SYSTEMS ENGINEERING

On decidability properties of two fragments of the asynchronous π-calculus

INGENIERÍA DE SISTEMAS

Sobre la decidibilidad de dos fragmentos del π-cálculo asincrónico

§ Jesús A. Aranda B.

Escuela de Ingeniería de Sistemas y Computación, Universidad del Valle, Cali, Colombia
§ jesus.aranda@correounivalle.edu.co

(Recibido: 27 de Abril de 2013-Aceptado: 09 de Septiembre de 2013)

Abstract

In (Cacciagrano, et al., 2008) the authors studied the expressiveness of persistence in the asynchronous *π-calculus,* henceforth Aπ. They considered Aπ and three sub-languages of it, each capturing one source of persistence: the persistent-input calculus (PIAπ), the persistent-output calculus (POAπ), and the persistent calculus (PAπ). They prove that, under some general conditions, there cannot be an encoding from Aπ into a (semi)-persistent calculus preserving the must-testing semantics, a semantics sensitive to divergence.

In this paper we support and strengthen the separation results of (Cacciagrano, et al., 2008) by showing that convergence and divergence are two decidable properties in a fragment of POAπ and PAπ, in contrast to what happen in Aπ. Thus, it is shown that there cannot be a (computable) encoding from Aπ into PAπ and in such a fragment of POAπ, preserving divergence or convergence. These impossibility results don't presuppose any condition on the encodings and involve directly convergence for first time in the study of the expressiveness of persistence of Aπ.

Keywords: *Expressiveness, divergence, convergence, process calculi.*

Resumen

En (Cacciagrano, et al., 2008) se estudió la expresividad de la persistencia en el *π-cálculo* asincrónico, Aπ. En dicho artículo, los autores consideraron Aπ y tres de sus fragmentos, cada uno de ellos capturando una fuente de persistencia: el fragmento con entradas persistentes (PIAπ), el fragmento con salidas persistentes (POAπ), y el fragmento con tanto entradas como salidas persistentes (PAπ). Ellos demostraron que, bajo ciertas condiciones generales, no puede existir una codificación desde Aπ en alguno de sus fragmentos preservando la semántica must-testing, una semántica sensible a la divergencia.

En este artículo se ratifican y fortalecen los resultados de separación de (Cacciagrano, et al., 2008) mostrando que tanto convergencia como divergencia son propiedades decidibles en un fragmento significativo de POAπ y en PAπ., a diferencia de lo que sucede en Aπ. Así, se establece formalmente la no existencia de una codificación (decidable) de Aπ en PAπ o en el fragmento de POAπ, preservando divergencia y convergencia. Estos resultados de separación no requieren de ninguna condición específica sobre las codificaciones e involucran directamente convergencia por primera vez en el estudio de la persistencia de Aπ.

Palabras clave: *Expresividad, divergencia, convergencia, cálculos de procesos.*

1. Introduction

1.1 The π-calculus

The *(polyadic)* π*-calculus* (Milner, 1999) is one of the most influential formalisms for modeling and analyzing the behavior of concurrent systems; i.e. systems consisting of multiple computing agents, called *processes* that interact with each other. Indeed, the π-calculus has attained a wide range of applications in different areas of computer science and engineering, among others: Biology (Eccher, & Priami, 2006) business processes (Puhlmann, 2007), object-oriented programming (Jones, 1993), security (Abadi, & Gordon 1999), session types (Gay & Hole, 2005), and service oriented computing (Lucchi & Mazzara, 2007).

The relevance of this calculus has given rise to several variants; these variants can focus on some specific features or to be tailored to a specific domain. The exploration of the limitations, redundancies and capabilities of several variants is central in the expressiveness studies on process calculi.

1.2 Expressiveness

Most works on the expressiveness of the π-calculus consider questions such as whether a given variant can express certain behaviors, whether a given variant is as expressive as another one w.r.t. certain equivalence relation, or whether for some property, a given fragment is as hard as the full language.

Unfortunately, the subject of expressiveness in the π-calculus, and process calculi at large, is not a well-established discipline, or even a stable craft. Several guiding principles and cogent classification criteria have been put forth in several works such as (Palamidessi, 2003; Gorla, 2006; Vigliotti, et al., 2005; Gorla, 2008; Fu, & Lu, 2010). Hitherto, however, we do not have a general agreement as to what are the properties that a taxonomy of process calculi must consider in the way we have for the linguistic formalisms of computability, where the notion of language (generation) can be taken as the canonical measure for expressiveness. This is perhaps due to the great diversity of observations and properties often used to reason about concurrent behavior (e.g., divergence, convergence, failures, traces, barbs, must testing, bisimilarity, etc.). It may be the case that rather than being absolute, a taxonomy of concurrent calculi ought to be parametric on the observations we wish to make of processes. After all, concurrency is a field with a myriad of aspects for which we may require different terms of discussion and analysis.

The purpose of this paper is to show an expressiveness study of linearity and persistence features on the main variant of the π-calculus, $A\pi$, considering divergence and convergence as the properties of interest.

Below, it is introduced the topic of this paper, some related works and our contributions.

1.3 Expressiveness issue: Linearity and persistence in the asynchronous π-calculus

In (Palamidessi, et al., 2006) the authors presented an expressiveness study of *linearity* and *persistence* of processes in the asynchronous version of the π-calculus, Aπ. Linearity (and persistence) is understood in a sense of that is similar to that used in (Girard 1987): the ability (incapability) of consuming a resource. The *replication* operator is central in (Palamidessi, et al., 2006) and plays a role similar to the "bang" operator from linear logic, also denoted as !.

The study in (Palamidessi, et al., 2006) is conducted in the *asynchronous* π *- calculus,* which naturally captures the notion of linearity and persistence also present in other calculi.

Let us for example consider the π-calculus system $\overline{x}(\vec{z}) \mid x(y).P \mid x(y).Q$

This system represents a *linear* message with a datum z, tagged with x, that can be *consumed* by either (linear) receiver $x(y).P$ or $x(y).Q$. Persistent messages (and receivers) can simply be specified using the *replication operator* that, as previously mentioned, creates an unbounded number of copies of a given process. One can then consider the existence of encodings from Aπ into three sub-languages of it, each capturing one source of persistence: the *persistent-input* calculus (*PIAπ*), defined as Aπ where inputs are replicated; the *persistent-output* calculus (*POAπ*), defined dually, i.e. outputs rather than inputs are replicated; the *persistent* calculus (*PAπ*), defined as Aπ but with all inputs and outputs replicated.

The main result in (Palamidessi, et al., 2006) basically states that we need one source of linearity, i.e. either on inputs (PIAπ) or outputs (POAπ) to encode the behavior of arbitrary Aπ processes via weak barbed congruence.

The notion of linearity (persistency) is present is several concurrency frameworks. *Persistence of messages* is present, e.g., in Concurrent Constraint Programming (*CCP*) (Saraswat, 1993), *SPL* (Crazzolara, & Winskel, 2001), and the *Spi Calculus* variants in (Amadio, et al., 200)). In all these formalisms messages cannot be consumed. In the π-calculus persistent receivers are used, for instance, to model functions, objects, higher-order communications, or procedure definitions. Furthermore, persistence of *both* messages and receivers arise in the context of *CCP* with universally-quantified persistent ask operations (Fages, et al., 2001; Olarte, & Valencia 2008) and in the context of calculi for security, persistent receivers can be used to specify protocols where principals are willing to run an unbounded number of times (and persistent messages to model the fact that every message can be remembered by the spy).

Now, the previously mentioned positive result in (Palamidessi, et al., 2006) may give insights in the context of the expressiveness of the above frameworks. The main drawback of the work (Palamidessi, et al., 2006) is, however, that the notion of correctness for the encodings is based on weak barbed bisimulation (congruence), which is not sensitive to *divergence*. In particular, the encoding provided in (Palamidessi, et al., 2006) from *Aπ* into *PIAπ* is weak barbed congruent preserving but not divergence preserving. Although in some situations divergence may be ignored, in general it is an important issue to consider in the correctness of encodings (Gorla, 2006; Palamidessi, 2003; Cacciagrano, et al., 2006).

In (Cacciagrano, et al., 2008), the study of linearity and persistence complements the work (Palamidessi, et al., 2006) proving that, under some general conditions, there cannot be an encoding from *Aπ* into a (semi-) persistent calculus preserving the must-testing semantics; this semantics is sensible to divergence.

Unfortunately, in concurrency theory there is no a unified notion about what a good encoding is. There are several works about what properties should have an encoding (Palamidessi, 2003; Gorla, 2006; Vigliotti, et al., 2005; Gorla, 2008; Fu, & Lu, 2010; Nestmann, 2000). Considering this, it would be legitimate to ask, at what extent, the separation results from Aπ into the (semi-) persistent calculi depends on the general conditions in (Cacciagrano, et al., 2008) or only concerns divergence.

The por our main contribution is to show formally that two fragments of the (semi-) persistent subcalculi of Aπ are not as expressive as Aπ when divergence or convergences are considered. Unlike (Cacciagrano, et al., 2008), the separation results showed in this paper does not rely on some particular properties that an encoding should satisfy. The results are the following: Convergence and divergence are decidable in *PAπ*, Convergence and divergence are decidable in the fragment of *POAπ* where the replication operator is restricted to input processes and output processes.

As convergence and divergence are undecidable in *Aπ*, see Remark 1 in Section 3., then, there is no (computable) encoding from Aπ into these fragments.

The relevance of this paper does not rely only on the discriminatory results between different fragments of Aπ, but identifies fragments where it is possible to define a range of processes where it would be possible to determine automatically properties such as convergence and divergence. Decidability results for process calculi as those presented in this paper are important steps towards the development of formal verification tools for concurrent systems.

2. Preliminaries

This section introduces some notions, notations that will be used in the rest of the paper.

2.1 The asynchronous π-calculus: Aπ

Communication in the π-calculus is considered synchronous. The key property relies on the fact that the output and the input prefix impose a precedence over the terms which are underneath, such that once a communication involving the output and the input prefix occurs, the terms which were underneath the prefixes are unguarded at the same time. This behavior can be seen as a kind of acknowledgement of the execution of the communication over the processes involved in it.

Asynchronous π-calculus (Aπ) is a variant of the π-calculus introduced in (Boudol, 1992; Honda, & Tokoro, 1991). In this variant the communication can be seen as asynchronous, in the sense that the act of sending a datum and the act of receiving it can be seen as separate, hence not simultaneous. Aπ is obtained by restricting the term underneath the output prefix to be 0 (the null process). In this way the kind of acknowledgement provided by the precedence in the output prefix is lost. Moreover, an unguarded occurrence of $\overline{x}(y)$ can be thought of as a datum y in an implicit communication medium, tagged with x to indicate that it is available to any unguarded term of the form $x(z)$.P. Thus, in the evolution of a term, the datum y can be considered to be sent when $\overline{x}(y)$ becomes unguarded, and to be received when $\overline{x}(y)$ disappears via an internal action.

2.1.1 Syntax

Names are the most primitive entities in the Aπ-calculus. We presuppose a countable set ***N*** of (port, links or channel) *names*, ranged over by , y,... . For each name *x*, we assume a *co-name* $\overline{x}$ thought of as *complementary*, so we decree that $\overline{\overline{x}}$= x. The other entity in the π-calculus is a *process*. Processes are built from names as follows.

Definition 1 (Syntax) Processes in Aπ-calculus are given respectively by

$$P,Q,... := 0 \mid x(y).P \mid \overline{x}\, y \mid v(x)P \mid P \mid Q \mid !Q$$

The process (summation) 0 does nothing. $\overline{x}\ y$ and x(y).P represent the output and input process respectively, $\overline{x}\ y$ is a process which can output a datum y on channel *x*. x(y).P is a process which can perform an input action on channel x and then it behaves like P{z/y}, the process which has replaced every occurrence of the name y, by the datum z received. {z/y} is a substitution of z by y, x(y) is called a guard or *(input) prefix*. In $P \mid Q$, the parallel composition of P and Q, P and Q can proceed independently or can synchronize via shared names. In (ν x)P , the name x is declared private to P , i.e. initially, components of P can use x to interact with one another but not with other processes, the scope of x could change as a result of interaction between processes as will be seen later. Finally, the replication !*P* can be thought of as unboundedly many P's in parallel $P \mid P \mid P \mid$ ..., replication is the means to express infinite behaviour.

In each of $x(y).P$ and $(v\ y)P$, the occurrence of y is *bound* with *scope* P . An occurrence of a name in a process is *bound* if it is under the scope of a binding occurrence of the name. An occurrence of a name is *free* if it is not bound. Given Q we define its *bound names* $bn(Q)$ as the set of names with a bound occurrence in Q, and its *free names* $fn(Q)$ as the set of names with a non-bound occurrence in Q, hence $n(Q) = fn(Q) \cup bn(Q)$ is the set of names of Q.

In this paper, we consider a version of Aπ without τ and choice as proposed in (Boudol, 1992; Honda, & Tokoro, 1991).

2.1.2 Semantics

The semantics of the language described above is made precise by a labeled transition system. A transition P $\xrightarrow{\alpha}$ Q says that P can perform an *action* α and evolve into Q. The set of actions used in the transition system is composed by $\bar{x}\,y$, $x\,y$, $\bar{x}(y)$, τ. $\bar{x}\,y$, *a free output*, sends the name y on the name x, $x\,y$, *an input*, receives the name y on the name x, $\bar{x}(y)$, *a bound output*, sends a fresh name on x and τ is an *internal action* .

Definition 2 *(Semantics) The labeled transition relation* α $\xrightarrow{\alpha}$ *is given by the rules in Table 1. Omitted from Table 1 are the symmetric forms of Par-L, Com- L and Close-L.*

2.2 Divergence and convergence

Definition 3 *We say that* P *is stable iff* P cannot perform any action α.

Definition 4 W*e say that P is* convergent *P*$\downarrow$*,* iff there is a stable process *Q* such that $P(\xrightarrow{\tau})^{*}$ *Q*. We say that *P* is divergent, *P*$\uparrow$, *iff* $P\,(\xrightarrow{\tau})^{\omega}$, i.e., there exists an infinite sequence $P = P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \ldots$

2.3 Semi-persistence in Aπ

Here we define the syntactic restrictions of Aπ that are considered in this paper.

The fragment of persistent-output calculus $POA\pi_{ri}$, arises as from Aπ by requiring all outputs to be replicated where the replication operator is restricted to input processes and output processes.

$$\text{Input } x(y).P \xrightarrow{xz} P\{z/y\}\, where\ x,y \in \boldsymbol{N}$$

$$\text{Output } \bar{x}\ y \xrightarrow{\bar{x}y} 0$$

$$\text{Open } \frac{P \xrightarrow{\bar{x}y} P\prime}{(v\,y)P \xrightarrow{\overline{x}(y)} P\prime}\ x \neq y \qquad \text{Res } \frac{P \xrightarrow{\alpha} P\prime}{(v\,y)P \xrightarrow{\alpha} (v\,y)P\prime}\ y \notin n(\alpha)$$

$$\text{Par-L } \frac{P \xrightarrow{\alpha} P\prime}{P \,\big|\, Q \xrightarrow{\alpha} P' \,\big|\, Q}\ \ bn(\alpha) \cap fn(Q) = \emptyset$$

$$\text{Com-L } \frac{P \xrightarrow{\bar{x}y} P',\ \ Q \xrightarrow{xy} Q'}{P \,\big|\, Q \xrightarrow{\tau} P' \big| Q'} \qquad \text{Close-L } \frac{P \xrightarrow{\overline{x}(y)} P',\ Q \xrightarrow{x\,y} Q'}{P \,\big|\, Q \xrightarrow{\tau} (v\,y)\,(P' \big| Q')}$$

$$\text{Rep-Act } \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

$$\text{Rep-Comm } \frac{P \xrightarrow{\overline{x}y} P',\ P' \xrightarrow{x\,y} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}$$

$$\text{Rep-Close } \frac{P \xrightarrow{\overline{x}(z)} P',\ P \xrightarrow{x\,y} P''}{!P \xrightarrow{\tau} ((v\,z)(P' \mid P'')) \mid !P}\ \ z \notin fn(P)$$

Table 1. *Operational Semantics for the Aπ-calculus*

Processes in $POA\pi_{ri}$ are generated by the following grammar:

$$P,Q,... := 0 \mid x(y).P \mid !x\,(y).P \mid !x\,y \mid (v\,x)P \mid (P \mid Q\,)$$

Finally, we have the *persistent* calculus π, a subset of Aπ where output and input processes must be replicated. Processes in *PAπ* are generated by the following grammar:

$$P,Q,... := 0 \mid !x\,(y).P \mid !\overline{x}\,y \mid (v\,x)P \mid (P \mid Q\,) \mid !P$$

The relation $\xrightarrow{\alpha}$ for $POA\pi_{ri}$ and *PA*π can be equivalently defined as in Table 1, with Output replaced with Output($POA\pi_{ri}$), and Input and Output replaced with Input(*PA*π) and Output(*PA*π) rules showed below. The new rules reflect the persistent-output nature of $POA\pi_{ri}$ (Rule Output($POA\pi_{ri}$)), and the persistent nature of PAπ (Rules Input and Output(PAπ)). Notice that these new rules can be derived directly from the application of the Rules Input, Output, and Rep-Act in Table 1.

$$\text{Output }(POA\pi_{ri})\ !\overline{x}\,y \xrightarrow{\overline{x}\,y} 0 \mid !\overline{x}\,y$$

$$\text{Input}(PA\pi)\ !x\,(y).P \xrightarrow{x\,z} P\,\{z/y\} \mid x(y).P \textit{ where } x, y \in N$$

$$\text{Output }(PA\pi)\ !\overline{x}\,y \xrightarrow{\overline{x}\,y} 0 \mid !\overline{x}\,y$$

Remark 1 Notice that, unlike to the presentation of Aπ, its syntactic restrictions does not include terms as !P explicitly. However, it is clear that all the terms from these restricted variants correspond to Aπ terms. Considering this, and for the sake of uniformity in the presentation of the proofs, we consider processes in Lemmas 1, 2, 3, and 4 as Aπ terms rather than restricted language terms. Clearly the results obtained from these lemmas are valid for the two syntactic variants.

3. Decidability results for $\mathbf{POA\pi_{ri}}$ and PAπ

We shall prove that there is no computable encoding preserving divergence or convergence from *Aπ* into $POA\pi_{ri}$ and *PAπ*. We do this by proving that unlike for *Aπ*, divergence and convergence are decidable for these $POA\pi_{ri}$ and *PAπ* processes.

Below, a remark on the undecidabiliy of convergence and divergence in *Aπ*.

Remark 2 Convergence and divergence are undecidable in *Aπ*. In (Busi, et al, 2009), it was proved the undecidability of convergence and divergence for the calculus *CCS*. That result is extended directly to *Aπ* by using the encoding from *π*-calculus with recursive functions into replication showed in (Sangiorgi, Walker, 2001). The encoding from guarded-choice *π*-calculus into choice-free π-calculus given in (Palamidessi, et al, 2006), and either Honda and Tokoro's encoding or Boudol's encoding from π-calculus into *Aπ* proposed in (Boudol, 1992) and (Honda, & Tokoro, 1991). All of these encodings preserve and reflect divergence and convergence.

We need to prove that the set of reachable processes through a *τ*-action can be computed; $Succ(P) = \{P' \mid P \xrightarrow{\tau}, P'\}$ is computable.

Without lose of generality, we assume that all the bound and free names are distinct in every process we consider in this section. Notice that every process can be transformed into an equivalent process with distinct names by using α-conversion (Sangiorgi, & Walker, 2001).

It is well known that the relation $\xrightarrow{\alpha}$ is image-finite (Sangiorgi, & Walker, 2001). Therefore the set of successors of a process P, Succ(P), is finite. Here we describe how to build this set.

3.1 Computing successors

Now, it will be illustrated how to calculate derivative processes fron any Aπ process *P* is computable. The approach is to show a function doing the corresponding calculations. Depending on the nature of the derivation, one of the following lemmas is the most suitable.

The computability of the derivative processes is extended trivially to the terms defined in its syntactic restrictions of Aπ, i.e $POA\pi_{ri}$ and *PAπ*.

Lemma 1 *For any Aπ P, Deriv*$_{x\,z}$ *(P) = {P' | P* $\longrightarrow$ *, P'}is computable.*

Proof.

Let us define inductively the set $Der(P)$ as follows:

- $P = 0{:}\ Der(P) := \emptyset.\ P = \bar{x}\,y{:}\ Der(P) := \{0\}.\ P = x(y).Q : Der(P) := \emptyset.$
- $P = Q|R{:}\ Der(P) := \{(Q'|R) | Q' \in Der(Q)\} \cup \{(Q|R') | R' \in Der(R)\}.$
- $P = (\nu y)Q : Der(P) := \{(\nu y)Q' | Q' \in Der(Q)\}\ if\ y \notin \{x,z\}, otherwise\ Der(P) := \emptyset.$
- $P = !Q : Der(P) \coloneqq \{(Q'|!Q) | Q' \in Der(Q)\}.$

It can be proved that $Der(P) = Deriv_{\bar{x}\,z}(P)$ by structural induction on P. ■

Lemma 2 *For any Aπ P,* $Deriv_{bound-output}(P) = \{(\alpha, P') | P \xrightarrow{\alpha} P'$

$where\ \alpha\ is\ a\ bound - output\}$ *is computable.*

Proof.

Let us define inductively the set $Der(P)$ as follows:

- $P = 0{:}\ Der(P) := \emptyset.P = \bar{x}\,y{:}\ Der(P) := \emptyset.P = x(y).Q : Der(P){:}\ \emptyset.$
- $P = Q|R{:}\ Der(P) := \{(\alpha, Q'|R) | (\alpha, Q') \in Der(Q)\} \cup \{(\alpha, Q|R') | (\alpha, R') \in Der(R)\}.$
- $P = (\nu y)Q : Der(P) \coloneqq \{(\bar{x}(y), (\nu y)Q') | Q' \in Deriv_{\bar{x}\,z}(P))\ where\ x \notin y\}$
- $P = !Q : Der(P) := \{(\alpha, Q' | !Q) | (\alpha, Q') \in Der(Q)\}.$

It can be proved that $Der(P) = Deriv_{bound-output}(P)$ by structural induction on P. ■

Lemma 3 *For any Aπ P,* $Deriv_{x\,z}(P) = \{P' | P \xrightarrow{x\,z} P'\}$ *is computable.*

Proof.

Let us define inductively the set $Der(P)$ as follows:

- $P = 0{:}\ Der(P) := \emptyset.P = \bar{x}\,y{:}\ Der(P) := \emptyset.P = x(y).Q : Der(P) := \{Q\{z/y\}\}.$
- $P = Q|R{:}\ Der(P) := \{(Q'|R) | Q' \in Der(Q)\} \cup \{(Q|R') | R' \in Der(R)\}.$
- $P = (\nu y)Q : Der(P) \coloneqq \{((\nu y)Q') | Q' \in Der(Q)\}\ if\ y \notin \{x,z\}, otherwise\ Der(P) \coloneqq \emptyset.$
- $P = !Q : Der(P) := \{(Q' | !Q) | Q' \in Der(Q)\}.$

It can be proved that $Der(P) = Deriv_{x\,z}(P)$ by structural induction on P. ■

Lemma 4 *For any Aπ P, Succ*$(P) = \{P' \mid P \xrightarrow{\tau} P'\}$ *is computable.*

Proof.
Let us define inductively the set $Der(P)$ as follows:

- $P = 0$: $Der(P) := \emptyset$.
- $P = \bar{x}\,y$: $Der(P) := \emptyset.P = x(y).Q$: $Der(P) := \{Q\{z/y\}\}$.

$P = Q|R$: $Der(P) := \{(Q'|R) | Q' \in Der(Q)\} \cup \{(Q|R') \,|R' \in Der(R)\} \cup Der_n(P)$

where $Der_n(P)$ represents the set of the derivative processes from P through a $\tau - action$ resulting from synchronization between Q and R. $Der_n(P)$ is defined as follows:

$Der_n(P) := \{(Q'|R')|Q' \in Deriv_{\bar{x}\,z}(Q), R' \in Deriv_{x\,z}(R)\ for\ some\ x, z \in fn(R)\}$

$\cup\ \{(Q'|R')|Q' \in Deriv_{x\,z}(Q), R' \in Deriv_{\bar{x}\,z}(R)\ for\ some\ x, z \in fn(R)\}\ \cup$

$\{(v\,z)(Q'|R')|\ (\bar{x}(z), Q') \in Deriv_{bound-output}(Q), R' \in Deriv_{x\,z}(R)\ for\ some\ x \in fn(Q), z \notin fn(R)\}$

$\{(v\,z)(Q'|R')|\ (\bar{x}(z), R') \in Deriv_{bound-output}(R), Q' \in Deriv_{x\,z}(R)\ for\ some\ x \in \cup\, fn(R), z \notin fn(Q)\}$.

- $P = (v\,y)Q : Der(P) := \{(\,(v\,y)\,Q')\ \mid Q' \in Der(Q)\}$
- $P = !Q : Der(P) := \{(Q' \mid !Q)\ \mid Q' \in Der(Q)\} \cup$

$\{((Q'|Q'')|!\,Q\,)|Q' \in Deriv_{\bar{x}\,z}(Q), Q'' \in Deriv_{x\,z}(Q)\ for\ some\ x, z \in fn(Q)\}$

$\{((v\,z)(Q'|Q'')\ |!\,Q\,)|\ (\bar{x}(z), Q') \in Deriv_{bound-output}(Q), Q'' \in Deriv_{x\,z}(Q)\ for\ some\ x \in \cup\, fn(Q), z \notin fn(Q)\}$

$\cup \{((Q''|Q')|!\,Q)\,|Q' \in Deriv_{\bar{x}\,z}(Q), Q'' \in Deriv_{x\,z}(Q)\ for\ some\ x, z \in fn(Q)\}\ \cup$

$\{((v\,z)(Q''|Q')\ |!\,Q\,)|\ (\bar{x}(z), Q') \in Deriv_{bound-output}(Q), Q'' \in Deriv_{x\,z}(Q)\ for\ some\ x \in fn(Q), z \notin fn(Q)\}$

The calculability of $Der(P)$ when $P = Q \mid R$ or $P = !Q$ relies on the calculability of $Deriv_{x\,z}(\_)$, $Deriv_{\bar{x}\,z}(\_)$ and $Deriv_{bound-output}(\_)$ which are shown in Lemmata 3, 1, and 2 respectively.
It can be proved that Der(P) = Succ(P) by induction on P. ■

By using the function $Succ$, we can now determine whether a process is convergent (divergent) or not.

3.2 Decidability of convergence and divergence

Now, we can show that convergence and divergence are decidable for $POA\pi_{ri}$ and $PA\pi$. The next theorem shows the decidability of divergence and convergence in $PA\pi$.

Theorem 1 *Divergence and convergence are decidable in* $PA\pi$.

Proof.
From the persistent nature of both input and output prefixes in $PA\pi$, we know that if a synchronisation happens in a $POA\pi_{ri}$ process then there must be an infinite τ-labelled computation from such a $PA\pi$ process. Hence a $PA\pi$ process P is divergent if and only if $Succ(P)>0$ and we can say that a $PA\pi$ process P is convergent if and only if $Succ(P)=0$. ■

Now, we consider the decidability of convergence and divergence for $POA\pi_{ri}$.

First we need to introduce the notion of *occurrence of linear input prefix*. A linear input prefix is the input prefix that is not under the scope of the replication operator.

Definition 6 (Occurrences of linear input prefix) *Let* $P \in POA\pi_{ri}$. The maximal number of occurrences of linear inputs in P, $LinearInp(P)$ is given inductively as follows*:* $LinearInp(0)=0$, $LinearInp(!x^{-})=0$, $LinearInp(\mathrm{a}(x).P)=1+LinearInp(P)$, $LinearInp(!P)=0$, $LinearInp((vx)P)=LinearInp(P)$, $LinearInp(P \mid Q)==LinearInp(P)+=LinearInp(Q)$.

The following proposition says that only input actions that come from linear input prefixes can participate, by synchronization, in a finite maximal sequence of τ-actions.

Proposition 1 Let $P,\ P' \in POA\pi_{ri}$ *such that* $P \xrightarrow{\tau} P'$ *and* P' *is convergent. Then* P *is convergent and each* τ *-move from* P *into* P' *is produced by a synchronization between an output and an input action coming from a linear input prefix.*

Proof.
To prove the first part, to obtain a contradiction, let us suppose that P is non-convergent, i.e. there is no maximal finite computation from P. As any maximal computation from P' can be seen as the ending part of a maximal computation from P passing through. Each maximal computation from P' must be infinite. Therefore. P' is non-convergent, a contradiction.

To prove the second part, to obtain a contradiction, let us suppose that there is a τ -move from P into (convergent) P' produced by a synchronisation between an output and an input action coming from an input prefix being under the scope of a replication operator. Since output actions are persistent at time in $POA\pi_{ri}$, i.e. once an output action can be performed, the same action can be executed at anytime later on, and the input actions coming from input prefix being under the scope of a replication operator are persistent as well, there is possible to perform a synchronization from P' and any of its τ-derivative processes. Hence P' cannot be convergent, a contradiction. ■

As a corollary from Proposition 1, we have the following proposition:

Proposition 2 *Let* $P,\ P' \in POA\pi_{ri}$ *such that such that* $P \xrightarrow{\tau} P'$ *and* P' *is convergent. Then* $LinearInp(P) \geq 1$.

A crucial observation to prove the decidability of convergence and divergence in $POA\pi_{ri}$ is that the number of occurrences of linear input prefix decreases as long as a finite computation is performed.

Proposition 3 *Let* $P,\ P' \in POA\pi_{ri}$ *such that such that*$P \xrightarrow{\tau} P'$ *and* P' *is convergent. Then* $LinearInp(P) = LinearInp(P) - 1$.

Proof.
From Proposition 1, we know that any τ-move from P into P' corresponds to a synchronisation where the input action comes from a linear input prefix. The participation of this kind of input action implies that an occurrence of a linear input prefix is consumed from P. Notice that although

the execution of this input action can substitute names in *P*, the linear or persistent nature of the rest of the process remains unchanged. As for the output action, the consumption of an output action does not alter the number of occurrences of linear input prefix; it is due to the asynchronous nature of the calculus. ■

The following Lemma gives an upper bound of the length of the maximal finite computations that depends on the number of occurrences of linear inputs prefix. Notice that the lower bound does not depend on this number, e.g. $a(x).0 \mid a(x).0 \mid ... \mid a(x).0$ is stable.

Lemma 5 *Let* $P \in POA\pi_{ri}$*. For each maximal finite* τ*-labeled computation c from P, length(c)* ≤ *LinearInp(P).*

Proof.
Let us consider any maximal finite computation from *P*:

$$P \xrightarrow{\tau}, P_1 \xrightarrow{\tau}, P_2 \xrightarrow{\tau}, P_3 \xrightarrow{\tau} ... \xrightarrow{\tau} P_n$$
$$\textit{where } P_n \textit{ is estable}$$

From Proposition 3 we know that *LinearInp*(P_1) = *LinearInp(P)*– 1, in general *LinearInp*(P_i) = *LinearInp(P)* – i. Consequentenly, *LinearInp*($P_{LinearInp(P)}$) – 0. From Proposition 2, $P_{LinearInp(P)}$ is stable. Therefore, *length(c)* ≤ *LinearInp(P).* ■

From the computable function *Succ*, we can define and calculate a function $Succ^i(P)=\{P \mid P' \xrightarrow{\tau} P''$ *for some* $P \in Succ^{i-1}(P)\}$ where $Succ^1(P) = Succ(P)$ and $Succ^0(P) = \{P\}$, in a similar way we can identify the stable processes derivable from P at *i* τ-actions by the function $SuccSt^{\,i}(P) =\{P'' \mid P' \xrightarrow{\tau} P''$ *for some* $P' \in Succ^{i-1}(P)$ and $\in Succ(P'') =\{\}\}$.

Now, it is easy to see from Lemma 5 and by using the function$Succ^i$ and $SuccSt^{\,i}$, which are computable from Lemma 4, that divergence and convergence are decidable.

Theorem 2 *Divergence is decidable in* $POA\pi_{ri}$ *.*

Proof.
From Lemma 5, a $POA\pi_{ri}$ *process P is divergent if and only if there is at least one computation from P whose length is greater than LinearInp(P). It can be checked whether such a computation exists by ver- ifying that* $|\,SuccSt^{\,LinearInp(P)+1}(P)| \geq 0$*. From Lemma 4 it is clear that* $SuccSt^{\,LinearInp(P)+1}(P)$ *can be straightforwardly calculated.* ■

Theorem 3 *Convergence is decidable in* $POA\pi_{ri}$ *.*
Proof. From Lemma 5, a $POA\pi_{ri}$ *process P is convergent if and only if there is at least one maximal computation from P whose length is less or equal to LinearInp(P), i.e. if there is at least one stable process derivable from P at most in LinearInp(P)* τ*-moves. It can be checked whether such a stable process exists by verifying that* $|SuccSt^{\,0}(P)| + |SuccSt^{\,1}(P)| + |\,SuccSt^{\,2}(P)| + ... + |\,SuccSt^{\,LinearInp(P)}(P\,)| \geq 1$*. From Lemma 4 it is clear that* $SuccSt^{\,i}(P)$ *for any natural number i can be straightforwardly calculated.* ■

As corollary from Theorem 2 and Theorem 3 and considering Remark 5.1.1 we obtain the following separation result:

Theorem 4 *There is no encoding preserving and reflecting divergence (convergence) from A*π *into* $POA\pi_{ri}$*.*

4. Conclusions

In this paper we studied the decidability of divergence and convergence in two fragments of *A*π: $POA\pi_{ri}$ and *PA*π. As main contribution we showed that these two properties are decidable in both fragments.

As the divergence-sensitive nature of Testing Semantics (Nicola, & Hennessy, 1984) and the failures-sensitive nature of Failures Semantics (Milner, 1989), these results support and strengthen the separation results from (Cacciagrano, et al., 2008). This paper does not take into account particular properties that an encoding should satisfy, in addition this paper extends the expressiveness gap considering for first time convergence.

Notice that although the results for $PA\pi$ could be obtained straightforwardly from the results for $POA\pi_{ri}$, the nature of the separation is different in both cases: the separation results for $PA\pi$ seems to rely on the persistence nature of the processes exclusively, thus, we claim that this result is valid for the full synchronous π-calculus, as the results of (Cacciagrano, et al., 2008). On the other hand, the expressiveness gap between $A\pi$ into$POA\pi_{ri}$ relies on asynchrony, and then this result seems to be specialized for this fragment.

Although it seems that the results from this paper are not unexpected, one of the most relevant and important contributions of this paper relies on identifying precisely a range of processes where it would be possible to determine automatically properties such as convergence and divergence. A process calculi term can be used to model several concurrent real-life systems. Decidability results for process calculi are an important contribution towards the development of formal verification tools for concurrent systems.

Although this paper shows strong impossibility results on two fragments, it is necessary to extend this work and to explore these properties on full $POA\pi$ and $PIA\pi$. In a near future, we expect to analyze how to model a Turing-equivalent model preserving divergence (convergence) into $PIA\pi$, as some evidence (Aranda, 2009) suggests that $PIA\pi$ is more expressive than $POA\pi$.

5. References

Abadi, M., & Gordon, A. (1999). *A calculus for cryptographic protocols: The spi calculus*. Inf. Comput. 148 (1), 1–70.

Alves-Foss, J. (2000). *An efficient secure authenticated group key exchange algo- rithm for large and dynamic groups.* In Proceedings of the 23rd National Information Systems Security Conference.

Amadio, R., Lugiez, D., & Vanackere,. V. (2003). On the symbolic reduction of processes with cryptographic functions. *Theor. Comput. Sci.* 290 (1), 695– 740.

Aranda, J., Di Giusto, C., Nielsen, M., & Valencia, F. (2007). *CCS* with replication in the Chomsky hierarchy: the expressive power of divergence. *Lecture Notes in Computer Science* 4807, 383–398.

Aranda, J., Valencia, F., & Versari, C. (2009). On the expressive power of restriction and priorities in ccs with replication. *Lecture Notes in Computer Science 5504,* 242–256.

Aranda, J. (2009). *On the expressivity of Infinite and local behaviour in fragments of the Π-calculus.* Tesis de Doctorado, Escuela de Ingeniería de Sistemas y Computación, Universidad del Valle, Cali, Colombia.

Borger, E., Gradel, E., & Gurevich, Y. (1997). *The Classical Decision Problem.* Springer Verlag, Berlin.

Boudol, G., *Asynchrony and the pi-calculus*. (1992). Technical Report RR-1702, INRIA Sophia Antipolis.

Busi, N. (2002) Analysis issues in petri nets with inhibitor arcs. *Theor. Comput.* Sci 275 (1-2), 127–177.

Busi, N., Gabbrielli, M., & Zavattaro, G. (2003) Replication vs. recursive definitions in channel based calculi. Lecture *Notes in Computer Science* 2719, 133–144.

Busi, N., Gabbrielli, M., & Zavattaro, G. (2004) Comparing recursion,replication, and iteration in process calculi. *Lecture Notes in Computer Science* 3142, 307–319.

Busi, N,. Gabbrielli, M,. & Zavattaro, G. (2009). On the expressive power of recursion, replication and iteration in process calculi. *Mathematical Structures in Computer Science* 19 (6), 1223-1263.

Cacciagrano, D., & Corradini, F. (2001). On synchronous and asynchronous communication paradigms. *Lecture Notes in Computer Science* 2202, 256–268.

Cacciagrano, D., Corradini, F., Aranda, J., & Valencia, F. (2008). *Linearity, persistence and testing semantics in the asynchronous pi-calculus.* Electr. Notes Theor. Comput. Sci. 194 (2), 59–84.

Cacciagrano, D., Corradini, F., & Palamidessi, C. (2006) Separation of synchronous and asynchronous communication via testing. *Electr. Notes Theor. Comput. Sci* 154 (3), 95–108.

Crazzolara, F., & Winskel, G. (2001). *Events in security protocols*. In ACM Conference on Computer and Communications Security, p. 96–105.

Eccher, C., & Priami, C. (2006). Design and implementation of a tool for translating sbml into the biochemical stochastic pi-calculus. *Bioinformatics* 22 (24), 3075–3081.

Fages, F., & Ruet, P., & Soliman, S. (2001). Linear Concurrent Constraint Programming: Operational and Phase Semantics. *Inf. Comput* 165(1), 14-41.

Fu, Y., & Lu, H. (2010). On the expressiveness of interaction. *Theoretical Computer Science* 411 (11-13), 1387-1451.

Gay, S., & Hole, M. (2005). *Subtyping for session types in the pi calculus*. *Acta Inf* 42 (2-3), 191–225.

Gorla, D. (2006). *On* the relative expressive power of asynchronous communication primitives. *Lecture Notes in Computer Science* 3921, 47–62.

Gorla, D. (2008). Towards a unified approach to encodability and separation results for process calculi. *Lecture Notes in Computer Science* 5201, 492-507.

Honda, K. & Tokoro, M. (1991). An object calculus for asynchronous communication. *Lecture Notes in Computer Science* 512 , 133–147.

Jones, C. (1993). A pi-calculus semantics for an object-based design notation. *Lecture Notes in Computer Science* 715, 158–172.

Lucchi, R., & Mazzara, M. (2007). A π-calculus based semantics for ws-bpel. J. *Log. Algebr. Program* 70 (1), 96–118.

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

Milner, R., (1999). *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press.

Nicola, R., & Hennessy, M. (1984). Testing equivalences for processes. *Theor. Comput. Sci.* 34, 83–133.

Nestmann, U., (2000). What is a “Good” Encoding of Guarded Choice? *Inf, Comput* 156 (1-2). 287-319.

Olarte, C., & Valencia, F. (2008). The expressivity of universal timed CCP: undecidability of Monadic FLTL and closure operators for security. PPDP, p. 8-19.

Palamidessi, C. (2003). Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science* 13 (5), 685–719.

Palamidessi, C., Saraswat, V., Valencia, F,. & Victor, B. (2006). *On the expressiveness of linearity vs persistence in the asychronous pi-calculus.* In *LICS*, p. 59–68. IEEE Computer Society.

Phillips, I. (2008). CCS with priority guards. *J. Log. Algebr. Program* 75 (1), 139– 165.

Puhlmann, F. (2007). Soundness verification of business processes specified in the pi-calculus. *Lecture Notes in Computer Science* 4803, 6–23.

Sangiorgi, D. & Walker, D. (2001). *The π–calculus: A Theory of Mobile Processes*. Cambridge University Press.

Saraswat. V. (1993). *Concurrent Constraint Programming*. The MIT Press.

Vigliotti, M., Phillips, I. & Palamidessi, C. (2005). Separation results via leader election problems. *Lecture Notes in Computer Science* 4111, 172–194.