions have more structure – they can also carry values. A special case of value-passing process algebra, namely the π-calculus, which supports communication of names, will be examined in Sections 4–6.

In the remainder of this paper, to simplify the notation, we omit the indication of the signature. We assume that there is a given signature Σ, and that all contexts and processes, as well as quantification over them, are, or refer to, contexts and processes in Σ. Thus, we shall call a Σ-context simply a context, and we shall abbreviate function $\mathscr{C}_\Sigma$ in (2) as $\mathscr{C}$. Also, we shall abbreviate $\mathscr{C}(\mathscr{R})$ as $\mathscr{R}^{\mathscr{C}}$ and $\mathscr{T}(\mathscr{R})$ as $\mathscr{R}^{\mathscr{T}}$ (that is, $\mathscr{R}^{\mathscr{C}}$ is the closure of $\mathscr{R}$ under faithful contexts and $\mathscr{R}^{\mathscr{T}}$ is the transitive closure of $\mathscr{R}$). In applications of our proof techniques, we shall often employ the sound function $\sim (-^{\mathscr{C}})^{\mathscr{T}} \sim$, which maps a relation $\mathscr{R}$ onto the relation $\sim (\mathscr{R}^{\mathscr{C}})^{\mathscr{T}} \sim$.

2.1.3. *Beyond faithfulness* Function $\mathscr{C}$ yields the closure with respect to the *faithful* contexts. You might reasonably think that the key property that makes $\mathscr{C}$ respectful is that faithful contexts preserve bisimilarity, and you may therefore wonder whether $\mathscr{C}$ could be strengthened to allow the closure under *all contexts that preserve bisimilarity*. Let us call $\mathscr{C}^\star$ this variant of $\mathscr{C}$. We show in this subsection that $\mathscr{C}^\star$ is not respectful.

Consider the simple process language

$$P \ := \ f(P) \ \big| \ a.P \ \big| \ \mathbf{0}$$

where $a.-$ is a CCS-like prefix, $\mathbf{0}$ is the inactive process and f is an operator whose behaviour is given by the rule

$$\frac{X \xrightarrow{a} X' \quad X' \xrightarrow{a} X''}{f(X) \xrightarrow{a} X''}.$$

Since the transition rules of the operators are in *tyft* format, all contexts of the language preserve bisimilarity (Groote and Vaandrager 1992). Note, in particular, that the transition rule for f uses a lookahead greater than one. Such lookaheads are allowed in the *tyft* format but are not in the De Simone format. We can show that, on this language, $\mathscr{C}^\star$ is not respectful. Take

$$\mathscr{R} \overset{\text{def}}{=} \{(a.\mathbf{0}, a.a.\mathbf{0})\}.$$

Processes $a.\mathbf{0}$ and $a.a.\mathbf{0}$ are not bisimilar. But the diagram

$$\begin{tikzcd}
a.\mathbf{0} \arrow[d, "a"'] & & & & & & a.a.\mathbf{0} \arrow[d, "a"'] \\
\mathbf{0} & \sim & f(a.\mathbf{0}) & \mathscr{C}^\star(\mathscr{R}) & f(a.a.\mathbf{0}) & \sim & a.\mathbf{0}
\end{tikzcd}$$

shows that $\mathscr{R} \rightarrowtail \sim \mathscr{C}^\star(\mathscr{R}) \sim$ holds: hence $\mathscr{C}^\star$ is not respectful since, otherwise, function $\sim(\mathscr{C}^\star(-))\sim$ would be so also, and we should have $\mathscr{R} \subseteq \sim$.

The counterexample above still does not show that $\mathscr{C}^\star$ itself is not sound. However, it does show that even if $\mathscr{C}^\star$ were sound its interest would be rather limited because it could not be combined with very simple functions such as the constant-to-$\sim$ function.

3. CCS: Operational semantics and proof techniques

We begin this section with a brief synopsis of its contents. We review the syntax and the operational semantics of CCS. A quick inspection of the transition rules of the CCS operators shows that all proof techniques for bisimilarity introduced in the previous section can be applied to CCS processes. We use the techniques to derive a proof, simpler than the one in Milner (1989), of a standard result of the calculus, namely the uniqueness of solutions of equations.

3.1. *The calculus*

We assume an infinite set *Names* $= \{a, b, \ldots, x, y, \ldots\}$ of *names* and a set of constant identifiers *Constants* ranged over by A. The special symbol τ does not occur in *Names* and in *Constants*. The class of the CCS processes is built from the operators of input prefix, output prefix, silent prefix, parallel composition, sum, restriction, inaction, and constants:

$$\begin{array}{lcl}
P & := & \alpha . P \;\Big|\; P_1 \mid P_2 \;\Big|\; P_1 + P_2 \;\Big|\; \boldsymbol{\nu}\, a\, P \;\Big|\; \mathbf{0} \;\Big|\; A \\
\alpha & := & a \;\Big|\; \overline{a} \;\Big|\; \tau\,.
\end{array}$$

Following π-calculus syntax (Section 4), we use $\boldsymbol{\nu}$ for restriction ($\boldsymbol{\nu}\, a\, P$ is normally written $P \setminus a$ in CCS), and we omit the relabelling operator (which, anyhow, would not bring complications into the theory we shall present). Moreover, for notational convenience, we limit ourselves to finite restrictions and finite sums. We suppose that for each constant A there is a defining equation of the form $A \overset{\text{def}}{=} P$. We refer the reader to Milner (1989) for details of the operators of the calculus. Sometimes we use $\overset{\text{def}}{=}$ as an abbreviation mechanism to assign a name to expressions or relations to which we want to refer later. In this section, P, Q, and R are CCS processes, and *Pr* is the class of all CCS processes.

The transition system describing the operational semantics of CCS process is shown in Table 1; we have omitted the symmetric form of the rules for parallel composition and summation. In a transition $P \xrightarrow{\mu} Q$, the label μ can be an input a, an output $\overline{a}$ or a silent move τ. We use α to range over prefixes and μ over actions. We distinguish between prefixes and actions in analogy with the π-calculus, in which the alphabets for prefixes and actions are different.

$$\begin{array}{llll}
\mathsf{pre:} & \alpha.P \xrightarrow{\alpha} P & \mathsf{sum:} & \dfrac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'} \\[2ex]
\mathsf{par:} & \dfrac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} & \mathsf{com:} & \dfrac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \\[2ex]
\mathsf{res:} & \dfrac{P \xrightarrow{\mu} P'}{\boldsymbol{\nu}\, a\, P \xrightarrow{\mu} \boldsymbol{\nu}\, a\, P'}\ \mu \neq a, \overline{a} & \mathsf{const:} & \dfrac{P \xrightarrow{\mu} P'}{A \xrightarrow{\mu} P'}\ \text{if } A \overset{\text{def}}{=} P
\end{array}$$

Table 1. *The transition system for CCS*

3.2. *Our proof techniques in CCS*

The operational semantics of CCS uses a standard labelled transition system. Hence, to apply the whole theory of proof techniques for bisimilarity developed in Section 2 to CCS, we only have to understand which contexts are faithful; these are needed in the definition of function $\mathscr{C}$ (closure under contexts).

Lemma 3.1. All CCS contexts are faithful.

Proof. The CCS language can be described with the signature

$$\Sigma \overset{\text{def}}{=} \{a., \overline{a}., \tau., \mid, \boldsymbol{\nu}\ , +, A \ : \ a \in \mathit{Names}, \text{ and } A \in \mathit{Constants}\}$$

whose symbols have the obvious meaning and the obvious arities. All functional operators in Σ, namely $\{a., \overline{a}., \tau., \mid, \boldsymbol{\nu}\ , +\}$ are defined by transition rules in De Simone format. By Corollary 2.21, all CCS contexts are faithful. $\square$

Therefore, the definition of function $\mathscr{C}$ in CCS becomes

$$\mathscr{C}(\mathscr{R}) \overset{\text{def}}{=} \bigcup_{C} \{(C[P], C[Q]) \ : \ (P, Q) \in \mathscr{R}\}.$$

Lemmas 3.1, 2.17 and Theorem 2.11 ensure the soundness of $\mathscr{C}$.

3.3. *An application: The proof of the uniqueness of solutions of equations*

An interesting example of the application of our proof techniques to CCS is the proof of the uniqueness of solutions of equations, as from Milner's book (Milner 1989). This result says that if a context C obeys certain conditions, then all processes P that satisfy the equation $P \sim C[P]$ are bisimilar with each other.

We use a tilde to denote a finite (and possibly empty) tuple. All notations we introduce are generalised to tuples componentwise: thus, $\widetilde{P} \mathrel{\mathscr{R}} \widetilde{Q}$ means that $P_i \mathrel{\mathscr{R}} Q_i$, for each component of the vectors $\widetilde{P}$ and $\widetilde{Q}$. For notational convenience, in this section we work with polyadic contexts, that is, contexts that may contain an arbitrary number of different holes $[\cdot]_1, \dots, [\cdot]_n$, and, moreover, each of these holes may appear more than once. If C contains at most holes $[\cdot]_1, \dots, [\cdot]_n$, then we say that C is an *n-ary* context. Moreover, if $\widetilde{P}$ is a vector of n processes, then $C[\widetilde{P}]$ is the process obtained by replacing each occurrence of the hole $[\cdot]_i$ with the i-th component of $\widetilde{P}$.

In Sections 2 and 3.2 we only considered the closure of a relation under monadic contexts, *i.e.*, contexts containing at most one hole; this closure was given by function $\mathscr{C}$.

We can recover the closure of a relation under polyadic contexts as the transitive closure of the closure under the monadic ones.

Lemma 3.2. If $(P_i, Q_i) \in \mathscr{R}$, $i \le i \le n$, and C is an n-ary context, then

$$(C[P_1,\ldots,P_n], C[Q_1,\ldots,Q_n]) \in (\mathscr{R}^{\mathscr{C}})^{\mathscr{T}}.$$

Proof. Let $\widetilde{P} \overset{\text{def}}{=} P_1,\ldots,P_n$ and $\widetilde{Q} \overset{\text{def}}{=} Q_1,\ldots,Q_n$. We have to show that $C[\widetilde{P}]$ and $C[\widetilde{Q}]$ are in the transitive closure of $\mathscr{R}^{\mathscr{C}}$. We proceed by induction on the structure of C. All cases are simple – we will only look at parallel composition. Suppose $C = C_1 \mid C_2$. By induction,

$$\left(C_1[\widetilde{P}], C_1[\widetilde{Q}]\right) \in \left(\mathscr{R}^{\mathscr{C}}\right)^{\mathscr{T}} \quad \text{and} \quad \left(C_2[\widetilde{P}], C_2[\widetilde{Q}]\right) \in \left(\mathscr{R}^{\mathscr{C}}\right)^{\mathscr{T}}.$$

Hence also

$$\left(C_1[\widetilde{P}] \mid C_2[\widetilde{P}], C_1[\widetilde{Q}] \mid C_2[\widetilde{P}]\right) \in \left(\mathscr{R}^{\mathscr{C}}\right)^{\mathscr{T}} \quad \text{and} \quad \left(C_1[\widetilde{Q}] \mid C_2[\widetilde{P}], C_1[\widetilde{Q}] \mid C_2[\widetilde{Q}]\right) \in \left(\mathscr{R}^{\mathscr{C}}\right)^{\mathscr{T}}.$$

Since $\left(\mathscr{R}^{\mathscr{C}}\right)^{\mathscr{T}}$ is transitive, we infer $\left(C_1[\widetilde{P}] \mid C_2[\widetilde{P}], C_1[\widetilde{Q}] \mid C_2[\widetilde{Q}]\right) \in \left(\mathscr{R}^{\mathscr{C}}\right)^{\mathscr{T}}$. □

We say that a context C is *weakly guarded* if each occurrence of each hole of C is within some subexpression of the form $\alpha.C'$. For instance, $\alpha.[\cdot]$ is weakly guarded, but $[\cdot] \mid \alpha.[\cdot]$ is not.

Lemma 3.3. (Milner 1989, Lemma 4.13) If C is weakly guarded and $C[\widetilde{P}] \xrightarrow{\mu} P'$, then P' is of the form $C'[\widetilde{P}]$, and, moreover, for any $\widetilde{Q}$, $C[\widetilde{Q}] \xrightarrow{\mu} C'[\widetilde{Q}]$.

Proof. The proof is by simple induction on the structure of C. Intuitively, since C is weakly guarded, the processes that fill the holes of C do not contribute to the first action produced. □

We write $\widetilde{C}$ for a tuple of contexts $C_1,\ldots,C_n$; then $\widetilde{C}[\widetilde{P}]$ is $C_1[\widetilde{P}],\ldots,C_n[\widetilde{P}]$.

Proposition 3.4. (Unique solution of equations (Milner 1989, Proposition 4.14(2))) Suppose $\widetilde{C}$ are weakly guarded contexts, with $\widetilde{P} \sim \widetilde{C}[\widetilde{P}]$ and $\widetilde{Q} \sim \widetilde{C}[\widetilde{Q}]$. Then $\widetilde{P} \sim \widetilde{Q}$.

Proof. Let n be the length of vectors $\widetilde{C}$, $\widetilde{P}$ and $\widetilde{Q}$, and take

$$\mathscr{R} \overset{\text{def}}{=} \{(P_i, Q_i) \ : \ 1 \le i \le n\},$$

and suppose $P_i \xrightarrow{\mu} P_i'$ (the case of a move from Q_i is symmetric). From Lemma 3.3 we deduce that there are C_i' and Q_i' such that the following two diagrams commute:

$$\begin{tikzcd} P_i \arrow[r, phantom, "\sim"] \arrow[d, "\mu"'] & C_i[\widetilde{P}] \arrow[d, "\mu"'] \\ P_i' \arrow[r, phantom, "\sim"] & C_i'[\widetilde{P}] \end{tikzcd} \qquad \begin{tikzcd} C_i[\widetilde{Q}] \arrow[r, phantom, "\sim"] \arrow[d, "\mu"'] & Q_i \arrow[d, "\mu"'] \\ C_i'[\widetilde{Q}] \arrow[r, phantom, "\sim"] & Q_i' \end{tikzcd}$$

By Lemma 3.2, this shows that $\mathscr{R} \rightarrowtail \sim(\mathscr{R}^{\mathscr{C}})^{\mathscr{T}}\sim$ holds. Since function $\sim(-^{\mathscr{C}})^{\mathscr{T}}\sim$ is sound, we infer $\mathscr{R} \subseteq \sim$, which proves the proposition. □

In the proof of Proposition 3.4, the cardinality of the relation $\mathscr{R}$ is the same as the cardinality of the vector of given contexts $\widetilde{C}$. In particular, if we are dealing with only one context (*i.e.*, only one equation), $\mathcal{R}$ consists of only *one* pair. For the proof of Proposition 3.4, Milner (Milner 1989) shows that

$$\mathcal{R}' \stackrel{\text{def}}{=} \bigcup_{C} \{(C[\widetilde{P}], C[\widetilde{Q}])\}$$

is a bisimulation up-to $\sim$ (that is, $\mathcal{R}' \rightarrowtail \sim\mathcal{R}'\sim$ holds), proceeding by induction on the structure of C. Note that in $\mathcal{R}'$ the contexts in the union are *all* contexts – including the unguarded ones.

4. The π-calculus

The π-calculus is an extension of CCS where names are exchanged as a result of a communication. This allows us to model systems with dynamic linkage reconfiguration and confers a remarkable expressiveness to the calculus, as testified, for instance, by various works on the encoding of λ-calculus, of higher-order calculi, of object-oriented languages and of non-interleaving behavioural equivalences (Milner 1991; Sangiorgi 1992; Sangiorgi 1996; Boreale and Sangiorgi 1995; Walker 1994).

We briefly review the syntax and the operational semantics of the π-calculus. We refer the reader to Milner *et al.* (1992) and Milner (1991) for more details. We maintain the notations introduced for CCS, which will not be repeated. With respect to CCS, π-calculus grammar differs in the prefixes, which now present an object part, and in the treatment of constants, which are now parametrised on a tuple of names. In addition, π-calculus grammar usually incorporates a matching construct to test for equality between names. There are two forms of output prefix: the *free output* $\overline{a}b.P$ and the *bound output* $\overline{a}(b).P$ (the latter is an abbreviation for $\boldsymbol{\nu}\, b\, \overline{a}b.P$). We admit bound outputs in the syntax of the calculus because of their important role in the operational semantic and in the algebraic theory.

$$\begin{array}{lcl} P & := & \alpha.P \;\big|\; P_1 \mid P_2 \;\big|\; P_1 + P_2 \;\big|\; \boldsymbol{\nu}\, a\, P \;\big|\; \mathbf{0} \;\big|\; A\langle\widetilde{b}\rangle \;\big|\; [a=b]P \\ \alpha & := & a(b) \;\big|\; \overline{a}b \;\big|\; \overline{a}(b) \;\big|\; \tau\,. \end{array}$$

Defining equations take the form $A \stackrel{\text{def}}{=} (\widetilde{c})P$, which can be thought of as a procedure with formal parameters $\widetilde{c}$; then $A\langle\widetilde{b}\rangle$ is like a procedure call with actual parameters $\widetilde{b}$. In the prefixes $a(b)$, $\overline{a}b$ and $\overline{a}(b)$ we call a the *subject*. The operators $a(b).P$, $\overline{a}(b).P$, $\boldsymbol{\nu}\, b\, P$ and $(\widetilde{b})P$ bind all free occurrences of the names b and $\widetilde{b}$ in P. We use $\mathrm{fn}(P)$ to denote the set of free names of P. For notational simplicity, we require that a process only has a finite number of free names and that in a constant definition $A \stackrel{\text{def}}{=} (\widetilde{c})P$, vector $\widetilde{c}$ contains all the free names of P. We suppose that it is always possible to alpha-convert bound names of an expression to 'fresh' ones. *We shall identify processes that only differ in the choice of the bound names.* The symbol $=$ will mean 'syntactic identity modulo alpha conversion'. A *substitution* is a function from names to names. We use the standard notation for substitutions, for example, $\{x/y\}$ is the function that sends y to x and is identity on all names but y. We use σ, ρ *etc.* to range over substitutions, and write $P\sigma$ for the agent obtained from P by replacing all free occurrences of any name x by $\sigma(x)$, with change

inp:	$a(c).P \xrightarrow{ab} P\{^{b}/_{c}\}$	pre:	$\alpha.P \xrightarrow{\alpha} P$, if α is not an input
sum:	$\dfrac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'}$	par:	$\dfrac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}$ if $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$
com:	$\dfrac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\overline{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$	close:	$\dfrac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\overline{a}(b)} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu}\, b\,(P' \mid Q')}$ if $b \notin \mathrm{fn}(P)$
res:	$\dfrac{P \xrightarrow{\mu} P'}{\boldsymbol{\nu}\, a\, P \xrightarrow{\mu} \boldsymbol{\nu}\, a\, P'}$ $a \notin \mathrm{n}(\mu)$	open:	$\dfrac{P \xrightarrow{\overline{a}b} P'}{\boldsymbol{\nu}\, b\, P \xrightarrow{\overline{a}(b)} P'}$ $a \neq b$
const:	$\dfrac{P\{^{\tilde{b}}/_{\tilde{c}}\} \xrightarrow{\mu} P'}{A\langle\tilde{b}\rangle \xrightarrow{\mu} P'}$ if $A \overset{\mathrm{def}}{=} (\tilde{c})P$	match:	$\dfrac{P \xrightarrow{\mu} P'}{[a=a]P \xrightarrow{\mu} P'}$

Table 2. *The transition system for the π-calculus*

of bound names if necessary to avoid captures. Similarly, $\alpha\sigma$ (or $\mu\sigma$) is the result of applying σ to the prefix α (or action μ), and does not affect a bound name in α (or μ), if any. Substitutions have precedence over the operators of the language. Also, $\sigma\rho$ is the composition of the two substitutions, in which σ is applied first; therefore $P\sigma\rho$ is $(P\sigma)\rho$.

The operational semantics of the calculus is defined by the transition rules of Table 2. The silent action $P \xrightarrow{\tau} Q$ has the same meaning as in CCS. An input action takes the form $P \xrightarrow{ab} Q$ and means 'P receives name b at a and evolves to Q'. Note that label ab does not have brackets around b, as in an input prefix $a(b)$: This is to show that in the input prefix name b is a binder (waiting to be instantiated), whereas in an input action b represents a value (with which an input binder has been instantiated). An output action can be either of the form $P \xrightarrow{\overline{a}b} Q$ or $P \xrightarrow{\overline{a}(b)} Q$; the latter means '$P$ sends the private (*i.e.*, 'fresh') name b at a'. Bound outputs are the central argument of transition rules open and close, the most original rules of the π-calculus with respect to CCS. All names in an action are free, except if the action is a bound output, say $\overline{a}(b)$, in which case a is free but b is bound. Bound and free names of an action μ (written $\mathrm{bn}(\mu)$ and $\mathrm{fn}(\mu)$, respectively) are defined accordingly. The *names* of μ (written $\mathrm{n}(\mu)$ for short) are $\mathrm{bn}(\mu) \cup \mathrm{fn}(\mu)$. We also work up to alpha conversion on processes in transition systems, for which *alpha convertible agents are deemed to have the same transitions.*

The reader familiar with the π-calculus will have noticed that we are using an *early transition system* (Sangiorgi 1992) – since the bound names of an input are instantiated as soon as possible, in the input rule – as opposed to a *late transition system* (Milner *et al.* 1992; Milner 1991) – where the instantiation is done later, in the communication rule. The adoption of an early transition system naturally leads to the adoption of an *early bisimilarity*, so christened in the literature to distinguish it from other formulations like the *late* and the *open* (see, for instance, Ferrari *et al.* (1996)). Our 'early' choice is not critical for the results we shall present, although some definitions (like that of function $\mathscr{C}_\Sigma$ in Section 5) depend upon this choice.

With the given early transition system, the definition of progression between relations on π-calculus processes only differs from the standard one (Definition 2.1) because a side condition is added to ensure the 'freshness' of bound names of actions, as in the following definition.

Definition 4.1. A progression $\mathscr{R} \rightarrowtail \mathscr{S}$, between two relations $\mathscr{R}$ and $\mathscr{S}$ on π-calculus processes, holds if for all $P \mathrel{\mathscr{R}} Q$

— whenever $P \xrightarrow{\mu} P'$ with $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$, there is Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \mathrel{\mathscr{S}} Q'$,

and the symmetric clause, on the actions by Q.

The definitions of a bisimulation relation and of bisimilarity are the same as those for CCS-like languages in Section 2. However, in contrast with CCS, in π-calculus bisimilarity is not a full congruence, since it is not preserved by input prefix. This failure arises because $\sim$ is not preserved by name instantiation. For instance, $[a = b]\overline{a}c.\mathbf{0} \sim \mathbf{0}$, but $([a = b]\overline{a}c.\mathbf{0})\{a/b\} \not\sim \mathbf{0}\{a/b\}$, since $([a = b]\overline{a}c.\mathbf{0})\{a/b\} = [a = a]\overline{a}c.\mathbf{0}$ is not a deadlocked process. In consequence, we also have $d(a).[a = b]\overline{a}c.\mathbf{0} \not\sim d(a).\mathbf{0}$. We therefore also consider the *congruence* $\sim^{\mathrm{c}}$ induced by $\sim$ (Milner *et al.* 1992).

Definition 4.2. (Congruence induced by $\sim$) We set $P \sim^{\mathrm{c}} Q$, pronounced 'P and Q are congruent', if $P\sigma \sim Q\sigma$, for all substitutions σ.

5. Proof techniques for the π-calculus

In comparison with CCS, actions are more structured in the π-calculus (there is an object part also) and the definitions of transition rules and progression involve alpha conversion and substitution on names. These differences require straightforward modifications to the theory of sound and respectful functions presented in Section 2. The only exception is the definition of function $\mathscr{C}$ (closure under contexts) and the proof of its respectfulness. The Definition 2.15 of faithful contexts (on which the definition of $\mathscr{C}$ is based) is limiting in the π-calculus, because it does not capture all contexts. For instance, $C \stackrel{\mathrm{def}}{=} a(x).[\cdot]$ is not faithful: if $P \stackrel{\mathrm{def}}{=} x(y).\mathbf{0}$, then $C[P] \xrightarrow{ab} P\{b/x\}$, but there is no $\widehat{\lambda}$ such that $P \xrightarrow{\widehat{\lambda}} P\{b/x\}$. The problem has to do with substitutions, which play an important role in the π-calculus and cannot be ignored. Besides substitutions, in the π-calculus a closure under contexts should arguably take into account the difference between bisimilarity and induced congruence. Intuitively, if we have to prove $C[P] \sim C[Q]$, then it is not sound, in general, to cut the common context C and prove $P \sim Q$, for $P \sim Q$ might not imply $C[P] \sim C[Q]$. One solution to this is to require that the hole occurs in C in a special position, so to guarantee that C preserves the bisimilarity between P and Q. Another solution is to prove that P and Q are congruent, rather than bisimilar.

We therefore revisit the definition of function $\mathscr{C}$ and the proof of its respectfulness for the π-calculus. We call the new function $\mathscr{C}_\pi$. We recall that a context C is weakly guarded if the possible occurrence of the hole $[\cdot]$ is within a subexpression of C of the form $\alpha.C'$; otherwise C is non-weakly-guarded. We set

$$\begin{aligned}\mathscr{C}_\pi(\mathscr{R}) \stackrel{\mathrm{def}}{=} & \bigcup\nolimits_{C \text{ non-weakly-guarded}} \{(C[P], C[Q]) \;:\; (P,Q) \in \mathscr{R}\} \;\bigcup \\ & \bigcup\nolimits_{C \text{ weakly guarded}} \{(C[P], C[Q]) \;:\; (P\sigma, Q\sigma) \in \mathscr{R}, \text{ for all } \sigma\}\,.\end{aligned}$$

Remark 5.1. Note that if $\mathscr{R}$ is closed under substitutions, then $\mathscr{C}_\pi(\mathscr{R})$ simply becomes

$$\bigcup_C \{(C[P], C[Q]) \ : \ (P,Q) \in \mathscr{R}\}.$$

Proposition 5.2. Function $\mathscr{C}_\pi$ is respectful.

Proof. Suppose that $\mathscr{R} \subseteq \mathscr{S}$ and $\mathscr{R} \rightarrowtail \mathscr{S}$. Then, clearly, $\mathscr{C}_\pi(\mathscr{R}) \subseteq \mathscr{C}_\pi(\mathscr{S})$. We also have to check that $\mathscr{C}_\pi(\mathscr{R}) \rightarrowtail \mathscr{C}_\pi(\mathscr{S})$ holds. For this, given $(C[P], C[Q]) \in \mathscr{C}_\pi(\mathscr{R})$ with $C[P] \xrightarrow{\mu} R$, we show that there are C', P' and Q' such that

$$R = C'[P'], \qquad C[Q] \xrightarrow{\mu} C'[Q'] \qquad \text{and} \qquad (C'[P'], C'[Q']) \in \mathscr{C}_\pi(\mathscr{S}). \tag{6}$$

We proceed by induction on the structure of C.

Case 1 $C = [\cdot]$.

Then $C[P] = P$, $C[Q] = Q$ and (6) follows from the hypothesis $\mathscr{R} \rightarrowtail \mathscr{S}$.

Case 2 $C = a(x).\,C'$.

Then $C[P] = a(x).\,C'[P]$, $C[Q] = a(x).\,C'[Q]$, $\mu = ab$, for some b, and

$$R = C'[P]\{b/x\} = C''[P\{b/x\}],$$

for $C'' = C'\{b/x\}$. Moreover, we have that $C[Q] \xrightarrow{ab} C''[Q\{b/x\}]$. Since C is weakly guarded, from the definition of $\mathscr{C}_\pi$ we deduce that $(P\{b/x\}\sigma, Q\{b/x\}\sigma) \in \mathscr{R}$ for all σ. This and the hypothesis $\mathscr{R} \subseteq \mathscr{S}$ demonstrate $(C''[P\{b/x\}], C''[Q\{b/x\}]) \in \mathscr{C}_\pi(\mathscr{S})$.

Case 3 $C = C_1 \mid T$, or $C = T \mid C_1$.

We will look at the case $C = C_1 \mid T$. There are three possibilities to consider, according to whether the action $C[P] \xrightarrow{\mu} R$ comes from $C_1[P]$ alone, from T alone or from an interaction between $C_1[P]$ and T. We only consider the first, since the other two are similar. So, suppose

$$C_1[P] \xrightarrow{\mu} R' \qquad \text{and} \qquad R = R' \mid T. \tag{7}$$

By definition of $\mathscr{C}_\pi$, $(C_1[P] \mid T, C_1[Q] \mid T) \in \mathscr{C}_\pi(\mathscr{R})$ implies

$$(C_1[P], C_1[Q]) \in \mathscr{C}_\pi(\mathscr{R}). \tag{8}$$

From (8) and (7), by induction, there are C_1', P' and Q' such that

$$R' = C_1'[P'], \qquad C_1[Q] \xrightarrow{\mu} C_1'[Q'] \qquad \text{and} \qquad (C_1'[P'], C_1'[Q']) \in \mathscr{C}_\pi(\mathscr{S}).$$

Moreover, using rule par, we have

$$C_1[Q] \mid T \xrightarrow{\mu} C_1'[Q'] \mid T. \tag{9}$$

Finally, since $(C_1'[P'], C_1'[Q']) \in \mathscr{C}_\pi(\mathscr{S})$ and the addition of a parallel component does not change the guardedness of a context, we get

$$(C_1'[P'] \mid T, C_1'[Q'] \mid T) \in \mathscr{C}_\pi(\mathscr{S}). \tag{10}$$

If $C' \overset{\text{def}}{=} C_1 \mid T$, then $R = C_1'[P'] \mid T$, (9) and (10) prove (6).

Case 4 $C = \overline{a}b.\,C'$, or $C = \tau.\,C'$ or $C = C_1 + T$, or $C = T + C_1$, or $C = \boldsymbol{\nu}\, a\, C'$, or $C = A\langle \tilde{b} \rangle$, or $C = [a = b]C'$.

These cases are easy.

$\square$

A useful fact, which is derived from Definition 4.2 of the congruence $\sim^{c}$, is the following corollary.

Corollary 5.3. Suppose that $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R})$ holds, for some sound function $\mathscr{F}$, and suppose that for two given processes P and Q, and for all substitutions σ, we have that $(P\sigma, Q\sigma) \in \mathscr{R}$. Then $P \sim^{c} Q$.

A special case of this corollary occurs when the relation $\mathscr{R}$ itself is closed under substitutions, in which case $P \sim^{c} Q$ holds for all pairs (P, Q) in $\mathscr{R}$.

5.1. *Closure of relation under injective substitutions on names*

A substitution σ on names is *injective on a set* V of names if for all $a, b \in V$, we have $\sigma(a) = \sigma(b)$ implies $a = b$. A substitution σ is *injective* if it is injective on the set of all names.

A primitive respectful function, very useful in the π-calculus, is one that allows us to work up to injective substitutions on names. It is called $\mathscr{S}ub$ and is defined as follows:

$$\mathscr{S}ub(\mathscr{R}) \overset{\text{def}}{=} \{(P\sigma, Q\sigma) \ : \ (P, Q) \in \mathscr{R} \text{ and } \sigma \text{ is injective on } \text{fn}(P, Q)\}\,.$$

We will show that $\mathscr{S}ub$ is respectful, but first we need the following lemma.

Lemma 5.4. Let σ be a substitution injective on a finite set V of names with $\text{fn}(P) \subseteq V$. Then there is an injective substitution ρ with $\sigma(a) = \rho(a)$ for all $a \in V$, such that:

1 If $P \xrightarrow{\mu} P'$, then $P\rho \xrightarrow{\mu\rho} P'\rho$.

2 If $P\rho \xrightarrow{\mu'} P''$, then there are P' and μ with $P \xrightarrow{\mu} P'$ and $\mu\rho = \mu'$, $P'\rho = P''$.

Proof. We define the function ρ. Then, checking that ρ satisfies Clauses (1) and (2) can be done by transition induction; all cases are simple, and we omit them. Let W, W^- and V^- be the following sets of names:

$$\begin{aligned} W &\overset{\text{def}}{=} \{\sigma(a) \ : \ a \in V\} \\ W^- &\overset{\text{def}}{=} W - V \ = \ \{a \ : \ a \in W \text{ and } a \notin V\} \\ V^- &\overset{\text{def}}{=} V - W \ = \ \{a \ : \ a \in V \text{ and } a \notin W\}\,. \end{aligned}$$

Since σ is injective on V, sets V and W have the same finite cardinality; hence also sets V^- and W^- have the same finite cardinality. Take an ordering of names in V^- and W^-, say

$$\begin{aligned} W^- &= \{a_1, \ldots, a_n\}\,, \\ V^- &= \{b_1, \ldots, b_n\}\,. \end{aligned}$$

The substitution ρ is specified as follows:

$$\rho(a) \overset{\text{def}}{=} \begin{cases} \sigma(a) & \text{if } a \in V \\ b_i & \text{if } a = a_i \in W^- \\ a & \text{otherwise, that is } a \notin (V \cup W^-)\,. \end{cases}$$

Function ρ is injective: first, notice that names in V are mapped onto distinct names of

W, and that names in W^- are mapped onto distinct names in V^-. Hence ρ, restricted to $V \cup W$, is an injective function from this set onto itself. Since names not in $V \cup W$ are mapped onto themselves, ρ is injective on all names (in fact, ρ is a bijection). $\square$

Proposition 5.5. Function $\mathscr{S}ub$ is respectful.

Proof. We have to show that if $\mathscr{R} \subseteq \mathscr{S}$ and $\mathscr{R} \rightarrowtail \mathscr{S}$, then $\mathscr{S}ub(\mathscr{R}) \subseteq \mathscr{S}ub(\mathscr{S})$ and $\mathscr{S}ub(\mathscr{R}) \rightarrowtail \mathscr{S}ub(\mathscr{S})$. The former is straightforward, so we only look at the latter.

Take $(P\sigma, Q\sigma) \in \mathscr{S}ub(\mathscr{R})$ for some $(P,Q) \in \mathscr{R}$ and σ injective on $\mathrm{fn}(P,Q)$. Suppose $P\sigma \xrightarrow{\mu'} P''$. We have to find Q'' such that

$$Q\sigma \xrightarrow{\mu'} Q'' \qquad \text{and} \qquad (P'', Q'') \in \mathscr{S}ub(\mathscr{S}). \tag{11}$$

Let ρ be the injective function that Lemma 5.4 associates to σ and the set of names $\mathrm{fn}(P) \cup \mathrm{fn}(Q)$; thus $P\rho = P\sigma$ and $Q\rho = Q\sigma$. By Lemma 5.4(2), there are μ and P' such that $P \xrightarrow{\mu} P'$, $\mu' = \mu\rho$ and $P'' = P'\rho$. Since $\mathscr{R} \rightarrowtail \mathscr{S}$, the diagram

$$\begin{tikzcd} P \arrow[r, phantom, "\mathscr{R}"] \arrow[d, "\mu"'] & Q \arrow[d, "\mu"'] \\ P' \arrow[r, phantom, "\mathscr{S}"] & Q' \end{tikzcd}$$

commutes, for some Q'. By Lemma 5.4(1), $Q\rho \xrightarrow{\mu\rho} Q'\rho$. Hence the diagram

$$\begin{tikzcd} P\rho \arrow[r, phantom, "\mathscr{S}ub(\mathscr{R})"] \arrow[d, "\mu\rho"'] & Q\rho \arrow[d, "\mu\rho"'] \\ P'\rho \arrow[r, phantom, "\mathscr{S}ub(\mathscr{S})"] & Q'\rho \end{tikzcd}$$

commutes also. Therefore, for $Q'' \overset{\text{def}}{=} Q'\rho$, since $P'\rho = P''$, $Q\rho = Q\sigma$ and $\mu\rho = \mu'$, this proves (11). $\square$

Having proved that $\mathscr{S}ub$ is respectful, we know that it is a sound function and, moreover, we can safely combine it with other respectful functions, following the indications in Section 2.

6. Applications of the proof techniques in the π-calculus

6.1. *Use of the closure under injective substitutions on names*

The closure under injective substitutions on names (that is, function $\mathscr{S}ub$ of Section 5.1) is useful for cases in which universal quantifications on substitutions are involved. For instance, such quantifications are present (implicitly) in the clause of progression for inputs and bound outputs (Definition 4.1), and (explicitly) in the definition of function $\mathscr{C}_\pi$.

As a simple example of the application of function $\mathscr{S}ub$, consider the processes

$$\begin{aligned} P &\overset{\text{def}}{=} a(x).\, \boldsymbol{\nu}\, b\, (\overline{x}b \mid \overline{b}x) \\ Q &\overset{\text{def}}{=} a(x).\, \boldsymbol{\nu}\, b\, \overline{x}b.\overline{b}x\,, \end{aligned}$$

and suppose we want to prove $P \sim Q$. If we were to look for a bisimulation relation containing P and Q as a pair, then at least we would need

$$\begin{aligned}\mathscr{R} \stackrel{\text{def}}{=} \; & \{(P,Q),(\mathbf{0} \mid \mathbf{0},\mathbf{0})\} \;\cup \\ & \textstyle\bigcup_{c\in \text{Names}} \left\{\left(\boldsymbol{\nu}\, d\,(\overline{c}d \mid \overline{d}c), \boldsymbol{\nu}\, d\,\overline{c}d.\overline{d}c\right) \;:\; d \neq c\right\} \;\cup \\ & \textstyle\bigcup_{c\in \text{Names}} \bigcup_{d\in \text{Names}} \left\{\left(\mathbf{0} \mid \overline{d}c, \overline{d}c\right) \;:\; d \neq c\right\}.\end{aligned}$$

Note that $\mathscr{R}$ contains three unions that range over the infinite set of names. These unions are needed because, for all names d and c with $d \neq c$, processes P and Q can perform an input action labelled ac and then a bound output action labelled $\overline{c}(d)$. Exploiting function $\mathscr{S}ub$, we can prove $P \sim Q$ by simply taking

$$\mathscr{R}' \stackrel{\text{def}}{=} \left\{\left(P,Q\right), \left(\boldsymbol{\nu}\, b\,(\overline{x}b \mid \overline{b}x), \overline{x}b.\overline{b}x\right), \left(\mathbf{0} \mid \overline{b}x, \overline{b}x\right), \left(\mathbf{0} \mid \mathbf{0}, \mathbf{0}\right)\right\},$$

where b and x are any pair of distinct names. $\mathscr{R}'$ only contains four pairs of processes. It is easy to check that $\mathscr{R}' \rightarrowtail \mathscr{S}ub(\mathscr{R}')$ holds, and hence $\{(P,Q)\} \subseteq \mathscr{R}' \subseteq \sim$.

We can do better than $\mathscr{R}'$ using a combination of function $\mathscr{S}ub$ and simple respectful functions for garbage collecting processes $\mathbf{0}$ from parallel compositions, and for discarding pairs of syntactically equal derivatives (it is easy to define respectful functions that do this). In this way, $P \sim Q$ can be proved by exhibiting a relation made of only two pairs of processes, namely (P,Q) and $(\boldsymbol{\nu}\, b\,(\overline{x}b \mid \overline{b}x), \overline{x}b.\overline{b}x)$.

6.2. *Unique solutions of equations*

As with our treatment of CCS, we can use the function $\sim(-^{\mathscr{C}_\pi})^{\mathscr{T}}\sim$ in the π-calculus to get a simpler proof of the uniqueness of solutions of equations. Both the assertion and the proof of the result are similar to those given for CCS in Section 3.3. There is, however, an additional ingredient in the π-calculus, namely the use of parameters in constant definitions and calls. Because of this, and because $\sim$ is not preserved by substitution of names, the uniqueness result must be proved with respect to the congruence $\sim^{\mathrm{c}}$, rather than the bisimilarity $\sim$. We omit the details.

6.3. *Normalisation of replications*

To express processes with an infinite behaviour, some presentations of the π-calculus use the *replication* operator $!P$ in place of recursive definitions. Intuitively, $!P$ stands for a countably infinite number of copies of P in parallel. It is easy to code replication up using recursive definitions[†]. And if the number of recursive definitions is finite, then the reverse direction holds also (Milner 1991).

The transition rule for replication is

$$\text{rep:}\;\frac{P \mid\, !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}.$$

In this and the following subsection, we exploit our proof techniques based on sound functions to demonstrate some results about the replication operator. The main result of

[†] The recursive definition for $!P$ would be $!P \stackrel{\text{def}}{=} P \mid\, !P$; in Section 1 replication was presented in this way.

this subsection is new. It says that, if we choose to have replication in the grammar of the π-calculus, a simple form of replication suffices, namely normalised replications of the form $!\alpha.P$. All 'free' replications $!P$ can be coded up using normalised replications, up to the bisimilarity congruence $\sim^{\mathrm{c}}$. The proof of this result is obtained in three steps, the first of which uses our proof techniques, whereas the other two use a standard structural induction. Subsection 6.4 considers certain distributivity properties of private replications, which were first proved by Milner (Milner 1991).

Throughout this and the next subsection, we assume that the syntax of the π-calculus expressions contains the replication operator $!P$ in place of recursive definitions. The definition of function $\mathscr{C}_\pi$ and the proof of its soundness (Proposition 5.2) remain unchanged if in the definition of $\mathscr{C}_\pi$ we require that the hole of a context cannot occur underneath a replication; this will suffice in the examples below. It is easy to extend this definition, and allow holes of contexts also underneath replications, by utilising polyadic contexts.

Definition 6.1. We say that a replication $!P$ is *normal* if P is of the form $\alpha.Q$. A process has *normalised replications* if all replications it contains are normal.

Normalised replications can be given the simple transition rule

$$\textsf{rep-nor:}\ \frac{\alpha.P \xrightarrow{\mu} P'}{!\alpha.P \xrightarrow{\mu} P' \mid !\alpha.P}$$

or, alternatively, the two rules

$$\textsf{rep-inp:}\quad !a(x).P \xrightarrow{ab} P\{^b\!/\!_x\} \mid !a(x).P$$

$$\textsf{rep-pre:}\quad !\alpha.P \xrightarrow{\alpha} P \mid !\alpha.P\,,\ \text{if } \alpha \text{ is not an input.}$$

Remark 6.2. As an aside, we wish to point out that rule rep-nor (as well as rep-inp and rep-pre) preserves the following pleasant property of π-calculus transition system in Table 2, and which we state here very informally: if two inference proofs of transitions $P \xrightarrow{\mu} P'$ and $P \xrightarrow{\mu} P''$ involve the same prefix(es) of P, then P' and P'' are syntactically the same (up to alpha conversion). This is a handy property to have, for example, when examining the set of derivatives of a process, because it makes it easier to reason by structural induction on processes. This property does not hold for rule rep. For instance, we can infer

$$!\overline{a}b.Q \xrightarrow{\overline{a}b} Q \mid !\overline{a}b.Q \qquad \text{and} \qquad !\overline{a}b.Q \xrightarrow{\overline{a}b} \overline{a}b.Q \mid Q \mid !\overline{a}b.Q\,;$$

in these transitions, the same prefix $\overline{a}b$ of $!\overline{a}b.Q$ is consumed, but the derivatives $Q \mid !\overline{a}b.Q$ and $\overline{a}b.Q \mid Q \mid !\overline{a}b.Q$ are syntactically different.

Lemma 6.3.

1 $P \mid !P \sim^{\mathrm{c}} !P$;
2 $!(P \mid Q) \sim^{\mathrm{c}} !P \mid !Q$;
3 $!(P + Q) \sim^{\mathrm{c}} !(P \mid Q)$.

Proof. Assertion (1) is trivial: because of the transition rule for replication, for each P, we have $!P \xrightarrow{\mu} P'$ iff $P \mid !P \xrightarrow{\mu} P'$. Assertions (2) and (3) can be proved by exhibiting the appropriate progressions, both of which are of the form $\mathscr{R} \rightarrowtail \sim\mathscr{R}^{\mathscr{C}_\pi}\sim$. For (2), the relation to use is

$$\mathscr{R}_2 \overset{\text{def}}{=} \bigcup_{P,Q} \{(\,!(P \mid Q),\ !P \mid \,!Q)\},$$

and for (3) it is

$$\mathscr{R}_3 \overset{\text{def}}{=} \bigcup_{P,Q} \{(\,!(P + Q),\ !(P \mid Q))\}\,.$$

Relations $\mathscr{R}_2$ and $\mathscr{R}_3$ are closed under substitutions, hence, by Corollary 5.3, they can be used to prove $\sim^{\mathrm{c}}$ equalities.

We consider the proof of $\mathscr{R}_3 \rightarrowtail \sim(\mathscr{R}_3)^{\mathscr{C}_\pi}\sim$ in detail. We check that $!(P \mid Q)$ can match the moves by $!(P+Q)$; the converse, on the actions by $!(P \mid Q)$, can be treated similarly. By transition induction, we prove that if $!(P+Q) \xrightarrow{\mu} T_1$, then there is R such that

$$T_1 \sim R \mid \,!(P+Q) \quad \text{and, for some } T_2,\ \ !(P \mid Q) \xrightarrow{\mu} T_2 \sim R \mid \,!(P \mid Q)\,. \tag{12}$$

This shows that $(T_1, T_2) \in \sim\mathscr{R}_3{}^{\mathscr{C}_\pi}\sim$, and we are done. Note that we use function $\mathscr{C}_\pi$ to cancel context $R \mid [\cdot]$: according to the definition of $\mathscr{C}_\pi$, this is legitimate because $R \mid [\cdot]$ is a non-weakly-guarded context (actually, in the case of relation $\mathscr{R}_3$ we could cancel *any* context because $\mathscr{R}_3$ is closed under substitutions on names – see Remark 5.1).

To infer $!(P+Q) \xrightarrow{\mu} T_1$, the last rule applied must have been of the form

$$\frac{(P+Q) \mid \,!(P+Q) \xrightarrow{\mu} T_1}{!(P+Q) \xrightarrow{\mu} T_1}\,.$$

Therefore, there are three cases to consider, depending on whether $(P+Q) \mid \,!(P+Q) \xrightarrow{\mu} T_1$ comes from $P+Q$ alone, from $!(P+Q)$ alone or from an interaction between $P+Q$ and $!(P+Q)$. We only look at the last case, assuming P is the summand of $P+Q$ that is used, and that it performs an input at a of the free name b. Thus we have, for some T_1' and P' such that $P \xrightarrow{ab} P'$:

$$\frac{P+Q \xrightarrow{ab} P' \qquad !(P+Q) \xrightarrow{\overline{a}b} T_1'}{(P+Q) \mid \,!(P+Q) \xrightarrow{\tau} T_1 = P' \mid T_1'}\,. \tag{13}$$

By the induction assumption, for some R', we have

$$T_1' \sim R' \mid \,!(P+Q) \tag{14}$$

and, for some T_2',

$$!(P \mid Q) \xrightarrow{\overline{a}b} T_2' \sim R' \mid \,!(P \mid Q)\,. \tag{15}$$

Therefore we can infer

$$\frac{\dfrac{P \mid Q \xrightarrow{ab} P' \mid Q \qquad !(P \mid Q) \xrightarrow{\overline{a}b} T_2'}{(P \mid Q) \mid \,!(P+Q) \xrightarrow{\tau} P' \mid Q \mid T_2'}}{!(P \mid Q) \xrightarrow{\tau} P' \mid Q \mid T_2'}\,. \tag{16}$$

By (15),

$$P' \mid Q \mid T_2' \sim P' \mid Q \mid R' \mid \,!(P \mid Q)\,. \tag{17}$$

L1	$\boldsymbol{\nu}\, a\,(P+Q)$	$\sim^{\mathrm{c}}$	$\boldsymbol{\nu}\, a\, P + \boldsymbol{\nu}\, a\, Q$	
L2	$\boldsymbol{\nu}\, a\,[b=c]P$	$\sim^{\mathrm{c}}$	$[b=c]\boldsymbol{\nu}\, a\, P$	if $a \notin \{b,c\}$
L3	$\boldsymbol{\nu}\, a\,[a=b]P$	$\sim^{\mathrm{c}}$	$\mathbf{0}$	if $a \neq b$
L4	$\boldsymbol{\nu}\, a\,[a=a]P$	$\sim^{\mathrm{c}}$	$\boldsymbol{\nu}\, a\, P$	
L5	$\boldsymbol{\nu}\, a\, \alpha.\, P$	$\sim^{\mathrm{c}}$	$\alpha.\, \boldsymbol{\nu}\, a\, P$	if $a \notin \mathrm{n}(\alpha)$
L6	$\boldsymbol{\nu}\, a\, \alpha.\, P$	$\sim^{\mathrm{c}}$	$\mathbf{0}$	if α is an input or an output at a
L7	$[a=b](P+Q)$	$\sim^{\mathrm{c}}$	$[a=b]P + [a=b]Q$	
L8	$!\,[a=b]P$	$\sim^{\mathrm{c}}$	$[a=b]\,!\,P$	
L9	$!\,\alpha.\, P$	$\sim^{\mathrm{c}}$	$\alpha.\,(P \mid \,!\,\alpha.\, P)$	if $\mathrm{bn}(\alpha) \cap \mathrm{fn}(\alpha.P) = \emptyset$

Table 3. *Some simple laws for the π-calculus*

Moreover, from associativity and commutativity of parallel composition, and Lemma 6.3(1–2) we get

$$\begin{aligned} P' \mid Q \mid R' \mid \,!\,(P \mid Q) &\sim P' \mid R' \mid Q \mid \,!\,P \mid \,!\,Q \\ &\sim P' \mid R' \mid \,!\,P \mid \,!\,Q \\ &\sim P' \mid R' \mid \,!\,(P \mid Q)\,. \end{aligned} \tag{18}$$

Now, define $R \overset{\mathrm{def}}{=} P' \mid R'$. From (13) and (14), we have $T_1 \sim R \mid \,!\,(P+Q)$, and, from (16–18), we have $!\,(P \mid Q) \xrightarrow{\tau} \sim R \mid \,!\,(P \mid Q)$. This proves (12). $\square$

In the proof of Assertions (2) and (3) of Lemma 6.3, the possibility of cutting contexts off, achieved through the closure under contexts, reduces the size of the relations to exhibit sensibly. Indeed, if we fix the processes P and Q to examine, and we content ourselves with proving bisimilarity (rather then congruence) results, then relations $\mathscr{R}_2$ and $\mathscr{R}_3$ would only contain *one* pair of processes. For instance, $\mathscr{R}_3$ would be

$$\{(\,!\,(P+Q),\,!\,(P \mid Q))\}\,.$$

Without the closure under contexts, the relations $\mathscr{R}_2$ and $\mathscr{R}_3$ in the proof of Lemma 6.3 would consist of pairs of processes with at least a further component. For instance, $\mathscr{R}_3$ would become

$$\mathscr{R}_3' \overset{\mathrm{def}}{=} \bigcup_{P,Q,R} \{(R \mid \,!\,(P+Q), R \mid \,!\,(P \mid Q))\}$$

($\mathscr{R}_3'$ progresses to $\sim\mathscr{R}_3'\sim$). Having $\mathscr{R}_3'$ in place of $\mathscr{R}_3$ does not make the proof conceptually more difficult, but it does make it more tedious.

Remark 6.4. Reasoning as above, one can prove the result $!\,P \mid \,!\,P \sim \,!\,P$, mentioned in Section 1, using the singleton relation $\mathscr{R} \overset{\mathrm{def}}{=} \{\,!\,P \mid \,!\,P,\ !\,P\}$, and showing that $\mathscr{R} \rightarrowtail \sim(\mathscr{R})^{\mathscr{C}_\pi}\sim$ holds.

Table 3 contains a few simple π-calculus laws, which will be used in Lemma 6.5. We shall also use the expansion law, as formulated in Parrow and Sangiorgi (1995), and which for ease of reference is reported in Table 4. We abbreviate the sum of processes P_i, $i \in I$, as $\sum_{i \in I} P_i$, and their parallel composition as $\prod_{i \in I} P_i$. We use M to range over (possibly empty) match sequences; thus if M is $[a=b][c=d]$, then MP is $[a=b][c=d]P$.

Let $P \stackrel{\text{def}}{=} \sum_i M_i\alpha_i.\, P_i$ and $Q \stackrel{\text{def}}{=} \sum_j N_j\beta_j.\, Q_j$ where no α_i (respectively, β_j) binds a name free in Q (respectively, P).
Then infer

$$P \mid Q \;\sim^{c}\; \sum_i M_i\alpha_i.\,(P_i \mid Q) + \sum_j N_j\beta_j.\,(P \mid Q_j) + \sum_{\alpha_i \;opp\; \beta_j} M_iN_j[x_i = y_j]\tau.\, R_{ij}$$

where x_i and y_j are the subjects of α_i and β_j, respectively, and α_i *opp* β_j and R_{ij} are defined as follows:

1 α_i is $\overline{x}_i u$ and β_j is $y_j(v)$; then R_{ij} is $P_i \mid Q_j\{^{u}\!/\!_{v}\}$;
2 α_i is $\overline{x}_i(u)$ and β_j is $y_j(v)$; then R_{ij} is $\boldsymbol{\nu}\, w\,(P_i\{^{w}\!/\!_{u}\} \mid Q_j\{^{w}\!/\!_{v}\})$, where w is a fresh name;
3 the converse of (1);
4 the converse of (2).

Table 4. *The expansion law for the π-calculus*

Lemma 6.5. For each process P there is a process Q of the form $\sum_{i\in I} M_i\alpha_i.\, P_i$ such that $P \;\sim^{c}\; Q$. Moreover, the maximal number of nestings of replications in P and in Q is the same.

Proof. The proof is by induction on the structure of P. The transformations we shall impose do not modify the nesting of replications. If $P = \alpha.\, P'$, there is nothing to prove. If $P = P_1 + P_2$, use induction twice. If $P = [a = b]P'$, use induction plus Law **L7**. If $P = P_1 \mid P_2$, use induction plus the expansion law. If $P = \boldsymbol{\nu}\, a\, P'$ use induction plus Laws **L1**–**L6** to push a restriction underneath a sum, a matching, and a prefix, plus (possibly) the laws

$$\begin{array}{lcl} [a = b]\mathbf{0} & \sim^{c} & \mathbf{0} \\ P + \mathbf{0} & \sim^{c} & \mathbf{0} \end{array}$$

to garbage collect $\mathbf{0}$ processes. We are left with the case of replication, that is, $P = \,!\,P'$. By induction, $P' \;\sim^{c}\; \sum_{j\in J} M_j\alpha_j.\, P'_j$, and we can deduce

$$\begin{array}{lcll} !\,P & \sim^{c} & !\big(\sum_{j\in J} M_j\alpha_j.\, P'_j\big) & \\ & \sim^{c} & !\big(\prod_{j\in J} M_j\alpha_j.\, P'_j\big) & \text{(Lemma 6.3(3))} \\ & \sim^{c} & \prod_{j\in J} !\,M_j\alpha_j.\, P'_j & \text{(Lemma 6.3(2))} \\ & \sim^{c} & \prod_{j\in J} M_j\,!\,\alpha_j.\, P'_j & \text{(law } \mathbf{L8}) \\ & \sim^{c} & \prod_{j\in J} M_j\alpha_j.\,(P_j \mid \,!\,P'_j) & \text{(law } \mathbf{L9}). \end{array}$$

Finally, $\prod_{j\in J} M_j\alpha_j.\,(P_j \mid \,!\,P'_j)$ can be rewritten into the form $\sum_{i\in I} M_i\alpha_i.\, P_i$ by means of the expansion law. □

Theorem 6.6. For every process P there is a process Q with normalised replications such that $P \;\sim^{c}\; Q$.

Proof. The proof is by induction on the maximal number of nested replications in P. If P does not have replications, there is nothing to prove. For the inductive case, we proceed by induction on the structure of P. The only interesting case is when $P = \,!\,P'$. By Lemma 6.5, $P' \;\sim^{c}\; \sum_{i\in I} M_i\alpha_i.\, P'_i$ and the two processes have the same maximal number of nested replications. By the induction on the number of nested replications, there are processes $P''_i \;\sim^{c}\; P'_i$ with normalised replications. We can thus derive

$$!\,P \;\sim^{c}\; !\Big(\sum_{i\in I} M_i\alpha_i.\, P''_i\Big),$$

and then, by Lemmas 6.3(2–3) and law **L8**,

$$\begin{aligned} &\sim^c \quad !\left(\textstyle\prod_{i\in I} M_i\alpha_i.\,P_i''\right)\\ &\sim^c \quad \textstyle\prod_{i\in I}\,!\,M_i\alpha_i.\,P_i''\\ &\sim^c \quad \textstyle\prod_{i\in I} M_i\,!\,\alpha_i.\,P_i''\,, \end{aligned}$$

which is a process with normalised replications. □

6.4. *Distributivity properties of private replications*

In Milner (1991), Milner shows certain distributivity properties for private replications with respect to parallel composition and replication. The importance of these properties has emerged in different situations, such as the correctness of the encodings of λ-calculus and higher-order calculi into the π-calculus (Milner 1991; Sangiorgi 1992) and in reasoning about data structures (Walker 1994).

The replication theorems Assume that a occurs free in R, P_1, P_2, and $\alpha.P$ only as subject of output prefixes. Then[†]

1 $\boldsymbol{\nu}\, a\big(\,!\,a(x).R \mid P \mid Q\big) \sim^c \boldsymbol{\nu}\, a(\,!\,a(x).R \mid P) \mid \boldsymbol{\nu}\, a(\,!\,a(x).R \mid Q)$;

2 $\boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid\ !\,\alpha.P) \sim^c\ !\,\boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid \alpha.\,P)$.

For the proof of these assertions, Milner (Milner 1991) uses relations $\mathscr{R}_1$ and $\mathscr{R}_2$, defined as below, and proves that they progress to $\sim\mathscr{R}_1\sim$ and $\sim\mathscr{R}_2\sim$, respectively. We let $\mathscr{N}$ be the set of all processes that contain name a free only as subject of output prefixes:

$$\mathscr{R}_1 \overset{\text{def}}{=} \bigcup_{P,Q,R\in\mathscr{N},\,\widetilde{b}\subseteq_{\text{fin}}\text{Names}} \left\{\left(\begin{array}{l} \boldsymbol{\nu}\,\widetilde{b}\,\boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid P \mid Q),\\ \boldsymbol{\nu}\,\widetilde{b}\big(\boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid P) \mid \boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid Q)\big) \end{array}\right)\right\}$$

$$\mathscr{R}_2 \overset{\text{def}}{=} \bigcup_{\alpha.P,Q,R\in\mathscr{N},\,\widetilde{b}\subseteq_{\text{fin}}\text{Names}} \left\{\left(\begin{array}{l} \boldsymbol{\nu}\,\widetilde{b}\,\boldsymbol{\nu}\, a\big(\,!\,a(x).\,R \mid\ !\,\alpha.P \mid Q\big),\\ \boldsymbol{\nu}\,\widetilde{b}\big(\,!\,\boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid \alpha.P) \mid \boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid Q)\big) \end{array}\right)\right\}$$

where $\widetilde{b} \subseteq_{\text{fin}}$ Names means that $\widetilde{b}$ is a finite tuple of names. Since $\mathscr{R}_1$ and $\mathscr{R}_2$ are closed under substitutions on names, they give us $\sim^c$ equalities (Corollary 5.3); and the assertions of the replication theorems follow for $\widetilde{b} = \emptyset$ and $Q = \mathbf{0}$.

The use of function $\mathscr{C}_\pi$ (closure under contexts) allows us a few simplifications. In the proof of (1), it allows us to eliminate the outermost vector of restrictions $\boldsymbol{\nu}\,\widetilde{b}$ from $\mathscr{R}_1$, and take

$$\mathscr{R}_1' \overset{\text{def}}{=} \bigcup_{P,Q,R\in\mathscr{N}} \Big\{\big(\boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid P \mid Q), \boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid P) \mid \boldsymbol{\nu}\, a(\,!\,a(x).\,R \mid Q)\big)\Big\}\,.$$

[†] To simplify the case analysis in the proof, in the assertion of the second replication theorem we have used a normalised replication $!\,\alpha.\,P$, in place of a 'free' replication $!\,P$ as used by Milner (Milner 1991). Some justification for this simplification comes from Theorem 6.6.

In the proof of (2), the use of $\mathscr{C}_\pi$ suggests a drastic simplification of $\mathscr{R}_2$, by taking

$$\mathscr{R}'_2 \stackrel{\text{def}}{=} \bigcup_{\alpha.P, R \in \mathscr{N}} \left\{ \left(\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R\,|\;!\,\alpha.P),\;!\,\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R\,|\,\alpha.P)\right) \right\}.$$

To see that $\mathscr{R}'_2$ progresses to $\sim(\mathscr{R}'_2)^{\mathscr{C}_\pi}\sim$, suppose $(Q_1, Q_2) \in \mathscr{R}'_2$, for

$$\begin{array}{lcl} Q_1 & \stackrel{\text{def}}{=} & \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R\,|\;!\,\alpha.\,P\,), \\ Q_2 & \stackrel{\text{def}}{=} & !\,\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R\,|\,\alpha.\,P\,). \end{array}$$

We assume that α is an output at a, say $\alpha = \overline{a}b$; all other cases are similar. The only moves that Q_1 and Q_2 can do (up to unfolding of replications) are

$$\begin{array}{lcll} Q_1 & \xrightarrow{\tau} & \boldsymbol{\nu}\, a\,(R\{^b\!/\!_x\}\;|\;!\,a(x).\,R\,|\,P\;|\;!\,\alpha.P\,) & \stackrel{\text{def}}{=} Q'_1 \\ Q_2 & \xrightarrow{\tau} & \boldsymbol{\nu}\, a\,(R\{^b\!/\!_x\}\;|\;!\,a(x).\,R\,|\,P\,)\,|\,Q_2 & \stackrel{\text{def}}{=} Q'_2\,. \end{array}$$

Now, let

$$C \stackrel{\text{def}}{=} \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R\,|\,R\{^b\!/\!_x\}\,|\,P\,)\,|\,[\cdot]\,.$$

We have, by the first replication theorem and commutativity and associativity of parallel composition

$$\begin{array}{lcll} Q'_1 & \sim & \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R\,|\,R\{^b\!/\!_x\}\,|\,P\;|\;!\,\alpha.P\,) & \\ & \sim & \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R\,|\,R\{^b\!/\!_x\}\,|\,P\,)\,|\,\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R\,|\;!\,\alpha.P\,) & = C[Q_1]\,, \\ Q'_2 & \sim & \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R\,|\,R\{^b\!/\!_x\}\,|\,P\,)\,|\,Q_2 & = C[Q_2]\,. \end{array}$$

This shows that $(Q'_1, Q'_2) \in \sim(\mathscr{R}'_2)^{\mathscr{C}_\pi}\sim$, and concludes the proof.

7. Conclusions and further developments

In this paper, we have studied generalisations of the bisimulation proof method that allow us to reduce the size of the relations to exhibit (and hence relieve the work needed) for establishing bisimilarity results. We have relaxed the self-progression requirement in the definition of a bisimulation relation, namely $\mathscr{R} \rightarrowtail \mathscr{R}$, and considered progressions of the form $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R})$, where $\mathscr{F}$ is a function on relations. The *sound* functions are those for which $\mathscr{R} \rightarrowtail \mathscr{F}(\mathscr{R})$ implies that $\mathscr{R}$ only contains pairs of bisimilar processes, for all $\mathscr{R}$. We have given a condition on functions, called *respectfulness*, which ensures soundness. We have shown that the class of respectful functions contains non-trivial functions and and that it enjoys closure properties with respect to important function constructors: thus, sophisticated sound functions (and hence sophisticated proof techniques) can be derived from simpler ones.

The usefulness of our proof techniques has been supported by various non-trivial examples (drawn from CCS and the π-calculus), which include the proof of the unique solution of equations and the proofs of a few properties of the replication operator. Among these, there is a novel result, which justifies the adoption of the simple form of replication $!\,\alpha.\,P$ as the only form of replication in the π-calculus.

One of our most useful primitive proof techniques is an 'up-to context' technique, which allows us to cancel a common context in the derivatives of two processes. We have shown that the associated function is respectful if the contexts cancelled are faithful, but that it loses respectfulness if the cancelled contexts are simply required to preserve bisimilarity – a property weaker than faithfulness. We have also seen that if the transition rules for the operators of the language are in unary De Simone format, all contexts of the language are faithful. It remains to find out how far beyond faithfulness and the De Simone format it is possible to go while preserving respectfulness. Groote and Vaandrager's *tyft* format (Groote and Vaandrager 1992) – but without lookaheads greater than one – and Bloom, Istrail and Meyer's *GSOS* format (Bloom *et al.* 1995) are examples of formats that would be interesting to examine. Lookaheads greater than one, present in the *tyft* format, must be disallowed in the light of the counterexample in Section 2.1.3.

Most of the respectful functions $\mathscr{F}$ that we have considered have the property that if a relation $\mathscr{R}$ progresses to $\mathscr{F}(\mathscr{R})$, then $\mathscr{F}(\mathscr{R})$ is a bisimulation relation: that is, the bisimulation relation is found after one application of the respectful function. However, the definition of respectfulness (Definition 2.5) allows us greater freedom: in the proof of soundness for respectful functions (Theorem 2.11), the bisimulation relation is constructed from a sequence of relations in which the respectful function is applied unboundedly many times. This suggests another direction to investigate, namely the search for other useful respectful functions and function constructors, to be added to those we have found.

In this paper, we have confined ourselves to strong bisimilarities, where all actions are treated equally. A natural development of our work is to look at *weak bisimilarities*, where a special action, called the *silent action*, is distinguished from the others and partially ignored in the bisimilarity clause. Often a weak bisimilarity is not preserved by *dynamic* operators, *i.e.*, operators like CCS or π-calculus sum that can be discharged when some action is performed. This introduces problems for the soundness of the up-to-context technique similar to those we had to face in Section 5 with the π-calculus (where bisimilarity is not a congruence) and which, therefore, might be dealt with in an analogous way. In the weak case it might also be harder to establish results about combinations of proof techniques (*i.e.*, to develop a theory of sound or respectful function constructors). The reason is that the soundness of some basic techniques for weak bisimilarities presents a few rather delicate points whose fragility might be enhanced in combinations of techniques (see, for instance, the study of 'weak bisimulations up-to weak bisimilarity' in Sangiorgi and Milner (1992)).

We believe that our proof techniques could be very advantageous in *higher-order calculi* like CHOCS (Thomsen 1990), or Higher-Order π-calculus (Sangiorgi 1992), *i.e.*, calculi in which terms can be exchanged in a communication. For instance, a few rather involved proofs in Sangiorgi (1992), dealing with the Higher-Order π-calculus, should become simpler using some form of 'bisimulation up-to context' (see Remark 6.6.18 in Sangiorgi (1992)). Our proof techniques should also be useful in higher-order functional languages, for instance, for reasoning about *applicative bisimilarity* of programs (Abramsky 1989).

The bisimulation proof method stems from the theory of fixed-points and the co-induction principle (Milner 1989; Milner and Tofte 1991). On a complete lattice (*i.e.*, a partial order with all joins) the co-induction principle says

> *Let $(D, <)$ be a complete lattice, and $\mathscr{G} : D \to D$ a monotone function with greatest fixed-point $\mu_{\mathscr{G}}$. To prove that $x < \mu_{\mathscr{G}}$ it suffices to prove that x is a post-fixed-point of $\mathscr{G}$, that is, $x < \mathscr{G}(x)$.*

When the bisimilarity relation $\sim$ is interpreted as the greatest fixed-point of a certain continuous function on relations (Milner 1989, Section 4.6), this translates into saying that to prove $\mathscr{R} \subseteq \sim$, it suffices to prove that $\mathscr{R}$ is a bisimulation relation. We would like to see whether our study of the bisimulation proof method leads to an interesting generalisation of the co-induction principle. A possible generalisation, suggested by the definition of respectful functions and the proof of Theorem 2.11, uses an auxiliary function $\mathscr{F}$ as follows:

Theorem 7.1. Let $(D, <)$ be a complete lattice, and $\mathscr{G} : D \to D$ a monotone function with greatest fixed-point $\mu_{\mathscr{G}}$. Suppose $\mathscr{F} : D \to D$ and that, for all $z, y \in D$, $z < y$ and $z < \mathscr{G}(y)$ implies $\mathscr{F}(z) < \mathscr{F}(y)$ and $\mathscr{F}(z) < \mathscr{G}(\mathscr{F}(y))$. Then to prove $x < \mu_{\mathscr{G}}$ it suffices to prove $x < \mathscr{G}(\mathscr{F}(x))$.

Theorem 2.11 is an instance of this theorem, and the proof is essentially the same. A more elegant but weaker formulation of Theorem 7.1 could require that $\mathscr{F}$ is monotone and that $\mathscr{F} \circ \mathscr{G} < \mathscr{G} \circ \mathscr{F}$ (that is, for all z, $(\mathscr{F} \circ \mathscr{G})(z) < (\mathscr{G} \circ \mathscr{F})(z)$). It is worth pointing out that if $\mathscr{F}$ is monotone, the condition $\mathscr{F} \circ \mathscr{G} < \mathscr{G} \circ \mathscr{F}$ is the same as the condition 'for all $z, y \in D$, $z < \mathscr{G}(y)$ implies $\mathscr{F}(z) < \mathscr{F}(\mathscr{G}(y))$'. In terms of respectful functions for bisimilarity, this formulation would amount to having the same conditions as in Remark 2.6.

Acknowledgements

The ideas in this paper were developed when I was visiting Jaco de Bakker and his group at CWI (Amsterdam). I have benefited from discussions with people in CWI, especially Jan Rutten and Daniele Turi. I would also like to thank Glenn Bruns, Martin Hofmann, Marcelo Fiore, Robin Milner, Andrew Pitts, Peter Sewell, David N. Turner and David Walker, whose suggestions helped me to improve the technical presentation. Thanks also to the anonymous referees for detailed comments (including those on TeX!). This research has been supported by the ESPRIT BRA project 6454 'CONFER'.

References

Abramsky, S. (1989) The Lazy Lambda Calculus. In: Turner, D. (ed) *Research Topics in Functional Programming*, Addison-Wesley 65–116.

Abramsky, S. (1991) A Domain Equation for Bisimulation. *Information and Computation* **92** 161–218.

Aczel, P. (1988) *Non-well-founded Sets*, CSLI lecture notes **14**.

Bergstra, J. and Klop, J. (1984) Process Algebra for Synchronous Communication. *Information and Computation* **60** 109–137.

Bloom, B., Istrail, S. and Meyer, A. (1995) Bisimulation can't be Traced. *Journal of the ACM* **42**(1) 232–268.

Boreale, M. and De Nicola, R. (1995) Testing Equivalence for Mobile Processes. *Information and Computation* **120** 279–303.

Boreale, M. and Sangiorgi, D. (1995) A fully abstract semantics for causality in the π-calculus. In: Mayr, E. and Puech, C. (eds) Proc. 12th Symposium on Theoretical Aspects of Computer Science (STACS'95). *Springer Verlag Lecture Notes in Computer Science* **900**. (To appear in *Acta Informatica*.)

Caucal, D. (1990) Graphes Canoniques de Graphes Algébriques. *Informatique Théorique et Applications (RAIRO)* **24**(4) 339–352.

Christensen, S., Hüttel, H. and Stirling, C. (1995) Bisimulation equivalence is decidable for all context-free processes. *Information and Computation* **121**(2).

De Simone, R. (1985) Higher Level Synchronising Devices in MEIJE-SCCS. *Theoretical Computer Science* **37** 245–267.

Ferrari, G., Montanari, U. and Quaglia, P. (1996) A π-calculus with Explicit Substitutions. *Theoretical Computer Science* **168**(1) 53–103.

Fiore, M. (1993) A Coinduction principle for Recursive Data Types Based on Bisimulation. In: *Proc. 8th LICS Conf.*, IEEE Computer Society Press.

Groote, J. (1990) Transition System Specifications with Negative Premises. In: Baeten, J. and Klop, J. (eds) Proc. CONCUR '90. *Springer Verlag Lecture Notes in Computer Science* **458** 332–341.

Groote, J. and Vaandrager, F. (1992) Structured Operational Semantics and Bisimulation as a Congruence. *Information and Computation* **100** 202–260.

Hennessy, M. and Milner, R. (1985) Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM* **32** 137–161.

Hirshfeld, Y., Jerrum, M. and Moller, F. (1996) A polynomial-time algorithm for deciding bisimulation equivalence of normed context-free processes. *Theoretical Computer Science* **158** 143–159.

Joyal, A., Nielsen, M. and Winskel, G. (1994) Bisimulation from Open Maps. Tech. rept. RS-94-7. BRICS. (Extract in *Proc. LICS'93*, IEEE Computer Society Press.)

Larsen, K. and Liu, X. (1991) Compositionality through an Operational Semantics of Contexts. *J. Logic Computat.* **1**(6) 761–795.

Milner, R. (1989) *Communication and Concurrency*, Prentice Hall.

Milner, R. (1991) The polyadic π-calculus: a tutorial. Tech. rept. ECS–LFCS–91–180. LFCS, Dept. of Comp. Sci., Edinburgh Univ. (Also in: Bauer, F. L., Brauer, W. and Schwichtenberg, H. (eds.) *Logic and Algebra of Specification*, Springer Verlag, 1993.)

Milner, R. and Tofte, M. (1991) Co-induction in relational semantics. *Theoretical Computer Science* **87** 209–220.

Milner, R., Parrow, J. and Walker, D. (1992) A Calculus of Mobile Processes (Parts I and II). *Information and Computation* **100** 1–77.

Parrow, J. and Sangiorgi, D. (1995) Algebraic Theories for Name-Passing Calculi. *Information and Computation* **120**(2) 174–197.

Pitts, A. (1994) A Co-induction Principle for Recursively Defined Domains. *Theoretical Computer Science* **124** 195–219.

Plotkin, G. (1981) A Structural Approach to Operational Semantics. DAIMI-FN-19. Computer Science Department, Aarhus University.

Rutten, J. and Turi, D. (1994) Initial Algebra and Final Coalgebra Semantics for Concurrency. In: Proc. Rex School/Symposium 1993 'A Decade of Concurrency – Reflexions and Perspectives'. *Springer Verlag Lecture Notes in Computer Science* **803**.

Sangiorgi, D. (1992) *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*, PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh.

Sangiorgi, D. (1995) Lazy functions and mobile processes. Tech. rept. RR-2515. INRIA-Sophia Antipolis. (To appear in 'Festschrift volume in honor of Robin Milner's 60th birthday', MIT Press.)

Sangiorgi, D. (1996) Locality and Non-interleaving Semantics in Calculi for Mobile Processes. *Theoretical Computer Science* **155** 39–83.

Sangiorgi, D. and Milner, R. (1992) The problem of 'Weak Bisimulation up to'. In: Cleveland, W. (ed) Proc. CONCUR'92. *Springer Verlag Lecture Notes in Computer Science* **630** 32–46.

Thomsen, B. (1990) *Calculi for Higher Order Communicating Systems*, Ph.D. thesis, Department of Computing, Imperial College.

Walker, D. (1994) Algebraic Proofs of Properties of Objects. Proc. ESOP'94. *Springer Verlag Lecture Notes in Computer Science* **788** 501–516.