Comparing Recursion, Replication, and Iteration in Process Calculi

Nadia Busi, Maurizio Gabbrielli, and Gianluigi Zavattaro

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura A.Zamboni 7, I-40127 Bologna, Italy.
busi,gabbri,zavattar@cs.unibo.it

Abstract. In [BGZ03] we provided a discrimination result between recursive definitions and replication in a fragment of CCS by showing that termination (i.e., all computations terminate) is undecidable in the calculus with recursion, whereas it turns out to be decidable in the calculus with replication. Here we extend the results in [BGZ03] by considering iteration, a third mechanism for expressing infinite behaviours. We show that convergence (i.e., the existence of a terminating computation) is undecidable in the calculus with replication, whereas it is decidable in the calculus with iteration. We also show that recursion, replication and iteration constitute a strict expressiveness hierarchy w.r.t. weak bisimulation: namely, 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 direction.

1 Introduction

In this paper we continue the investigation we have started in [BGZ03], devoted to the comparison of different mechanisms used in the context of channel-based process calculi for extending finite processes with infinite behaviours. More precisely, we focus on three classical mechanisms, namely, *recursion*, *replication*, and *iteration*.

We adopt process constants to express recursion: we assume that 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 behaviour, since process copies can be nested at an arbitrary depth by using constant application. On the other hand, the replication operator $!P$ allows to create an unbounded number of parallel copies of a process P, thus providing an "in width" infinite behaviour, 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 behaviour is supported, since the copies are executed one after the other.

In [BGZ03] we proved a discrimination result between recursion and replication in the context of a fragment of CCS [Mil89] with guarded choice and without relabelling. We showed that *termination*, i.e. all computations terminate, is undecidable in the calculus with recursion, whereas it turns out to be decidable in the calculus with replication.

In this paper we extend our previous work by taking into account other interesting properties of processes, and by investigating the decidability of these properties. More precisely, we consider process *convergence*, i.e. there exists a computation that terminates, *barb*, i.e. a process has the ability to perform a synchronization on a certain channel after a (possibly empty) internal computation, and *weak bisimulation*. We say that weak bisimulation is decidable if given any pair of processes, it is decidable whether those two processes are weakly bisimilar. The results of our investigation are reported in the following Table:[1]

	Recursion	Replication	Iteration
Termination	undecidable[BGZ03]	decidable[BGZ03]	decidable
Convergence	undecidable	undecidable	decidable
Barb	undecidable	decidable	decidable
Weak bisimulation	undecidable	undecidable	decidable

The undecidability results are proved by presenting an encoding of Random Access Machines [SS63] (RAMs), a well known deterministic Turing powerful formalism. In [BGZ03] we showed an encoding of RAMs in the calculus with recursion. The encoding is deterministic, i.e. it presents a unique possible computation that reflects the computation of the corresponding RAM. This proves that termination and convergence are undecidable. By exploiting a slightly different encoding, it is possible to 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 behaviour.

The decidability of process termination for the calculus with replication implies the impossibility to provide a termination preserving encoding of RAMs. The existence of encodings that preserve "weaker" properties was left as an open problem. In this paper, we answer positively to this question by showing how to model RAMs in a nondeterministic manner. The encoding is nondeterministic in the following sense: computations which do not follow the expected behaviour of the modeled RAM are introduced by the encoding, but 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, simply by following a technique

[1] In the present paper we consider the fragment of CCS with general choice, whereas in [BGZ03] we considered the calculus with guarded choice. Decidability results presented in [BGZ03] can be easily adapted to the calculus with general choice. Clearly, the undecidability results of [BGZ03] continue to hold also in this extended calculus.

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

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

Table 1. The transition system for finite core CCS (symmetric rules of omitted).

similar to the one described above for the calculus with recursion. Interestingly, we have that even if weak bisimulation is undecidable under replication, barb turns out to be decidable. This is proved by resorting to the theory of well structured transition systems [FS01].

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.

2 The Calculi

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

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:*

$$P \ ::= \ \mathbf{0} \ \mid \ \alpha.P \ \mid \ P+P \ \mid \ P|P \ \mid \ (\nu x)P \qquad \alpha \ ::= \ \tau \ \mid \ x \ \mid \ \overline{x}$$

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 synchronization are available, the output $\overline{x}$ or the input x. The sum construct $+$ is used to make choice among 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 1 contains the set of the transition rules for finite CCS.

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

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

Table 2. The rules for the $\surd$ transitions.

Definition 2. **($\mathbf{CCS}_D$)** *We assume a set of constants, ranged over by D. The class of CCS_D processes is defined by adding the production $P ::= D\langle\tilde{x}\rangle$ to the grammar of Definition 1. It is assumed that each constant D has a unique defining equation of the form $D \stackrel{def}{=} (\tilde{x})P$, where $(\tilde{x})$ is a binder for the names in the sequence of names in $\tilde{x}$. Both in a constant definition $D \stackrel{def}{=} (\tilde{x})P$ and in a constant application $D\langle\tilde{x}\rangle$, the parameter $\tilde{x}$ is a tuple of all distinct names. As usual, in case the sequence $\tilde{x}$ is empty, we omit the surrounding parentheses. Moreover, we assume that $fn(P) \subseteq n(\tilde{x})$ where $n(\tilde{x})$ denotes the set of names in the sequence $\tilde{x}$.*

The transition rules for constant is

$$\frac{P\{\tilde{x}/\tilde{y}\} \xrightarrow{\alpha} P'}{D\langle\tilde{x}\rangle \xrightarrow{\alpha} P'} \quad \text{if } D \stackrel{def}{=} (\tilde{y})Q$$

where $P\{\tilde{x}/\tilde{y}\}$ is the term obtained by replacing all the free occurrences of the names in $\tilde{y}$ with the corresponding names in $\tilde{x}$.

Definition 3. **($\mathbf{CCS}_!$)** *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

$$\frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

Definition 4. **($\mathbf{CCS}_*$)** *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 behaviour 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 transition $P \xrightarrow{\surd} \bullet$, where $\surd \notin$ *Name* is a new label and $\bullet$ is an auxiliary operator. 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 $P \xrightarrow{\surd} \bullet$. Formally, the axioms and rules for $\surd$ transitions are reported in Table 2. The transition rules for iteration are

$$\frac{P \xrightarrow{\alpha} P'}{P^* \xrightarrow{\alpha} P';P^*} \qquad \frac{P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\alpha} P';Q} \qquad \frac{P \xrightarrow{\surd} \bullet \qquad Q \xrightarrow{\alpha} Q'}{P;Q \xrightarrow{\alpha} Q'}$$

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 $\prod_n P$ denotes the empty process $\mathbf{0}$).

Given a process Q, its internal runs $Q \longrightarrow Q_1 \longrightarrow Q_2 \longrightarrow \ldots$ are given by its reduction steps, (denoted with $\longrightarrow$), i.e. by those transitions $\longrightarrow$ that the process can perform in isolation, independently of the context. The internal transitions $\longrightarrow$ correspond to the transitions labeled with τ plus the ending $\surd$ transitions, i.e. $P \longrightarrow P'$ iff $P \xrightarrow{\tau} P'$ or $P \xrightarrow{\surd} P'$. 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: $Deriv(P) = \{Q \mid P \longrightarrow^* Q\}$.

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 $\{P_i\}_{i \in \mathbb{N}}$, 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 Undecidability Results for $\mathrm{CCS}_!$

We prove that $\mathrm{CCS}_!$ is powerful enough to model, at least in a nondeterministic way, any Random Access Machine [SS63] (RAM), a well known register based Turing powerful formalism.

A RAM (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. Given a configuration $(i, c_1, \ldots, c_n)$, its computation proceeds by executing the 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; in this case we say that the configuration $(i, c_1, \ldots, c_n)$ terminates.

In [Min67] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : Succ(r_j))$: adds 1 to the contents 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.

Our encoding is nondeterministic because it introduces computations which do not follow the expected behaviour 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 has the following property; at its end it executes the action $\overline{w}$, after which it cannot produce any further observable action. Thus, if P is the encoding of a RAM R, then R terminates if and only if $P \approx \tau.P + \overline{w}$. This proves that *weak bisimulation* is undecidable.

In this section we reason up to a structural congruence $\equiv$ in order to rearrange the order of parallel composed processes and to abstract away from the terminated processes $\mathbf{0}$. We define $\equiv$ as the least congruence relation satisfying the usual axioms $P|Q \equiv Q|P$, $P|(Q|R) \equiv (P|Q)|R$, and $P|\mathbf{0} \equiv P$.

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 message $\overline{p}_i$ 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 process which is guarded by an input operation p_i. Once activated, the instruction performs its operation on the registers, then waits for an acknowledgement 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.

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

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

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. Let TI the set of the terminating indexes given by $m + 1$ plus all those indexes greater than m that are target of some jump instruction. For each index $i \in TI$ we will assume the presence of a process $p_i.\overline{w}$ able to consume the program counter message and communicate program termination 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:

$$
\begin{array}{ll}
[\![r_j = c_j]\!] \; : \; \overline{nr_j} \mid !nr_j.\,(\nu\; m\; i\; d\; u)(& \\
\qquad \overline{m} \mid !m.(inc_j.\overline{i} + dec_j.\overline{d}) \mid !i.(\overline{m} \mid \overline{inc} \mid \overline{u} \mid d.u.(\overline{m} | \overline{dec})) \mid & \\
\qquad d.(\overline{zero} \mid u.DIV \mid \overline{nr_j}) \mid \prod_{c_j} (\overline{u} \mid d.u.(\overline{m} \mid \overline{dec})) \quad) &
\end{array}
$$

where DIV is a process able to activate an infinite observable computation, for instance $\overline{w'} \mid !w'.\overline{w'}$.

Observe that the content c_j of the register is modeled by the parallel composition of a corresponding number of processes $(\overline{u} \mid d.u.(\overline{m} | \overline{dec}))$; the term $\overline{u}$ represents a unit inside the register, while $d.u.(\overline{m} | \overline{dec})$ is an auxiliary term that is responsible for removing the unit when the register is decremented.

The name nr_j is used to activate and restart the register. This is because the register is modeled as a replicated process, and the message $\overline{nr_j}$ is used to spawn a new replica. The name m is used to activate the so-called manager of the register: $(inc_j.\overline{i} + dec_j.\overline{d})$. The manager handles an increment or a decrement request and produces the local name $\overline{i}$ or $\overline{d}$, respectively. The modeling of register increment is easy, while register decrement introduces nondeterminism.

Two different processes may synchronize with $\overline{d}$: either $(d.u.(\overline{m} \mid \overline{dec}))$ or $(d.(\overline{zero} \mid u.DIV \mid \overline{nc}))$. In the first case the register is actually decremented while in the second case a jump is executed. Observe that the jump could occur even if the register is not empty, nevertheless, if the register is not empty this means that at least one instance of $\overline{u}$ is available. This ensures that the computation cannot terminate as the process $u.DIV$ is spawn. In case the the register is actually empty, the old instance of the register remains represented by the deadlocked "garbage" process

$$
G_j \; : \quad (\nu\; m\; i\; d\; u)(!m.(inc_j.\overline{i} + dec_j.\overline{d}) \mid !i.(\overline{m} \mid \overline{inc} \mid \overline{u} \mid d.u.\overline{dec}) \mid u.DIV)
$$

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*

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

where the modeling of program instructions $[\![(i : I_i)]\!]$, the modeling of registers $[\![r_j = c_j]\!]$, the set of terminating indexes TI, and the garbage $G_1, \ldots, G_n$ have been defined above, and $k_1 \ldots k_n$ are natural numbers. Observe that due to the *presence of $k_1 \dots k_n$ 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.*

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

This proves that *convergence* and *weak bisimulation* are undecidable in $\mathrm{CCS}_!$.

4 Decidability results for $\mathrm{CCS}_!$

We show that barb is a decidable property in the calculus with replication. This result is based on the theory of well-structured transition systems [FS01]; first of all, we recall the alternative semantics for $\mathrm{CCS}_!$ defined in [BGZ03], that is equivalent to the one presented in Section 2, but is based on a finitely branching transition system. Then, by exploiting the theory developed in [FS01], we show that barb is decidable for $\mathrm{CCS}_!$ processes.

We start recalling some basic definitions and results of [FS01], concerning well-structured transition systems, that will be used in the following. A *quasi-ordering* is a reflexive and transitive relation over a set X. Given a quasi-ordering $\leq$ 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 $\uparrow x = \{y \in X \mid x \leq y\}$. Given $Y \subseteq X$, we define $\uparrow Y = \bigcup_{y \in Y} \uparrow y$. A *finite basis* of an upward-closed set I is a finite set B such that $I = \bigcup_{x \in B} \uparrow x$.

Definition 7. *A* well-quasi-ordering *(wqo) is a quasi-ordering $\leq$ over a set X such that, for any infinite sequence $x_0, x_1, x_2, \dots$ 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 $x_0, x_1, x_2, \dots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \dots$ (with $i_0 < i_1 < i_2 < \dots$).

Transition systems can be formally defined as follows.

Definition 8. *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 write $Succ(s)$ (resp. $Pred(s)$) to denote the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors (resp. $\{s' \in S \mid s' \rightarrow s\}$ of immediate predecessors) of s. We write $\rightarrow^+$ (resp, $\rightarrow^*$) for the transitive (resp. the reflexive and transitive) closure of $\rightarrow$. We write $Pred^*(s)$ to denote the set $\{s' \in S \mid s' \rightarrow^* s\}$. TS is* finitely branching *if all $Succ(s)$ are finite.*

We restrict to finitely branching transition systems. Well-structured transition system, defined as follows, provide the key tool to decide properties of computations.

Definition 9. *A* well-structured transition system with strong compatibility *is a transition system $TS = (S, \rightarrow)$, equipped with a quasi-ordering $\leq$ on S, such that the two following conditions hold:*

1. **well-quasi-ordering**: *$\leq$ is a well-quasi-ordering, and*
2. **strong compatibility**: *$\leq$ is (upward) compatible with $\rightarrow$, i.e., 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 $s_2 \leq t_2$.*

Definition 10. *A well-structured transition system has* effective pred-basis *if there exists an algorithm accepting any state $s \in S$ and returning $pb(s)$, a finite basis of $\uparrow Pred(\uparrow s)$.*

The following proposition (a special case of a result in [FS01]) will be used to obtain our decidability result.

Proposition 1. *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.*

As the results on well-structured transition systems apply to finitely branching transition systems, we need to use the alternative semantics for $\mathrm{CCS}_!$ defined in [BGZ03], that is based on a finitely branching transition system and that is equivalent to the semantics presented in Section 2. The new semantics is obtained by reformulating the (non finitely branching) semantics of replication defined in Definition 3. The new transition relation $\stackrel{\alpha}{\mapsto}$ over $\mathrm{CCS}_!$ processes is the least relation satisfying all the axioms and rules of Table 1 (where $\stackrel{\alpha}{\mapsto}$ is substituted for $\stackrel{\alpha}{\longrightarrow}$), plus the following rules `REPL1` and `REPL2`.

$$\texttt{REPL1}: \frac{P \stackrel{\alpha}{\mapsto} P'}{!P \stackrel{\alpha}{\mapsto} P' |\, !P} \qquad\qquad \texttt{REPL2}: \frac{P \stackrel{\alpha}{\mapsto} P' \quad P \stackrel{\overline{\alpha}}{\mapsto} P''}{!P \stackrel{\tau}{\mapsto} P' |\, P'' |\, !P}$$

As done for the standard transition system, we assume that the reductions $\mapsto$ of the new semantics corresponds to the τ–labeled transitions $\stackrel{\tau}{\mapsto}$. Barbs in the new semantics are defined in the obvious way: $P \Downarrow x$ iff there exists P', P'', α s.t. $P \mapsto^* P' \stackrel{\alpha}{\mapsto} P''$ and $n(\alpha) = \{x\}$. We have the following result:

Proposition 2. *Let $P \in CCS_!$. Then $P \Downarrow x$ iff $P \Downarrow x$.*

In [BGZ03] we defined a preorder $\preceq$ on $\mathrm{CCS}_!$processes and, by exploiting Higman's lemma [Hig52], we proved that $\preceq$ is a well-quasi-ordering compatible with $\mapsto$, thus obtaining a well-structured transition system. In this section we show that the obtained well-structured transition system has an effective pred-basis. Thus, exploiting the Proposition 1 we show that $P \Downarrow x$ is decidable.

We start recalling the definition of $\preceq$ and the results of [BGZ03] that will be used to prove the decidability of barb. The definition of the wqo on processes needs the following structural congruence, that turns out to be compatible with $\mapsto$.

Definition 11. *We define $\equiv$ as the least congruence relation satisfying the following axioms:* $\quad P|Q \equiv Q|P \quad P|(Q|R) \equiv (P|Q)|R \quad P|\mathbf{0} \equiv P$

Proposition 3. *Let $P, Q \in CCS_!$. If $P \equiv Q$ and $Q \overset{\alpha}{\mapsto} Q'$ then there exists P' such that $P \overset{\alpha}{\mapsto} P'$ and $P' \equiv Q'$.*

Now we are ready to define the preorder on processes:

Definition 12. *Let $P, Q \in CCS_!$. 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 $P \equiv P' | \prod_{i=1}^{n} (\nu x_i) P_i$, $Q \equiv P' | R | \prod_{i=1}^{n} (\nu x_i) Q_i$, and $P_i \preceq Q_i$ for $i = 1, \ldots, n$.*

Definition 13. *Let $P \in CCS_!$. With $d_\nu(P)$ we denote the maximum number of nested restrictions in process P:*

$d_\nu(\mathbf{0}) = 0$	$d_\nu(\alpha.P) = d_\nu(P)$
$d_\nu(P + Q) = max(\{d_\nu(P), d_\nu(Q)\})$	$d_\nu(P\|Q) = max(\{d_\nu(P), d_\nu(Q)\})$
$d_\nu((\nu x)P) = 1 + d_\nu(P)$	$d_\nu(!P) = d_\nu(P)$

The set of sequential and bang subprocesses of P is defined as:

$Sub(\mathbf{0}) = \emptyset$	$Sub(\alpha.P) = \{\alpha.P\} \cup Sub(P)$
$Sub(P + Q) = \{P + Q\} \cup Sub(P) \cup Sub(Q)$	$Sub(P\|Q) = Sub(P) \cup Sub(Q)$
$Sub((\nu x)P) = Sub(P)$	$Sub(!P) = \{!P\} \cup Sub(P)$

Definition 14. *Let n be a natural number and P a process. With $\mathcal{P}_{P,n}$ we denote the set of $CCS_!$ processes whose sequential subprocesses, bang subprocesses and bound names are contained in the corresponding elements of P, and with a nesting level of restrictions not greater than n:*
$\mathcal{P}_{P,n} = \{Q \in CCS_! \mid Sub(Q) \subseteq Sub(P) \wedge bn(Q) \subseteq bn(P) \wedge d_\nu(Q) \leq n\}$.

We define the set of processes that can immediately perform a labelled move; we show that this set is upward-closed and we provide this set with a finite basis.

Definition 15. *Let $P \in CCS_!$. The set of processes $Now_\alpha(P)$ is defined as $\{Q \in \mathcal{P}_{P,d_\nu(P)} \mid Q \overset{\alpha}{\mapsto}\}$.*

Proposition 4. *Let $P \in CCS_!$. The set of processes $Now_\alpha(P)$ is upward-closed.*

Definition 16. *Let $P \in CCS_!$. The set $fbNow_\alpha(P)$ is defined as follows:*

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

Proposition 5. *Let $P \in CCS_!$ and $\alpha \neq \tau$. Then the set $fbNow_\alpha(P)$ is a finite basis of $Now_\alpha(P)$.*

It is possible to provide a method to construct a finite basis for the set of predecessors of a given process w.r.t. a transition $\overset{\alpha}{\mapsto}$; hence, the following holds:

Theorem 2. *Let $P \in CCS_!$. 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.*

As a consequence of this result, by Proposition 1 it is possible to compute a finite basis of $Pred^*(I)$ for any upward-closed set I specified through a finite basis. The possibility to compute a finite basis of $Pred^*(I)$, together with the decidability of $\preceq$, provides a method to decide if a given process Q belongs to $Pred^*(I)$, as it is sufficient to verify if there exists a process in the finite basis that is smaller than Q. Since the set of processes $Now_\alpha(P)$ – that can immediately perform a (not silent) move α – is upward-closed, and we provided a finite basis for it in Definition 16, we have that it is possible to decide if a process belongs to $Pred^*(Now_\alpha(P))$.

From the following Proposition, which provides a characterisation of $P\Downarrow x$ in terms of belonging to $Pred^*(Now_\alpha(P))$, we obtain the decidability of $P\Downarrow x$.

Proposition 6. *Let $P \in CCS_!$. $P\Downarrow x$ iff $P \in Pred^*(Now_x(P))$ or $P \in Pred^*(Now_{\overline{x}}(P))$.*

Corollary 1. *Let $P \in CCS_!$. Then $P\Downarrow x$ is decidable.*

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

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

Definition 18. *The function size on CCS_*processes is defined as follows:*

$$size(\mathbf{0}) = 1 \qquad size(\alpha.P) = 1 + size(P)$$
$$size(P + Q) = 1 + size(P) + size(Q) \qquad size(P|Q) = size(P) \times size(Q)$$
$$size((\nu x)P) = size(P) \qquad size(P*) = size(P) + 1$$

Proposition 7. *Let $P \in CCS_*$. Then $|Reach(P)| \leq size(P)$.*

Corollary 2. *Let $P \in CCS_*$. The set $Reach(P)$ is finite.*

As a consequence of the above corollary, we obtain that termination, convergence and barb are decidable in CCS$_*$, as well as weak bisimulation [KS90,PT87].

6 Conclusion and Related Work

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 $[\![!P]\!] = D\langle\tilde{x}\rangle$ with $D = (\tilde{x})([\![P]\!]|D\langle\tilde{x}\rangle)$

where $\tilde{x}$ is a list containing the free names of P. 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 following the typical encoding of the continuation passing style. 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 [GSV04] Giambagi, Schneider and Valencia consider other infinite operators in the setting of CCS. 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 this paper we close this open problem proving that such an encoding does not exist. In [NPV02] it is shown that replication is strictly less expressive than recursive definitions in the context of timed concurrent constraint languages. Because of the very different underlying computational model, the proof techniques exploited in that paper cannot be applied directly in the context of CCS. Recently, the undecidability of weak bisimulation has been proved by Srba [S03] also for PA processes. PA and our calculus are incomparable as PA considers sequential composition and does not contain restriction.

References

[BGZ03] N. Busi, M. Gabbrielli, and G. Zavattaro. Replication vs. Recursive Definitions in Channel Based Calculi. In *Proc. ICALP'03*, LNCS 2719, pages 133–144, Springer-Verlag, 2003.

[FS01] A. Finkel and Ph. Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science*, 256:63–92, 2001.

[GSV04] P. Giambiagi, G. Schneider and F.D. Valencia. On the Expressiveness of CCS-like Calculi In Proceedings of FOSSACS 04. LNCS 2987, pages 226–240, Springer-Verlag, 2004.

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. In *Proc. London Math. Soc.*, vol. 2, pages 236–366, 1952.

[KS90] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990

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

[Min67] M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Englewood Cliffs, 1967.

[NPV02] M. Nielsen, C. Palamidessi, and F. D. Valencia. On the Expressive Power of Temporal Concurrent Constraint Programming Languages. In *Proc. of PPDP'02*. ACM Press, 2002.

[PT87] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987

[SS63] J. C. Shepherdson and J. E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10:217–255, 1963.

[S03] J. Srba. Undecidability of Weak Bisimilarity for PA-Processes. In Proc. of DLT'02, LNCS 2450, pages 197–208, Springer-Verlag, 2003.