Under consideration for publication in Math. Struct. in Comp. Science

On the expressive power of recursion, replication and iteration in process calculi

Nadia Busi Maurizio Gabbrielli Gianluigi Zavattaro

Dip. di Scienze dell'Informazione, Univ. di Bologna, Mura A.Zamboni 7, 40127 Bologna, Italy.

Received March 2008

We investigate the expressive power of three alternative approaches for the definition of infinite behaviors in process calculi, namely, recursive definitions, replication and iteration. We prove several discrimination results between the calculi obtained from a core CCS by adding the three mechanisms mentioned above. These results are obtained by considering the decidability of four basic properties: termination (i.e. all computations are finite), convergence (i.e. the existence of a finite computation), barb (i.e. the ability of performing a synchronization) and weak bisimulation.
Our results, summarized in Table 1, show that the three calculi form a strict expressiveness hierarchy, since all the mentioned properties are undecidable in CCS with recursion, while termination and barb are decidable in CCS with replication and all the properties are decidable in CCS with iteration.
As a corollary we obtain also a strict expressiveness hierarchy w.r.t. weak bisimulation, since there exist weak bisimulation preserving encodings of iteration in replication and of replication in recursion, whereas there exist no weak bisimulation preserving encoding in the other directions.

1. Introduction

Process calculi are continuously updated with new languages, dialects and variants which range from general frameworks such as CCS and π-calculus to formalism tailored on specific class of applications (biological ones are probably the most common today). Given such a rich variety it is important to formally compare the existing languages in order to understand precisely their relative expressive power. Several notions of expressive power are meaningful in this context: The classical notion based on the ability to express recursive functions can be further refined by considering, for example, compositionality properties for the encoding of a language into another or the ability to express some patterns of behaviors (typically connected to mobility).

An important aspect which, in general, may significantly affect the expressive power of a language is the mechanism adopted for extending finite processes in order to express infinite behaviors: Three classical mechanisms which are used in process calculi are *recursion*, *replication*, and *iteration*. Recursion can be supported by using process constants: Each process constant D has an associated (possibly recursive) definition $D \stackrel{def}{=} P$. By using recursively defined process constants one can obtain an "in depth" infinite behavior,

since process copies in this case can be nested at an arbitrary depth by using constant application. On the other hand, the replication operator $!P$ allows one to create an unbounded number of parallel copies of a process P, thus providing an "in width" infinite behavior, since the copies are placed at the same level. Finally, the iteration operator P^* permits to iterate the execution of a process P, i.e. at the end of the execution of one copy of P another copy can be activated. In this case, a "repetitive" infinite behavior is supported, since the copies are executed one after the other.

It is well known that recursion and replication are equivalent (for any reasonable notion of expressive power) for full π-calculus (MPW92), as the ability to communicate free names together with replication and restriction allows to simulate process constants (replication can easily be simulated by recursive definitions, provided one admits enough constants). On the other hand, there is a common agreement on the fact that recursion cannot be replaced by replication when name mobility is not allowed, as in the case of CCS, even though this result has not been formally proved so far.

In this paper we compare the expressive power of recursion, replication and iteration in the context of CCS. More precisely, we consider the three dialects CCS_D, $\text{CCS}_!$ and CCS_* obtained by adding recursion, replication and iteration, respectively, to a core CCS language corresponding to the finite fragment of CCS (Milner 1989) without relabeling. For these three dialects we investigate the decidability of the following four relevant properties of processes: *termination*, i.e. the non-existence of a divergent computation (equivalently, all the computations are finite); *convergence*, i.e. the existence of a computation that terminates; *barb*, i.e. the ability to perform a synchronization on a certain channel after a (possibly empty) internal computation; *weak bisimulation* (here we say that weak bisimulation is decidable if, given any pair of processes, it is decidable whether those two processes are weakly bisimilar).

We first prove that all these properties are undecidable in CCS_D. These undecidability results are obtained by providing an encoding of Random Access Machines (Shepherdson and Sturgis 1963), RAMs for short, a well known deterministic Turing powerful formalism, into CCS_D. The encoding is deterministic, i.e. it presents a unique possible computation that reflects the computation of the corresponding RAM. This proves immediately that termination and convergence are undecidable. By exploiting a slightly different (deterministic) encoding, we prove the undecidability of barb as well as of weak bisimulation: The idea is to extend the modeling of RAMs with an observable action that can be performed on program termination; in this way we reduce the problem of testing the termination of a RAM to the problem of detecting an observable behavior.

Next we show that convergence and weak bisimulation remain undecidable in the language $\text{CCS}_!$ with replication. These results are obtained by providing a nondeterministic encoding of RAMs in $\text{CCS}_!$. The encoding is nondeterministic in the following sense: computations which do not follow the expected behavior of the modeled RAM can be introduced by the encoding, however all these computations are infinite. This proves that a process modeling a RAM has a terminating computation, i.e. converges, if and only if the corresponding RAM terminates. Thus, process convergence is undecidable for the calculus with replication. The nondeterministic modeling of RAMs under replication permits us to prove that also weak bisimulation is undecidable, by following a technique similar to

	Recursion	Replication	Iteration
Termination	undecidable	decidable	decidable
Convergence	undecidable	undecidable	decidable
Barb	undecidable	decidable	decidable
Weak bisimulation	undecidable	undecidable	decidable

Table 1. *Summary of the results*

the one described above for the calculus with recursion. Interestingly, while convergence and weak bisimulation are undecidable under replication, we prove that termination as well barb turn out to be decidable properties. These decidability results are obtained by resorting to the theory of well structured transition systems (Finkel and Schnoebelen 2001). It is also worth noting that the decidability of process termination for the calculus with replication implies the impossibility to provide a termination preserving encoding of RAMs into $\text{CCS}_!$.

For the calculus with process iteration we have that all the properties are decidable. This is a consequence of the fact that the processes of this calculus are finite state. Intuitively, this follows from the fact that each iteration activates one copy at a time (thus only a predefined number of processes can be active at the same time) and all the copies share the same finite set of possible states.

These results of our investigation are summarized in Table 1.

As a final corollary we obtain also that recursion, replication and iteration constitute a strict expressiveness hierarchy w.r.t. weak bisimulation. In fact, we show that there exist weak bisimulation preserving encodings of iteration in replication and of replication in recursion, whereas we prove that there exist no weak bisimulation preserving encoding in the other directions.

The remaining of this paper is structured as follows. In Section 2 we present the syntax and the semantics of the considered calculi. Section 3 contains the deterministic RAM encoding and the undecidability results for the calculus with recursion. In Section 4 we first show the non deterministic RAM encoding for the calculus with replication and prove the undecidability results. Then we prove the decidability of termination and barb for $\text{CCS}_!$. In Section 5 we prove that the processes of the calculus with process iteration are finite states, while Section 6 concludes by discussing the weak-bisimulation preserving encoding and by mentioning some relevant related work.

Preliminary versions of this paper appeared in (Busi *et al.* 2003; Busi *et al.* 2004).

2. The Calculi

We start considering the finite fragment of the core of CCS (that we sometimes call simply CCS for brevity). After that, we present the three infinite extensions.

$$\texttt{PRE}: \quad \alpha.P \xrightarrow{\alpha} P \qquad \texttt{PAR}: \quad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \texttt{SUM}: \quad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$$

$$\texttt{RES}: \quad \frac{P \xrightarrow{\alpha} P'}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \; x \notin n(\alpha) \qquad \texttt{COM}: \quad \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

Table 2. *The transition system for finite core CCS (symmetric rules of* PAR*,* SUM*, and* COM *omitted).*

Definition 1. (finite core CCS) Let $Name$, ranged over by x, y, ..., be a denumerable set of channel names. The class of finite core CCS processes is described by the following grammar:

$$\begin{array}{lcl} P & ::= & \mathbf{0} \mid \alpha.P \mid P+P \mid P|P \mid (\nu x)P \\ \alpha & ::= & \tau \mid x \mid \overline{x} \end{array}$$

The term $\mathbf{0}$ denotes the empty process while the term $\alpha.P$ has the ability to perform the action α (which is either the unobservable τ action or a synchronization on a channel x) and then behaves like P. Two forms of synchronizing action are available, the output $\overline{x}$ or the input x. The sum construct $+$ is used to make choice between the summands while parallel composition $|$ is used to run parallel programs. Restriction $(\nu x)P$ makes the name x local in P. We denote the process $\alpha.\mathbf{0}$ simply with α, and the process $(\nu x_1)(\nu x_2)\ldots(\nu x_n)P$ with $(\nu \tilde{x})P$ where $\tilde{x}$ is the sequence of names $x_1, x_2, \ldots, x_n$.

For input and output actions, we write $\overline{\alpha}$ for the complementary of α; that is, if $\alpha = x$ then $\overline{\alpha} = \overline{x}$, if $\alpha = \overline{x}$ then $\overline{\alpha} = x$. We write $fn(P)$, $bn(P)$ for the *free names* and the *bound names* of P. The *names* of P, written $n(P)$, is the union of the free and bound names of P. The names in a label α, written $n(\alpha)$ is the set of names in α, i.e. the empty set if $\alpha = \tau$ or the singleton $\{x\}$ if α is either x or $\overline{x}$. Table 2 contains the set of the transition rules for finite CCS.

Definition 2. ($\mathbf{CCS}_D$) We assume a denumerable set of constants, ranged over by D. The class of CCS_D processes is defined by adding the production $P ::= D$ to the grammar of Definition 1. It is assumed that each constant D has a unique defining equation of the form $D \stackrel{def}{=} P$.

The transition rule for constant is

$$\texttt{CONST}: \quad \frac{P \xrightarrow{\alpha} P' \qquad D \stackrel{def}{=} P}{D \xrightarrow{\alpha} P'}$$

$$\mathbf{0} \xrightarrow{\surd} \qquad P^* \xrightarrow{\surd} \qquad \frac{P \xrightarrow{\surd}}{(\nu x)P \xrightarrow{\surd}}$$

$$\frac{P \xrightarrow{\surd} \quad Q \xrightarrow{\surd}}{P|Q \xrightarrow{\surd}} \qquad \frac{P \xrightarrow{\surd} \quad Q \xrightarrow{\surd}}{P + Q \xrightarrow{\surd}} \qquad \frac{P \xrightarrow{\surd} \quad Q \xrightarrow{\surd}}{P;Q \xrightarrow{\surd}}$$

Table 3. *Definition of the termination predicate.*

It is worth noting that this rule for the semantics of constants has been adopted also in (Giambiagi *et al.* 2004). In that paper, it is observed that no α-conversion is to be considered in calculi with this form of constant definition, and this causes name captures and scoping to be dynamic.

Definition 3. (CCS$_!$) The class of CCS$_!$ processes is defined by adding the production $P ::= \ !P$ to the grammar of Definition 1.

The transition rule for replication is

$$\texttt{REPL}: \quad \frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

Definition 4. (CCS$_*$) The class of CCS$_*$ processes is defined by adding the production $P ::= \ P^*$ to the grammar of Definition 1.

Intuitively, the process P^* has the ability to iterate the behavior of the process P an arbitrary number of time (possibly zero times). In order to formally describe the semantics of iteration we explicitly represent the ending of process P with the predicate $P \xrightarrow{\surd}$. We also exploit an auxiliary operator $P;Q$ denoting the sequential composition of processes. Informally, given the process $P;Q$ we have that the process Q can start only if $P \xrightarrow{\surd}$. Formally, the axioms and rules defining the predicate for $P \xrightarrow{\surd}$ are reported in Table 3. The transition rules for iteration are

$$\texttt{ITER}: \quad \frac{P \xrightarrow{\alpha} P'}{P^* \xrightarrow{\alpha} P';P^*}$$

$$\texttt{SEQ1}: \quad \frac{P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\alpha} P';Q} \qquad \texttt{SEQ2}: \quad \frac{P \xrightarrow{\surd} \quad Q \xrightarrow{\alpha} Q'}{P;Q \xrightarrow{\alpha} Q'}$$

In the following we adopt the following notation. We use $\prod_{i\in I} P_i$ to denote the parallel composition of the indexed processes P_i, while we use $\prod_n P$ to denote the parallel composition of n instances of the process P (if $n = 0$ then $\prod_n P$ is the empty process $\mathbf{0}$).

Given a process Q, its internal runs $Q \longrightarrow Q_1 \longrightarrow Q_2 \longrightarrow \ldots$ are given by the sequences of τ labeled transitions, i.e., those transitions that the process can perform without requiring interaction with the context. Formally, $P \longrightarrow P'$ iff $P \xrightarrow{\tau} P'$. We denote with $\longrightarrow^*$ the reflexive and transitive closure of $\longrightarrow$. With $Deriv(P)$ we denote the set of processes reachable from P with a sequence of reduction steps, formally $Deriv(P) = \{Q \mid P \longrightarrow^* Q\}$.

A process Q is *dead* if there exists no Q' such that $Q \longrightarrow Q'$. We say that a process P *converges* if there exists a dead process P' in $Deriv(P)$. We say that P *terminates* if all its internal runs terminate, i.e., the process P cannot give rise to an infinite computation: formally, P *terminates* iff there exist no $\{P_i\}_{i\in\mathbf{N}}$, s.t. $P_0 = P$ and $P_j \longrightarrow P_{j+1}$ for any j. Observe that *process termination* implies *process convergence* while the vice versa does not hold.

Barbs are used to observe whether a process has the ability to perform, possibly after an internal run, an observable action on a specific channel; formally $P \Downarrow x$ iff there exist P' and P'' s.t. $P \longrightarrow^* P' \xrightarrow{\alpha} P''$ and $n(\alpha) = \{x\}$.

Definition 5. (weak bisimulation) A binary, symmetric relation $\mathcal{R}$ on processes is a *weak bisimulation* if $(P, Q) \in \mathcal{R}$ implies that, if $P \xrightarrow{\alpha} P'$, then one of the following holds:
— there exist Q', Q'', Q''' s.t. $Q \longrightarrow^* Q' \xrightarrow{\alpha} Q'' \longrightarrow^* Q'''$ and $(P', Q''') \in \mathcal{R}$;
— $\alpha = \tau$ and there exists Q' s.t. $Q \longrightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$.

Two processes P and Q are *weakly bisimilar*, written $P \approx Q$, if there exists a weak bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

3. Results for $\mathbf{CCS}_D$

A RAM (Shepherdson and Sturgis 1963), denoted in the following with R, is a computational model composed of a finite set of registers $r_1, \ldots, r_n$, that can hold arbitrary large natural numbers, and by a program composed by indexed instructions $(1 : I_1), \ldots, (m : I_m)$, that is a sequence of simple numbered instructions, like arithmetical operations (on the contents of registers) or conditional jumps. An internal state of a RAM is given by $(i, c_1, \ldots, c_n)$ where i is the program counter indicating the next instruction to be executed, and $c_1, \ldots, c_n$ are the current contents of the registers $r_1, \ldots, r_n$, respectively.

Without loss of generality, we assume that the registers contain the value 0 at the beginning of the computation and that the execution of the program begins with the first instruction $(1 : I_1)$. In other words, the initial configuration is $(1, 0, \ldots, 0)$. The computation continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached. More formally, we indicate by $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ the fact that the configuration of the RAM R changes from $(i, c_1, \ldots, c_n)$ to $(i', c'_1, \ldots, c'_n)$ after the execution of the i-th instruction.

In (Minsky 1967) it is shown that the following two instructions are sufficient to model every recursive function:

— $(i : Succ(r_j))$: adds 1 to the content of register r_j;
— $(i : DecJump(r_j, s))$: if the contents of register r_j is not zero then decreases it by 1 and go to the next instruction, otherwise jumps to instruction s.

3.1. *Undecidability of convergence and termination*

In the remainder of this section we will reason up-to the following structural congruence $\equiv_R$ used to remove terminated processes equal to $\mathbf{0}$ as well as unnecessary restrictions. Formally, $\equiv_R$ is the least congruence relation satisfying the following axioms:

$$P|\mathbf{0} \equiv_R P$$
$$(\nu x)P \equiv_R P \quad \text{if } x \notin fn(P)$$

The following proposition shows that processes congruent according to $\equiv_R$ have the same operational semantics; this allows us to reason up-to this structural congruence.

Proposition 1. Let $P, Q \in \mathrm{CCS}_D$ with $P \equiv_R Q$. If $P \xrightarrow{\alpha} P'$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv_R Q'$.

Proof. By induction on the proof of the relation $P \equiv_R Q$. □

We now show how to reduce RAM termination to either convergence or termination in CCS_D. Let R be a RAM with registers $r_1, \ldots, r_n$, and instructions $(1 : I_1), \ldots, (m : I_m)$; we model R as described in the following. For each $1 \leq i \leq m$, we model the i-th instruction $(i : I_i)$ of R with a program constant $Inst_i$ defined as follows.

$$Inst_i \stackrel{def}{=} \overline{inc_j}.Inst_{i+1} \qquad \text{if } I_i = Succ(r_j)$$
$$Inst_i \stackrel{def}{=} \overline{dec_j}.ack.Inst_{i+1} + \overline{zero_j}.Inst_s \qquad \text{if } I_i = DecJump(r_j, s)$$

In the first case, the process $Inst_i$ simply increments the register r_j (by firing the $\overline{inc_j}$ prefix) and activates the subsequent instruction; in the second case the process $Inst_i$ tries either to decrement or to test whether the register r_j is empty. According to the prefix which is fired ($\overline{dec_j}$ or $\overline{zero_j}$, respectively) the corresponding subsequent instruction is activated ($Inst_{i+1}$ or $Inst_s$, respectively). In the case of decrement, the process waits for an acknowledgment *ack* before activating the next instruction; this is necessary in order to activate the next instruction only after the actual update of the register.

In order to model RAM termination, we also need to take under consideration the instruction indexes outside $\{1, \ldots, m\}$ that can be reached as subsequent instructions of one of the defined instructions $(1 : I_1), \ldots, (m : I_m)$. For each of these indexes i we consider a corresponding constant defined as follows.

$$Inst_i \stackrel{def}{=} \mathbf{0} \qquad \text{if } I_i \text{ undefined (i.e. } i \notin \{1, \ldots, m\})$$

As far as the modeling of the registers is concerned, we represent each register r_j, which is initially empty, with a constant Z_j. The constant Z_j is defined in terms of two

other constants O_j and E_j:

$$\begin{array}{lcl} Z_j & \stackrel{def}{=} & zero_j.Z_j \ + \ inc_j.(\nu x)(O_j \mid x.\overline{ack}.Z_j) \\ O_j & \stackrel{def}{=} & dec_j.\overline{x} \ + \ inc_j.(\nu y)(E_j \mid y.\overline{ack}.O_j) \\ E_j & \stackrel{def}{=} & dec_j.\overline{y} \ + \ inc_j.(\nu x)(O_j \mid x.\overline{ack}.E_j) \end{array}$$

The idea behind this modeling of the registers is to exploit a chain of nested restrictions with a length corresponding to the content of the register. More precisely, the term Z_j represents the register when empty, while O_j and E_j model the register when it has an odd or an even content, respectively. Each time the register is incremented, the length of the chain of restrictions augments due to the creation of a new name. Observe that in order to avoid name collisions, the two names x and y are alternatively exploited. The use of two different names requires also the exploitation of the two different constants O_j and E_j.

Example 1. Consider a RAM with two registers r_1 and r_2 and the following program:

$$\begin{array}{l} (1 : Succ(r_1)) \\ (2 : Succ(r_1)) \\ (3 : DecJump(r_1, 5)) \\ (4 : DecJump(r_2, 3)) \end{array}$$

Assuming that the registers are initially both empty, and that the computation starts from the instruction with index 1, the computation of the RAM above consists of two increments of r_1, followed by a first decrement of r_1, a jump from instruction 4 to 3, a second decrement of r_1, a jump from instruction 4 to 3, and the final jump from instruction 3 to the undefined instruction 5. As will be formalized below, we will model the above RAM with the following process:

$$P \ = \ Inst_1 \mid Z_1 \mid Z_2$$

We have that P has the following 2 deterministic reduction steps corresponding to the two increment instructions:

$$\begin{array}{lcl} P & \longrightarrow & Inst_2 \mid (\nu x)(O_1 \mid x.\overline{ack}.Z_1) \mid Z_2 \\ & \longrightarrow & Inst_3 \mid (\nu x)((\nu y)(E_1 \mid y.\overline{ack}.O_1) \mid x.\overline{ack}.Z_1) \mid Z_2 \ = \ Q \end{array}$$

Note that at this point of the computation the first register contains the value 2; this is represented by the nesting of the two restrictions on the names x and y. The computation continues with the following 3 deterministic reduction steps corresponding to the first decrement:

$$\begin{array}{lcl} Q & \longrightarrow & ack.Inst_4 \mid (\nu x)((\nu y)(\overline{y} \mid y.\overline{ack}.O_1) \mid x.\overline{ack}.Z_1) \mid Z_2 \\ & \longrightarrow & ack.Inst_4 \mid (\nu x)((\nu y)(\mathbf{0} \mid \overline{ack}.O_1) \mid x.\overline{ack}.Z_1) \mid Z_2 \\ & \longrightarrow & Inst_4 \mid (\nu x)((\nu y)(\mathbf{0} \mid O_1) \mid x.\overline{ack}.Z_1) \mid Z_2 \\ & \equiv_R & Inst_4 \mid (\nu x)(O_1 \mid x.\overline{ack}.Z_1) \mid Z_2 \ = \ R \end{array}$$

Note that at this point the first register contains the value 1; this is represented by the fact that the inner restriction on y can be removed by the structural congruence. The

computation is then completed by the following 6 deterministic reduction steps:

$$\begin{array}{rcl} R & \longrightarrow & Inst_3 \mid (\nu x)(O_1 \mid x.\overline{ack}.Z_1) \mid Z_2 \\ & \longrightarrow & ack.Inst_4 \mid (\nu x)(\overline{x} \mid x.\overline{ack}.Z_1) \mid Z_2 \\ & \longrightarrow & ack.Inst_4 \mid (\nu x)(\mathbf{0} \mid \overline{ack}.Z_1) \mid Z_2 \\ & \longrightarrow & Inst_4 \mid (\nu x)(\mathbf{0} \mid Z_1) \mid Z_2 \ \equiv_R Inst_4 \mid Z_1 \mid Z_2 \\ & \longrightarrow & Inst_3 \mid Z_1 \mid Z_2 \\ & \longrightarrow & Inst_5 \mid Z_1 \mid Z_2 \end{array}$$

where this last process is dead because $Inst_5 \stackrel{def}{=} \mathbf{0}$.

We are now ready to formally define the process $R_j^{c_j}$ used to model the register r_j when its content is c_j.

$$\begin{array}{lcll} R_j^{c_j} & \stackrel{def}{=} & Z_j & \text{if } c_j = 0 \\ R_j^{c_j} & \stackrel{def}{=} & \underbrace{(\nu x)\big((\nu y)\big(\ldots(\nu x)}_{c_j \text{ restrictions}} \big(O_j \mid \underbrace{x.\overline{ack}.E_j\big)\ldots\big) \mid\ y.\overline{ack}.O_j\big) \mid\ x.\overline{ack}.Z_j\big)}_{c_j \text{ processes}} & \text{if } c_j > 0 \text{ is odd} \\ R_j^{c_j} & \stackrel{def}{=} & \underbrace{(\nu x)\big((\nu y)\big(\ldots(\nu y)}_{c_j \text{ restrictions}} \big(E_j \mid \underbrace{y.\overline{ack}.O_j\big)\ldots\mid\ y.\overline{ack}.O_j\big) \mid\ x.\overline{ack}.Z_j\big)}_{c_j \text{ processes}} & \text{if } c_j > 0 \text{ is even} \end{array}$$

Definition 6. Let R be a RAM with program instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$. Given the configuration $(i, c_1, \ldots, c_n)$ of R we define

$$[\![(i, c_1, \ldots, c_n)]\!]_D \ = \ Inst_i | R_1^{c_1} | \ldots | R_n^{c_n}$$

with $Inst_i$ and $R_j^{c_j}$ defined as above.

As we have already discussed, the computation of the encoding of RAMs proceeds deterministically, and corresponds exactly to the computation of the corresponding RAM; thus the encoding terminates if and only if the considered RAM terminates, as stated by the following theorem.

The remainder of this subsection is dedicated to the formal proof of this correctness result.

We say that a process P performs a deterministic reduction step if it is not dead and if $P \longrightarrow P'$ and $P \longrightarrow P''$ then $P' \equiv_R P''$. A process performs a deterministic computation if all of its derivatives are either dead or perform a deterministic computation step.

We are now ready to state the correspondence result between the computation of one RAM and of its encoding.

Proposition 2. Let R be a RAM with program instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$. We have that given the configuration $(i, c_1, \ldots, c_n)$ one of the following holds:

— $i \notin \{1, \ldots, m\}$ and $[\![(i, c_1, \ldots, c_n)]\!]_D$ is dead;
— $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$ and $[\![(i, c_1, \ldots, c_n)]\!]_D$ performs either one or three deterministic reduction steps reaching a process $Q \equiv_R [\![(i', c'_1, \ldots, c'_n)]\!]_D$.

Proof. It is immediate to see that $[\![(i, c_1, \ldots, c_n)]\!]_D$ is dead if and only if the instruction of index i is undefined (i.e. $i \notin \{1, \ldots, m\}$). In this case the first item holds.

If the instruction with index i is defined then there exists $(i', c'_1, \ldots, c'_n)$ such that $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$. This RAM computational step is either an increment, or a decrement, or a test-for-zero action. In case of increment or test-for-zero we have that $[\![(i, c_1, \ldots, c_n)]\!]_D$ has a deterministic reduction step leading to $[\![(i', c'_1, \ldots, c'_n)]\!]_D$. In case of decrement, $[\![(i, c_1, \ldots, c_n)]\!]_D$ performs three deterministic steps before reaching $[\![(i', c'_1, \ldots, c'_n)]\!]_D$. In this case the second item holds. □

As a simple corollary of the above Proposition we have that convergence is undecidable as we have that a RAM terminates if and only if the corresponding encoding has a terminating computation.

Theorem 1. Let R be a RAM with program instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$. Given the initial configuration $(1, 0, \ldots, 0)$ of R we have that R terminates if and only if the process $[\![(1, 0, \ldots, 0)]\!]_D$ converges.

Proof. As a consequence of Proposition 2, we have that $(1, 0, \ldots, 0) \rightarrow_R (i_1, c_1^1, \ldots, c_n^1) \rightarrow_R \ldots \rightarrow_R (i_k, c_1^k, \ldots, c_n^k)$ if and only if $[\![(1, 0, \ldots, 0)]\!]_D \longrightarrow^+ [\![(i_1, c_1^1, \ldots, c_n^1)]\!]_D \longrightarrow^+ \ldots \longrightarrow^+ [\![(i_k, c_1^k, \ldots, c_n^k)]\!]_D$. This can be proved by induction on the length k of the computation.

Assume now that the RAM terminates. In this case we have that $(1, 0, \ldots, 0) \rightarrow_R (i_1, c_1^1, \ldots, c_n^1) \rightarrow_R \ldots \rightarrow_R (i_k, c_1^k, \ldots, c_n^k)$ with the instruction of index i_k undefined. By the first item of Proposition 2 we have that $[\![(i_k, c_1^k, \ldots, c_n^k)]\!]_D$ is dead, thus $[\![(1, 0, \ldots, 0)]\!]_D$ converges due to the computation $[\![(1, 0, \ldots, 0)]\!]_D \longrightarrow^+ [\![(i_1, c_1^1, \ldots, c_n^1)]\!]_D \longrightarrow^+ \ldots \longrightarrow^+ [\![(i_k, c_1^k, \ldots, c_n^k)]\!]_D$.

Assume now that the RAM does not terminate. By contradiction we prove that $[\![(1, 0, \ldots, 0)]\!]_D$ does not converge. Assume that $[\![(1, 0, \ldots, 0)]\!]_D$ converges. In this case we have that $[\![(1, 0, \ldots, 0)]\!]_D \longrightarrow^+ [\![(i_1, c_1^1, \ldots, c_n^1)]\!]_D \longrightarrow^+ \ldots \longrightarrow^+ [\![(i_k, c_1^k, \ldots, c_n^k)]\!]_D$ with the instruction of index i_k undefined. By the first item of Proposition 2 we have that the considered RAM terminates due to the computation $(1, 0, \ldots, 0) \rightarrow_R (i_1, c_1^1, \ldots, c_n^1) \rightarrow_R \ldots \rightarrow_R (i_k, c_1^k, \ldots, c_n^k)$. This contradicts the initial assumption, i.e., that the RAM does not terminate. □

3.2. *Undecidability of barb and weak bisimulation*

Here, we slightly modify the above modeling of RAMs in order to prove that also barb and weak bisimulation are undecidable. The modeling is essentially the same with the unique difference that we add outer restrictions, on all the names used by the encoding, in order to keep unobservable the internal actions, and we add a unique observable action $\overline{w}$, that communicates termination. In this way, we have that the modeling of a RAM R is weakly bisimilar to $\overline{w}$ (resp. has a barb on the channel w) if and only if R terminates thus proving that weak bisimulation (resp. barb) is undecidable.

All the definitions of the encoding are as in the previous Subsection excluding the

definition of the undefined instructions which is now as follows.

$$Inst_i \stackrel{def}{=} \overline{w} \qquad \text{if } I_i \text{ undefined (i.e. } i \notin \{1,\ldots,m\})$$

Recall that instructions outside the range $\{1,\ldots,m\}$ can be reached only on program termination.

We can now prove the undecidability of weak barb.

Theorem 2. Let R be a RAM with program instructions $(1 : I_1),\ldots,(m : I_m)$ and registers $r_1,\ldots,r_n$. We have that R terminates if and only if $[\![(1,0,\ldots,0)]\!]_D \Downarrow w$.

Proof. As we have already discussed in the previous Subsection, the computation of the encoding of RAMs proceeds deterministically, and corresponds exactly to the computation of the corresponding RAM.

Assume now that the RAM R terminates. In this case the encoding has a deterministic internal run leading to a process that contain the encoding of an undefined instruction $Inst_i$, i.e., the process $\overline{w}$. We can conclude that the encoding has a barb on w.

Assume now that the RAM R does not terminate. In this case the encoding has an infinite deterministic internal run, thus no process $Inst_i$, with corresponding instruction I_i undefined, can be produced. As only such processes can perform actions on the channel w, we can conclude that the encoding has no barb on w. □

The above theorem proves that barb is undecidable in CCS_D. As a consequence of this theorem, and as a consequence of the fact that the encoding has only one deterministic internal run, we have that also weak bisimulation is undecidable. In fact, it is sufficient to add outer restrictions to the encoding, on all the names used by the encoding except w, in order to obtain a process which is weakly equivalent to $\overline{w}$ if and only if the encoding has a barb on w. Formally, we have that:

$$[\![(1,0,\ldots,0)]\!]_D \Downarrow w$$
$$\text{if and only if}$$
$$(\nu\ inc_1, dec_1, zero_1, \ldots, inc_n, dec_n, zero_n, ack)\big([\![(1,0,\ldots,0)]\!]_D\big) \approx \overline{w}$$

Hence, also weak bisimulation is undecidable.

4. Results for $\mathrm{CCS}_!$

4.1. *Undecidability of convergence and weak bisimulation*

We prove that $\mathrm{CCS}_!$ is powerful enough to model, at least in a nondeterministic way, any Random Access Machine. Our encoding is nondeterministic because it introduces computations which do not follow the expected behavior of the modeled RAM. However, all these computations are infinite. This ensures that, given a RAM, its modeling has a terminating computation if and only if the RAM terminates. This proves that *convergence* is undecidable.

Exploiting the encoding, we also prove that weak bisimulation is undecidable. The idea is to use only two observable actions, namely $\overline{w}$ and $\overline{w'}$. The former makes visible the fact that the program counter has reached an index outside the original range $\{1,\ldots,m\}$

of program instructions. The latter makes visible the activation of an incorrect infinite computation. In this way, we have that a correct terminating run of the encoding includes one and only one observable action $\overline{w}$, while all incorrect computations include also the other observable action $\overline{w'}$ Thus, if P is the encoding of a RAM R, then R terminates if and only if $P \approx \tau.P + \tau.\overline{w}$. This proves that *weak bisimulation* is undecidable.

In this section we reason up-to an extended version of the structural congruence $\equiv_R$ that allows also for the reordering of processes in parallel compositions. The new definition includes also the following axioms

$$P|Q \equiv_R Q|P \qquad\qquad P|(Q|R) \equiv_R (P|Q)|R$$

As we proved that the previous version of $\equiv_R$ preserves the operational semantics of CCS_D. This new extended version of $\equiv_R$ preserves the operational semantics of the calculus with replication.

Proposition 3. Let $P, Q \in \mathrm{CCS}_!$ with $P \equiv_R Q$. If $P \xrightarrow{\alpha} P'$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv_R Q'$.

Proof. By induction on the proof of the relation $P \equiv_R Q$. □

Let R be a RAM with registers $r_1, \ldots, r_n$, and instructions $(1 : I_1), \ldots, (m : I_m)$. We model separately registers and instructions.

The program counter is modeled with a process $\overline{p}_i$ indicating that the i-th instruction is the next to be executed. For each $1 \leq i \leq m$, we model the i-th instruction $(i : I_i)$ of R with a replicated process which is guarded by an input operation p_i. Once activated, the instruction performs its operation on the registers, then waits for an acknowledgment indicating that the operation has been performed, and finally updates the program counter by producing $\overline{p_{i+1}}$ (or $\overline{p_s}$ in case of jump).

Formally, for any $1 \leq i \leq m$, the instruction $(i : I_i)$ is modeled by $[\![(i : I_i)]\!]$ which is a shorthand notation for the following processes.

$$[\![(i : I_i)]\!] \quad : \quad !p_i.(\overline{inc_j} \mid ack.\overline{p_{i+1}}) \qquad \text{if } I_i = Succ(r_j)$$

$$[\![(i : I_i)]\!] \quad : \quad !p_i.(\overline{dec_j} \mid (ack.\overline{p_{i+1}} \; + \; jmp.ack.\overline{p_s})) \qquad \text{if } I_i = DecJump(r_j, s)$$

It is worth noting that a program counter message $\overline{p_i}$, with the index i outside the range $\{1, \ldots, m\}$, is produced on program termination. It is not restrictive to assume that the unique of these indexes is $m+1$. For this index $m+1$ we assume the presence of a process $p_{m+1}.\overline{w}$ able to make termination observable on the channel w.

We model each register r_j, when it contains c_j, with the following process simply denoted with $[\![r_j = c_j]\!]$ in the following:

$$\begin{array}{rll} [\![r_j = c_j]\!] & : & (\nu\ m, u)\ \big(\\ & & \quad \prod_{c_j} \overline{u} \mid \\ & & \quad inc_j.(\overline{m} \mid \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_j})) \mid \\ & & \quad !m.(\overline{ack} \mid inc_j.(\overline{m} \mid \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_j}))) \big) \mid \\ & & !nr_j.(\nu\ m, u)\ \big(\\ & & \quad \overline{m} \mid !m.(\overline{ack} \mid inc_j.(\overline{m} \mid \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_j}))) \big) \end{array}$$

where DIV is a process able to activate the following infinite observable computation:

$$DIV \ : \quad \overline{w'} \mid !w'.\overline{w'}$$

Example 2. We consider a RAM with only one register r_j, initially empty, and the following program instructions:

$$(1 : Succ(r_1))$$
$$(2 : DecJump(r_1, 2))$$

The RAM computation consists simply of an increment followed by a decrement. The encoding of this RAM includes the encoding of the program counter, of the instructions, and of the register:

$$\begin{array}{rl} P \ = & \overline{p_1} \mid !p_1.(\overline{inc_1}|ack.\overline{p_2}) \mid !p_2.(\overline{dec_1} \mid (ack.\overline{p_3} \ + \ jmp.ack.\overline{p_2})) \mid \\ & (\nu\ m,u)\ \big(\\ & \quad inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1})) \mid \\ & \quad !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \ \big) \mid \\ & !nr_1.(\nu\ m,u)\ \big(\\ & \quad \overline{m} \mid !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \ \big) \end{array}$$

This process has 4 deterministic reduction steps (corresponding to the execution of the increment instruction) leading to a process which is structurally equivalent to the following one:

$$\begin{array}{rl} Q \ = & \overline{p_2} \mid !p_1.(\overline{inc_1}|ack.\overline{p_2}) \mid !p_2.(\overline{dec_1} \mid (ack.\overline{p_3} \ + \ jmp.ack.\overline{p_2})) \mid \\ & (\nu\ m,u)\ \big(\overline{u} \mid \\ & \quad inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1})) \mid \\ & \quad !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \ \big) \mid \\ & !nr_1.(\nu\ m,u)\ \big(\\ & \quad \overline{m} \mid !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \ \big) \end{array}$$

The above process Q has two deterministic reduction steps leading to a process structurally equivalent to the following one:

$$\begin{array}{rl} R \ = & (ack.\overline{p_3} \ + \ jmp.ack.\overline{p_2}) \mid !p_1.(\overline{inc_1}|ack.\overline{p_2}) \mid !p_2.(\overline{dec_1} \mid (ack.\overline{p_3} \ + \ jmp.ack.\overline{p_2})) \mid \\ & (\nu\ m,u)\ \big(\overline{u} \mid \\ & \quad (u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1})) \mid \\ & \quad !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \ \big) \mid \\ & !nr_1.(\nu\ m,u)\ \big(\\ & \quad \overline{m} \mid !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \ \big) \end{array}$$

The above process R has two possible reduction steps: either a synchronization on u or a synchronization on $jump$. In the first case the computation proceeds deterministically until a dead process with the program counter $\overline{p_3}$ is reached. In the second case the

following process is reached:

$$\begin{array}{rl} S \;=\; & ack.\overline{p_2} \mid !p_1.(\overline{inc_1}|ack.\overline{p_2}) \mid !p_2.(\overline{dec_1} \mid (ack.\overline{p_3} \;+\; jmp.ack.\overline{p_2})) \mid \\ & (\nu\ m,u)\ \big(\overline{u} \mid \\ & \quad (u.DIV \mid \overline{nr_1}) \mid \\ & \quad !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \big) \mid \\ & !nr_1.(\nu\ m,u)\ \big(\\ & \quad \overline{m} \mid !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \big) \end{array}$$

It is easy to see that the above process does not converge because it is not possible to remove the possibility for the processes $\overline{u}$ and $u.DIV$ to synchronize. This synchronization activates the divergent process DIV. This is made clear assuming that such synchronization is delayed for at least three reduction steps. In this case, the computation continues with three synchronizations on the channels nr_1, m, and ack, and the following process is reached:

$$\begin{array}{rl} T \;=\; & \overline{p_2} \mid !p_1.(\overline{inc_1}|ack.\overline{p_2}) \mid !p_2.(\overline{dec_1} \mid (ack.\overline{p_3} \;+\; jmp.ack.\overline{p_2})) \mid \\ & (\nu\ m,u)\ \big(\overline{u} \mid u.DIV \mid \\ & \quad !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \big) \mid \\ & (\nu\ m,u)\ \big(\\ & \quad inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1})) \mid \\ & \quad !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \big) \mid \\ & !nr_1.(\nu\ m,u)\ \big(\\ & \quad \overline{m} \mid !m.(\overline{ack} \mid inc_1.(\overline{m} \mid \overline{u}) + dec_1.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_1}))) \big) \end{array}$$

The processes $\overline{u}$ and $u.DIV$ occurring on the second line of the definition above cannot interact with any other processes, thus the unique actions they can perform is a synchronization that activates the divergent process DIV.

Observe that the content c_j of the register is modeled by the parallel composition of a corresponding number of processes $(\overline{u} \mid d.u.\overline{m})$; the term $\overline{u}$ represents a unit inside the register, while $d.u.\overline{m}$ is an auxiliary term that is responsible for removing the unit when the register is decremented. The register is blocked until either the prefix inc_j or the prefix dec_j fires. In the former case, the effect is that a new instance of the process $(\overline{u} \mid d.u.\overline{m})$ is spawn (and the $\overline{ack}$ is produced after having re-activated the process able to perform either the inc_j or the dec_j actions). In the latter case, the computation proceeds nondeterministically. Either the register is actually decremented (activating one of the processes $d.u.\overline{m}$), or the decision to jump is taken. In this case, a possibly diverging process $u.DIV$ is spawn and a new register instance is activated executing $\overline{nr_j}$. The new instance of the register guarantees that if the register was not empty at the time of the jump, the process $u.DIV$ eventually executes the u action (performing a synchronization with one of the still available units $\overline{u}$) and starts its infinite computation.

It is worth to note that even in correct computations (i.e., those corresponding to the RAM computation) some garbage is left when a jump instruction is executed. The garbage is due to the activation of a new instance of the register. The garbage processes

(one for each register) is as follows.

$$G_j \;:\quad (\nu\ m,u)\ \big(\ u.DIV \mid !m.(\overline{ack} \mid inc_j.(\overline{m} \mid \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_j}))) \ \big)$$

Note that the a garbage process is dead as it is composed of processes prefixed by input actions on restricted names.

We are now ready to formally define the RAM encoding.

Definition 7. Let R be a RAM with program instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$. Given the configuration $(i, c_1, \ldots, c_n)$ of R we define

$$\begin{array}{l}[\![(i, c_1, \ldots, c_n)]\!]_R \;= \\ (\nu\ p_1, \ldots, p_m, p_{m+1}, inc_1, dec_1, nr_1, \ldots, inc_n, dec_n, nr_n, ack, jmp) \\ (\ \ \overline{p_i} \mid [\![(1 : I_1)]\!] \mid \ldots \mid [\![(m : I_m)]\!] \mid p_{m+1}.\overline{w} \mid \\ \quad [\![r_1 = c_1]\!] \mid \ldots \mid [\![r_n = c_n]\!] \mid \prod_{k_1} G_1 \mid \ldots \mid \prod_{k_n} G_n \ \ )\end{array}$$

where the modeling of program instructions $[\![(i : I_i)]\!]$, the modeling of registers $[\![r_j = c_j]\!]$, and the garbage processes $G_1, \ldots, G_n$ have been defined above, and $k_1 \ldots k_n$ are natural numbers. Observe that due to the presence of $k_1, \ldots, k_n$ the target of the encoding is not a unique process but it is a class of processes which differ only in the amount of garbage.

We are now ready to state the correspondence result between the computations of one RAM and of its encoding.

Proposition 4. Let R be a RAM with program instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$. We have that given the configuration $(i, c_1, \ldots, c_n)$ one of the following holds:

— $i \notin \{1, \ldots, m\}$ and each process in $[\![(i, c_1, \ldots, c_n)]\!]_R$ has a deterministic reduction step leading to a dead process,

— I_i is an increment instruction, $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$, and each process in $[\![(i, c_1, \ldots, c_n)]\!]_R$ performs four deterministic reduction steps reaching a process Q which is structurally congruent w.r.t. $\equiv_R$ to some process in $[\![(i', c'_1, \ldots, c'_n)]\!]_R$,

— I_i is a decrement instruction, $(i, c_1, \ldots, c_n) \rightarrow_R (i', c'_1, \ldots, c'_n)$, and each process in $[\![(i, c_1, \ldots, c_n)]\!]_R$ performs two deterministic reduction steps reaching a process Q for which the following holds: Q is not dead and for each Q' such that $Q \longrightarrow Q'$, either Q' has a sequence of deterministic reduction steps (of length either 2 or 3) to a process Q'' which is structurally congruent w.r.t. $\equiv_R$ to some process in $[\![(i', c'_1, \ldots, c'_n)]\!]_R$, or Q' does not converge.

Proof. The proof for the first two items is immediate. In the third case, we observe that either the computation of the encodings in $[\![(i, c_1, \ldots, c_n)]\!]_R$ faithfully reproduces the RAM computation (and in this case the first part holds) or a "wrong jump" (i.e. a synchronization on the channel jmp is executed even if the tested register c_j is not empty) is executed. In this case, the process

$$(\nu\ m,u)\ \big(\prod_{c_j} \overline{u} \mid u.DIV \mid \overline{nr_j}) \mid !m.(\overline{ack} \mid inc_j.(\overline{m} \mid \overline{u}) + dec_j.(u.\overline{m} + \overline{jmp}.(u.DIV \mid \overline{nr_j}))) \ \big)$$

appears at top level (i.e. is not guarded by any prefix). We have that, as $c_j > 0$, any process including the above process at top level does not terminate. □

As a simple corollary of the above Proposition we have that convergence is undecidable as we have that a RAM terminates if and only if the corresponding encoding has a terminating computation.

Theorem 3. Let R be a RAM with program $(1 : I_1), \ldots, (m : I_m)$ and state $(i, c_1, \ldots, c_n)$, and let the process P be in $[\![(i, c_1, \ldots, c_n)]\!]_R$. Then $(i, c_1, \ldots, c_n)$ terminates if and only if P converges.

Proof. Similar to the proof of the Theorem 1 in which Proposition 4 is used instead of Proposition 2. □

This proves that convergence is undecidable in $\text{CCS}_!$.

It is worth to note that differently from the RAM encoding presented for CCS_D, we cannot conclude that also termination is undecidable. In fact, the processes produced by the new encoding are nondeterministic as additional infinite computations are added also to processes that encode a terminating RAM. Hence, it is no longer true that for these processes convergence and termination coincide.

We conclude this section proving the also weak bisimulation is undecidable.

Theorem 4. Let R be a RAM with program $(1 : I_1), \ldots, (m : I_m)$ and state $(i, c_1, \ldots, c_n)$, and let the process P be in $[\![(i, c_1, \ldots, c_n)]\!]_R$. Then $(i, c_1, \ldots, c_n)$ terminates if and only if $P \approx \tau.P + \tau.\overline{w}$.

Proof. Assume that $(i, c_1, \ldots, c_n)$ does not terminate. Consider one of the processes P in $[\![(i, c_1, \ldots, c_n)]\!]_R$. By the Theorem 3 we have that any P cannot converge. In order to prove that $P \not\approx \tau.P + \tau.\overline{w}$ we proceed by contradiction assuming that $P \approx \tau.P + \tau.\overline{w}$. In this case, as $\tau.P + \tau.\overline{w} \xrightarrow{\tau} \overline{w} \xrightarrow{\overline{w}} \mathbf{0}$, we have that also $P \longrightarrow^* P' \xrightarrow{\overline{w}} P'' \longrightarrow^* P'''$ with $P''' \approx \mathbf{0}$. The unique way in which P' can perform a transition labeled with $\overline{w}$ is by activation of the process $p_{m+1}.\overline{w}$. Moreover, as P cannot converge, also P' (which is one of its derivatives) cannot. It is easy to see that any infinite computation starting from P' cannot include reductions executed by an instruction (in fact, the program counter process $\overline{p_i}$ is definitively consumed by the input action p_{m+1}). Thus, all computations are due to the presence in P' of some DIV process. It is easy to see that such process is present also in P''. Hence all derivatives of P'', thus also P''', can perform actions on the observable channel w' that cannot be mimicked by $\mathbf{0}$. Hence, $P''' \not\approx \mathbf{0}$ which is in contradiction with what we observed above.

Assume now that $(i, c_1, \ldots, c_n)$ terminates. Consider a process P in $[\![(i, c_1, \ldots, c_n)]\!]_R$. We prove that $P \approx \tau.P + \tau.\overline{w}$ showing that the following relation

$$\begin{aligned}
\mathcal{R} \;=\;& Id \;\cup\; \{(P, \tau.P + \tau.\overline{w})\} \\
\cup\;& \{(Q, \overline{w}) \mid Q \equiv_R Q'' \text{ with } Q' \longrightarrow Q'' \text{ and } Q' \in [\![(m+1, c_1, \ldots, c_n)]\!]_R \\
& \qquad \text{for some } c_1, \ldots, c_n\} \\
\cup\;& \{(Q, \mathbf{0}) \mid Q \equiv_R Q''' \text{ with } Q' \longrightarrow Q'' \xrightarrow{\overline{w}} Q''' \text{ and } Q' \in [\![(m+1, c_1, \ldots, c_n)]\!]_R \\
& \qquad \text{for some } c_1, \ldots, c_n\}
\end{aligned}$$

(where Id is the identity relation) is a weak bisimulation.

We first consider the pair $(P, \tau.P + \tau.\overline{w})$. Any transition executable by P can be mimicked by the $\tau.P$ branch. The process $\tau.P+\tau.\overline{w}$ can perform two kinds of τ transitions. The first one can be mimicked by P with no transition. The second one, can be mimicked by P by executing the whole terminating computation (that exists by Proposition 4) leading to a process structurally equivalent to a process in $[\![(m+1, c_1, \ldots, c_n)]\!]_R$. This process can perform an additional reduction (i.e. the synchronization on the channel p_{m+1}) and reach one of those terms Q such that, by the second line of the definition of $\mathcal{R}$, $(Q, \overline{w})$ is in $\mathcal{R}$.

We now consider one of the pairs $(Q, \overline{w})$ that are in $\mathcal{R}$ due to the second line in the definition. Both processes have a unique outgoing transition labeled with $\overline{w}$, leading respectively to a process Q' and the empty process $\mathbf{0}$ such that, by the last line of the definition of $\mathcal{R}$, $(Q', \mathbf{0})$ is in $\mathcal{R}$.

We complete the proof simply observing that, given one of the pairs $(Q, \mathbf{0})$ that are in $\mathcal{R}$ due to the last line in the definition, we have that both processes have no outgoing transitions. □

This proves that also weak bisimulation is undecidable in CCS$_!$.

4.2. *Decidability of termination*

We have already observed that the undecidability result proved in the Theorem 3 applies to convergence and not to termination. In this subsection we prove that, differently from the calculus with general recursion, in CCS$_!$ termination is indeed decidable. This result is based on the theory of well-structured transition systems, so we start this section by recalling some basic definitions and results from (Finkel and Schnoebelen 2001) that will be used in the following. Then, since these results are valid for finitely branching transition systems, we provide an alternative finitely branching semantics for CCS$_!$ which is equivalent w.r.t. termination to the one presented in Section 2. This allows us to prove that termination is decidable for CCS$_!$ processes by defining a suitable well-structured transition system for CCS$_!$.

4.3. *Well-Structured Transition System*

The following results and definitions are from (Finkel and Schnoebelen 2001) unless differently specified. Recall that a *quasi-order* (or, equivalently, preorder) is a reflexive and transitive relation.

Definition 8 (Well-quasi-order). A *well-quasi-order* (wqo) is a quasi-order $\leq$ over a set X such that, for any infinite sequence $x_0, x_1, x_2, \ldots$ in X, there exist indexes $i < j$ such that $x_i \leq x_j$.

Note that if $\leq$ is a wqo then any infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \ldots$ (with $i_0 < i_1 < i_2 < \ldots$). Thus well-quasi-orders exclude the possibility of having infinite strictly decreasing sequences.

We also need a formal definition for (finitely branching) transition systems. This can be given as follows. Here and in the following $\rightarrow^+$ (resp. $\rightarrow^*$) denotes the transitive (resp. the reflexive and transitive) closure of the relation $\rightarrow$.

Definition 9. A *transition system* is a structure $TS = (S, \rightarrow)$, where S is a set of *states* and $\rightarrow \subseteq S \times S$ is a set of *transitions*. We define $Succ(s)$ as the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of S. We say that TS is *finitely branching* if, for each $s \in S$, $Succ(s)$ is finite.

We also define $Pred(s)$ as the set $\{s' \in S \mid s' \rightarrow s\}$ of immediate predecessors of s, while $Pred^*(s)$ denotes the set $\{s' \in S \mid s' \rightarrow^* s\}$ (of predecessors of s).

The functions $Succ$, $Pred$ and $Pred^*$ will be used also on sets by assuming that in this case they are defined by the point-wise extension of the above definitions.

The key tool to decide several properties of computations is the notion of well-structured transition system. This is a transition systems equipped with a well-quasi-order on states which is (upward) compatible with the transition relation. Here we will use a strong version of compatibility, hence the following definition.

Definition 10 (Well-structured transition system with strong compatibility). A *well-structured transition system with strong compatibility* is a transition system $TS = (S, \rightarrow)$, equipped with a quasi-order $\leq$ on S, such that the two following conditions hold:

1 $\leq$ is a well-quasi-order;
2 $\leq$ is strongly (upward) compatible with $\rightarrow$, that is, for all $s_1 \leq t_1$ and all transitions $s_1 \rightarrow s_2$, there exists a state t_2 such that $t_1 \rightarrow t_2$ and $s_2 \leq t_2$ holds.

The following theorem is a special case of Theorem 4.6 in (Finkel and Schnoebelen 2001) and will be used to obtain our decidability result.

Theorem 5. Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with strong compatibility, decidable $\leq$ and computable $Succ$. Then the existence of an infinite computation starting from a state $s \in S$ is decidable.

We will need also a result due to Higman which allows to extend a well-quasi-order from a set S to the set of the finite sequences on S. To be more precise, given a set S let use denote by S^* the set of finite sequences built by using elements in S. We can define a quasi order on S^* as follows.

Definition 11. Let S be a set and $\leq$ a quasi order over S. The relation $\leq_*$ over S^* is defined as follows. Let $t, u \in S^*$, with $t = t_1 t_2 \ldots t_m$ and $u = u_1 u_2 \ldots u_n$. We have that $t \leq_* u$ iff there exists an injection f from $\{1, 2, \ldots, m\}$ to $\{1, 2, \ldots, n\}$ such that $t_i \leq u_{f(i)}$ and $i \leq f(i)$ for $i = 1, \ldots, m$.

The relation $\leq_*$ is clearly a quasi order over S^*. It is also a wqo, since we have the following result.

Lemma 1. [Higman] Let S be a set and $\leq$ a wqo over S. Then the relation$\leq_*$ is a wqo over S^*.

Finally we will use also the following proposition, whose proof is immediate.

Proposition 5. Let S be a finite set. Then the equality is a wqo over S.

4.3.1. *A finitely branching transition system for CCS!*

Due to the rule for replication, the transition systems for CCS! defined in Section 2 is not finitely branching. As previously mentioned, Theorem 5 applies only to finitely branching transition systems. Hence, in order to use it, we need to define a new finitely branching transition system for CCS! which is equivalent to the one presented in Section 2 w.r.t. termination. This new transition system can be obtained by reformulating the rule for replication as follows.

Let us define a new transition relation $\overset{\alpha}{\mapsto}$ over CCS! processes as the least relation satisfying all the axioms and rules of Table 2 (where $\xrightarrow{\alpha}$ is substituted by $\overset{\alpha}{\mapsto}$) plus the following rules `REPL1` and `REPL2`.

$$\texttt{REPL1}: \frac{P \overset{\alpha}{\mapsto} P'}{!P \overset{\alpha}{\mapsto} P' \mid !P} \qquad \texttt{REPL2}: \frac{P \overset{\alpha}{\mapsto} P' \quad P \overset{\overline{\alpha}}{\mapsto} P''}{!P \overset{\tau}{\mapsto} P' \mid P'' \mid !P}$$

Rule REPL of Section 2 caused an infinitely branching transition system because in the premise one could use an unbound number of copies of the process P. So, for example, from $(\alpha.P)!$ one could obtain the transitions

$$\begin{array}{l} !(\alpha.P) \xrightarrow{\alpha} P|(\alpha.P)!, \\ !(\alpha.P) \xrightarrow{\alpha} P|P|!(\alpha.P) \\ \vdots \end{array}$$

This behavior is ruled out by using `REPL1`, where the premise does not involve the !. Rule `REPL2` is added because some computations depend on the possibility of synchronizing two copies of P generated by $!P$.

As in the case of the standard transition system, we assume that the reductions $\mapsto$ of the new semantics corresponds to the τ–labeled transitions $\overset{\tau}{\mapsto}$. Moreover, also for the new semantics, we say that a process P terminates if and only if all its computations are finite, i.e. it cannot give rise to an infinite sequence of $\mapsto$ reductions.

In order to prove the equivalence w.r.t. termination of the semantics of CCS! presented in Section 2 with the alternative semantics presented here, we use the following congruence on processes which allows us to simplify the proofs.

Definition 12. We define $\equiv_T$ as the least congruence relation satisfying the following axioms:

$$\begin{array}{l} P|Q \equiv_T Q|P \\ P|(Q|R) \equiv_T (P|Q)|R \\ P|\mathbf{0} \equiv_T P \\ P|!P \equiv_T !P \end{array}$$

The following proposition shows that congruent processes are equivalent w.r.t. termination when considering the $\longrightarrow$ transitions.

Proposition 6. Let $P, Q \in \mathrm{CCS}_!$ with $P \equiv_T Q$. If $P \xrightarrow{\alpha} P'$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv_T Q'$.

Proof. By induction on the proof of the relation $P \equiv_T Q$. □

The same hold also in the case of $\mapsto$ transitions, as shown by the following.

Proposition 7. Let $P, Q \in \mathrm{CCS}_!$ with $P \equiv_T Q$. If $P \overset{\alpha}{\mapsto} P'$ then there exists Q' such that $Q \overset{\alpha}{\mapsto} Q'$ and $P' \equiv_T Q'$.

Proof. By induction on the proof of the relation $P \equiv_T Q$. □

Then we have the following two propositions which show that $\longrightarrow$ and $\mapsto$ induce the same derivations up to $\equiv_T$ congruence.

Proposition 8. Let $P \in \mathrm{CCS}_!$. If $P \xrightarrow{\alpha} P'$ then there exists P'' such that $P \overset{\alpha}{\mapsto} P''$ and $P' \equiv_T P''$.

Proof. By induction on the proof of the derivation $P \xrightarrow{\alpha} P'$. □

Proposition 9. Let $P \in \mathrm{CCS}_!$. If $P \overset{\alpha}{\mapsto} P'$ then there exists P'' such that $P \xrightarrow{\alpha} P''$ and $P' \equiv_T P''$.

Proof. By induction on the proof of the derivation $P \overset{\alpha}{\mapsto} P'$. □

From this four propositions it is immediate to obtain the result stating the equivalence w.r.t. termination.

Corollary 1. Let $P \in \mathrm{CCS}_!$. Then P terminates according to the semantics (defined by) $\longrightarrow$ iff P terminates according to the new semantics $\mapsto$.

4.3.2. *Termination is decidable in CCS*$_!$

The last, and more complex, step in order to prove that termination is decidable for $(\mathrm{CCS}_!, \longrightarrow)$ is to equip the transition system $(\mathrm{CCS}_!, \mapsto)$ with a well-quasi-order compatible with $\mapsto$. The desired result then follows from Theorem 5 and Corollary 1.

To define the well-quasi-order we first introduce another congruence con processes, which is simpler than $\equiv_T$ and which turns out to be compatible with $\mapsto$. In fact, differently from the case of the previous section, here the congruence is needed only to simplify the definition of the quasi-order, hence we do not need to take into account the replication operator.

Definition 13. We define $\equiv_w$ as the least congruence relation satisfying the following axioms:

$$P|Q \equiv_w Q|P$$
$$P|(Q|R) \equiv_w (P|Q)|R$$
$$P|\mathbf{0} \equiv_w P$$

We can now define the relation $\preceq$ which will be proven to be a well-quasi-order.

Definition 14. Let $P, Q \in \mathrm{CCS}_!$. We write $P \preceq Q$ iff there exist n, $x_1, \ldots, x_n$,P', R, $P_1, \ldots, P_n$, $Q_1, \ldots, Q_n$ such that $P \equiv_w P' | \prod_{i=1}^{n} (\nu x_i) P_i$,
$Q \equiv_w P' | R | \prod_{i=1}^{n} (\nu x_i) Q_i$, and $P_i \preceq Q_i$ for $i = 1, \ldots, n$.

Intuitively $P \preceq Q$ holds if Q can be obtained, up to $\equiv_w$, from P by adding some parallel processes while preserving the nesting structure given by restrictions. In order to show that the relation $\preceq$ is indeed a quasi order we need some more notation and a preliminary lemma.

First we define the maximum number $d_\nu(P)$ of nested restrictions in a process P.

Definition 15. Let $P \in \mathrm{CCS}_!$. We define $d_\nu(P)$ inductively as follows:

$$
\begin{aligned}
&d_\nu(\alpha.P) = d_\nu(P) \\
&d_\nu(P + Q) = max(\{d_\nu(P), d_\nu(Q)\}) \\
&d_\nu(P|Q) = max(\{d_\nu(P), d_\nu(Q)\}) \\
&d_\nu((\nu x)P) = 1 + d_\nu(P) \\
&d_\nu(!P) = d_\nu(P)
\end{aligned}
$$

Then we need also a notation for indicating all the sequential and bang subprocesses of P.

Definition 16. Let $P \in \mathrm{CCS}_!$. The set $Sub(P)$ containing all the sequential and bang subprocesses of P is defined inductively as follows:

$$
\begin{aligned}
&Sub(\alpha.P) = \{\alpha.P\} \cup Sub(P) \\
&Sub(P + Q) = \{P + Q\} \cup Sub(P) \cup Sub(Q) \\
&Sub(P|Q) = Sub(P) \cup Sub(Q) \\
&Sub((\nu x)P) = Sub(P) \\
&Sub(!P) = \{!P\} \cup Sub(P)
\end{aligned}
$$

Finally with $\mathcal{P}_{P,n}$ we denote the set of all those $\mathrm{CCS}_!$ processes whose nesting level of restrictions is not greater than n and such that their sequential subprocesses, bang subprocesses and and bound names are contained in the corresponding elements of P. More precisely we have the following definition.

Definition 17 ($\mathcal{P}_{P,n}$). Let n be a natural number and P a process. We define $\mathcal{P}_{P,n}$ as follows:

$$\mathcal{P}_{P,n} = \{Q \in \mathrm{CCS}_! \mid Sub(Q) \subseteq Sub(P) \wedge bn(Q) \subseteq bn(P) \wedge d_\nu(Q) \leq n\}$$

The notion of $\mathcal{P}_{P,n}$ is important because processes contained in $\mathcal{P}_{P,n}$ can be written in a sort of normal form (up to $\equiv_w$) which allows us to simplify the proofs. This is the content of the following lemma.

Lemma 2. Let $P \in \mathrm{CCS}_!$, $n \leq d_\nu(P)$ and $Q \in \mathcal{P}_{P,n}$. Suppose that $|bn(P)| = m$ and

$bn(P) = \{x_1, \ldots, x_m\}$. Then there exist l, $k_1, \ldots, k_m$ such that

$$Q \equiv_w \prod_{i=1}^{l} Q_i | \prod_{j=1}^{m} (\prod_{h=1}^{k_j} (\nu x_j) R_{j,h})$$

for some
$Q_i \in Sub(P)$ for $i = 1, \ldots, l$
$R_{j,h} \in \mathcal{P}_{P,n-1}$ for $j = 1, \ldots, m$ (and corresponding $h = 1, \ldots k_j$)

Proof. By induction on the structure of Q. □

Using this normal form we can prove that $\preceq$ is a quasi order.

Proposition 10. The relation $\preceq$ is a quasi order over CCS$_!$ processes.

Proof. Transitivity of $\preceq$ is a consequence of the following fact.
If $P \preceq Q$, by definition of $\preceq$ and by Lemma 2 it follows that that

$$P \equiv_w \prod_{i=1}^{l} P_i | \prod_{j=1}^{m} \prod_{h=1}^{k_j} (\nu x_j) R_{j,h}$$

and

$$Q \equiv_w \prod_{i=1}^{l+l'} P_i | \prod_{j=1}^{m} \prod_{h=1}^{k_j+k'_j} (\nu x_j) R'_{j,h}$$

with $P_i \in Sub(P|Q)$ for $i = 1, \ldots, l + l'$ and $R_{j,h} \preceq R'_{j,h}$ for $j = 1, \ldots, m$ (and corresponding $h = 1, \ldots, k_j$). □

To prove that $\preceq$ is a well-quasi-order we need two more preliminary results. The first, rather obvious, states that the set of sequential and bang subprocesses of a process is finite.

Proposition 11. Given a process $P \in$ CCS$_!$ the set $Sub(P)$ is finite.

Proof. By induction on the structure of P. □

The second one shows that when performing a derivation step we obtain a process which is not more "complicated" than the original one. This is an important feature of CCS$_!$ which is due to the absence of recursive definitions. Essentially this is the key property that allows us to obtain the decidability of termination.

Proposition 12. Let $P \in$ CCS$_!$ and $Q \in \mathcal{P}_{P,n}$. If $Q \stackrel{\alpha}{\mapsto} Q'$ then $Q' \in \mathcal{P}_{P,n}$.

Proof. By induction on the proof of transition $Q \stackrel{\alpha}{\mapsto} Q'$. □

An immediate consequence of the above proposition is that all the processes reachable from P with a sequence of reduction steps belong to $\mathcal{P}_{P,n}$, where n is the maximum level of nesting of P. We state this fact as an explicit corollary since we will need it later. So, we denote by $Deriv(P)$ the processes reachable from P with a sequence of reduction steps.

Definition 18. Let $P \in \mathrm{CCS}_!$. Then we define

$$Deriv(P) = \{Q \mid P \mapsto^* Q\}$$

Then we have the following.

Corollary 2. Let $P \in \mathrm{CCS}_!$. Then $Deriv(P) \subseteq \mathcal{P}_{P,d_\nu(P)}$ holds.

Proof. Immediate from Proposition 12. □

Now we are ready to prove that $\preceq$ is a well-quasi-ordering. The key idea is the following: we use Lemma 2 to transform each derivative of P in $1 + m$ (finite) sequences, where m is the cardinality of $bn(P)$. The first sequence is over $Sub(P)$ which is a finite set, whereas the other sequences are over processes that are "simpler" than P, in the sense that the nesting level of restrictions in those processes is strictly smaller than $d_\nu(P)$. The result is proved proceeding by induction on the nesting level of restrictions and using the Higman's lemma.

Theorem 6. Let $P \in \mathrm{CCS}_!$ and $n \geq 0$. The relation $\preceq$ is a wqo over $\mathcal{P}_{P,n}$.

Proof. The proof is by induction on n.

Let $n = 0$.

Take an infinite sequence $P_1, P_2, \ldots, P_i, \ldots$, with $P_i \in \mathcal{P}_{P,0}$ for $i > 0$.

By Lemma 2, for any i we have that $P_i \equiv_w \prod_{j=1}^{n_i} P_{i,j}$, with $P_{i,j} \in Sub(P)$.

Hence, we have an infinite sequence of elements of $Sub(P)^*$; as $Sub(P)$ is finite (by Proposition 11), by Proposition 5 and Higman's Lemma (Lemma 1) we have that $=_*$ is a wqo over $Sub(P)^*$ (the relation $=_*$ on sequences is defined according to Definition 11, where we consider equality as the quasi order on the starting set).

It's easy to see that if $P_{i,1}P_{i,2}\ldots P_{i,n_i} =_* P_{k,1}P_{k,2}\ldots P_{k,n_k}$ then $P_i \preceq P_k$.

For the inductive step, let $n > 0$ and take an infinite sequence $P_1, P_2, \ldots, P_i, \ldots$, with $P_i \in \mathcal{P}_{P,n}$ for any $i > 0$.

By Lemma 2, there exists m such that, for any i we have that

$$P_i \equiv_w \prod_{j=1}^{n_i} P_{i,j} \mid \prod_{j=1}^{m} \prod_{h=1}^{k_{i,j}} (\nu x_j) R_{i,j,h}$$

with $P_{i,j} \in Sub(P)$ and $R_{i,j,h} \in \mathcal{P}_{P,n-1}$.

Hence, each P_i can be seen as composed of $m + 1$ finite sequences:

$$\begin{array}{l} P_{i,1} \ldots P_{i,n_i} \\ R_{i,1,1} \ldots R_{i,1,k_{i,1}} \\ \vdots \\ R_{i,m,1} \ldots R_{i,m,k_{i,m}} \end{array}$$

We note that the first sequence is composed of elements from the finite set $Sub(P)$, whereas the other m sequences are composed of elements in $\mathcal{P}_{P,n-1}$. We know from the base case that $=_*$ is a wqo over $Sub(P)^*$.

By inductive hypothesis, we have that $\preceq$ is a wqo on $\mathcal{P}_{P,n-1}$; hence, by Higman's Lemma

we have that $\preceq_*$ is a wqo on $\mathcal{P}^*_{P,n-1}$.
We start extracting an infinite subsequence from $P_1 \dots P_i \dots$ making the finite sequences $P_{i,1} \dots P_{i,n_i}$ increasing w.r.t. $=_*$; then, we extract an infinite subsequence from the subsequence obtained in the previous step, that makes the finite sequences $R_{i,1,1} \dots R_{i,1,k_{i,1}}$ increasing w.r.t. $\preceq_*$, and so on.
At the end of the process we obtain an infinite subsequence of $P_1 \dots P_i \dots$ that is ordered w.r.t. $\preceq$. □

As the last step in order to obtain our decidability result, we need to show that the relation $\preceq$ of Definition 14 is strongly compatible with $\stackrel{\alpha}{\mapsto}$. This is the content of theorem 7 below which uses the following proposition.

Proposition 13. Let $P, Q \in \mathrm{CCS}_!$. If $P \equiv_w Q$ and $Q \stackrel{\alpha}{\mapsto} Q'$ then there exists P' such that $P \stackrel{\alpha}{\mapsto} P'$ and $P' \equiv_w Q'$.

Proof. By induction on the proof of the relation $P \equiv_T Q$. □

Theorem 7. Let $P, Q, P' \in \mathrm{CCS}_!$. If $P \stackrel{\alpha}{\mapsto} P'$ and $P \preceq Q$ then there exists Q' such that $Q \stackrel{\alpha}{\mapsto} Q'$ and $P' \preceq Q'$.

Proof. The proof is by induction on $d_\nu(P)$.
By definition of $\preceq$ we have that $P \equiv_w \overline{P} | \prod_{i=1}^n (\nu x_i) P_i$ and $Q \equiv_w \overline{P} | R | \prod_{i=1}^n (\nu x_i) Q_i$, with $P_i \preceq Q_i$ for $i = 1, \dots, n$.
First of all, note that $d_\nu(P_i) < d_\nu(P)$ for $i = 1, \dots, n$.
As $P \stackrel{\alpha}{\mapsto} P'$, by Proposition 13 also $\overline{P} | \prod_{i=1}^n (\nu x_i) P_i \stackrel{\alpha}{\mapsto} P''$ wih $P'' \equiv_w P'$. The proof proceeds by case analysis on the last rule applied in the proof of transition $\overline{P} | \prod_{i=1}^n (\nu x_i) P_i \stackrel{\alpha}{\mapsto} P''$. □

We can then state the main result of this section.

Theorem 8. Let $P \in \mathrm{CCS}_!$. Then the transition system $(Deriv(P), \mapsto, \preceq)$ is a finitely branching well-structured transition system with strong compatibility, decidable $\preceq$ and computable *Succ*.

Proof. The fact that $(Deriv(P), \mapsto)$ is finitely branching derives from an inspection of the transition rules (in particular `REPL1` and `REPL2`). The fact that $\preceq$ is a well-quasi-order on $Deriv(P)$ is a consequence of Corollary 2 and Theorem 6 (taking $n = d_\nu(P)$). Strong compatibility has been proven in Theorem 7. □

Corollary 3. Let $P \in \mathrm{CCS}_!$. The termination of process P is decidable.

Proof. Immediate from Theorem 5, Corollary 1 and Theorem 8. □

4.4. *Decidability of barb*

In this section we show that the ability of a process to perform, possibly after some internal moves, an observable action on a given channel is a decidable property in the calculus with replication. As previously mentioned such a property will be called barb, for short.

Also this result is based on the theory of well-structured transition systems and we need here some additional definitions and results from (Finkel and Schnoebelen 2001).

Recall that given a quasi-order $\leq$ over X, an *upward-closed set* is a subset $I \subseteq X$ such that the following holds: $\forall x, y \in X : (x \in I \wedge x \leq y) \Rightarrow y \in I$. Given $x \in X$, we define its upward closure as $\uparrow x = \{y \in X \mid x \leq y\}$. This notion can be extended to sets in the obvious way: given a set $Y \subseteq X$ we define its upward closure as $\uparrow Y = \bigcup_{y \in Y} \uparrow y$.

Definition 19 (Finite basis). A *finite basis* of an upward-closed set I is a finite set B such that $I = \bigcup_{x \in B} \uparrow x$.

In our case the notion of basis is particularly important when considering the basis of the predecessor of a state in a transition system. More precisely, we are interested in effective pred-basis as defined below. Recall from Definition 9 that $Pred(S)$ denote the immediate predecessors of a set of states S while $Pred^*(S)$ are the predecessors.

Definition 20 (Effective pred-basis). A well-structured transition system has *effective pred-basis* if there exists an algorithm such that, for any any state $s \in S$, it returns the set $pb(s)$ which is a finite basis of $\uparrow Pred(\uparrow s)$.

The following proposition is a special case of Proposition 3.5 in (Finkel and Schnoebelen 2001).

Proposition 14. Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with strong compatibility, decidable $\leq$ and effective pred-basis. It is possible to compute a finite basis of $Pred^*(I)$ for any upward-closed set I given via a finite basis.

This proposition can be used to prove that $P \Downarrow x$ is decidable by exploiting the following idea: Clearly a process can perform an action α (in any number of steps) iff such a process is a predecessor of a process which can perform α immediately (i.e. in one step). If one can show that the set S consisting of those processes which can immediately perform α is upward closed, previous result allows us to compute effectively a finite pred-basis of $Pred^*(S)$ and therefore to decide whether a process Q belongs to $Pred^*(S)$: To this aim, in fact, it is sufficient to decide whether in the (finite) basis there exist a process which is smaller than Q (this is possible because the quasi-order $\leq$ is decidable).

In order to develop this idea we first need to perform three steps.

First we have to define a finitely branching, well-structured transition system with strong compatibility, decidable $\preceq$ and computable $Succ$. Such a system has already been obtained in the previous subsection, so we simply use here that system. †.

Next we have to show that when considering the new relation $\mapsto$ (used in the finitely branching semantics) rather than the old one $\longrightarrow$ the notion of barb does not change. This is the content of the following proposition, where we assume that barbs in the new semantics are defined in the obvious way: We say that $P \Downarrow x$ iff there exists P', P'', α such that $P \mapsto^* P' \stackrel{\alpha}{\mapsto} P''$ and $n(\alpha) = \{x\}$.

† To be more precise, in the transition system here we have to consider $\mathcal{P}_{P, d_\nu(P)}$ rather than $Deriv(P)$, because $Deriv(P)$ does not include all the possible states that we are dealing with (e.g. in the basis). However the proofs are the same.

Proposition 15. Let $P \in \mathrm{CCS}_!$. Then $P \Downarrow\!\!\Downarrow x$ iff $P \Downarrow x$.

Proof. Analogous to that one of Corollary 1. □

The third, and more substantial, step needed to prove that $P \Downarrow\!\!\Downarrow x$ (and therefore $P \Downarrow x$) is decidable consist in the definition of an effective pred-basis for our finitely branching, well-structured transition system. This will be done in the next subsection.

4.4.1. *An effective pred-basis for* $(\mathcal{P}_{P,d_\nu(P)}, \mapsto, \preceq)$

We start by defining the set of processes $Now_\alpha(P)$ that can immediately perform the labeled action α and which are constructed by using sequential and bang subprocesses of P (in the sense made precise by Definition 17).

Definition 21. Let $P \in \mathrm{CCS}_!$. The set of processes $Now_\alpha(P)$ is defined as $\{Q \in \mathcal{P}_{P,d_\nu(P)} \mid Q \stackrel{\alpha}{\mapsto}\}$.

Next we show that this set is upward-closed.

Proposition 16. Let $P \in \mathrm{CCS}_!$. Then $Now_\alpha(P) =\uparrow Now_\alpha(P)$ holds.

Proof. Immediate from the definition of $\preceq$. □

We can now provide a finite basis for the previously defined set. This is possible because the set of sequential and bang subprocesses of a process is finite

Definition 22. Let $P \in \mathrm{CCS}_!$. The set $fbNow_\alpha(P)$ is defined as follows:

$$fbNow_\alpha(P) = \{(\nu x_1 \ldots x_m)Q \mid Q \in Sub(P), m \leq d_\nu(P), x_1 \ldots x_m \subseteq bn(P), Q \stackrel{\alpha}{\mapsto}, n(\alpha) \notin \{x_1, \ldots, x_m\}\}$$

Proposition 17. Let $P \in \mathrm{CCS}_!$ and $\alpha \neq \tau$. Then the set $fbNow_\alpha(P)$ is a finite basis of $Now_\alpha(P)$.

Proof. The set $fbNow_\alpha(P)$ is finite by construction and by proposition 11. Moreover it is also a basis, since if $R \in Now_\alpha(P)$ then we can show by induction on the structure of R that there exists $Q \in fbNow_\alpha(P)$ such that $Q \preceq R$. □

Using the above definitions and results we can provide a method to construct a finite basis for the set of predecessors of a given process w.r.t. a transition $\stackrel{\alpha}{\mapsto}$. The idea is to first consider sequential and bang processes (for which, as we will show, it is possible to compute directly a pred-basis) and then consider restricted and parallel processes (for which we proceed by induction on their syntactical structure). In order to compute directly the pred-basis for a sequential process Q, we simply observe that given any transition $S \stackrel{\alpha}{\mapsto} S'$ such that $Q \preceq S'$, we have that either Q was already present (at top level) in S or not. In the first case a pred-basis can be obtained by application of proposition 17 by simply considering the processes $Q|R$ with $R \in fbNow_\alpha(P)$. In the second case, the process Q appears at top level only in S': this means that it is "activated" by one sequential or bang process, let us call it R. The activation occurs either because R performs directly a transition labeled with α or because it synchronizes with a parallel

process (in the case $\alpha = \tau$). In the first case, it is enough to consider the sequential or bang subprocesses of P able to activate Q by performing a transition labeled with α, in the second case it is necessary to consider sequential of bang subprocesses of P able to activate Q after performing a synchronization with another process in parallel.

Definition 23. Let $P \in \mathrm{CCS}_!$. Given a process $Q \in \mathcal{P}_{P,d_\nu(P)}$, we define

$$\begin{aligned} basic_\alpha(Q) = \quad & \{Q|R \mid R \in fbNow_\alpha(P)\} \cup \\ & \{R \in Sub(P) \mid \exists R' : R \overset{\alpha}{\mapsto} R' \wedge Q \preceq R'\} \cup \\ & synchbasic_\alpha(Q) \end{aligned}$$

where:

$$\begin{aligned} synchbasic_\tau(Q) = \{Q_1|Q_2 \mid \quad & Q_1 \in Sub(P) \wedge Q_2 \in fbNow_\alpha(P) \wedge \\ & \exists Q_1' : Q_1 \overset{\overline{\alpha}}{\mapsto} Q_1' \wedge Q \preceq Q_1' \ \} \\ synchbasic_\alpha(Q) = \emptyset \quad & \text{if } \alpha \neq \tau \end{aligned}$$

The pred-basis $pb_\alpha(Q)$ of a process Q w.r.t. α is defined by induction on the structure of the process as follows:

$$\begin{aligned} & pb_\alpha(\mathbf{0}) = basic_\alpha(\mathbf{0}) \\ & pb_\alpha(\alpha'.Q) = basic_\alpha(\alpha'.Q) \\ & pb_\alpha(Q_1 + Q_2) = basic_\alpha(Q_1 + Q_2) \\ & pb_\alpha((\nu x)Q) = \quad basic_\alpha((\nu x)Q') \\ & \qquad\qquad\quad \cup \\ & \qquad\qquad\quad \{(\nu x)Q' \mid Q' \in pb_\alpha(Q) \wedge x \neq n(\alpha)\} \\ & pb_\alpha(Q_1|Q_2) = \quad basic_\alpha(Q_1|Q_2) \\ & \qquad\qquad\quad \cup \\ & \qquad\qquad\quad \{Q_1'|Q_2 \mid Q_1' \in pb_\alpha(Q_1)\} \\ & \qquad\qquad\quad \cup \\ & \qquad\qquad\quad \{Q_1|Q_2' \mid Q_2' \in pb_\alpha(Q_2)\} \\ & \qquad\qquad\quad \cup \\ & \qquad\qquad\quad sync_\alpha(Q_1, Q_2) \\ & pb_\alpha(!Q) = basic_\alpha(!Q) \\ & sync_\tau(Q_1, Q_2) = \{Q_1'|Q_2' \mid \exists \alpha \in n(P) : Q_1' \in pb_\alpha(Q_1) \wedge Q_2' \in pb_{\overline{\alpha}}(Q_2')\} \\ & sync_\alpha(Q_1, Q_2) = \emptyset \qquad \text{if } \alpha \neq \tau \end{aligned}$$

The above definition provides a procedure to compute a pred-basis of a process as shown by the following Lemma.

Lemma 3. Let $P \in \mathrm{CCS}_!$ and $Q \in \mathcal{P}_{P,d_\nu(P)}$. Then $pb_\tau(Q)$ is a computable finite basis of $\uparrow Pred(\uparrow Q)$.

Proof. By induction on the structure of Q. □

Using the previous Lemma we can obtain the following Theorem.

Theorem 9. Let $P \in \mathrm{CCS}_!$. Then the transition system $(\mathcal{P}_{P,d_\nu(P)}, \mapsto, \preceq)$ is a well-structured transition system with strong compatibility, decidable $\preceq$ and effective pred-basis.

Proof. The first part is the content of Theorem 8 (note that in Theorem 8 $Deriv(P)$ rather than $\mathcal{P}_{P,d_\nu(P)}$ was used. However the proof of such a theorem works also when considering $\mathcal{P}_{P,d_\nu(P)}$). The second part follows from Lemma 3. □

We can now prove the main result of this section following the argument previously illustrated. First we characterize barbs in terms of predecessors.

Proposition 18. Let $P \in \text{CCS}_!$. $P \Downarrow x$ iff $P \in Pred^*(Now_x(P))$ or $P \in Pred^*(Now_{\overline{x}}(P))$.

Proof. Immediate by definition of $P \Downarrow x$, of $Pred^*$ and by definition 21. □

Corollary 4. Let $P \in \text{CCS}_!$. Then $P \downarrow x$ is decidable.

Proof. First observe that from Theorem 9 and Proposition 14 it follows that it is possible to compute a finite basis of $Pred^*(I)$ for any upward-closed set I specified by means of a finite basis. Next note that the set $Now_\alpha(P)$ of processe that can immediately perform a (not silent) move α is upward-closed (Proposition 16) and we provided a finite basis for it (Proposition 17). Hence, since $\preceq$ is decidable, we can decide whether a given process Q belongs to $Pred^*(Now_\alpha(P))$ by verifying if there exists a process in the finite basis (of $Pred^*(Now_\alpha(P))$) that is smaller than Q. This and Proposition 18 imply that $P \Downarrow x$ is decidable and therefore the thesis follows from Proposition 15. □

5. Decidability results for CCS_*

We show that the set of processes reachable from a given process P is finite. Hence, all the properties considered in this paper are decidable in CCS_*.

Definition 24. $Reach(P)$ is the set of terms reachable from P with a sequence of transitions: $Reach(P) = \{Q \mid \exists n \geq 0, \alpha_1, \ldots, \alpha_n \ s.t. \ P \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q\}$.

We provide an upper bound to the number of reachable processes:

Definition 25. The function $size$ on CCS_* processes is defined as follows:

$size(\mathbf{0}) = 1$ $\quad$ $size(\alpha.P) = 1 + size(P)$

$size(P + Q) = 1 + size(P) + size(Q)$ $\quad$ $size(P|Q) = size(P) \times size(Q)$

$size((\nu x)P) = size(P) + 1$ $\quad$ $size(P^*) = size(P) + 1$

In order to prove that $size$ is actually an upper bound to the number of processes reachable from a given process, we need the following Lemma.

Lemma 4. Let $P \in \text{CCS}_*$. For any $n > 0$, we have that if $P^* \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q$ or $R; P^* \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q$ with $R \in Reach(P)$ then $Q = Q'; P^*$ with $Q' \in Reach(P)$.

Proof. We proceed by induction on n.

We divide the base case, i.e. $n = 1$, in two parts. In the first part we consider $P^* \xrightarrow{\alpha_1} Q$. In this case the transition is inferred by the rule `ITER`, thus $P \xrightarrow{\alpha_1} Q'$ (hence $Q' \in Reach(P)$) and $Q = Q'; P^*$. In the second part we consider $R; P^* \xrightarrow{\alpha_1} Q$ (with $R \in Reach(P)$). In this case the transition is inferred by either the rule `SEQ1` or `SEQ2`. If it is

`SEQ1` it is easy to see that the lemma holds. If it is `SEQ2`, we have that $P^* \xrightarrow{\alpha_1} Q$, hence we can reason as in the first part.

If $n > 1$, by inductive hypothesis we have that $P^* \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} T \xrightarrow{\alpha_n} Q$ where $T = Q'; P^*$ with $Q' \in \mathit{Reach}(P)$. If we consider the last transition $Q'; P^* \xrightarrow{\alpha_n} Q$ we can prove that the lemma holds simply resorting to the second part of the base case. □

Proposition 19. Let $P \in \mathrm{CCS}_*$. Then $|\mathit{Reach}(P)| \leq \mathit{size}(P)$.

Proof. By induction on the structure of P. The unique nontrivial case is $P = Q^*$ in the inductive step. In this case we consider the above Lemma that allows us to conclude that $\mathit{Reach}(Q^*) = \{Q^*\} \cup \{R; Q^* \mid R \in \mathit{Reach}(Q)\}$. □

As a trivial corollary we have that the set of processes reachable from any process in CCS_* is finite.

Corollary 5. Let $P \in \mathrm{CCS}_*$. The set $\mathit{Reach}(P)$ is finite.

As a consequence of the above corollary, we obtain that termination, convergence, and barb are decidable. In fact, in a finite labelled transition system, termination can be checked verifying the absence of sequences of τ labelled transitions starting from the initial state that includes a loop, convergence can be checked verifying the presence of a sequence of τ labelled transitions starting form the initial state and leading to a state without outgoing τ labelled transitions, and a barb on w can be checked verifying the presence of a sequence of τ labelled transitions starting from the initial state and leading to a state with an outgoing transition labelled with w or $\overline{w}$. Moreover, we have also that weak bisimulation is decidable as we can test weak bisimilarity of two finite labelled transition systems using, e.g., the algorithm proposed by Kanellakis and Smolka (Kanellakis and Smolka 1990).

6. Conclusion and Related Work

In this paper we have investigated the expressive power of three different constructs for the modeling of infinite behaviors in process calculi. More precisely, we have considered a finite fragment of CCS that we have extended with recursive definitions, replication, and iteration, respectively. We have considered four different properties for processes; process termination (i.e. all runs are finite), process convergence (i.e. there exists a finite completed run), barb (a process has the ability to perform visible actions on a specific channel), and weak bisimulation between processes. For each of these properties we have proved whether they are decidable or not in the three considered calculi; the results are reported in the Table presented in the Introduction.

As a consequence of the results we have proved in the paper there exists a strict hierarchy of expressiveness w.r.t. weak bisimulation among the three considered infinite operators. In fact, there exist encodings of replication in recursion, and of iteration in replication that preserve weak bisimulation, while the vice versa does not hold.

To encode replication using recursive definitions, we consider an encoding $[\![\ ]\!]$ which is

homomorphic except for replication:

$$[\![!P]\!] \;=\; D \qquad \text{with } D \stackrel{def}{=} [\![P]\!]|D$$

In order to model iteration using replication it is simply necessary to spawn replicas only on termination of the previous one. This can be done detecting the termination of the execution of one iteration before activating the subsequent one. The idea is to use new fresh channel names in such a way that when a process terminates, it communicates termination on the corresponding channel; this triggers the subsequent iteration. Formally, we consider the encoding $[\![P]\!] = (\nu x)([\![P]\!]_x)$ where $[\![\ ]\!]_x$ is an encoding function indexed on a fresh name x (i.e. $x \notin fn(P)$). The name x indicates the channel on which termination should be communicated. The indexed encoding is defined as follows:

$$\begin{array}{l}
[\![\mathbf{0}]\!]_x = \overline{x} \\
[\![\alpha.P]\!]_x = \alpha.[\![P]\!]_x \\
[\![P+Q]\!]_x = [\![P]\!]_x + [\![Q]\!]_x \\
[\![P|Q]\!]_x = (\nu\, y, z)(y.z.\overline{x} \mid [\![P]\!]_y \mid [\![Q]\!]_z) \qquad y, z \notin fn(P) \cup fn(Q) \\
[\![(\nu y)P]\!]_x = (\nu y)[\![P]\!]_x \\
[\![P^*]\!]_x = (\nu y)(\overline{y} \mid !y.([\![P]\!]_y + \overline{x})) \qquad y \notin fn(P)
\end{array}$$

The encodings in the opposite direction do not exist. Replication cannot be encoded in terms of iteration because weak bisimulation is decidable only under iteration; recursion cannot be encoded into replication because barb is decidable only under replication and weak bisimulation preserves barbs.

In a related paper (Giambiagi *et al.* 2004) Giambagi, Schneider and Valencia consider other infinite operators in the setting of CCS, namely recursive expressions with static binding and parameterless constants with dynamic binding. The former is proved to be as expressive as replication, while the latter is proved to be as expressive as constants with parameters. In their approach, two calculi are equally expressive if there exists a weak bisimulation preserving encoding of one calculus in the other, and vice versa. In their paper they leave as an open problem the existence of a weak bisimulation preserving encoding from recursion to replication. In (Busi *et al.* 2004), one of the two conference papers which this paper is based on, we closed this open problem proving that such an encoding does not exist.

A comparison of different mechanisms for describing infinite behaviors is reported in (Nielsen *et al.* 2002), where the expressive power of several timed concurrent constraint languages is investigated. In particular, one of the results in that paper shows that the language with replication is strictly less expressive than the language with recursive definitions of processes (in case process constants have parameters). Because of the very different underlying computational model, the proof techniques exploited in that paper cannot be applied directly in the context of CCS.

The undecidability of weak bisimulation has been proved by Srba (Srba 2003) also for PA processes. PA is a minimal process algebra comprising sequential and parallel composition, as well as recursion. Even if PA is minimal, the Srba's result cannot be applied in our setting because PA considers sequential composition, which is more general than our prefix operator. Also the vice versa does not hold (we cannot directly apply our

undecidability results to PA) because we consider the restriction operator, which is not part of PA.

Acknowledgments – We thank Lucia Acciai for pointing out an error in a preliminary version of the paper, and the anonymous referees for their comments useful to improve the presentation.

References

N. Busi, M. Gabbrielli, and G. Zavattaro. (2003) Replication vs. Recursive Definitions in Channel Based Calculi. In *Proc. ICALP'03*, LNCS 2719, pages 133–144, Springer-Verlag.

N. Busi, M. Gabbrielli, and G. Zavattaro. (2004) Comparing Recursion, Replication, and Iteration in Process Calculi. In *Proc. ICALP'04*, LNCS 3142, pages 307–319, Springer-Verlag.

A. Finkel and Ph. Schnoebelen. (2001) Well-Structured Transition Systems Everywhere! *Theoretical Computer Science*, 256:63–92.

P. Giambiagi, G. Schneider and F.D. Valencia. (2004) On the Expressiveness of CCS-like Calculi In Proceedings of FOSSACS 04. LNCS 2987, pages 226–240, Springer-Verlag.

G. Higman. (1952) Ordering by divisibility in abstract algebras. In *Proc. London Math. Soc.*, vol. 2, pages 236–366.

P.C. Kanellakis and S.A. Smolka. (1990) CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68.

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

R. Milner, J. Parrow, D. Walker. A calculus of mobile processes. *Journal of Information and Computation*, 100:1–77. Academic Press, 1992.

M. L. Minsky. (1967) *Computation: finite and infinite machines*. Prentice-Hall, Englewood Cliffs.

M. Nielsen, C. Palamidessi, and F. D. Valencia. (2002) On the Expressive Power of Temporal Concurrent Constraint Programming Languages. In *Proc. of PPDP'02*. ACM Press.

J. C. Shepherdson and J. E. Sturgis. (1963) Computability of recursive functions. *Journal of the ACM*, 10:217–255.

J. Srba. (2003) Undecidability of Weak Bisimilarity for PA-Processes. In Proc. of DLT'02, LNCS 2450, pages 197–208, Springer-Verlag.