# On the Expressiveness of CCS-like Calculi

Pablo Giambiagi[1], Gerardo Schneider[2⋆], and Frank D. Valencia[2⋆⋆]

[1] KTH Royal Institute of Technology, IMIT, Electrum 229, 164 40 Kista, Sweden
{pgiamb@imit.kth.se}

[2] Uppsala University, Dept. of Computer Systems, Box 337, 751 05 Uppsala, Sweden
{gerardos@it.uu.se; frankv@it.uu.se}

**Abstract.** In the literature there are several CCS-like process calculi, or CCS variants, differing in the constructs for the specification of infinite behavior and in the scoping rules w.r.t. channel names. In this paper we study various representatives of these calculi based upon both their relative expressiveness and the decidability of *divergence* (i.e., the existence of a *divergent* computation). We regard any two calculi as being *equally expressive* iff for every process in each calculus, there exists a *weakly bisimilar* process in the other.

By providing *weak bisimilarity* preserving mappings among the various variants, we show that in the context of *relabeling-free* and *finite summation* calculi: (1) CCS with *parameterless* (or *constant*) definitions is equally expressive to the variant with *parametric* definitions. (2) The CCS variant with *replication* is equally expressive to that with *recursive expressions* and *static* scope. We also state that the divergence problem is undecidable for the calculi in (1) but decidable for those in (2). We obtain this from (un)decidability results by Busi, Gabbrielli and Zavattaro, and by showing the relevant mappings to be computable and to preserve divergence and its negation. From (1) and the well-known fact that parametric definitions can replace injective relabelings, we show that injective relabelings are redundant (i.e., derived) in CCS (which has constant definitions only).

## 1 Introduction

The study of concurrency is often conducted with the aid of process calculi. Undoubtedly CCS [8], a calculus for synchronous communication, remains as a standard representative. In fact, many foundational ideas in the theory of concurrency have grown out of this calculus.

Nevertheless, there are several variants of CCS, or CCS-like calculi, in the literature. This is reasonable as a variant may simplify the presentation of the calculus or be tailored to specific applications. Given two variants, a legitimate question is whether they are equally *expressive.* To answer this question one has to agree on what it means for one variant to be as expressive as the other. A natural way of doing this in CCS is by comparing w.r.t. some standard process equivalence such as *(weak) bisimilarity*: If for every process $P$ in one variant there is a $Q$ in the other variant such that $Q$ is (weakly) bisimilar to $P$ then we say that the latter variant is at least as expressive as the former.

⋆ Work supported by European project ADVANCE, Contract No. IST–1999–29082.
⋆⋆ Work supported by European project PROFUNDIS.

Another legitimate question, given a variant, is whether some fundamental property such as *divergence* (i.e., the existence of divergent computations) becomes simpler or harder to analyze.

In this paper, we study both the relative expressiveness w.r.t. weak bisimilarity and the decidability of divergence for various CCS-like calculi, or CCS variants. We shall focus upon two sources of variation found in the CCS literature: The constructs used to express infinite behavior and the way in which scoping of channel (port) names is dealt with. As for the constructs for finite behavior, in all the calculi we confine our attention to prefix, finite sums, restriction, and parallel composition. The calculi here studied can be described as follows:

- $\mathrm{CCS}_{\mathtt{k}}$: Infinite behavior is given by a *finite* set of *constant* (i.e., parameterless) definitions of the form $A \stackrel{\mathrm{def}}{=} P$. The calculus is essentially CCS [8] without relabelling nor infinite summations.
- $\mathrm{CCS}_{\mathtt{p}}$: Like $\mathrm{CCS}_{\mathtt{k}}$ but using *parametric definitions* of the form $A(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} P$. The calculus is the variant in [9], Part I.
- $\mathrm{CCS}_{!}$: Infinite behavior given by *replication* of the form $!P$. This variant is presented in [2].
- $\mathrm{CCS}_{\mu}$: Infinite behavior given by *recursive expressions* of the form $\mu X.P$ as in [8]. However, we adopt *static scoping* of channel names in the sense discussed in [4].

In particular, we show that (1) $\mathrm{CCS}_{\mathtt{k}}$ is exactly as expressive as $\mathrm{CCS}_{\mathtt{p}}$ while (2) $\mathrm{CCS}_{\mu}$ is exactly as expressive as $\mathrm{CCS}_{!}$. We use recent work by Busi et al. [2] to also state that (3) the divergence problem is undecidable for the calculi in (1) but decidable for those in (2). The results (1-3) are summarized in Figure 1.

Also, as a consequence of (1), we prove that (4) injective relabelings, from the expressiveness point of view, are redundant operators in CCS. More precisely, the behavior of any CCS process involving relabelings (all of them being injective) can be expressed up-to strong bisimilarity by a $\mathrm{CCS}_{\mathtt{k}}$ process. Furthermore, we also illustrate that $\mathrm{CCS}_{\mathtt{k}}$ exhibits *dynamic scoping* of channel names and that it does not satisfy $\alpha$-conversion. By dynamic scoping we mean that, unlike the static case, the occurrence of a name can get dynamically (i.e., during execution) captured under a restriction.

Let us now elaborate on the significance and implications of the above results. A noteworthy aspect of (1) is that any finite set of parametric (possibly mutually recursive) definitions can be replaced by an also *finite* set of parameterless definitions using neither infinite summations nor relabelings. This arises as a result of the restricted nature of communication in CCS (e.g., absence of mobility). Related to this result is that of [8] which shows that, in the context of value-passing CCS, a parametric definition can be encoded using an *infinite* set of constant definitions and infinite sums.

Regarding (1) some readers may feel that given a process $P$ with a parametric definition $D$, one could simply create as many constant definitions as permutations of possible parameters w.r.t. the finite set of names in $P$ and $D$. This would not work for $\mathrm{CCS}_{\mathtt{p}}$; an unfolding of $D$ within a restriction may need $\alpha$-conversions to avoid name captures, thus generating new names (i.e., names not in $P$ nor $D$) during execution.

The interesting point about (4) is that injective relabelings are perhaps the most used kind of relabelings (e.g., in [8] the linking operators, buffers, counters, stacks, etc, ). In

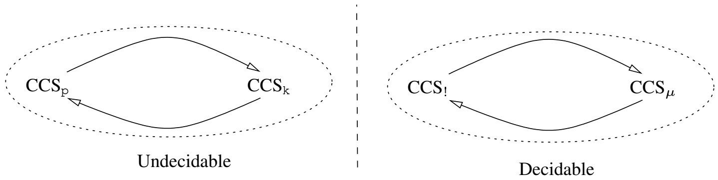


**Fig. 1.** Classification of CCS variants. An arrow from $X$ to $Y$ indicates that $Y$ is at least as expressive as $X$. (Un)decidable is understood w.r.t. the existence of divergent computations.

fact, [8] points out that the CCS laws for equational reasoning with injective relabelings as side conditions can usually be applied as one mostly works with this kind of relabeling. In the context of SCCS, another CCS variant where interaction is synchronous, in [7] the author shows that *idempotent relabelings* are redundant. The author also shows that under some natural assumptions, general relabelings are redundant in SCCS.

Another noteworthy aspect of our results is the qualitative distinction between static and dynamic name scoping for the calculi under consideration. Static scoping renders the calculus decidable (w.r.t. the divergence problem) and as expressive as that with replication. In contrast, dynamic scoping renders the calculus undecidable and as expressive as that with parametric definitions. This is interesting, since as we shall see, the difference between the calculi with static or dynamic scoping is very subtle. Using static scoping for recursive expressions was discussed in the context of ECCS [4], an extension of CCS whose ideas lead to the design of the $\pi$-calculus [9].

It should be noticed that preservation of divergence is not a requirement for equality of expressiveness; *weak bisimilarity does not preserve divergence*. Hence, although the results in [2] prove that divergence is decidable for CCS$_!$ (and undecidable for CCS$_\mathrm{p}$), it does not follow directly from the arrows in Figure 1 that it is also decidable for CCS$_\mu$.

Finally, it is worth pointing out that, as nicely exposed in [6], decidability of divergence does not imply lack of *Turing* expressiveness (see [6] for details). In fact the authors in [1] show that CCS$_!$ is Turing-complete. But this does not imply that CCS$_!$ is equally expressive to CCS$_\mathrm{p}$ either; the notions of expressiveness used in concurrency theory may not coincide with those in computability. For example, [10] shows that under some reasonable assumptions the asynchronous version of the $\pi$-calculus, which can certainly encode Turing Machines, is strictly less expressive than the synchronous one.

Overall, the general contribution of this paper is to provide and clarify some qualitative and semantics distinctions among various CCS variants.

## 2 CCS-like Calculi

We shall classify CCS-like calculi that differ in their way of specifying infinite behavior and name scope. Let us begin with their common finite fragment.

In CCS, processes can perform actions or synchronize on them. These actions can be either offering port *names* for communication, or the so-called *silent* action $\tau$. We presuppose a countable set $\mathcal{N}$ of port *names*, ranged over by $a, b, x, y \ldots$ and their primed versions. We then introduce a set of *co-names* $\overline{\mathcal{N}} = \{\overline{a} \mid a \in \mathcal{N}\}$ disjoint from $\mathcal{N}$. The set of *labels*, ranged over by $l$ and $l'$, is $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}}$. The set of *actions* $Act$, ranged over by $\alpha$ and $\beta$, extends $\mathcal{L}$ with a new symbol $\tau$. Actions $a$ and $\overline{a}$ are thought of as *complementary*, so we decree that $\overline{\overline{a}} = a$. We also decree that $\overline{\tau} = \tau$.

The processes specifying finite behavior are given by:

$$P, Q \ldots ::= \sum_{i \in I} \alpha_i.P_i \;\big|\; P \backslash a \;\big|\; P \parallel Q \tag{1}$$

Intuitively $\sum_{i \in I} \alpha_i.P_i$, where $I$ is a finite set of indexes, represents a process able to perform one–but only one–of its $\alpha_i$'s actions and then behave as the corresponding $P_i$. We write the summation as $\mathbf{0}$ if $|I| = 0$, and drop the "$\sum_{i \in I}$" if $|I| = 1$. The restriction $P \backslash a$ behaves as $P$ except that it can offer neither $a$ nor $\bar{a}$ to its environment. The names $a$ and $\bar{a}$ in $P$ are said to be *bound* in $P \backslash a$. The *bound names* of $P$, $bn(P)$, are those with a bound occurrence in $P$, and the *free names* of $P$, $fn(P)$, are those with a not bound occurrence in $P$. Finally, $P \parallel Q$ represents parallelism; either $P$ or $Q$ may perform an action, or they can also synchronize when performing complementary actions.

The above description is made precise by the operational semantics in Table 1. A transition $P \xrightarrow{\alpha} Q$ says that $P$ can perform $\alpha$ and evolve into $Q$.

$$\text{SUM}\ \frac{}{\sum_{i \in I} \alpha_i.P_i \xrightarrow{\alpha_j} P_j} \text{ if } j \in I \qquad \text{RES}\ \frac{P \xrightarrow{\alpha} P'}{P \backslash a \xrightarrow{\alpha} P' \backslash a} \text{ if } \alpha \notin \{a, \overline{a}\}$$

$$\text{PAR}_1\ \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \quad \text{PAR}_2\ \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \quad \text{COM}\ \frac{P \xrightarrow{l} P' \quad Q \xrightarrow{\overline{l}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

**Table 1.** An operational semantics for finite processes.

In the literature there are at least four alternatives to extend the above syntax to express infinite behavior. We describe them next.

### 2.1 Parametric Definitions: $\mathbf{CCS_p}$

A typical way of specifying infinite behavior is by using parametric definitions [9]. In this case we extend the syntax of finite processes (Equation 1) as follows:

$$P, Q, \ldots := \ldots \mid A(y_1, \ldots, y_n) \tag{2}$$

Here $A(y_1, \ldots, y_n)$ is an *identifier* (also *call*, or *invocation*) of arity $n$. We assume that every such an identifier has a unique, possibly recursive, *definition* $A(x_1, \ldots, x_n) \stackrel{\texttt{def}}{=} P_A$ where the $x_i$'s are pairwise distinct, and the intuition is that $A(y_1, \ldots, y_n)$ behaves as its *body* $P_A$ with each $y_i$ replacing the *formal parameter* $x_i$. We denote by $\mathcal{D}$ the set of all definitions. We often use the notation $\boldsymbol{x}$ as an abbreviation of $x_1, x_2, \ldots, x_n$.

**Convention 1 (Finitary $\mathcal{D}$.)** *Similar to [12], we shall require any process to depend only on finitely many definitions. Below we formalize this requirement.*

Given $A(\boldsymbol{x}) \stackrel{\mathtt{def}}{=} P_A$ and $B(\boldsymbol{y}) \stackrel{\mathtt{def}}{=} P_B$ in $\mathcal{D}$, we say that $A$ (directly) *depends* on $B$, written $A \rightsquigarrow B$, if there is an invocation $B(\boldsymbol{z})$ in $P_A$. The above requirement can be then formalized by requiring the strict ordering induced by $\rightsquigarrow^*$ (the reflexive and transitive closure of $\rightsquigarrow$)[1] to be well founded. We also need the following requirement.

**Convention 2** *For each $A(x_1, \ldots, x_n) \stackrel{\mathtt{def}}{=} P_A$, we require $fn(P_A) \subseteq \{x_1, \ldots, x_n\}$.*

We shall use $\mathrm{CCS_p}$ to denote the calculus with parametric definitions with the above syntactic restrictions. The rules for $\mathrm{CCS_p}$ are those in Table 1 plus the rule:

$$\text{CALL}\ \frac{P_A[y_1, \ldots, y_n/x_1, \ldots, x_n] \xrightarrow{\alpha} P'}{A(y_1, \ldots, y_n) \xrightarrow{\alpha} P'}\ \text{if } A(x_1, \ldots, x_n) \stackrel{\mathtt{def}}{=} P_A \qquad (3)$$

As usual $P[y_1 \ldots y_n/x_1 \ldots x_n]$ results from syntactically replacing every free occurrence of $x_i$ with $y_i$ renaming bound names, i.e., *name $\alpha$-conversion*, wherever needed to avoid capture. (Of course if $n = 0$, $P[y_1 \ldots y_n/x_1 \ldots x_n] = P$.)

It follows from [9] that in $\mathrm{CCS_p}$ we can identify process expression differing only by renaming of bound names; i.e., *name $\alpha$-equivalence* (so $P\backslash a$ is the same as $P[b/a]\backslash b$).

### 2.2 Constant Definitions: $\mathrm{CCS_k}$

We now consider the alternative for infinite behavior given in CCS [8]. We refer to identifiers with arity zero and their corresponding definitions as *constant* and *constant* (or *parameterless*) definitions, respectively. We omit the "( )" in $A(\ )$.

Given $A \stackrel{\mathtt{def}}{=} P$, requiring all names in $fn(P)$ to be formal parameters, as we did in $\mathrm{CCS_p}$ (Convention 2), would be too restrictive—$P$ would not have visible actions. Consequently, let us drop the requirement in Convention 2 to consider a fragment allowing *only* constant definitions but *with possible occurrence of free names in their bodies*. The rules for this fragments are those of $\mathrm{CCS_p}$. We shall refer to this fragment as $\mathrm{CCS_k}$. In this case Rule CALL, which for $\mathrm{CCS_k}$ we prefer to call CONS, takes the form

$$\text{CONS}\ \frac{P_A \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}\ \text{if } A \stackrel{\mathtt{def}}{=} P_A \qquad (4)$$

i.e., no $\alpha$-conversion involved; thus allowing name captures. As illustrated in the next section, this causes scoping to be dynamic and $\alpha$-equivalence not to hold.

***Relabelings.*** The reader familiar with process algebras may have noticed that $\mathrm{CCS_k}$ is basically CCS except for the absence of *relabeling*. A relabeling $f : Act \to Act$ is the identity for all but finitely many actions. Furthermore, $f$ satisfies $\overline{f(a)} = f(\overline{a})$, $f(a) \neq \tau$ and $f(\tau) = \tau$. The process $P(f)$ performs each $\alpha$ performed by $P$ but renamed as $f(\alpha)$. More precisely:

$$\text{REL}\ \frac{P \xrightarrow{\alpha} P'}{P(f) \xrightarrow{f(\alpha)} P'(f)}$$

[1] The relation $\rightsquigarrow^*$ is a preordering. By induced strict ordering we mean the strict component of $\rightsquigarrow^*$ modulo the equivalence relation obtained by taking the symmetric closure of $\rightsquigarrow^*$.

*Remark 1.* It is well known that the behavior specified by any process involving only *injective* relabelings can be equivalently specified (up to strong bisimilarity) by a relabeling-free process with the help of parametric definitions [11]. This is important since, as pointed out in [8], one usually works with injective relabelings. □

### 2.3 Recursion Expressions: CCS$_\mu$

Hitherto we have seen process expression whose recursive behavior is specified in an underlying set of definitions. It is often convenient, however, to have expressions which can specify recursive behavior on their own. Let us now extend the finite processes (Equation 1) to include such recursive expressions. The extended syntax is given by:

$$P, Q, \ldots := \ldots \mid X \mid \mu X.P \tag{5}$$

Here $\mu X.P$ binds the occurrences of the *process variable* $X$ in $P$. As for bound and free names, we define the *bound variables* of $P$, $bv(P)$ are those with a bound occurrence in $P$, and the *free variables* of $P$, $fv(P)$ are those with a not bound occurrence in $P$. An expression generated by the above syntax is said to be a *process (expression)* iff it is closed (i.e., it contains no free variables). The process $\mu X.P$ behaves as $P$ with the free occurrences of $X$ replaced by $\mu X.P$. Applying *variable* $\alpha$-conversions wherever necessary to avoid captures. The semantics $\mu X.P$ is given by the rule:

$$\text{REC} \; \frac{P[\mu X.P/X] \stackrel{\alpha}{\longrightarrow} P'}{\mu X.P \stackrel{\alpha}{\longrightarrow} P'} \tag{6}$$

***Static and Dynamic Scope.*** An interesting issue of the substitution $[\mu X.P/X]$ applied to $P$ is whether it *also requires* the renaming of *bound names* in $P$ to avoid captures (i.e., *name* $\alpha$*-conversion*). Such a requirement seems necessary should we want to identify process up-to $\alpha$*-equivalence*. In fact, the requirement gives us *static* scope of names. Let us illustrate this with an example.

*Example 1.* Consider $\mu X.P$ with $P = (a \parallel (\overline{a}.b \parallel X)\backslash a)$. First, let us assume we perform name $\alpha$-conversions to avoid captures. So, $[\mu X.P/X]$ in $P$ renames the bound $a$ by a fresh name, say $c$, thus avoiding the capture of $P's$ free $a$ in the replacement: I.e,

$$P[\mu X.P/X] = (a \parallel (\bar{c}.b \parallel \mu X.P)\backslash c) = (a \parallel (\bar{c}.b \parallel \mu X.(a \parallel (\overline{a}.b \parallel X)\backslash a))\backslash c)$$

The reader may care to verify (using the rules in Table 1 plus Rule REC) that $b$ will not be performed; i.e., there is no $\mu X.P \stackrel{\alpha_1}{\longrightarrow} P_1 \stackrel{\alpha_2}{\longrightarrow} \ldots$ s.t. $\alpha_i = b$.

Now let assume that the substitution makes no name $\alpha$-conversion, thus causing a free occurrence of $a$ in $P$, shown below in the dashed box, to get bound, *dynamically in the scope* of the outermost restriction: I.e.,

$$P[\mu X.P/X] = (a \parallel (\bar{a}.b \parallel \mu X.P)\backslash a) = (a \parallel (\bar{a}.b \parallel \mu X.(\boxed{a} \parallel (\overline{a}.b \parallel X)\backslash a))\backslash a).$$

The reader can verify that now $b$ can eventually be performed. Such an execution of $b$ cannot be performed by $\mu X.Q$ where $Q$ is $(a \parallel (\overline{c}.b \parallel X)\backslash c)$ i.e, $P$ with the binding and bound occurrence of $a$ syntactically replaced with $c$. This shows that name $\alpha$*-equivalence does not hold in this dynamic scope case*. □

*Remark 2.* It should be pointed out that using recursive expressions with no name $\alpha$-conversion is in fact equivalent to using instead constant definitions as in the previous calculus $\mathrm{CCS}_{\mathtt{k}}$. In fact, in presenting CCS, [8] uses alternatively both kinds of constructions; using Rule REC, with no name $\alpha$-conversion, for one and Rule CONS for the other. For example, by taking $A \stackrel{\text{def}}{=} P$ with $P$ as in Example 1 one can verify that in $\mathrm{CCS}_{\mathtt{k}}$, $A$ exhibits exactly the same dynamic scoping behavior illustrated in the above example. So, *name $\alpha$-equivalence does not hold in CCS*. Notice that the above observations imply some semantics differences between CCS and the $\pi$-calculus. The former does not satisfy name $\alpha$-equivalence because of the dynamic nature of name scoping—see Example 1. The latter uses static scoping and satisfies $\alpha$-equivalence. $\square$

We shall then use $\mathrm{CCS}_{\mu}$ as the calculus with the syntax in Equation 5 and whose semantics is given by the rules in Table 1 plus Rule REC (Equation 6) with the static approach (i.e., name $\alpha$-conversion in the substitution of process variables to avoid capture). From [4] it follows that in $\mathrm{CCS}_{\mu}$ we can identify processes up-to name $\alpha$-equivalence.

Furthermore, we make a typical assumption on $\mathrm{CCS}_{\mu}$ process variables; they need to be guarded. We say that an expression is *guarded* in $P$ iff it lies within some sub-expression of $P$ of the form $\alpha.Q$.

**Convention 3 (Guarded Recursion.)** *We shall confine ourselves to $\mathrm{CCS}_{\mu}$ processes whose variables are all guarded in them.*

### 2.4 Replication: $\mathrm{CCS}_{!}$

One simple way of expressing infinite is by using replication. Although, mostly found in calculus for mobility, it is also studied in the context of CCS in [1,2]. In this case the syntax of finite processes (Equation 1) is extended by:

$$P, Q, \ldots := \ldots \mid !P \tag{7}$$

Intuitively $!P$ behaves as $P \parallel P \parallel \ldots \parallel P \parallel !P$; as many copies of $P$ as you wish. We call $\mathrm{CCS}_{!}$ the calculus that results from the above syntax. The operational rules for $\mathrm{CCS}_{!}$ are those in Table 1 plus the following rule:

$$\text{REP}\ \frac{P \parallel !P \stackrel{\alpha}{\longrightarrow} P'}{!P \stackrel{\alpha}{\longrightarrow} P'} \tag{8}$$

From [9] we know that in $\mathrm{CCS}_{!}$ one can identify process up to name $\alpha$-equivalence.

### 2.5 Summary of Calculi

We described several calculi based on the literature of CCS. We have $\mathrm{CCS}_{\mathtt{p}}$ the calculus with parametric definitions and $\mathrm{CCS}_{\mathtt{k}}$ the calculus with constant (or parameterless) definitions. We also have $\mathrm{CCS}_{\mu}$ the calculus with recursive expressions with static scope—the version with dynamic scope instead coincides with $\mathrm{CCS}_{\mathtt{k}}$. Finally, we have the calculus with replication, $\mathrm{CCS}_{!}$.

**Convention 4** *Henceforth, we use $\Sigma$ to denote the signature $\{\mathtt{p}, \mathtt{k}, \mu, !\}$ of our calculi sub-indexes. We shall use $\sigma, \sigma', \ldots$ to range over $\Sigma$. In the following sections, we shall index sets and relations with the appropriate symbol from $\Sigma$ to make it clear what is the calculus under consideration. For example, $\xrightarrow{\alpha}_\sigma$ means that the transitions under consideration are those of $CCS_\sigma$. Similarly, we shall use $Proc_\sigma$ to denote the set of $CCS_\sigma$ processes. Often, we shall omit the indexes when they are unimportant or clear from the context.*

## 3 Expressiveness and Classification Criteria

Here we introduce the means we shall use to compare and classify the various calculi.

***Comparing Calculi: Bisimilarity.*** We wish to compare the behavior of two given processes $P$ and $Q$ w.r.t. the standard notion of (weak) bisimilarity [8]. However, $P$ and $Q$ may belong to two different calculi $CCS_\sigma$, $CCS_{\sigma'}$, respectively. We then find it convenient to state the standard notion as below. First, recall that the *converse* of a binary relation $\mathcal{S}$ is $\mathcal{S}^{-1} = \{(e', e) \mid (e, e') \in \mathcal{S}\}$

**Definition 1 (Bisimilarity).** *A relation $\mathcal{S} \subseteq Proc_\sigma \times Proc_{\sigma'}$, with $\sigma, \sigma' \in \Sigma$, is said to be a* (strong) simulation *iff for all $(P, Q) \in \mathcal{S}$:*

$$\textit{whenever } P \xrightarrow{\alpha}_\sigma P' \textit{ then, for some } Q', Q \xrightarrow{\alpha}_{\sigma'} Q' \textit{ and } (P', Q') \in \mathcal{S}.$$

*The relation $\mathcal{S}$ is called a* (strong) bisimulation *if both $\mathcal{S}$ and its converse are simulations. Furthermore, we say that $P \in Proc_\sigma$ and $Q \in Proc_{\sigma'}$ are strongly bisimilar (w.r.t., $\sigma$ and $\sigma'$), written $P \sim_\sigma^{\sigma'} Q$ (or simply $P \sim Q$), iff there exists a bisimulation $\mathcal{S} \subseteq Proc_\sigma \times Proc_{\sigma'}$, such that $(P, Q) \in \mathcal{S}$. The relation $\sim$ is called (strong) bisimilarity.* $\square$

Let us now recall the weaker notion of bisimilarity which abstracts away from silent (i.e., $\tau$) actions. We need some little notation. Define $\overset{s}{\Longrightarrow}$,with $s = \alpha_1.\alpha_2.\ldots \in Act^*$, as $(\xrightarrow{\tau})^* \xrightarrow{\alpha_1} (\xrightarrow{\tau})^* \ldots (\xrightarrow{\tau})^* \xrightarrow{\alpha_n} (\xrightarrow{\tau})^*$. The notions of *weak (bi)simulation* and *weak bisimilarity* can be derived from the strong versions by replacing in Definition 1 $\xrightarrow{\alpha}$ and $\sim$ with $\overset{s}{\Longrightarrow}$ and $\approx$, respectively. We can now make precise our criterion for expressiveness.

**Definition 2.** *We say that $CCS_\sigma$ is* as expressive as *$CCS_{\sigma'}$ iff for every $P \in Proc_\sigma$, there exists $Q \in Proc_{\sigma'}$ such that $P$ and $Q$ are weakly bisimilar (w.r.t. $\sigma$ and $\sigma'$).* $\square$

To prove equivalence on expressiveness, we shall provide *(weak) bisimulation preserving* mappings $[\![\cdot]\!]$, which we call *encodings*, from the processes of one calculus into the processes of another. Furthermore, we are interested in encodings preserving a pleasant property; *divergence*. It should be noticed that unlike strong bisimulation, weak bisimulation identifies some divergent processes with non-divergent ones. Let us formalize the notion of divergence.

***Classifying Calculi: Decidability of Divergence.*** Divergence will play a key role in this paper as it will also be used to discriminate among the various calculi. A process is deemed divergent if it can engage in an infinite sequence of $\tau$ (i.e., silent) transitions.

**Definition 3.** *We say that $P$ is* divergent *(or that it diverges) iff $P(\xrightarrow{\tau})^\omega$, i.e., there exists an infinite sequence $P = P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots$.* □

We shall classify the various calculi according to whether divergence is *decidable* for the calculus. By divergence being decidable for $\mathrm{CCS}_\sigma$, we mean that there exists an algorithm which can fully determine, given $P \in \mathit{Proc}_\sigma$, whether $P$ is divergent.

## 4 Encodings

In this section we give the various encodings. Furthermore, in order classify the calculi w.r.t. to the decidability of divergence, we shall also prove the appropriate encodings to be divergence-preserving and computable.

### 4.1 Encoding $\mathrm{CCS_p}$ into $\mathrm{CCS_k}$

Here we give an encoding $[\![\cdot]\!] : \mathrm{CCS_p} \to \mathrm{CCS_k}$. For the sake of presentation, we consider only unary parametric definitions. The encoding can be easily generalized to the $n$-ary case by extending our concepts and definition from names to vector of names.

First we need some notation and assumptions. For simplicity and w.l.o.g we assume there is a definition of the form $M^P(x) \stackrel{\mathrm{def}}{=} P \in D_P$ with $M^P$ not occurring in $P$ and $D_P$ being the *finite* set of definitions arising from the identifiers in $P$—think of $M^P$ as the "main" procedure of $P$. Formally, $D_P$ is the set of definitions for the identifiers in the closure under $\leadsto$ of $\{M^P\}$ (See Convention 1). For the time being, let $\widehat{P}$ be simply the process in $\mathrm{CCS_k}$ that results from replacing in $P$ each occurrence of $B(y)$ with $B_y$.

We shall simply define $[\![P]\!] = \widehat{M^P(x)}$. Of course we also need to specify the set $[\![D_P]\!]$ of (constant) definitions induced by $[\![P]\!]$—i.e, $D_{[\![P]\!]}$.

Intuitively, $[\![D_P]\!]$ is the set computed from $D_P$ as follows: Initially $[\![D_P]\!]$ is empty. (I) Start by adding a constant definition $M^P_x \stackrel{\mathrm{def}}{=} \widehat{P}$ to $[\![D_P]\!]$. Notice that $\widehat{P}$ may contain occurrences of some $B_y$, which corresponds to some $B(y)$, within $P$ where $B(x) \stackrel{\mathrm{def}}{=} P_B$. Then (II) add $B_y \stackrel{\mathrm{def}}{=} \widehat{P_B[y/x]}$ to $[\![D_P]\!]$. However, the substitution $[y/x]$ may involve an $\alpha$-conversion introducing some new (or *fresh*) name, say $z$. Hence, there could also be some identifier $C_z$ within $\widehat{P_B[y/x]}$, where $C(x) \stackrel{\mathrm{def}}{=} P_C$. Similarly, there could also be identifiers $C'_w$, with $C'(x) \stackrel{\mathrm{def}}{=} P_{C'}$, within $\widehat{P_B[y/x]}$ where $w$ is a name already occurring in $P_B$ (i.e., unlike $z$, the name $w$ does not result from $\alpha$-conversion). Hence, we need to go back to step (II) thus adding to $[\![D_P]\!]$ both $C_z \stackrel{\mathrm{def}}{=} \widehat{P_C[z/x]}$ and ${C'}_w \stackrel{\mathrm{def}}{=} \widehat{P_{C'}[w/x]}$. We repeat (II) until no more new definitions can be added to $[\![D_P]\!]$. Despite the generation of identifiers of the form $C_z$ with $z$ fresh (i.e. resulting from $\alpha$-conversion) we shall show that such a set of definitions $[\![D_P]\!]$ can be computed.

For soundness and technical purposes, we want $\widehat{\cdot}$ to be invariant under name $\alpha$-equivalence. Consequently, we find it convenient to redefine $\widehat{\cdot}$ as below. The intuition of our approach, however, remains the same.

**Definition 4.** *The function* $\widehat{\cdot}: CCS_{\mathrm{p}} \to \mathcal{P}(CCS_{\mathrm{k}})$ *is inductively defined over the structure of its parameter:*

$$\widehat{P} = \begin{cases} \{\mathbf{0}\} & \text{if } P = \mathbf{0} \\ \{\alpha.Q \mid Q \in \widehat{P'}\} & \text{if } P = \alpha.P' \\ \{Q_1 \parallel Q_2 \mid Q_i \in \widehat{P}_i\ ,\ i = 1,2\} & \text{if } P = P_1 \parallel P_2 \\ \{\Sigma_{i\in I}\alpha_i.Q_i \mid Q_i \in \widehat{P}_i\ ,\ i \in I\} & \text{if } P = \Sigma_{i\in I}\alpha_i.P_i \\ \{Q\backslash\beta \mid P \equiv_\alpha P'\backslash\beta \wedge Q \in \widehat{P'}\} & \text{if } P = P''\backslash\alpha \\ \{A_y\} & \text{if } P = A(y). \end{cases}$$

*Example 2.* If $P = a.\overline{b}.\mathbf{0} + B(b)$ then $\widehat{P}$ is the singleton $\{a.\overline{b}.\mathbf{0} + B_b\}$. □

*Example 3.* If $P = (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A(z))\backslash z$ then $\widehat{P}$ contains (among many others) the elements $(z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A_z)\backslash z$ and $(y.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A_y)\backslash y$. □

*Remark 3.* The definition of $\widehat{\cdot}$ is invariant under $\alpha$-conversions. More generally, it can be shown that $P \equiv_\alpha Q$ iff $\widehat{P} = \widehat{Q}$. □

We can now give a succinct characterization of $[\![D_P]\!]$.

**Definition 5.** *Given a process* $P \in CCS_{\mathrm{p}}$ *with associated definition set* $D_P$*, an* encoding *of* $P$ *in* $CCS_{\mathrm{k}}$ *is defined as the* $CCS_{\mathrm{k}}$ *constant* $M_x^P$ *(called* $[\![P]\!]$*) together with an underlying set of definitions* $[\![D_P]\!]$*, satisfying the following two conditions:*

*(I)* $[\![D_P]\!]$ *contains a definition* $(M_x^P \stackrel{\text{def}}{=} P_0)$ *for some* $P_0 \in \widehat{P}$.
*(II) If* $(A_y \stackrel{\text{def}}{=} Q_A) \in [\![D_P]\!]$, $B_z$ *occurs in* $Q_A$ *and* $(B(x) \stackrel{\text{def}}{=} P_B) \in D_P$ *, then there is* $Q_B \in \widehat{P_B[z/x]}$ *s.t.* $(B_z \stackrel{\text{def}}{=} Q_B) \in [\![D_P]\!]$.

*We understand a set of definitions to contain at most one definition per process constant. A set of definitions satisfying conditions (I) and (II) is called an* encoding set. □

Observe that, according to the definition, there are (infinitely) many encodings for a given process $P$. Not only can an encoding be extended with definitions and still remain an encoding, but also condition (II) allows for many different definitions for constant $B_z$. If, say, $Q_B, Q'_B \in \widehat{P_B[z/x]}$, then an encoding $[\![D_P]\!]$ may contain either the definition $B_z \stackrel{\text{def}}{=} Q_B$ or the definition $B_z \stackrel{\text{def}}{=} Q'_B$ (but not both).

The following lemma characterizes the shape of minimal encoding sets[2].

**Lemma 1.** *Given an encoding set* $[\![D_P]\!]$*, the set* $\mathbf{D} = \{(A_y \stackrel{\text{def}}{=} Q_A) \in [\![D_P]\!] \mid Q_A \in \widehat{P_A[y/x]}\}$*, is an encoding set (included in* $[\![D_P]\!]$*).*

*Proof.* $\mathbf{D}$ contains all the definitions from $[\![D_P]\!]$ required by conditions (I) and (II). □

Recall that $D_P$ contains finitely many definitions. We shall show that an encoding can be effectively constructed (so that the resulting set of definitions $[\![D_P]\!]$ is also finite). First let us illustrate the construction with the following example.

[2] See Corollary 3 in the Appendix for a proof of existence of minimal encoding sets.

*Example 4.* Let $P = A(x)$ with $D_P = \{A(x) \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A(z))\backslash z\}$. We proceed to define an encoding by constructing a set $[\![D_P]\!]$ so that it satisfies conditions (I) and (II). To satisfy condition (I), let $M_x^P \stackrel{\text{def}}{=} (z.x.\mathbf{0} \parallel \overline{x}.\mathbf{0} \parallel A_z)\backslash z \in [\![D_P]\!]$. Then, condition (II) requires a definition such as: $A_z \stackrel{\text{def}}{=} (z_1.z.\mathbf{0} \parallel \overline{z}.\mathbf{0} \parallel A_{z_1})\backslash z_1 \in [\![D_P]\!]$. Notice that due to $\alpha$-conversion in equation $A_z$ we have obtained a new name $z_1$ and hence we have to give a new definition for $A_{z_1}$. Of course because of the $\alpha$-conversion we could have chosen another fresh name $z_2$, but that would only lead to a different but equally useful encoding. Using condition (II) again: $A_{z_1} \stackrel{\text{def}}{=} (z.z_1.\mathbf{0} \parallel \overline{z_1}.\mathbf{0} \parallel A_z)\backslash z \in [\![D_P]\!]$, and we are done; no other definition needs to be added to $[\![D_P]\!]$. It is easy to check that the resulting set satisfies conditions (I) and (II), and therefore constitutes an encoding of $P$ in $\mathrm{CCS}_{\mathrm{k}}$. □

We now show that for any $P$, one can compute an encoding set $[\![D_P]\!]$.

**Theorem 1.** *For any $P \in CCS_{\mathrm{p}}$ with a finite set $D_P$ of associated definitions, one can effectively construct an encoding set $[\![D_P]\!]$.*

*Proof.* Let $\mathit{Var}(D_P)$ be the set of all the names occurring in $D_P$. For each $A(x) \stackrel{\text{def}}{=} P_A \in D_P$ and each $y \in \mathit{Var}(D_P)$, choose a $P_A^y$ so that $P_A^y \in \widehat{P_A[y/x]}$. Define $S = \{A_y \stackrel{\text{def}}{=} P_A^y \mid (A(x) \stackrel{\text{def}}{=} P_A) \in D_P \wedge y \in \mathit{Var}(D_P)\}$. Notice that $S$ is a finite set. Proceed by defining $\mathcal{F} = \{z \mid \exists \text{ constant } B_z\,.\, B_z \text{ occurs in } S \wedge B_z \text{ is not defined in } S\}$, and notice that $\mathcal{F}$ is a finite set too. Observe that, for each definition $A(x) \stackrel{\text{def}}{=} P_A \in D_P$ and for each $y \in \mathcal{F}$, the substitution $P_A[y/x]$ requires no alpha-conversion. Consequently it is possible to choose $P_A^y \in \widehat{P_A[y/x]}$ so that for each constant $B_z$ occurring in $P_A^y$, $z \in (\mathit{Var}(D_P) \cup \mathcal{F})$. We have now a candidate $\Sigma_{D_P}$ for the set of definitions in the encoding of $P$. It is simply defined as $\Sigma_{D_P} = \{A_y \stackrel{\text{def}}{=} P_A^y \mid (A(x) \stackrel{\text{def}}{=} P_A) \in D_P \wedge y \in (\mathit{Var}(D_P) \cup \mathcal{F})\}$. Since $(M_x^P \stackrel{\text{def}}{=} P_0) \in S \subseteq \Sigma_{D_P}$, with $P_0 \in \widehat{P}$, our candidate set satisfies condition (I) in Def. 5. It remains to be shown that $\Sigma_{D_P}$ also satisfies condition (II). Assume now that $(A_y \stackrel{\text{def}}{=} Q_A) \in \Sigma_{D_P}$, that $B_z$ occurs in $Q_A$ and that $(B(x) \stackrel{\text{def}}{=} P_B) \in D_P$. By construction, $z \in (\mathit{Var}(D_P) \cup \mathcal{F})$, and therefore $(B_z \stackrel{\text{def}}{=} P_B^z) \in \Sigma_{D_P}$. This shows that $\Sigma_{D_P}$ satisfies condition (II). Therefore, our effectively constructed candidate $\Sigma_{D_P}$ is indeed an encoding $[\![D_P]\!]$. □

We now state the correctness of the encoding up to (strong) bisimilarity. The theorem actually says that parametric definitions are not more expressive than constant definitions.

**Theorem 2.** *Given a process $P \in CCS_{\mathrm{p}}$ with associated set of definitions $D_P$, any encoding $[\![P]\!]$ with definition set $[\![D_P]\!]$ satisfies $P \sim [\![P]\!]$.*

*Proof.* See the appendix. □

*Remark 4.* It follows from Remark 1 and the above theorem that injective relabeling are redundant in CCS (up-to strong bisimilarity).

Now, [2] shows that divergence is undecidable for $\mathrm{CCS_p}$. Furthermore, we also showed that the above encoding is computable. Since strong bisimilarity is invariant under divergence, we can then conclude the following result.

**Theorem 3.** *The divergence problem is undecidable for* $CCS_{\mathrm{k}}$. □

### 4.2 Encoding $\mathbf{CCS_k}$ into $\mathbf{CCS_p}$

Intuitively, if the free names are treated dynamically, then they could equivalently be passed as parameters. Thus, we can define the encoding as follows:

**Definition 6.** *Given* $P \in CCS_{\mathrm{k}}$ *with a set of associated constant definitions of the form* $A \stackrel{\text{def}}{=} P_A$, *the encoding of* $P$ *into* $CCS_\mu$ *is a process* $[\![P]\!]$ *with associated set of definitions*

$$\left\{A(x_1,\ldots,x_n) \stackrel{\text{def}}{=} [\![P_A]\!] \mid (A \stackrel{\text{def}}{=} P_A) \in Dp \wedge fn(P_A) = \{x_1,\ldots,x_n\}\right\} .$$

*The encoding function* $[\![\cdot]\!]\colon Proc_k \rightarrow Proc_p$, *which is an homomorphism over all other operators, satisfies* $[\![A]\!] = A(x_1,\ldots,x_n)$ *where* $fn(P_A) = \{x_1,\ldots,x_n\}$. □

The following theorem states that constant definitions with dynamic scope are not more expressive that parametric definitions with static scope.

**Theorem 4.** *For every process* $P$ *in* $CCS_{\mathrm{k}}$, $[\![P]\!] \sim_k^p P$.

*Proof.* By verifying that $\{(P, [\![P]\!]) \mid P \in Proc_k\}$ is a strong bisimulation. □

### 4.3 Encoding $\mathbf{CCS_\mu}$ into $\mathbf{CCS_!}$

The main idea behind this encoding is to associate a replicated process $!x.P'$ to each occurrence of the recursion operator, $\mu X.P$. In the past a similar approach has been used to show that, in the $\pi$-calculus, recursion can be expressed using replication [12]. While in [12] each $\pi$-calculus process and its encoding happen to be strongly bisimilar, this is not the case for $\mathrm{CCS}_\mu$. Although in general a $\mathrm{CCS}_\mu$ process is only weakly bisimilar to its encoding, we show that divergence properties are always preserved.

Our definition assumes that process variables are indexed by $I$, i.e. $\{X_i \mid i \in I\}$:

**Definition 7.** *Let* $[\![\cdot]\!]\colon Proc_\mu \rightarrow Proc_!$ *be the encoding function that is homomorphic over all operators in the sub-calculus defining finite behavior and is otherwise defined as follows:*

$$\begin{aligned} [\![X_i]\!] &= \overline{x_i}.\mathbf{0} \\ [\![\mu X_i.P]\!] &= (!x_i.[\![P]\!] \parallel \overline{x_i}.\mathbf{0})\backslash x_i \end{aligned}$$

*where the names* $\{x_i \mid i \in I\}$ *are fresh.* □

The freshness condition on the variables $x_i$ is meant to guarantee that every time we apply $[\![P]\!]$, $P$ mentions none of them.

*Remark 5.* The above encoding would not work had we adopted dynamic scoping in the Rule REC for $\text{CCS}_\mu$ (see Remark 2). The $\mu X.P$ in Example 1 actually gives us a counter-example. $\square$

The following example illustrates why a $\text{CCS}_\mu$ process may not be strongly bisimilar to its encoding.

*Example 5.* Consider the $\text{CCS}_\mu$ process $P = \mu X.a.X$ with corresponding encoding $[\![P]\!] = (!x.a.\bar{x} \parallel \bar{x})\backslash x$. They are clearly not strongly bisimilar, as $P$ has the single trace

$$\mu X.a.X \xrightarrow{a}_\mu \mu X.a.X \xrightarrow{a}_\mu \mu X.a.X \ldots \tag{9}$$

while $[\![P]\!]$ only produces

$$(!x.a.\bar{x} \parallel \bar{x})\backslash x \xrightarrow{\tau}_\mu (!x.a.\bar{x} \parallel a.\bar{x})\backslash x \xrightarrow{a}_\mu (!x.a.\bar{x} \parallel \bar{x})\backslash x \xrightarrow{\tau}_\mu \ldots \tag{10}$$

Observe that each transition in (9) uses rule REC, and that every other step in (10) reflects explicitly, as an internal transition, each recursive call. $\square$

In comparing $\text{CCS}_\mu$ and $\text{CCS}_!$, we find it convenient to consider yet another variant calculus, as an intermediate step, which we call $\text{CCS}_\tau$: Its syntax agrees entirely with $\text{CCS}_\mu$'s (i.e. $Proc_\tau = Proc_\mu$), and its semantics differs from $\text{CCS}_\mu$'s only by a replacement of REC with a rule in which the unfolding performs a $\tau$ action—hence the name $\text{CCS}_\tau$:

$$\text{REC'} \quad \frac{}{\mu X.P \xrightarrow{\tau}_\tau P[\mu X.P/X]}$$

*Example 6.* Consider process $P$ as given in Example 5 but this time within $\text{CCS}_\tau$ (which is possible thanks to $Proc_\tau = Proc_\mu$). The only trace exhibited by $P$ is:

$$\mu X.a.X \xrightarrow{\tau}_\tau a.(\mu X.a.X) \xrightarrow{a}_\tau \mu X.a.X \xrightarrow{\tau}_\tau \ldots \tag{11}$$

and therefore $P \sim^!_\tau [\![P]\!]$. $\square$

In fact, the property illustrated by the previous example holds in general, as stated in the following theorem. The proof is essentially an adaptation of the one given by Sangiorgi and Walker in [12].

**Theorem 5.** *If $P \in CCS_\tau$, then $P \sim^!_\tau [\![P]\!]$.* $\square$

Because strong bisimilarity is known to preserve expressiveness and divergence, the above theorem lets us reduce the problem of studying the encoding to investigating the relation between $\text{CCS}_\tau$ and $\text{CCS}_\mu$.

We start by comparing rules REC and REC'. Consider first a process of the form $\mu X.P$. Using REC', we derive $\mu X.P \xrightarrow{\tau}_\tau P[\mu X.P/X]$. Now, if $P[\mu X.P/X] \xrightarrow{\alpha}_\tau P'$ has a derivation that does not use the rule REC', then an identical derivation serves to prove $P[\mu X.P/X] \xrightarrow{\alpha}_\mu P'$. Applying REC to the latter renders a derivation of $\mu X.P \xrightarrow{\alpha}_\mu P'$. This is a simple result, but can be generalized to all transitions of the form $Q \xrightarrow{\tau}_\tau Q'$ which depend on rule REC':

**Lemma 2.** *For $Q \in Proc_\tau$, if $Q \xrightarrow{\tau}_\tau Q'$ involves the application of rule REC' and $Q' \xrightarrow{\alpha}_\mu Q''$, then $Q \xrightarrow{\alpha}_\mu Q''$.* □

(By "involving" an application of a given rule in some transition, we mean that the rule is applied at some step of the derivation that produces the transition.)

The above lemma supports the intuition that each application of rule REC' could eventually correspond to an application of REC. A further intuition is that each $\rightarrow_\mu$ transition justified by REC may involve several recursive invocations, hence it could be mirrored by several transitions using REC' in CCS$_\tau$.

To formalize such intuitions define a binary relation $\mathcal{R} \in (Proc_\mu \times Proc_\tau)$ as follows: $P \mathcal{R} Q$ iff there exist $n \geq 0$ such that $P = Q_0 \xrightarrow{\tau}_\tau Q_1 \xrightarrow{\tau}_\tau \ldots Q_n = Q$, where each derivation $Q_i \xrightarrow{\tau}_\tau Q_{i+1}$ involves rule REC'.

We show that besides being a weak bisimulation relation, $\mathcal{R}$ also relates processes with equal divergence properties. Both proofs are intertwined. As a first step, notice that each $\xrightarrow{\alpha}_\mu$ transition can be mimicked by $\mathcal{R}$-related processes in CCS$_\tau$ after possibly some $\tau$ transitions (which corresponds to recursive invocations involving rule REC').

**Lemma 3.** *If $P \mathcal{R} Q$ and $P \xrightarrow{\alpha}_\mu P'$ then there exists $Q'$ such that $Q(\xrightarrow{\tau}_\tau)^* \xrightarrow{\alpha}_\tau Q'$ and $P' \mathcal{R} Q'$.* □

*Remark 6.* It is important at this point to recall that we have restricted our attention to processes where all variables are guarded. Without that assumption divergence would not be preserved by our encoding; $\mu X.X$ diverges in CCS$_\tau$ but deadlocks in CCS$_\mu$. □

Instead, the guarded recursion assumption forces $\mathcal{R}$ to be almost a strong simulation of CCS$_\tau$ by CCS$_\mu$. The catch is that the simulation holds only for $\rightarrow_\tau$ transitions that are also $\rightarrow_\mu$ transitions.

**Lemma 4.** *If $P \mathcal{R} Q$ and $Q \xrightarrow{\alpha}_\mu Q'$, then there exists $P'$ s.t. $P \xrightarrow{\alpha}_\mu P'$ and $P' \mathcal{R} Q'$.*

The previous lemma is a direct consequence of Lemma 2 and of the definition of $\mathcal{R}$. Since, for derivations that do not use rule REC', the transition relation $\rightarrow_\tau$ coincides with $\rightarrow_\mu$, we obtain:

**Corollary 1.** *If $P \mathcal{R} Q$ and there is a derivation of $Q \xrightarrow{\alpha}_\tau Q'$ which does not involve the application of rule REC', then there exists $P'$ s.t. $P \xrightarrow{\alpha}_\mu P'$ and $P' \mathcal{R} Q'$.* □

By now, we have all the ingredients needed to show that two identical processes, interpreted in CCS$_\mu$ and resp. CCS$_\tau$, are weakly bisimilar. Recall the definition of weak bisimulation (Section 3): One direction is provided by Lemma 3, the other follows by a combination of Corollary 1 and the definition of $\mathcal{R}$ (to cover the case in which $Q \xrightarrow{\alpha}_\tau Q'$ does use rule REC'). The result is summarized by our next theorem.

**Theorem 6.** *Given a process $P$ in CCS$_\mu$, $P \approx_\mu^\tau P$.* □

Observe that this is still not enough to show that $\mathcal{R}$ relates processes with the same divergence properties. If $P \mathcal{R} Q$ and $Q$ diverges, Corollary 1 is not even strong enough to show that $P$ may execute a single $\tau$ transition. However, it turns out that $Q$ cannot diverge by executing only recursive calls (again, a result of our assumptions on guarded summation and guarded recursion; see Remark 6); see Lemma 7 in the Appendix. So, if after some finite execution trace, $Q$ performs a $\tau$ transition that does not involve REC', we can apply Corollary 1 to deduce that $P$ may also perform a $\tau$ transition. Since this process can be repeated endlessly it must be concluded that divergence in $\mathrm{CCS}_\tau$ forces divergence in $\mathrm{CCS}_\mu$. The converse is an easy consequence of Lemma 2. That is, we have shown:

**Proposition 1.** *For $P \in CCS_\mu$, $P(\xrightarrow{\tau}_\mu)^\omega$ iff $P(\xrightarrow{\tau}_\tau)^\omega$.* $\square$

Our journey from $\mathrm{CCS}_\mu$ to $\mathrm{CCS}_\tau$ to $\mathrm{CCS}_!$ has then rendered the following result.

**Corollary 2.** *For $P \in Proc_\mu$, $P \approx^!_\mu [\![P]\!]$. Moreover, $P$ diverges iff $[\![P]\!]$ diverges.* $\square$

From the above corollary, the fact that the encoding is computable, and the result of [2] showing that divergence is decidable for $\mathrm{CCS}_!$ we conclude the following:

**Theorem 7.** *The divergence problem is decidable for $CCS_\mu$.* $\square$

### 4.4 Encoding $\mathbf{CCS_!}$ into $\mathbf{CCS_\mu}$

Except for the syntax and our restriction to guarded recursion, this encoding is essentially that given in [12] for replication into parametric definitions for the $\pi$-calculus.

**Definition 8.** *Let $[\![\cdot]\!]\colon Proc_! \to Proc_\mu$ be the encoding function that is homomorphic over all operators except for replication for which $[\![!P]\!] = \mu X.([\![P]\!] \parallel \tau.X)$.* $\square$

In fact, the proof of the following theorem is along the lines of that in [12].

**Theorem 8.** *For $P \in Proc_!$, $P \approx^\mu_! [\![P]\!]$.* $\square$

Observe that, because our restriction to guarded recursion, the encoding does not preserve divergence. For instance, if $P = !\mathbf{0}$ then $P$ is deadlocked in $\mathrm{CCS}_!$; but

$$[\![P]\!] = \mu X.(\mathbf{0} \parallel \tau.X) \xrightarrow{\tau}_\mu \mathbf{0} \parallel \mu X.(\mathbf{0} \parallel \tau.X) \xrightarrow{\tau}_\mu \mathbf{0} \parallel \mathbf{0} \parallel \mu X.(\mathbf{0} \parallel \tau.X) \xrightarrow{\tau}_\mu \dots .$$

## 5 Concluding Remarks

We studied the relative expressiveness w.r.t. weak bisimilarity and the decidability of existence of divergent computations for some CCS-like calculi. The calculi differ on the constructs used to express infinite behavior and on the treatment of scoping of channel names; the finite core being the same. We showed that parameters can be removed from recursive definitions without loss of expressiveness provided dynamic name scoping is applied. We also showed that the expressiveness of recursive expression with static scope corresponds precisely to that of replication. We partitioned the calculi into two groups: For one, divergence is undecidable (i.e., constant and parametric definitions), whereas it is decidable for the other (i.e., replication and recursive expressions with static scope). Figure 1, in the Introduction, illustrates these results.

As a consequence of our results, we proved that a substantial family of relabelings, the injective ones, is redundant in CCS. We also showed that a slightly different interpretation of Rule REC, namely performing also name $\alpha$-conversions in substitutions, can render decidable (w.r.t. divergence) an otherwise undecidable calculus (see Remark 2). We illustrated that CCS exhibits dynamic name scope and that it does not preserve $\alpha$-equivalence.

*Related Work.* Most of the related work was already discussed in the Introduction. The most closely related work is [2] which shows the (un)decidability of divergence for $\mathrm{CCS_p}$ and $\mathrm{CCS_!}$. Here we extend these results to the corresponding equally expressive calculi. The work on ECCS [4], perhaps the most immediate predecessor of the $\pi$-calculus, advocates the static scoping of names. In contrast, the work on CHOCS [13] advocates dynamic name scoping in the context of higher-order CCS. Furthermore, the CCS variant in [9] uses parametric definitions with static scope while the Edinburgh Concurrency Workbench tool [3] uses dynamic scoping for parametric definitions.

The work in [5] shows that that in CCS, non-injective relabelings lead to a sensible different treatment of asynchrony w.r.t the injective ones. We believe that it would be interesting to investigate more qualitative distinctions for these two kinds of relabelings.

*Acknowledgments.* We are indebted to Catuscia Palamidessi, Davide Sangiorgi, Jean-Jacques Levy, Maurizio Gabbrielli, Sergio Maffeis and Rosario Pugliese for insightful discussions on the topics of this paper.

## References

1. N. Busi, M. Gabbrielli, and G. Zavattaro. The expressive power of replication in ccs. Draft, 2003.
2. N. Busi, M. Gabbrielli, and G. Zavattaro. Replication vs. recursive definitions in channel based calculi. In *ICALP'03*, volume 2719 of *LNCS*, pages 133–144. Springer Verlag, 2003.
3. R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A semantics based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, 1993.
4. U. Engberg and M. Nielsen. A calculus of communicating systems with label-passing. Technical report, University of Aarhus, 1986.
5. R. Nicola M. Boreale and R. Pugliese. Trace and testing equivalence on asynchronous processes. *Information and Computation*, 2002.
6. S. Maffeis and I. Phillips. On the computational strength of pure ambient calculi. In *EXPRESS'03*, 2003.
7. R. Milner. Calculi for synchrony and asynchrony. Technical Report CSR-104-82, University of Edinburgh, 1982.
8. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
9. R. Milner. *Communicating and Mobile Systems: the $\pi$-calculus*. Cambridge University Press, 1999.
10. C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In ACM Press, editor, *POPL'97*, pages 256–265, 1997.
11. J. Parrow. An introduction to the pi-calculus. In *Handbook of Process Algebra*. 2001.
12. D. Sangiorgi and D. Walker. *The $\pi-$calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
13. B. Thomsen. A calculus of higher order communicating systems. In *POPL*, ACM, pages 143–154, 1989.

# A Proofs

## A.1 Lemmas and Theorems of Section 4.1

Even though encoding sets are in general not comparable, we can group them according to the minimal encoding sets they contain. Minimality is here understood with respect to the set inclusion order.

**Lemma 5.** *If $\mathbf{D}, \mathbf{E} \subseteq [\![D_P]\!]$ are all encoding sets, then so is $\mathbf{D} \cap \mathbf{E}$.*

*Proof.* That $\mathbf{D} \cap \mathbf{E}$ satisfies condition (I) is trivial. For condition (II), notice that if $(B_z \stackrel{\text{def}}{=} Q_B) \in \mathbf{D}$ and $(B_z \stackrel{\text{def}}{=} Q'_B) \in \mathbf{E}$, then $Q_B = Q'_B$ for $[\![D_P]\!]$ can at most contain a single definition for constant $B_z$. $\square$

**Corollary 3.** *Every encoding set $[\![D_P]\!]$ contains a single minimal encoding set.*

The following is a corollary of Lemma 1:

**Corollary 4.** *The elements of every minimal encoding set are of the form $A_y \stackrel{\text{def}}{=} Q_A$ for some $Q_A \in \widehat{P_A[y/x]}$.*

This lemma is used in the proof of Theorem 2:

**Lemma 6.** *Given processes $P, P', Q \in CCS_{\mathrm{p}}$ such that $P \equiv_\alpha P'$, every derivation of $P \stackrel{\alpha}{\to} Q$ can be transformed into a derivation of equal depth of $P' \stackrel{\alpha}{\to} Q'$, for some $Q' \equiv_\alpha Q$.*

**Theorem 2.** *Given a process $P \in CCS_{\mathrm{p}}$ with associated set of definitions $D_P$, any encoding $[\![P]\!]$ with definition set $[\![D_P]\!]$ satisfies*

$$P \sim [\![P]\!] \,.$$

*Proof.* We assume w.l.o.g. that $[\![D_P]\!]$ is a minimal encoding set, and show that the following relation $\mathcal{S} \subseteq \mathrm{CCS}_{\mathrm{p}} \times \mathrm{CCS}_{\mathrm{k}}$ is indeed a bisimulation

$$\mathcal{S} = \{(T, U) \mid U \in \widehat{T} \wedge \mathrm{def}_{[\![D_P]\!]}(U)\}$$

Where $\mathrm{def}_{[\![D_P]\!]}(R)$ means that all the constant identifiers occurring in $R$ are defined in $[\![D_P]\!]$. We prove it as usual, by induction on the depth of the inference of $T \stackrel{\alpha}{\to}_p T'$. We will only prove here that $\mathcal{S}$ is a (strong) simulation, that is that whenever $T \;\mathcal{S}\; U$, if $T \stackrel{\alpha}{\to}_p T'$ then $U \stackrel{\alpha}{\to}_k U'$ and $T' \;\mathcal{S}\; U'$. Let $T \stackrel{\alpha}{\to}_p T'$ be the conclusion of the last step in the inference. While most cases consist of routine checks, we specially call the attention of the reader to the last two cases (corresponding to the application of rule RES, resp. CALL).

- If $T \xrightarrow{\alpha}_p T'$ has been introduced by PAR1, then $T$ is of the form $T_1 \parallel T_2$ and $T' = (T_1' \parallel T_2)$; so we know that $T_1 \xrightarrow{\alpha}_p T_1'$. By assumption we have that $(T_1 \parallel T_2)\ \mathcal{S}\ U$. Therefore, $U \in \widehat{(T_1 \parallel T_2)}$; and, by definition of $\widehat{\cdot}$, there exist $U_1$ and $U_2$ s.t. $U = (U_1 \parallel U_2)$, where $U_1 \in \widehat{T_1}$ and $U_2 \in \widehat{T_2}$. Notice that $\mathrm{def}_{[\![D_P]\!]}(U_1)$ and then $T_1\ \mathcal{S}\ U_1$. By I.H. we have that $U_1 \xrightarrow{\alpha}_k U_1'$ with $T_1'\ \mathcal{S}\ U_1'$. Now, by PAR1 in $\mathrm{CCS_k}$ we have that $(U_1 \parallel U_2) \xrightarrow{\alpha}_k (U_1' \parallel U_2)$ and by definition of $\widehat{\cdot}$ we have that $(U_1' \parallel U_2) \in \widehat{T_1' \parallel T_2}$. Since $\mathrm{def}_{[\![D_P]\!]}(U_1')$ and $\mathrm{def}_{[\![D_P]\!]}(U_2)$, we have that $\mathrm{def}_{[\![D_P]\!]}(U_1' \parallel U_2)$. Hence, $(T_1' \parallel T_2)\ \mathcal{S}\ (U_1' \parallel U_2)$.
- If $T \xrightarrow{\alpha}_p T'$ has been introduced by PAR2, the proof is similar to the previous case.
- If $T \xrightarrow{\tau}_p T'$ has been introduced by COM, then $T$ is again of the form $T_1 \parallel T_2$, $T' = T_1' \parallel T_2'$ and we know that $T_1 \xrightarrow{\alpha}_p T_1'$ and $T_2 \xrightarrow{\bar{\alpha}}_p T_2'$. Since $T\ \mathcal{S}\ U$ we know that $U \in \widehat{T_1 \parallel T_2}$ so there exist $U_1$ and $U_2$ s.t. $U = (U_1 \parallel U_2)$, where $U_i \in \widehat{T_i}$, for $i = 1, 2$. Notice that $\mathrm{def}_{[\![D_P]\!]}(U_i)$ and then $T_i\ \mathcal{S}\ U_i$. By I.H. $U_i \xrightarrow{\alpha}_k U_i'$ with $T_i'\ \mathcal{S}\ U_i'$. By rule COM in $\mathrm{CCS_k}$ we have that $(U_1 \parallel U_2) \xrightarrow{\tau}_k (U_1' \parallel U_2')$. By definition of $\widehat{\cdot}$ we have that $(U_1' \parallel U_2') \in \widehat{T_1 \parallel T_2}$. Since $\mathrm{def}_{[\![D_P]\!]}(U_1')$ and $\mathrm{def}_{[\![D_P]\!]}(U_2')$, we have that $\mathrm{def}_{[\![D_P]\!]}(U_1' \parallel U_2')$. Hence, $(T_1' \parallel T_2')\ \mathcal{S}\ (U_1' \parallel U_2')$.
- Suppose that $T \xrightarrow{\alpha}_p T'$ has been introduced by SUM: $T$ is of the form $\Sigma_{i\in I}\alpha_i.Q_i$ and then $T' = Q_j$ and $\alpha_j = \alpha$, for some $j \in I$. Since $U \in \widehat{\Sigma_{i\in I}\alpha_i.Q_i}$, there exist $U_i$ (for all $i \in I$) s.t. $U = \Sigma_{i\in I}\alpha_i.U_i$ and $U_i \in \widehat{T_i}$. Let $U' = U_j$, so that $\alpha.T'\ \mathcal{S}\ \alpha.U'$. Then, by I.H., $T'\ \mathcal{S}\ U'$, and by rule SUM in $\mathrm{CCS_k}$, $U \xrightarrow{\alpha}_k U'$.
- Suppose now that $T \xrightarrow{\alpha}_p T'$ has been introduced by RES: Then $T$ is of the form $T_1\backslash b$, with $b \neq \alpha$, $b \neq \overline{\alpha}$; and $T'$ is of the form $T_1'\backslash b$ satisfying $T_1 \xrightarrow{\alpha}_p T_1'$. By assumption $T\ \mathcal{S}\ U$, and therefore $U = U_1\backslash c$ for some process $U_1$ and name $c$.
By Lemma 6, we can assume that $b = c$ (otherwise, we alpha-convert $T$ into $T_2\backslash c$ and work with a derivation of $T_2\backslash c \xrightarrow{\alpha}_p T_2'\backslash c$ of the same depth, noting that $T_2'\backslash c \equiv_\alpha T'$ and that the composed relation $\equiv_\alpha \mathcal{S}$ equals $\mathcal{S}$).
Since $U = U_1\backslash b$ and $U \in \widehat{T}$, according to the definition of $\widehat{\cdot}$, there must be a $V_1 \in \mathrm{CCS_p}$ such that $U_1 \in \widehat{V_1}$ and $T_1\backslash b \equiv_\alpha V_1\backslash b$. From the latter, $T_1 \equiv_\alpha V_1$; and, by Remark 3, $U_1 \in \widehat{T_1}$. We conclude that $T_1\ \mathcal{S}\ U_1$. By I.H., there exists $U_1'$ s.t. $U_1 \xrightarrow{\alpha}_k U_1'$ and $T_1'\ \mathcal{S}\ U_1'$. Hence, $U \xrightarrow{\alpha}_k U_1'\backslash b$ (by rule RES in $\mathrm{CCS_k}$), and it is easy to check that $T'\ \mathcal{S}\ U_1'\backslash b$.
- Suppose that $T \xrightarrow{\alpha}_p T'$ has been introduced by rule CALL: In this case, $T = A(y) \xrightarrow{\alpha}_p T'$ (where $(A(x) \stackrel{\mathrm{def}}{=} P_A) \in D_P$). By assumption we have that $A(y)\ \mathcal{S}\ U$. Then $U \in \widehat{A(y)}$. We deduce that $U = A_y$ and $\mathrm{def}_{[\![D_P]\!]}(A_y)$. By Corollary 4, there is a definition $(A_y \stackrel{\mathrm{def}}{=} Q_A) \in [\![D_P]\!]$ with $Q_A \in \widehat{P_A[y/x]}$. In order to apply the inductive hypothesis, we need to check that $P_A[y/x]\ \mathcal{S}\ Q_A$, i.e. we only need to show that $\mathrm{def}_{[\![D_P]\!]}(Q_A)$. This follows easily from condition (II) in Def. 5.
According to rule CONS, we have a derivation of $P_A[y/x] \xrightarrow{\alpha}_p T'$. By I.H., there exists $U'$ s.t. $Q_A \xrightarrow{\alpha}_k U'$ and $T'\ \mathcal{S}\ U'$. Finally, by rule CONS in $\mathrm{CCS_k}$ we have that $U = A_y \xrightarrow{\alpha}_k U'$.

¿From all the above cases we conclude that $\mathcal{S}$ is a (strong) simulation. In a similar form we can prove that $\mathcal{S}^{-1}$ is a simulation. To conclude, by condition (I) in Def. 5, $[\![D_P]\!]$ contains a definition for $M_x^P$. Since $M_x^P \in \widehat{M^P(x)}$, $M^P(x)\ \mathcal{S}\ M_x^P$. Noting that $M^P(x) = P$ and $M_x^P = [\![P]\!]$, this means that $P \sim [\![P]\!]$ as wanted. □

### A.2 Lemmas and Theorems of Section 4.3

**Lemma 2.** *For $Q \in Proc_\tau$, if $Q \xrightarrow{\tau}_\tau Q'$ involves the application of rule REC' and $Q' \xrightarrow{\alpha}_\mu Q''$, then $Q \xrightarrow{\alpha}_\mu Q''$.*

*Proof.* A simple induction on the depth of the derivation of $Q \xrightarrow{\tau}_\tau Q'$ where the only cases to consider are derivations ending in an application of the rules REC', PAR$_i$ or RES. □

*Remark 7.* Consider the binary relation $\mathcal{R} \in (Proc_\mu \times Proc_\tau)$ as defined on p. 14.

1. $\mathcal{R}$ includes the identity relation.
2. It is easy to show that for $P = P'\backslash a$, $P\ \mathcal{R}\ Q$ iff there is a $Q'$ such that $Q = Q'\backslash a$ and $P'\ \mathcal{R}\ Q'$.
3. Similarly, for $P = P_1 \parallel P_2$, $P\ \mathcal{R}\ Q$ iff there are $Q_1$ and $Q_2$ such that $Q = Q_1 \parallel Q_2$ and $P_i\ \mathcal{R}\ Q_i$, $i = 1, 2$.

**Lemma 3.** *If $P\ \mathcal{R}\ Q$ and $P \xrightarrow{\alpha}_\mu P'$ then there exists $Q'$ such that $Q(\xrightarrow{\tau}_\tau)^* \xrightarrow{\alpha}_\tau Q'$ and $P'\ \mathcal{R}\ Q'$.*

*Proof.* Consider the trace $P = Q_0 \xrightarrow{\tau}_\tau \ldots \xrightarrow{\tau}_\tau Q_n = Q$ that witnesses $P\ \mathcal{R}\ Q$. The proof proceeds by induction on the depth of the derivation of $P \xrightarrow{\alpha}_\mu P'$:

- REC: There is $T$ such that $P = \mu X.T$ and $T[\mu X.T/X] \xrightarrow{\alpha}_\mu P'$. Consider first the case $n = 0$ (i.e. $P = Q$): then $Q \xrightarrow{\tau}_\tau T[\mu X.T/X]$ (using REC'). By Remark 7.1, $T[\mu X.T/X]\ \mathcal{R}\ T[\mu X.T/X]$, and by I.H. there is $Q'$ s.t. $P'\ \mathcal{R}\ Q'$ and $T[\mu X.T/X](\xrightarrow{\tau}_\tau)^* \xrightarrow{\alpha}_\tau Q'$. Therefore, $Q(\xrightarrow{\tau}_\tau)^* \xrightarrow{\alpha}_\tau Q'$. If $n > 0$, note that $Q_1 = T[\mu X.T/X]$, and therefore $T[\mu X.T/X]\ \mathcal{R}\ Q$. The property follows directly from the inductive hypothesis.
- COM: There are $P_i$ and $P_i'$ ($i = 1, 2$) such that $P = P_1 \parallel P_2$, $P' = P_1' \parallel P_2'$, $\alpha = \tau$, $P_1 \xrightarrow{a}_\mu P_1'$ and $P_2 \xrightarrow{\bar{a}}_\mu P_2'$. By Remark 7.3, $Q = Q_1 \parallel Q_2$ with $P_i\ \mathcal{R}\ Q_i$. By the I.H. there are $Q_1'$ and $Q_2'$ such that $P_i'\ \mathcal{R}\ Q_i'$, $Q_1(\xrightarrow{\tau}_\tau)^* \xrightarrow{a}_\tau Q_1'$ and $Q_2(\xrightarrow{\tau}_\tau)^* \xrightarrow{\bar{a}}_\tau Q_2'$. Using zero or more applications of rules PAR$_i$, followed by an application of COM, it is easy to see that $Q(\xrightarrow{\tau}_\tau)^* \xrightarrow{\tau}_\tau Q_1' \parallel Q_2'$. That $Q' = Q_1' \parallel Q_2'$ satisfies $P'\ \mathcal{R}\ Q'$ is a simple consequence of Remark 7.3.

The cases PAR$_i$ and RES are analogous to the COM case, also relying on Remark 7. Finally, the SUM case is simpler, for then it must be that $P = Q$. □

**Lemma 4.** *If $P \mathcal{R} Q$ and $Q \xrightarrow{\alpha}_\mu Q'$, then there exists $P'$ s.t. $P \xrightarrow{\alpha}_\mu P'$ and $P' \mathcal{R} Q'$.*

*Proof.* By induction on the length $n$ of the trace $P = Q_0 \xrightarrow{\tau}_\tau \ldots \xrightarrow{\tau}_\tau Q_n = Q$:

- When $n = 0$, $P = Q$. Then $P \xrightarrow{\alpha}_\mu Q'$, and $Q' \mathcal{R} Q'$ by a previous observation.
- If $n > 0$, consider $Q_{n-1} \xrightarrow{\tau}_\tau Q \xrightarrow{\alpha}_\mu Q'$. By Lemma 2, $Q_{n-1} \xrightarrow{\alpha}_\mu Q'$. Since $P \mathcal{R} Q_{n-1}$, the result follows from the inductive hypothesis. □

**Lemma 7.** *If $Q = Q_0 \xrightarrow{\tau}_\tau Q_1 \xrightarrow{\tau}_\tau \ldots$ then, for some $i$, the derivation of $Q_i \xrightarrow{\tau}_\tau Q_{i+1}$ does not involve rule REC'.*

*Proof.* Consider the following measure function $\mathcal{M} \colon Proc_\tau \to \mathbb{N}$

$$
\begin{aligned}
&\mathcal{M}(\textstyle\sum_{i\in I} \alpha_i.P_i) = 0\\
&\mathcal{M}(P \parallel Q) \qquad = \mathcal{M}(P) + \mathcal{M}(Q)\\
&\mathcal{M}(P\backslash a) \qquad\; = \mathcal{M}(P)\\
&\mathcal{M}(\mu X.P) \qquad = 1 + \mathcal{M}(P)
\end{aligned}
$$

It is easy to prove, by induction on the structure of $P \in Proc_\tau$, that $\mathcal{M}(P[R/X]) = \mathcal{M}(P)$.

To prove this lemma, it suffices to show that each transition $P \xrightarrow{\tau}_\tau P'$, which involves rule REC', induces an strict reduction in measure (i.e. $\mathcal{M}(P') < \mathcal{M}(P)$).

The proof proceeds by induction on the depth of the derivation of $P \xrightarrow{\tau}_\tau P'$. We need only consider the cases in which the last step in the derivation corresponds to the application of rules REC', PAR$_i$, or RES.

- REC': In this case, $P = \mu X.R$ and $P' = R[\mu X.R/X]$. Then,
$$\mathcal{M}(P') = \mathcal{M}(R) < 1 + \mathcal{M}(R) = \mathcal{M}(P)\,.$$
- PAR$_i$: Consider only the case of rule PAR$_1$, the other being completely analogous. We can assume that $P = P_1 \parallel P_2$, $P' = P_1' \parallel P_2$, and $P_1 \xrightarrow{\tau}_\tau P_1'$. Moreover, the derivation of the latter transition must involve the rule REC'. By our inductive hypothesis, $\mathcal{M}(P_1') < \mathcal{M}(P_1)$. Then
$$\mathcal{M}(P') = \mathcal{M}(P_1') + \mathcal{M}(P_2') < \mathcal{M}(P_1) + \mathcal{M}(P_2) = \mathcal{M}(P)\,.$$
- RES: Assume $P = R\backslash a$, $P' = R'$ and $R \xrightarrow{\tau}_\tau R'$. By I.H., $\mathcal{M}(R') < \mathcal{M}(R)$. Then
$$\mathcal{M}(P') = \mathcal{M}(R') < \mathcal{M}(R) = \mathcal{M}(P)\,.$$

□

**Proposition 1.** *For $P \in CCS_\mu$, $P(\xrightarrow{\tau}_\mu)^\omega$ iff $P(\xrightarrow{\tau}_\tau)^\omega$.*

*Proof.* Notice first that, by Remark 7.1, $P \mathcal{R} P$.

⇒: Assume the infinite trace $P = P_0 \xrightarrow{\tau}_\mu P_1 \xrightarrow{\tau}_\mu \ldots$. Using Lemma 3 iteratively construct the also infinite trace $P = Q_0(\xrightarrow{\tau}_\tau)^* \xrightarrow{\tau}_\tau Q_1(\xrightarrow{\tau}_\tau)^* \xrightarrow{\tau}_\tau$ (where $P_i \mathrel{\mathcal{R}} Q_i$, for all $i \geq 0$).

⇐: Assume $P = Q_0 \xrightarrow{\tau}_\tau Q_1 \xrightarrow{\tau}_\tau Q_2 \xrightarrow{\tau}_\tau \ldots$. By Lemma 7, there is a subsequence $\{i_j\}_{j \in \mathbb{N}}$ of the natural numbers such that:

$$P = Q_{i_0}(\xrightarrow{\tau}_\tau)^* Q_{i_1 - 1} \xrightarrow{\tau}_\tau Q_{i_1}(\xrightarrow{\tau}_\tau)^* Q_{i_2 - 1} \xrightarrow{\tau}_\tau Q_{i_2} \ldots$$

where each derivation in the subtrace $Q_{i_{(j-1)}}(\xrightarrow{\tau}_\tau)^* Q_{i_j - 1}$ involves the application of rule REC', and each derivation $Q_{i_j - 1} \xrightarrow{\tau}_\tau Q_{i_j}$ does not use REC'.

To show that $P(\xrightarrow{\tau}_\mu)^\omega$ we construct, in an inductive fashion, an infinite trace $P = P_0 \xrightarrow{\tau}_\mu P_1 \xrightarrow{\tau}_\mu \ldots$ where $P_j \mathrel{\mathcal{R}} Q_{i_j}$, for all $j \in \mathbb{N}$:

- Let $P_0 = P$.
- For $j > 0$, assume $P_{j-1} \mathrel{\mathcal{R}} Q_{i_{(j-1)}}$ and notice that $P_{j-1} \mathrel{\mathcal{R}} Q_{i_j - 1}$ (by the definition of $\mathcal{R}$). Then, apply Corollary 1 to choose $P_j$ such that $P_{j-1} \xrightarrow{\tau}_\mu P_j$ and $P_j \mathrel{\mathcal{R}} Q_{i_j}$.

□