On the Bisimulation Proof Method

Davide Sangiorgi

Laboratory for Foundations of Computer Science

Department of Computer Science, University of Edinburgh,

The King's Buildings, Edinburgh EH9 3JZ, U.K.

Email: sad@dcs.ed.ac.uk.

November 20, 1994

Abstract

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

We study generalisations of the method aimed at reducing the size of the relations to exhibit and hence relieving the proof work needed to establish bisimilarity results. We allow a relation $\mathcal{R}$ to progress to a different relation $\mathcal{F}(\mathcal{R})$, where $\mathcal{F}$ is a function on relations. Functions which can be safely used in this way (i.e., such that if $\mathcal{R}$ progresses to $\mathcal{F}(\mathcal{R})$, then $\mathcal{R}$ only includes pairs of bisimilar processes) are *sound*. We give a simple condition which ensures soundness. We show that the class of sound functions contains non-trivial functions and we prove closure properties of the class w.r.t. 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 which 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 and final coalgebras [Acz88, Abr91, HM85, RT94]. Bisimilarity has also been advocated outside concurrency theory; for instance, co-induction principles based on bisimilarity have been proposed to reason about equality between elements of recursively defined domains and data types [Fio93, Pit94].

We first consider bisimilarity on standard labelled transition systems: Their 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 $\mathcal{R}$ on processes s.t.

$$\text{if } (P, Q) \in \mathcal{R} \text{ and } P \xrightarrow{\mu} P', \text{ then there is } Q' \text{ s.t. } Q \xrightarrow{\mu} Q' \text{ and } (P', Q') \in \mathcal{R}\,. \qquad (*)$$

($\sim$ can also be viewed as the greatest fixed-point of a certain monotone function on relations, whose definition closely follows clause $(*)$.) A relation $\mathcal{R}$ 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 $\sim$ — 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 $\mathcal{R}$ 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 $\mathcal{R}$. We cannot discard any such pair of derivatives from $\mathcal{R}$, 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 which are needed; and clause $(*)$ must be checked on all pairs introduced.

As an example, take a CCS-like language including an action prefix operator $\alpha.\, P$, parallel composition $P_1 \mid P_2$, and replication $!\, P$ which, intuitively, stands for a countable number of copies of P in parallel. The transition rule for replication is

$$\frac{P \mid\, !\, P \xrightarrow{\mu} P'}{!\, P \xrightarrow{\mu} P'}\,.$$

A property that we naturally expect to hold is that duplication of replication has no behavioural effect. Thus, let P be a non-deadlocked process of the language, and suppose we want to prove $!\, P \mid\, !\, P \sim\, !\, P$. For this, we would like to use the *singleton* relation

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

But $\mathcal{R}$ is easily seen not to be a bisimulation relation. If we add pairs of processes to $\mathcal{R}$ so to make it into a bisimulation relation, then we end up with a relation which at least contains the *infinite* set

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

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

The study reported in this paper aims at relieving the work involved with the bisimulation proof method. We anticipate that on the previous example our proof techniques allow us to prove the property $!\,P \mid !\,P \sim !\,P$ simply using the singleton $\mathcal{R}$. We generalise the bisimulation proof method by relaxing the bare recursion in $(*)$. First, we introduce the notion of *progression*: A symmetric relation $\mathcal{R}$ progresses to a relation $\mathcal{S}$, abbreviated $\mathcal{R} \rightarrowtail \mathcal{S}$, if:

$$(P, Q) \in \mathcal{R} \text{ and } P \stackrel{\mu}{\longrightarrow} P' \text{ imply that there is } Q' \text{ s.t. } Q \stackrel{\mu}{\longrightarrow} Q' \text{ and } (P', Q') \in \mathcal{S}.$$

(Therefore, a relation $\mathcal{R}$ is a bisimulation relation iff $\mathcal{R} \rightarrowtail \mathcal{R}$ holds.) We examine progressions of the form $\mathcal{R} \rightarrowtail \mathcal{F}(\mathcal{R})$, where $\mathcal{F}$ is a function from relations to relations. We are interested in functions $\mathcal{F}$ which are *sound* w.r.t. $\sim$, i.e. s.t. $\mathcal{R} \rightarrowtail \mathcal{F}(\mathcal{R})$ implies $\mathcal{R} \subseteq \sim$. 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 $\mathcal{F}$ on relations. This condition requires that if $\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{R} \rightarrowtail \mathcal{S}$ hold, then $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{S})$ and $\mathcal{F}(\mathcal{R}) \rightarrowtail \mathcal{F}(\mathcal{S})$ must hold too. A very useful property about 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 $\mathcal{C}$ which gives us the closure of a relation $\mathcal{R}$ under contexts; i.e., $\mathcal{R} \rightarrowtail \mathcal{C}(\mathcal{R})$ holds if $(P, Q) \in \mathcal{R}$ and $P \stackrel{\mu}{\longrightarrow} P'$ imply that

$$\begin{array}{l} \text{there are processes } P'', Q'' \text{ and a context } C \text{ s.t.} \\ P' = C[P''],\ Q \stackrel{\mu}{\longrightarrow} C[Q''] \text{ and } (P'', Q'') \in \mathcal{R}\,. \end{array} \qquad (**)$$

Function $\mathcal{C}$ 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 $\mathcal{C}$. These conditions are met in familiar process algebras like ACP [BK84] and CCS [Mil89].

Examples of respectful functions easily derivable from our primitive ones are: The function which returns the transitive closure of a relation; the function which returns the closure of a relation under polyadic contexts (i.e., contexts which might have more than one hole); the function mapping a relation

$\mathcal{R}$ onto $\sim \mathcal{R} \sim$, where $\sim \mathcal{R} \sim$ is the composition of the three relations (this function gives us Milner's *bisimulation up-to* $\sim$ technique [Mil89]; 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 which 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 standardtheorems 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 [San94, BS94], namely the full abstraction of certain semantics of true-concurrent behavioural equivalences in the π-calculus.

Our interest for the π-calculus is motivated, besides by its relevance as a process algebra, by certain peculiarities of its transition system, which deviates from a standard system, like the one for CCS, in some important aspects: Firstly, 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. Secondly, π-calculus transition rules utilise alpha conversion and substitution on names ("name" is a synonymous for "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 $\mathcal{C}$ (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, then 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 which 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 already mentioned Milner's *bisimulation up-to* $\sim$ technique [Mil89], 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 *silent action*, is distinguished from the others and partially ignored in the bisimilarity clause) has been studied by Milner and Sangiorgi [SM92].

Two special cases of the up-to-context technique had been previously put forward: In [Cau90], Caucal defines a notion of *self-bisimulation* and uses it to establish a decidability result for the class of BPA processes, which can be viewed as the processes generated by a context-free grammar. Self-bisimulation is used to eliminate common prefixes and suffixes in the derivatives of two processes. Another form of up-to-context technique is Milner, Parrow and Walker' *bisimulation up-to restriction* [MPW92], with 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 [BD92], and Milner, Parrow and Walker [MPW92].

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 our proof techniques based on respectful functions to it. 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 reason 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 $(\mathcal{P}r, Act, \longrightarrow)$ with domain $\mathcal{P}r$, set of *actions* (or *labels*) Act and transition relation $\longrightarrow \subseteq \mathcal{P}r \times Act \times \mathcal{P}r$. We use P, Q and R to range over $\mathcal{P}r$ 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 $\mathcal{R}$ and $\mathcal{S}$ range over binary relations on processes, i.e., if $\wp$ denotes the powerset construct, then $\mathcal{R}$ and $\mathcal{S}$ are elements of $\wp(\mathcal{P}r \times \mathcal{P}r)$. The union of relations $\mathcal{R}$ and $\mathcal{S}$ is $\mathcal{R} \cup \mathcal{S}$, and their composition is $\mathcal{R}\mathcal{S}$ (i.e., $(P, P') \in \mathcal{R}\mathcal{S}$ holds if for some P'', both $(P, P'') \in \mathcal{R}$ and $(P'', P') \in \mathcal{S}$ hold). We often use the infix notation for relations; hence $P \mathcal{R} Q$ means $(P, Q) \in \mathcal{R}$. We use letters I and J for countable indexing sets in unions and sums.

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

1. *whenever $P \xrightarrow{\mu} P'$, there is Q' s.t. $Q \xrightarrow{\mu} Q'$ and $P' \mathcal{S} Q'$;*
2. *the converse, i.e., whenever $Q \xrightarrow{\mu} Q'$, there is P' s.t. $P \xrightarrow{\mu} P'$ and $P' \mathcal{S} Q'$.*

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

Definition 2.2 *$\mathcal{R}$ is* bisimulation relation *if $\mathcal{R}$ progresses to itself, i.e. $\mathcal{R} \rightarrowtail \mathcal{R}$ holds.*

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

Therefore, if $\mathcal{R}$ progresses to itself, then $\mathcal{R}$ 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 $\mathcal{R}$ which progresses to itself and which includes the pair of given processes.

However, self-progressions $\mathcal{R} \rightarrowtail \mathcal{R}$ are special cases of progressions, but not the only ones by which process bisimilarities can be inferred. In the paper, we look for general conditions on progressions which 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 $\mathcal{R} \rightarrowtail \mathcal{F}(\mathcal{R})$ where $\mathcal{F}$ is a function on relations, i.e. a function from $\wp(\mathcal{P}r \times \mathcal{P}r)$ to $\wp(\mathcal{P}r \times \mathcal{P}r)$. We call these *first-order functions*, briefly *functions*. Below, $\mathcal{F}$ and $\mathcal{G}$ range over such functions.

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

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

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

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

(a) $\mathcal{R} \subseteq \mathcal{S}$ implies $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{S})$, and

(b) $\mathcal{R} \rightarrowtail \mathcal{S}$ implies $\mathcal{F}(\mathcal{R}) \rightarrowtail \mathcal{F}(\mathcal{S})$,

then we would get a stronger definition (i.e, a stronger condition on $\mathcal{F}$) which would not capture important sound functions, like the function $\mathcal{C}$ 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 [Mil89, 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 why 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 $\mathcal{R} \stackrel{\text{def}}{=} \bigcup_{i \in I} \mathcal{R}_i$ and suppose for all $i \in I$ there is $j \in I$ s.t. $\mathcal{R}_i \rightarrowtail \mathcal{R}_j$ holds. Then $\mathcal{R}$ is a bisimulation relation.* $\square$

Lemma 2.9

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

2. *if, for all* $i \in I$, $\mathcal{R}_i \rightarrowtail \mathcal{S}$, *then also* $\left(\bigcup_{i\in I}\mathcal{R}_i\right) \rightarrowtail \mathcal{S}$. $\square$

Corollary 2.10 *If for all* $i \in I$ *there is* $j \in J$ *s.t.* $\mathcal{R}_i \rightarrowtail \mathcal{S}_j$ *holds, then also* $\left(\bigcup_{i\in I}\mathcal{R}_i\right) \rightarrowtail \left(\bigcup_{j\in J}\mathcal{S}_j\right)$. $\square$

Theorem 2.11 (soundness of respectful functions) *If* $\mathcal{F}$ *is respectful, then* $\mathcal{F}$ *is sound.*

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

$$\begin{array}{rcl} \mathcal{R}_0 & \stackrel{\text{def}}{=} & \mathcal{R}\,, \\ \mathcal{R}_{n+1} & \stackrel{\text{def}}{=} & \mathcal{F}(\mathcal{R}_n) \cup \mathcal{R}_n\,. \end{array}$$

Fact: For all $n \geq 0$, it holds that

1. $\mathcal{R}_n \subseteq \mathcal{R}_{n+1}$;
2. $\mathcal{R}_n \rightarrowtail \mathcal{R}_{n+1}$.

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

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

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

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

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

"whenever $\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{R} \rightarrowtail \mathcal{S}$ hold, then $\mathcal{F}(\mathcal{R}) \rightarrowtail \mathcal{F}(\mathcal{S})$ 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 which 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.

Two simple primitive respectful functions are the following:

$$\begin{array}{rcl} \mathcal{I}(\mathcal{R}) & \stackrel{\mathrm{def}}{=} & \mathcal{R} \\ \mathcal{U}(\mathcal{R}) & \stackrel{\mathrm{def}}{=} & \sim \end{array}$$

$\mathcal{I}$ is the identity function. $\mathcal{U}$ 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 which returns the closure of a relation under contexts (Section 2.1); the other is a function which 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 ($\frown$), so defined:

$$\begin{array}{rcl} (\mathcal{G}\circ\mathcal{F})(\mathcal{R}) & \stackrel{\mathrm{def}}{=} & \mathcal{G}(\mathcal{F}(\mathcal{R})) \\ (\bigcup_{i\in I}\mathcal{F}_i)(\mathcal{R}) & \stackrel{\mathrm{def}}{=} & \bigcup_{i\in I}(\mathcal{F}_i(\mathcal{R})) \\ (\mathcal{G}^\frown\mathcal{F})(\mathcal{R}) & \stackrel{\mathrm{def}}{=} & \mathcal{G}(\mathcal{R})\,\mathcal{F}(\mathcal{R}) = \{(P,P') \ :\ \text{for some } P'',\ (P,P'')\in\mathcal{G}(\mathcal{R}) \ \text{ and } (P'',P')\in\mathcal{F}(\mathcal{R})\,\} \end{array}$$

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:

$$\begin{array}{rcl} \text{for } n>0,\quad \mathcal{D}_n & \stackrel{\mathrm{def}}{=} & \mathcal{I}^\frown\ldots{}^\frown\mathcal{I}\,, \quad n \text{ times} \\ \mathcal{B} & \stackrel{\mathrm{def}}{=} & \mathcal{U}^\frown\mathcal{I}^\frown\mathcal{U} \\ \mathcal{T} & \stackrel{\mathrm{def}}{=} & \bigcup_{n>0}\mathcal{D}_n \end{array}$$

Function $\mathcal{D}_n$ takes a function $\mathcal{R}$ and makes the composition of $\mathcal{R}$ with itself n times. Function $\mathcal{B}$ represents the classical *bisimulation up-to* $\sim$, as in Milner's book [Mil89] (where the proof of the soundness of $\mathcal{B}$ is by checking that $\mathcal{R}\subseteq\mathcal{B}(\mathcal{R})$ and that $\mathcal{B}(\mathcal{R})$ is a bisimulation relation). Function $\mathcal{T}$ returns the transitive closure of a relation. The plain definitions of these functions are:

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

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

$$\begin{array}{rcl} \mathcal{F}^n(\mathcal{R}) & \stackrel{\mathrm{def}}{=} & \mathcal{F}((\ldots(\mathcal{F}(\mathcal{R}))\ldots))\,, \quad n \text{ times} \\ \mathcal{F}^*(\mathcal{R}) & \stackrel{\mathrm{def}}{=} & \bigcup_n\mathcal{F}^n(\mathcal{R}) \end{array}$$

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

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

Lemma 2.14 (composition) *Composition is a respectful constructor.*

PROOF: We have to show that if $\mathcal{F}$ and $\mathcal{G}$ are respectful, then also $\mathcal{G}\circ\mathcal{F}$ is respectful. If $\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{R} \rightarrowtail \mathcal{S}$ then, by respectfulness of $\mathcal{F}$, also $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{S})$ and $\mathcal{F}(\mathcal{R}) \rightarrowtail \mathcal{F}(\mathcal{S})$. From this, by respectfulness of $\mathcal{G}$, we derive $\mathcal{G}(\mathcal{F}(\mathcal{R})) \subseteq \mathcal{G}(\mathcal{F}(\mathcal{S}))$ and $\mathcal{G}(\mathcal{F}(\mathcal{R})) \rightarrowtail \mathcal{G}(\mathcal{F}(\mathcal{S}))$, which means $(\mathcal{G}\circ\mathcal{F})(\mathcal{R}) \subseteq (\mathcal{G}\circ\mathcal{F})(\mathcal{S})$ and $(\mathcal{G}\circ\mathcal{F})(\mathcal{R}) \rightarrowtail (\mathcal{G}\circ\mathcal{F})(\mathcal{S})$. $\square$

Lemma 2.15 (union) *Union is a respectful constructor.*

PROOF: We have to show that if, for all $i \in I$, $\mathcal{F}_i$ is respectful, then also $\bigcup_{i\in I} \mathcal{F}_i$ is respectful. Suppose $\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{R} \rightarrowtail \mathcal{S}$. For all $i \in I$, $\mathcal{F}_i$ is respectful, hence $\mathcal{F}_i(\mathcal{R}) \subseteq \mathcal{F}_i(\mathcal{S})$ and $\mathcal{F}_i(\mathcal{R}) \rightarrowtail \mathcal{F}_i(\mathcal{S})$ hold. From the former, we derive $\bigcup_{i\in I} \mathcal{F}_i(\mathcal{R}) \subseteq \bigcup_{i\in I} \mathcal{F}_i(\mathcal{S})$, and from the latter plus Corollary 2.10, we get $\bigcup_{i\in I} \mathcal{F}_i(\mathcal{R}) \rightarrowtail \bigcup_{i\in I} \mathcal{F}_i(\mathcal{S})$; that is, $(\bigcup_{i\in I} \mathcal{F}_i)(\mathcal{R}) \subseteq (\bigcup_{i\in I} \mathcal{F}_i)(\mathcal{S})$ and $(\bigcup_{i\in I} \mathcal{F}_i)(\mathcal{R}) \rightarrowtail (\bigcup_{i\in I} \mathcal{F}_i)(\mathcal{S})$. $\square$

Lemma 2.16 (chaining) *Chaining is a respectful constructor.*

PROOF: Suppose $\mathcal{F}$ and $\mathcal{G}$ are respectful. We check that also $\mathcal{G}^\frown\mathcal{F}$ is respectful. Suppose $\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{R} \rightarrowtail \mathcal{S}$. Then $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{S})$ and $\mathcal{G}(\mathcal{R}) \subseteq \mathcal{G}(\mathcal{S})$, which gives $(\mathcal{G}^\frown\mathcal{F})(\mathcal{R}) \subseteq (\mathcal{G}^\frown\mathcal{F})(\mathcal{S})$. We also have to check that $(\mathcal{G}^\frown\mathcal{F})(\mathcal{R}) \rightarrowtail (\mathcal{G}^\frown\mathcal{F})(\mathcal{S})$. Take $(P, P') \in (\mathcal{G}^\frown\mathcal{F})(\mathcal{R})$ with $P \xrightarrow{\mu} P_1$. If $(P, P') \in (\mathcal{G}^\frown\mathcal{F})(\mathcal{R})$, then there is P'' s.t. $(P, P'') \in \mathcal{G}(\mathcal{R})$ and $(P'', P') \in \mathcal{F}(\mathcal{S})$; moreover, since by respectfulness of $\mathcal{G}$ and $\mathcal{F}$ it holds that $\mathcal{G}(\mathcal{R}) \rightarrowtail \mathcal{G}(\mathcal{S})$ and $\mathcal{F}(\mathcal{R}) \rightarrowtail \mathcal{F}(\mathcal{S})$, for some P_1'' and P_1' the following diagram commutes:

$$\begin{array}{ccccc}
P & \mathcal{G}(\mathcal{R}) & P'' & \mathcal{F}(\mathcal{R}) & P' \\
{\scriptstyle\mu}\downarrow & & {\scriptstyle\mu}\downarrow & & {\scriptstyle\mu}\downarrow \\
P_1 & \mathcal{G}(\mathcal{S}) & P_1'' & \mathcal{F}(\mathcal{S}) & P_1'
\end{array}$$

This shows that $(P_1, P_1') \in (\mathcal{G}^\frown\mathcal{F})(\mathcal{S})$. In a symmetric way, one can show that if $P' \xrightarrow{\mu} P_1'$, then there is P_1 s.t. $P \xrightarrow{\mu} P_1$ and $(P_1, P_1') \in (\mathcal{G}^\frown\mathcal{F})(\mathcal{S})$. We conclude that $(\mathcal{G}^\frown\mathcal{F})(\mathcal{R}) \rightarrowtail (\mathcal{G}^\frown\mathcal{F})(\mathcal{S})$. $\square$

We saw that functions $\mathcal{B}$, $\mathcal{D}_n$ and $\mathcal{T}$, and constructors $\mathcal{F}^n$ and $\mathcal{F}^*$ are definable in terms of the primitive functions $\mathcal{I}$ and $\mathcal{U}$, and of the primitive constructors composition, chaining and union. Therefore, as a consequence of Lemmas 2.13–2.16, these derived functions and constructors are respectful.

2.1 Closure of a relation under contexts

We now consider the case — 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 $\mathcal{P}r_\Sigma$, is the least set of strings which satisfy :

- if f is an operator in Σ with arity 0, then f is in $\mathcal{P}r_\Sigma$;
- if f is an operator in Σ with arity $n > 0$, and $t_1, \ldots, t_n$ are already in $\mathcal{P}r_\Sigma$, then $f(t_1, \ldots, t_n)$ is in $\mathcal{P}r_\Sigma$.

Thus, having a signature Σ, the process language is $\mathcal{P}r_\Sigma$ and a process is an element of $\mathcal{P}r_\Sigma$.

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

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

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

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

Definition 2.17 *A set Cont of Σ-contexts is a* faithful context-set *if for all $C \in$ Cont and $P \in \mathcal{P}r_\Sigma$ whenever $C[P] \xrightarrow{\mu} R$, there exist $C' \in$ Cont s.t. either*

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

(b) there are $P' \in \mathcal{P}r_\Sigma$ and $\lambda \in Act$ s.t. $P \xrightarrow{\lambda} P'$ and $R = C'[P']$ and, moreover, for all $Q, Q' \in \mathcal{P}r_\Sigma$ s.t. $Q \xrightarrow{\lambda} Q'$, 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 2.18 The use of Definition 2.17 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 $P' \in \mathcal{P}r_\Sigma$ and $\widehat{\lambda}$ s.t. $P \xrightarrow{\widehat{\lambda}} P'$ and $R = C'[P']$ and, moreover, for all $Q, Q' \in \mathcal{P}r_\Sigma$ s.t. $Q \xrightarrow{\widehat{\lambda}} Q'$ 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 Xinxin *1-to-1 contexts* [LX91] (1-to-1 meaning that these contexts have exactly one hole and that they produce one action at a time).

The transition relation for the processes of the language generated by a signature Σ can be defined structurally [Plo81], assigning a set of *transition rules* to each symbol in Σ. A possible format for such transition rules is the one below. We call it the *unary De Simone format over* Σ; we shall often just call it *De Simone format.* It is a simplified version of the format introduced by De Simone [DS85] (our main restriction is that only one action at a time is observable). In rule (3) below, X_r, $1 \le r \le n$, and Y_j, $j \in J$, are metavariables which are instantiated with processes when the rule is applied.

Definition 2.19 (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, $1 \le r \le n$, and Y_j, $j \in J$, are distinct variables;*
- *t is a term in $\mathcal{P}r_\Sigma(X'_1, \ldots, X'_n)$, where for all $1 \le r \le n$, 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 2.20 ((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. *$C \in \mathcal{P}r_\Sigma$ (i.e., C is a process), or*
2. *$C = [\cdot]$, or*
3. *$C = f(P_1, \ldots, P_{i-1}, C', P_{i+1}, \ldots, P_n)$, where*
   - *$f \in \Sigma'$,*
   - *n is the arity of f,*
   - *$1 \le i \le n$,*
   - *$P_r \in \mathcal{P}r_\Sigma$ for $r \in \{1, \ldots, n\} - \{i\}$,*
   - *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' (i.e., 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 2.21 *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 2.17 proceeding by induction on the structure of C. The basic case, when $C \in \mathcal{P}r_\Sigma$ or $\Sigma = [\cdot]$, is trivial.

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

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

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'; 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' s.t.

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

and moreover, for all $Q, Q' \in \mathcal{P}r_\Sigma$ with $Q \xrightarrow{\widehat{\lambda}} Q'$, also

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

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

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

Summarising, we have found that if $C[P] \xrightarrow{\mu} R$, then there are P', $\widehat{\lambda}$ and a (Σ, Σ')-context D s.t. $P \xrightarrow{\widehat{\lambda}} P'$, $R = D[P']$ and for all $Q, Q' \in \mathcal{P}r_\Sigma$ with $Q \xrightarrow{\widehat{\lambda}} Q'$, also $C[Q] \xrightarrow{\mu} D[Q']$. This concludes the proof. $\square$

Corollary 2.22 *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 2.21. $\square$

Corollary 2.22 applies to well-know process algebras like CCS (see Lemma 3.1) and ACP. The De Simone format excludes, for instance, operators which, in order to release some action, may require the release of more than one action from some of their arguments (i.e., using the terminology in [GV92], 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 [BIM88, Gro90]. 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 the 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 $\mathcal{C}_\Sigma$ in (2) as $\mathcal{C}$.

Lemma 2.23 (closure under contexts) *The function $\mathcal{C}$ is respectful.*

PROOF: Suppose $\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{R} \rightarrowtail \mathcal{S}$. Clearly, also $\mathcal{C}(\mathcal{R}) \subseteq \mathcal{C}(\mathcal{S})$. Thus, we only have to prove $\mathcal{C}(\mathcal{R}) \rightarrowtail \mathcal{C}(\mathcal{S})$. For this, we have to show that if $P \mathcal{R} Q$ holds, C is a faithful context and $C[P] \xrightarrow{\mu} P''$, then there are P', Q' and a faithful context C' s.t. $P'' = C'[P']$, $Q \xrightarrow{\mu} C'[Q']$ and $P' \mathcal{S} Q'$. 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 $\mathcal{R} \rightarrowtail \mathcal{S}$ and $\mathcal{R} \subseteq \mathcal{S}$, for some Q' the diagram

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \widehat{\lambda} \downarrow & & \widehat{\lambda} \downarrow \\ P' & \mathcal{S} & Q' \end{array}$$

commutes. (Note that the hypothesis $\mathcal{R} \subseteq \mathcal{S}$ 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

$$\begin{array}{ccc} C[P] & \mathcal{C}(\mathcal{R}) & C[Q] \\ \mu \downarrow & & \mu \downarrow \\ C'[P'] & \mathcal{C}(\mathcal{S}) & C'[Q'] \end{array}$$

commutes, and concludes the proof. $\square$

In the sequel, we often abbreviate $\mathcal{C}(\mathcal{R})$ as $\mathcal{R}^{\mathcal{C}}$ and $\mathcal{T}(\mathcal{R})$ as $\mathcal{R}^{\mathcal{T}}$ (that is, $\mathcal{R}^{\mathcal{C}}$ is the closure of $\mathcal{R}$ under contexts and $\mathcal{R}^{\mathcal{T}}$ is the transitive closure of $\mathcal{R}$). In applications of our proof techniques, we shall often employ the sound function $\sim (-^{\mathcal{C}})^{\mathcal{T}} \sim$, which maps a relation $\mathcal{R}$ onto the relation $\sim (\mathcal{R}^{\mathcal{C}})^{\mathcal{T}} \sim$.

3 CCS: Operational semantics and proof techniques

We first give a brief synopsis of the section. We review the syntax and the operational semantics of CCS. A quick inspection at 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 that the one in [Mil89], 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,

pre:	$\alpha.\, P \xrightarrow{\alpha} P$	sum:	$\dfrac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}$
par:	$\dfrac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}$	com:	$\dfrac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$
res:	$\dfrac{P \xrightarrow{\mu} P'}{\boldsymbol{\nu}\, a\, P \xrightarrow{\mu} \boldsymbol{\nu}\, a\, P'}\ \mu \neq a, \overline{a}$	const:	$\dfrac{P \xrightarrow{\mu} P'}{A \xrightarrow{\mu} P'}$ if $A \stackrel{\mathrm{def}}{=} P$

Table 1: The transition system for CCS

sum, restriction, inaction, and constants:

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

Following π-calculus syntax (Section 4), we use $\boldsymbol{\nu}$ for restriction ($\boldsymbol{\nu}\, a\, P$ is normally written $P \backslash a$ in CCS), and we omit the relabeling 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. It is supposed that for each constant A there is a defining equation of the form $A \stackrel{\mathrm{def}}{=} P$. We refer to [Mil89] for details on the operators of the calculus. Sometimes, we use $\stackrel{\mathrm{def}}{=}$ 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 $\mathcal{P}r$ is the class of all CCS processes.

The transition system describing the operational semantics of CCS process is shown in Table 1. 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 for analogy with the π-calculus, in which the alphabets for prefixes and actions are different.

3.2 Our proof techniques in CCS

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

Lemma 3.1 *All CCS contexts are faithful.*

Proof: The CCS language can be described with the signature $\Sigma \stackrel{\mathrm{def}}{=} \{a.\,, \overline{a}.\,, \tau.\,, \mid, \boldsymbol{\nu}\,, +\,, A \;:\; a \in$ *Names*, and $A \in$ *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 2.22, all CCS contexts are faithful. $\square$

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

$$\mathcal{C}(\mathcal{R}) \stackrel{\mathrm{def}}{=} \bigcup_{C} \{(C[P], C[Q]) \;:\; (P, Q) \in \mathcal{R}\}\,.$$

Lemmas 3.1, 2.23 and Theorem 2.11 ensure the soundness of $\mathcal{C}$.

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

An interesting example of application of our proof techniques to CCS is the proof of uniqueness of solutions of equations, as from Milner's book [Mil89]. This result says that if a context C obeys certain conditions, then all processes P which 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, $\widetilde{P} \mathrel{\mathcal{R}} \widetilde{Q}$ means that $P_i \mathrel{\mathcal{R}} Q_i$, for each component of vectors $\widetilde{P}$ and $\widetilde{Q}$. For notational convenience, in this section we work with polyadic contexts, i.e., contexts which may contain an arbitrary number of different holes $[\cdot]_1, \ldots, [\cdot]_n$, and, moreover, each of these holes may appear more than once. If C contains at most holes $[\cdot]_1, \ldots, [\cdot]_n$, 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 $\mathcal{C}$. 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 $(P_i, Q_i) \in \mathcal{R}$, $i \leq i \leq n$, and C is an n-ary context, then* $(C[P_1, \ldots, P_n], C[Q_1, \ldots, Q_n]) \in \left(\mathcal{R}^{\mathcal{C}}\right)^{\mathcal{T}}$.

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

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

Hence also

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

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

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 (Lemma 4.13 in [Mil89]) *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: Simple induction on the structure of C. Intuitively, since C is weakly guarded, the processes which fill the holes of C do not contribute to the first action produced. □

Lemma 3.4 (Proposition 4.14(1) in [Mil89]) *If $A \stackrel{\text{def}}{=} P$, then $A \sim P$.*

PROOF: Immediate from the transition rule for constants. $\square$

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

Proposition 3.5 (unique solution of equations, Proposition 4.14(2) in [Mil89]) *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

$$\mathcal{R} \stackrel{\text{def}}{=} \{(P_i, Q_i) \; : \; 1 \leq i \leq n\} \, ,$$

and suppose $P_i \xrightarrow{\mu} P_i'$ (the case of a move from Q_i is symmetric). From Lemmas 3.3 and 3.4 we deduce that there are C_i' and Q_i' s.t. the following two diagrams commute:

$$\begin{array}{ccc} P_i & \sim & C_i[\widetilde{P}] \\ \mu \downarrow & & \mu \downarrow \\ P_i' & \sim & C_i'[\widetilde{P}] \end{array} \qquad\qquad \begin{array}{ccc} C_i[\widetilde{Q}] & \sim & Q_i \\ \mu \downarrow & & \mu \downarrow \\ C_i'[\widetilde{Q}] & \sim & Q_i' \end{array}$$

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

In the proof of Proposition 3.5, the cardinality of the relation $\mathcal{R}$ 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), then $\mathcal{R}$ consists of *one* only pair. For the proof of Proposition 3.5, Milner [Mil89] shows that

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

is a bisimulation up-to $\sim$ (i.e., $\mathcal{R}' \rightarrowtail \sim \mathcal{R}' \sim$ holds), proceeding on 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 [Mil91, San92, San94, BS94, Wal94].

We briefly review the syntax and the operational semantics of the π-calculus. We refer to [MPW92, Mil91] for more details. We maintain the notations introduced for CCS, which will not be repeated. W.r.t. 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 $\boldsymbol{\nu}\, b\, \overline{a}b. P$.

`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`:	$\dfrac{P\{\widetilde{b}/\widetilde{c}\} \xrightarrow{\mu} P'}{A\langle\widetilde{b}\rangle \xrightarrow{\mu} P'}$ if $A \stackrel{\mathrm{def}}{=} (\widetilde{c})P$	`MATCH`:	$\dfrac{P \xrightarrow{\mu} P'}{[a=a]P \xrightarrow{\mu} P'}$

Table 2: The transition system for the π-calculus

We admit bound output 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 $A \stackrel{\mathrm{def}}{=} (\widetilde{c})P$, which can be thought as a procedure with formal parameters $\tilde{c}$; 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 denote by $\mathrm{fn}(P)$ the set of free names of P. For notational simplicity, we impose that a process only has a finite number of free names and that in a constant definition $A \stackrel{\mathrm{def}}{=} (\widetilde{c})P$, vector $\widetilde{c}$ contains all 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 which only differ on the choice of the bound names.* The symbol $=$ will mean "syntactic identity modulo alpha conversion". We denote by $\mathcal{P}r_\pi$ the class of all π-calculus processes.

A *substitution* is a function from names to names. We use the standard notation for substitutions, e.g. $\{x/y\}$ is the function which 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 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 evidence 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 w.r.t 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, respectively written $bn(\mu)$ and $fn(\mu)$, are defined accordingly. The *names* of μ, briefly $n(\mu)$, are $bn(\mu) \cup fn(\mu)$. We work up to alpha conversion on processes also in transition systems, for which *alpha convertible agents are deemed to have the same transitions.*

The reader familiar with the π-calculus would have noticed that we are using an *early transition system* [San92] — since the bound names of an input are instantiated as soon as possible, in the input rule — as opposed to a *late transition system* [MPW92, Mil91] — 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* [FMQ94]. Our "early" choice is not critical for the results we shall present, although some definitions (like that of function $\mathcal{C}_\Sigma$ 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 follows:

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

- *whenever $P \xrightarrow{\mu} P'$ with $\mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$, there is Q' s.t. $Q \xrightarrow{\mu} Q'$ and $P' \mathcal{S} Q'$,*

and the symmetric clause, on the actions by Q.

The definitions of a bisimulation relation and of bisimilarity are as those for CCS-like languages, in Section 2. However, in contrast with CCS, in π-calculus bisimilarity is not a full congruence, since 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* $\sim^c$ induced by $\sim$ [MPW92].

Definition 4.2 (congruence induced by $\sim$) *We set $P \sim^c Q$, pronounced "P and Q are congruent", if $P\sigma \sim Q\sigma$, for all substitutions σ.*

5 Proof techniques for the π-calculus

W.r.t CCS, in the π-calculus actions are more structured — there is also an object part — 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 $\mathcal{C}$ (closure under contexts) and the proof of its respectfulness. The Definition 2.17 of faithful contexts — on which the definition of $\mathcal{C}$ is based — is limitative in the π-calculus, because it does not capture all contexts. For instance, $C \stackrel{\text{def}}{=} a(x).\,[\cdot]$ is not faithful: If $P \stackrel{\text{def}}{=} x(y).\,\mathbf{0}$, then $C[P] \xrightarrow{ab} P\{b/x\}$, but there is no $\widehat{\lambda}$ s.t. $P \xrightarrow{\widehat{\lambda}} P\{b/x\}$. 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 $\mathcal{C}$ and the proof of its respectfulness for the π-calculus. We call the new function $\mathcal{C}_\pi$. We recall that a context C is guarded if the possible occurrence of the hole $[\cdot]$ is within a subexpression of C of the form $\alpha.\,C'$; otherwise C is non-guarded. We set:

$$\begin{aligned} \mathcal{C}_\pi(\mathcal{R}) \stackrel{\text{def}}{=} & \bigcup_{C \text{ non-guarded}} \{(C[P], C[Q]) \,:\, (P,Q) \in \mathcal{R}\} \quad \bigcup \\ & \bigcup_{C \text{ guarded}} \{(C[P], C[Q]) \,:\, (P\sigma, Q\sigma) \in \mathcal{R}, \text{ for all substitutions } \sigma\} \end{aligned}$$

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

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

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

Proof: Suppose that $\mathcal{R} \subseteq \mathcal{S}$ and $\mathcal{R} \rightarrowtail \mathcal{S}$. Then, clearly, $\mathcal{C}_\pi(\mathcal{R}) \subseteq \mathcal{C}_\pi(\mathcal{S})$. We also have to check that $\mathcal{C}_\pi(\mathcal{R}) \rightarrowtail \mathcal{C}_\pi(\mathcal{S})$ holds. For this, given $(C[P], C[Q]) \in \mathcal{C}_\pi(\mathcal{R})$ with $C[P] \xrightarrow{\mu} R$, we show that there are C', P' and Q' s.t.

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

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 $\mathcal{R} \rightarrowtail \mathcal{S}$.

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

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

We 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 remaining two are similar. So, suppose

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

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

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

From (8) and (7), by induction, there are C_1', P' and Q' s.t.

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

Moreover, using rule **PAR**, we have

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

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

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

If $C' \stackrel{\text{def}}{=} C_1 \mid T$, 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 $C = A\langle\widetilde{b}\rangle$, or $C = [a = b]C'$.

These cases are easy.

$\square$

A useful fact, which derives from the definition (4.2) of the congruence $\sim^{c}$, is the following:

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

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

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 $a, b \in V$, it holds that $\sigma(a) = \sigma(b)$ 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 which allows us to work up to injective substitutions on names. It is called $\mathcal{S}ub$ and is so defined:

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

We show that $\mathcal{S}ub$ is respectful. We first need a lemma:

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

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 first define the function ρ. Let W, W^- and V^- be the following sets of names:

$$\begin{array}{rcl} W & \stackrel{\text{def}}{=} & \{\sigma(a) \ : \ a \in V\} \\ W^- & \stackrel{\text{def}}{=} & W - V \ = \ \{a \ : \ a \in W \text{ and } a \notin V\} \\ V^- & \stackrel{\text{def}}{=} & V - W \ = \ \{a \ : \ a \in V \text{ and } a \notin W\} \end{array}$$

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

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

The substitution ρ is so specified:

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

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. Indeed, ρ is a bijection on names, and hence we can consider its inverse ρ^{-1}.

Now we prove clause (1) of the lemma by transition induction. The proof of clause (2) is similar and is omitted. Below, by alpha conversion we can assume that if $x \in \mathrm{bn}(P)$, then $x \notin V \cup W$; hence $\rho(x) = x$, and also $\rho(y) \neq x$, for all $y \neq x$.

Case 1 $P = a(x).\, Q$, $\mu = ab$, $P' = Q\{{}^{b}\!/\!{}_{x}\}$.

If $\rho(a) = a'$ and $\rho(b) = b'$, then we have $P\rho = a'(x).\, Q\rho$ and $P\rho \xrightarrow{a'b'} Q\rho\{{}^{b'}\!/\!{}_{x}\} = Q\{{}^{b}\!/\!{}_{x}\}\rho = P'\rho$.

Case 2 $P = \overline{a}b.\, Q$, or $P = \overline{a}(b).\, Q$, or $P = \tau.\, Q$, or $P = Q_1 + Q_2$, or $P = \boldsymbol{\nu}\, a\, Q$, or $P = [a = b]Q$.

Easy.

Case 3 $P = Q_1 \mid Q_2$.

We only consider the case of rule **PAR**, when Q_1 performs the action:

$$Q_1 \mid Q_2 \xrightarrow{\mu} Q_1' \mid Q_2\,, \text{ for some } Q_1' \text{ s.t. } Q_1 \xrightarrow{\mu} Q_1'\,.$$

By induction, $Q_1\rho \xrightarrow{\mu\rho} Q_1'\rho$, hence

$$(Q_1 \mid Q_2)\rho = Q_1\rho \mid Q_2\rho \xrightarrow{\mu\rho} Q_1'\rho \mid Q_2\rho = (Q_1' \mid Q_2)\rho\,.$$

Case 4 $P = A\langle\widetilde{b}\rangle$, for $A \stackrel{\text{def}}{=} (\widetilde{c})Q$.

The last inference rule applied is

$$\frac{Q\{\widetilde{b}/\widetilde{c}\} \xrightarrow{\mu} P'}{A\langle\widetilde{b}\rangle \xrightarrow{\mu} P'} .$$

By induction, $Q\{\widetilde{b}/\widetilde{c}\}\rho \xrightarrow{\mu\rho} P'\rho$. Since $\mathrm{fn}(Q) \subseteq \widetilde{c}$, we have $Q\{\widetilde{b}/\widetilde{c}\}\rho = Q\{\rho(\widetilde{b})/\widetilde{c}\}$ (where ρ is defined on tuples componentwise), and therefore we can infer, using rule **CONST**:

$$A\langle\widetilde{b}\rangle\rho = A\langle\rho(\widetilde{b})\rangle \xrightarrow{\mu\rho} P'\rho .$$

$\square$

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

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

Take $(P\sigma, Q\sigma) \in \mathcal{S}ub(\mathcal{R})$, for some $(P, Q) \in \mathcal{R}$ and σ injective on $\mathrm{fn}(P, Q)$. Suppose $P\sigma \xrightarrow{\mu'} P''$. We have to find Q'' s.t.

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

Let ρ be the injective function which 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' s.t. $P \xrightarrow{\mu} P'$, $\mu' = \mu\rho$ and $P'' = P'\rho$. Since $\mathcal{R} \rightarrowtail \mathcal{S}$, the diagram

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ \mu \downarrow & & \mu \downarrow \\ P' & \mathcal{S} & Q' \end{array}$$

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

$$\begin{array}{ccc} P\rho & \mathcal{S}ub(\mathcal{R}) & Q\rho \\ \mu\rho \downarrow & & \mu\rho \downarrow \\ P'\rho & \mathcal{S}ub(\mathcal{S}) & Q'\rho \end{array}$$

commutes too. For $Q'' \stackrel{\text{def}}{=} Q'\rho$, since $P'\rho = P''$, $Q\rho = Q\sigma$ and $\mu\rho = \mu'$, this proves (11). $\square$

Having proved that $\mathcal{S}ub$ is respectful, we know that it is a sound function and, moreover, we can safely combine it with other respectful functions, according to the modalities indicated 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 (i.e., function $\mathcal{S}ub$ 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 $\mathcal{C}_\pi$.

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

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

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:

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

Note that $\mathcal{R}$ contains three unions which 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 $\mathcal{S}ub$ we can prove $P \sim Q$ by simply taking

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

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

We can do better than $\mathcal{R}'$ using a combination of function $\mathcal{S}ub$ 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 which 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 showed for CCS, so in the π-calculus the function $\sim(-^{\mathcal{C}_\pi})^{\mathcal{T}}\sim$ can be used 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 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 w.r.t. the congruence $\sim^{c}$, 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 countable 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 other way round holds too [Mil91].

The transition rule for replication is

$$\texttt{REP:}\ \frac{P \mid\, !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}.$$

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 this subsection is new. It says that, if we choose to have replication in the grammar of the π-calculus, then 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 $\sim^{c}$. 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, first proved by Milner [Mil91].

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 $\mathcal{C}_\pi$ and the proof of its soundness (Proposition 5.2) remain unchanged if in the definition of $\mathcal{C}_\pi$ 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

$$\texttt{REP-NOR:}\ \frac{\alpha.P \xrightarrow{\mu} P'}{!\,\alpha.P \xrightarrow{\mu} P' \mid\, !\,\alpha.\,P}$$

or, alternatively, the two rules

$$\texttt{REP-INP:}\ \ !\,a(x).\,P \xrightarrow{ab} P\{b/x\} \mid\, !\,a(x).\,P \qquad \texttt{REP-PRE:}\ \ !\,\alpha.\,P \xrightarrow{\alpha} P \mid\, !\,\alpha.\,P\,,\ \text{if } \alpha \text{ is not an input}$$

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 $P \xrightarrow{\mu} P''$ consume 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 instance 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. $P \mid\, !\,P \sim^{c}\, !\,P$;

2. $!(P \mid Q) \sim^{c} !P \mid !Q$*;*

3. $!(P+Q) \sim^{c} !(P \mid Q)$*.*

Proof: Assertion (1) is trivial: Due to 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 $\mathcal{R} \rightarrowtail \sim \mathcal{R}^{\mathcal{C}_\pi} \sim$. For (2), the relation to use is

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

and for (3) it is

$$\mathcal{R}_3 \stackrel{\text{def}}{=} \bigcup_{P,Q} \{(!(P+Q), !(P \mid Q))\}$$

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

We consider the proof of $\mathcal{R}_3 \rightarrowtail \sim (\mathcal{R}_3)^{\mathcal{C}_\pi} \sim$ 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 s.t.

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

This shows that $(T_1, T_2) \in \sim \mathcal{R}_3{}^{\mathcal{C}_\pi} \sim$, and we are done. Note that we use function $\mathcal{C}_\pi$ to cancel context $R \mid [\cdot]$; according to the definition of $\mathcal{C}_\pi$, this is legitimate because $R \mid [\cdot]$ is a non-guarded context (actually, in the case of relation $\mathcal{R}_3$ we could cancel *any* context because $\mathcal{R}_3$ 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 $P+Q$ which is used, and that it performs an input at a of the free name b. Thus we have, for some T_1' and P' s.t. $P \xrightarrow{ab} P'$:

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

By the inductive 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),

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

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 $R \stackrel{\text{def}}{=} P' \mid R'$. 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 of proving bisimilarity — rather then congruence — results, then relations $\mathcal{R}_2$ and $\mathcal{R}_3$ would only contain *one* pair of processes. For instance, $\mathcal{R}_3$ would be

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

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

$$\mathcal{R}_3' \stackrel{\text{def}}{=} \bigcup_{P,Q,R} \{(R \mid !(P+Q), R \mid !(P \mid Q))\}$$

($\mathcal{R}_3'$ progresses to $\sim \mathcal{R}_3' \sim$). Having $\mathcal{R}_3'$ in place of $\mathcal{R}_3$ 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 the introductory Section 1, using the singleton relation $\mathcal{R} \stackrel{\text{def}}{=} \{!P \mid !P, !P\}$, and showing that $\mathcal{R} \rightarrowtail \sim(\mathcal{R})^{\mathcal{C}_\pi} \sim$ 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 [PS93], and which for easy 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 (possible empty) match sequences; thus if M is $[a=b][c=d]$, then MP is $[a=b][c=d]P$.

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

PROOF: 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

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

Let $P \stackrel{\mathrm{def}}{=} \sum_i M_i\alpha_i.\,P_i$ and $Q \stackrel{\mathrm{def}}{=} \sum_j N_j\alpha_j.\,Q_j$ where no α_i (resp. β_i) binds a name free in Q (resp. P). Then infer:

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

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

$P = [a = b]P'$, use induction plus the law **L7**. If $P = P_1 \mid P_2$, use induction plus the expansion law. If $P = \boldsymbol{\nu}\, a\, P'$ use induction plus the 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, i.e. $P = !\,P'$. By induction, $P' \sim^{c} \sum_{j \in J} M_j \alpha_j .\, P'_j$, and we can deduce:

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

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 s.t. $P \sim^{c} Q$.*

PROOF: By induction on the maximal number of nested replications in P. If P does not have replications, then 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

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

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

$$\begin{array}{cl} \sim^{c} & !\left(\prod_{i \in I} M_i \alpha_i .\, P''_i\right) \\ \sim^{c} & \prod_{i \in I} !\, M_i \alpha_i .\, P''_i \\ \sim^{c} & \prod_{i \in I} M_i\, !\, \alpha_i .\, P''_i \end{array}$$

which is a process with normalised replications. □

6.4 Distributivity properties of private replications

In [Mil91], Milner shows certain distributivity properties for private replications w.r.t. parallel composition and replication. The importance of these properties has emerged in different situations, like the correctness of the encodings of λ-calculus and higher-order calculi into the π-calculus [Mil91, San92] and in reasoning about data structures [Wal94].

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]

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

For the proof of these assertions, Milner [Mil91] uses relations $\mathcal{R}_1$ and $\mathcal{R}_2$, defined as below, and proves that they progress to $\sim \mathcal{R}_1 \sim$ and $\sim \mathcal{R}_2 \sim$, respectively. We call $\mathcal{N}$ be the set of all processes which contain name a free only as subject of output prefixes:

$$
\begin{aligned}
\mathcal{R}_1 &\stackrel{\mathrm{def}}{=} \bigcup_{P,Q,R\in\mathcal{N}} \Big\{\big(\boldsymbol{\nu}\,\widetilde{b}\,\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid P \mid Q), \boldsymbol{\nu}\,\widetilde{b}\,\big(\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid P) \mid \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid Q)\big)\big)\Big\}\\
\mathcal{R}_2 &\stackrel{\mathrm{def}}{=} \bigcup_{\alpha.P,Q,R\in\mathcal{N}} \Big\{\big(\boldsymbol{\nu}\,\widetilde{b}\,\boldsymbol{\nu}\, a\,\big(\,!\,a(x).\,R \mid \,!\,\alpha.P \mid Q\big), \boldsymbol{\nu}\,\widetilde{b}\,\big(\,!\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid \alpha.P) \mid \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid Q)\big)\big)\Big\}
\end{aligned}
$$

Since $\mathcal{R}_1$ and $\mathcal{R}_2$ are closed under substitutions on names, they give us $\sim^{\mathrm{c}}$ equalities (Corollary 5.3); and the assertions of the replication theorems follow for $\widetilde{b} = \emptyset$ and $Q = \mathbf{0}$.

The use of function $\mathcal{C}_\pi$ (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 $\mathcal{R}_1$, and take

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

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

$$
\mathcal{R}_2' \stackrel{\mathrm{def}}{=} \bigcup_{\alpha.P,R\in\mathcal{N}} \Big\{\big(\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid \,!\,\alpha.P), \,!\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid \alpha.P)\big)\Big\}\,.
$$

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

$$
\begin{aligned}
Q_1 &\stackrel{\mathrm{def}}{=} \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid \,!\,\alpha.\,P)\,,\\
Q_2 &\stackrel{\mathrm{def}}{=} \,!\boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid \alpha.\,P)\,.
\end{aligned}
$$

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

$$
\begin{aligned}
Q_1 &\xrightarrow{\;\tau\;} \boldsymbol{\nu}\, a\,(R\{b/x\} \mid \,!\,a(x).\,R \mid P \mid \,!\,\alpha.P) &&\stackrel{\mathrm{def}}{=} Q_1'\\
Q_2 &\xrightarrow{\;\tau\;} \boldsymbol{\nu}\, a\,(R\{b/x\} \mid \,!\,a(x).\,R \mid P) \mid Q_2 &&\stackrel{\mathrm{def}}{=} Q_2'\,.
\end{aligned}
$$

Now, let

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

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

$$
\begin{aligned}
Q_1' &\sim \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid R\{b/x\} \mid P \mid \,!\,\alpha.P)\\
&\sim \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid R\{b/x\} \mid P) \mid \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid \,!\,\alpha.P) &&= C[Q_1]\,,\\
Q_2' &\sim \boldsymbol{\nu}\, a\,(\,!\,a(x).\,R \mid R\{b/x\} \mid P) \mid Q_2 &&= C[Q_2]\,.
\end{aligned}
$$

This shows that $(Q_1', Q_2') \in \sim (\,\mathcal{R}_2'\,)^{\mathcal{C}_\pi}\!\sim$, and concludes the proof.

[1] 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 [Mil91]. Some justification for this simplification comes from Theorem 6.6.

7 Conclusions and further developments

In this paper, we have studied generalisations of the bisimulation proof method which 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 $\mathcal{R} \rightarrowtail \mathcal{R}$, and considered progressions of the form $\mathcal{R} \rightarrowtail \mathcal{F}(\mathcal{R})$, where $\mathcal{F}$ is a function on relations. The *sound* functions are those for which $\mathcal{R} \rightarrowtail \mathcal{F}(\mathcal{R})$ implies that $\mathcal{R}$ only contains pairs of bisimilar processes, for all $\mathcal{R}$. We have given a condition on functions, called *respectfulness*, which ensures soundness. We have showed that the class of respectful functions contains non-trivial functions and and that it enjoys closure properties w.r.t. 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 if the transition rules for the operators of the language are in unary De Simone format, then this technique is sound. The proof of this result uses the fact that the operators specifiable with transition rules in unary De Simone format preserve bisimilarity. There are other formats of transition rules which go beyond the De Simone format and which enjoy this property. For instance, Groote and Vaandrager' *tyft* format [GV92], and Bloom, Istrail and Meyer' *GSOS* format [BIM88]. It would be interesting to examine whether a soundness result for the up-to-context technique also holds for the *tyft* and the *GSOS* formats. We think that some constraints would have to be imposed on them: For instance one might have to disallow lookaheads greater than one in the *tyft* format (lookaheads greater than one allow the definition of operators which, in order to release some action, require the release of more than one actions from some of their arguments).

Most of the respectful functions $\mathcal{F}$ we have considered have the property that if a relation $\mathcal{R}$ progresses to $\mathcal{F}(\mathcal{R})$, then $\mathcal{F}(\mathcal{R})$ 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 of other useful respectful functions and function constructors, to be added to those we found.

In this paper, we 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 *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 which 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 analogous way. In the weak case it might also be less easy 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 [SM92]).

We believe that our proof techniques could be very advantageous in *higher-order calculi* like CHOCS [Tho90], or Higher-Order π-calculus [San92], i.e calculi in which terms can be exchanged in a communication. For instance, a few rather involved proofs in [San92], dealing with the Higher-Order π-calculus, should become simpler using some form of "bisimulation up-to context" (see Remark 6.6.18 in [San92]). Our proof techniques should also be useful in higher-order functional languages, for instance to reason about *applicative bisimilarity* of programs [Abr89].

The bisimulation proof method stems from the theory of fixed-points and the co-induction principle [Mil89, MT91]. On a complete lattice (i.e., a partial order with all joins) the co-induction principle says:

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

When the bisimilarity relation $\sim$ is interpreted as the greatest fixed-point of a certain continuous function on relations [Mil89, Section 4.6], this translate into saying that to prove $\mathcal{R} \subseteq\sim$ it suffices to prove that $\mathcal{R}$ is a bisimulation relation. We would like to see whether our study of the bisimulation proof method leads to 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 $\mathcal{F}$ as follows:

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

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 $\mathcal{F}$ is monotone and that $\mathcal{F}\circ\mathcal{G} < \mathcal{G}\circ\mathcal{F}$ (i.e., for all z, $(\mathcal{F}\circ\mathcal{G})(z) < (\mathcal{G}\circ\mathcal{F})(z)$). It is worth pointing out that if $\mathcal{F}$ is monotone, then the condition $\mathcal{F}\circ\mathcal{G} < \mathcal{G}\circ\mathcal{F}$ is the same as the condition "for all $z, y \in D$, $z < \mathcal{G}(y)$ implies $\mathcal{F}(z) < \mathcal{F}(\mathcal{G}(y))$". In terms of respectful functions for bisimilarity, this formulation would amount to having the same conditions of 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, David N. Turner and David Walker, whose comments helped me to improve the technical presentation.

This research has been supported by the ESPRIT BRA project 6454 "CONFER".

References

[Abr89] S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1989.

[Abr91] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991.

[Acz88] P. Aczel. *Non-well-funded Sets*. CSLI lecture notes; no. 14, 1988.

[BD92] M. Boreale and R. De Nicola. Testing equivalence for mobile processes. Tec. report SI-92/04, Dipartimento di Scienze dell'Informazione, Università degli studi di Roma "La Sapienza", 1992. To appear in *Information and Computation*.

[BIM88] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't' be traced: preliminary report. In *Conference Record of the 15th ACM Symposium on Principle of Programming Languages (POPL)*, pages 229–239, 1988.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

[BS94] M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the π-calculus. Technical Report ECS–LFCS–94–297, LFCS, Dept. of Comp. Sci., Edinburgh Univ., 1994.

[Cau90] D. Caucal. Graphes canoniques de graphes algébriques. *Informatique Théorique et Applications (RAIRO)*, 24(4):339–352, 1990.

[DS85] R. De Simone. Higher level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[Fio93] M. Fiore. A coinduction principle for recursive data types based on bisimulation. In *8th LICS Conf.* IEEE Computer Society Press, 1993.

[FMQ94] G. Ferrari, U. Montanari, and P. Quaglia. A π-calculus with explicit substitutions: the late semantics. In *Proc. MFCS'94*, Lecture Notes in Computer Science. Springer Verlag, 1994. To appear.

[Gro90] J.F. Groote. Transition system specifications with negative premises. In J.C.M. Baeten and J.W. Klop, editors, *Proc. CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 332–341, 1990.

[GV92] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, 1992.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

[LX91] K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. *J. Logic Computat.*, 1(6):761–795, 1991.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, Dept. of Comp. Sci., Edinburgh Univ., October 1991. Also in *Logic and Algebra of Specification*, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[MT91] R. Milner and M. Tofte. Co-induction in relational semantics. *Theoretical Computer Science*, 87:209–220, 1991.

[Pit94] A.M. Pitts. A co-induction principle for recursively defined domains. *Theoretical Computer Science*, 124:195–219, 1994.

[Plo81] G.D Plotkin. A structural approach to operational semantics. DAIMI-FN-19, Computer Science Department, Aarhus University, 1981.

[PS93] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. Technical Report ECS–LFCS–93–262, LFCS, Dept. of Comp. Sci., Edinburgh Univ., 1993. To appear in *Information and Computation.* Short version in *Proc. REX Summer School/Symposium 1993*, LNCS 803, Springer Verlag.

[RT94] J. Rutten and D. Turi. Initial algebra and final coalgebra semantics for concurrency. In *Proc. Rex School/Symposium 1993 "A Decade of Concurrency — Reflexions and Perspectives"*, volume 803 of *Lecture Notes in Computer Science.* Springer Verlag, 1994.

[San92] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.

[San94] D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. Technical Report ECS–LFCS–94–282, LFCS, Dept. of Comp. Sci., Edinburgh Univ., 1994. An extract appeared in *Proc. TACS '94*, Lecture Notes in Computer Science 789, Springer Verlag.

[SM92] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In W.R. Cleveland, editor, *Proceedings of CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer Verlag, 1992.

[Tho90] B. Thomsen. *Calculi for Higher Order Communicating Systems.* PhD thesis, Department of Computing, Imperial College, 1990.

[Wal94] D. Walker. Algebraic proofs of properties of objects. In *Proc. CAAP/ESOP'94*, Lecture Notes in Computer Science. Springer Verlag, 1994.