Encoding asynchronous interactions using open Petri nets*

Paolo Baldan[1] and Filippo Bonchi[2,3] and Fabio Gadducci[3]

[1] Dipartimento di Matematica Pura e Applicata, Università di Padova
[2] Centrum voor Wiskunde en Informatica, Amsterdam
[3] Dipartimento di Informatica, Università di Pisa

Abstract. We present an encoding for (bound) processes of the asynchronous CCS with replication into open Petri nets: ordinary Petri nets equipped with a distinguished set of *open* places. The standard token game of nets models the reduction semantics of the calculus; the exchange of tokens on open places models the interactions between processes and their environment. The encoding preserves strong and weak CCS asynchronous bisimilarities: it thus represents a relevant step in establishing a precise correspondence between asynchronous calculi and (open) Petri nets. The work is intended as fostering the technology transfer between these formalisms: as an example, we discuss how some results on expressiveness can be transferred from the calculus to nets and back.

Keywords: Asynchronous calculi, bisimilarity, decidability, open Petri nets.

1 Introduction

Distributed systems often rely on asynchronous communication, where the operation of sending messages is non-blocking: a process may send a message without any agreement with the receiver, and continue its execution while the message travels to destination. After the introduction of the *asynchronous π-calculus* [1, 2], many process calculi have been proposed that embody some asynchronous communication mechanism (see [3–5], among others). Due to the asymmetry between sending and receiving, behavioural equivalences for asynchronous systems (see e.g. [4, 6–9]) exhibit subtle differences with respect to their synchronous counterparts. Indeed, since sending is non-blocking, an external observer (interacting with a system by message exchanges) cannot know if a message has been received and thus message reception is considered, to some extent, unobservable.

In this paper we aim at establishing a formal correspondence between asynchronous calculi and Petri nets [10]. Perhaps due to their longevity, Petri nets are the best known and most widely used formalism for the visual specification of distributed systems. Besides being used in countless verification tools, suitable

* Partly supported by the EU FP6-IST IP 16004 SENSORIA and carried out during the second author's tenure of an ERCIM "Alain Bensoussa" Fellowship Programme.

net instances have been successfully adopted as a specification domain in many areas: for their relevance, we mention web services and workflow nets [11].

Petri nets exhibit both synchronous and asynchronous features in their behaviour. On the one hand, a transition having more than one place in its pre-set can be seen as a synchronisation point: it can be executed only if all the needed tokens are available in its pre-set at the same time. On the other hand, a token is first produced by a transition and it then remains available until another transition consumes it, in an asynchronous fashion.

The correspondence between synchronous calculi and Petri nets has been thoroughly investigated (see e.g. [12–16]). Typically, a set of operations is identified on a class of nets, and used for a denotational mapping of the calculus. The archetypal example is maybe the *simple calculus of nets* [13]: each occurrence of a CCS prefix in a process is mapped into a net transition, labelled by the name of the corresponding channel; and basic transitions are used for the parallel and non-deterministic operators. The resulting encodings are often quite syntactical, and forced to restrict their attention to the finite fragment of a calculus. More set-theoretical encodings, basically mapping each process onto a net whose places represent its sub-processes, whilst transitions simulate the control flow of the process, were considered [17]. The dichotomy is denoted [18] as *label* vs *location* oriented encoding, where it is argued that nets with inhibitor arcs should be considered for calculi equipped with both sums and restrictions.

Here we focus on the relation between *asynchronous* calculi and Petri nets. We propose an approach to process encoding which differs from those outlined above, as it relies on *reduction semantics* [19, 20] and *open nets* [21–23]. Open nets are ordinary P/T Petri nets equipped with a set of *open places*, i.e., a set of places visible from the environment: a net may then interact with the environment by exchanging tokens on these places. Open nets are closely related to previous approaches to reactivity and compositionality for Petri nets (see, e.g., [24–27], to mention a few), which, interestingly enough, have been often inspired by the search of encodings of calculi into nets or, more generally, by the investigation of the relation between Petri nets and process calculi.

Specifically, we encode an asynchronous variant of CCS [7] into open nets, in such a way that each process reduction corresponds to a transition firing in the net, and vice versa. The key idea is to exploit openness of places in order to account for name restriction. The free names of a process correspond to the open places of the associated net, and message exchanging between a process and the environment on a channel corresponds to token exchanging on open places. As the set of places in a net is fixed, the encoding applies to *bound* processes, i.e., processes where no restriction operator occurs under the scope of a replication (thus avoiding the generation of an unbounded number of restricted names).

Summarizing, the main features of our encoding are

1. it preserves the structural congruence of processes,
2. a bijective relation holds between process reductions and net firings,

3. the interaction between processes and environment is naturally modeled by the built-in interaction mechanism of open nets, thus formalising the fact that net interaction on open places is eminently *asynchronous*;
4. it preserves and reflects both strong and weak asynchronous bisimilarity.

As far as we know, the latter is the first result of this kind, raising the correspondence between reductions and firing steps up-to the level of asynchronous observational equivalences. Furthermore, it seems noteworthy that, while we consider the asynchronous equivalences for the calculus, the equivalences for open nets exploit the standard (either weak or strong) bisimulation game.

We believe that our encoding and its properties establish the fundamental correspondence between asynchronous calculi and open nets, thus paving the way for a fruitful "technology transfer" between the two formalisms.

In this paper, we exploit the encoding of (bound) asynchronous CCS processes into open nets in order to answer some questions about expressiveness of (fragments of) the two models. In an independent work, the recent paper [16], building upon [28], offers some results concerning the expressive power of restriction and its interplay with replication in synchronous calculi. Here we prove that analogous results can be given for asynchronous calculi. We first show that for bound asynchronous CCS processes strong and weak asynchronous bisimilarities (as well as many other behavioural equivalences) are undecidable. Exploiting the encoding, we immediately obtain that these bisimilarities are undecidable also for open nets. This fact falls outside the known undecidability of bisimilarity for Petri nets [29], as we only observe the interaction with the environment: internal transitions are indistinguishable for strong equivalences and unobservable for weak equivalences (e.g., all standard, closed Petri nets are weakly bisimilar in our setting). In the other direction, using the fact that reachability is decidable for Petri nets, through the encoding we prove that reachability and convergence are decidable for bound asynchronous CCS (which, thus, is not Turing powerful).

As mentioned before, in the study of the relation between Petri nets and process calculi, asynchronous calculi has received less attention than synchronous ones. In general terms, most of the proposals we are aware of put their emphasis on the preservation of the operational behaviour, while behavioural equivalences are seldom studied. This is e.g. the pattern followed in [30], which considers possible encodings of the join calculus, where communication is asynchronous, into Petri nets. In particular, the fragment of the join calculus with no name passing and process generation is shown to correspond to ordinary P/T Petri nets, while, in order to encode wider classes of join processes, high-level Petri nets, ranging from coloured nets to dynamic nets must be considered. The encoding share some ideas with ours, e.g., the fact that Petri net places are partitioned into public and private places, even if it does not tackle the relations between process and net behavioural equivalences. Some related work has been done in the direction of encoding several brands of coordination languages, where processes communicate through shared dataspaces, as Petri nets. The papers [30, 31] exploit the encoding to compare the expressiveness of Linda-like calculi with various communication primitives. In [32] an encoding of KLAIM, a Linda-like language

with primitives for mobile computing, into high-level Petri nets is provided. The long-term goal there is to reuse in the context of KLAIM the techniques available for Petri net verification. Concrete results in this direction are obtained in [33], where finite control π-calculus processes are encoded as safe Petri nets and verified using an unfolding-based technique.

The paper is structured as follows. In Section 2 we recall the syntax and the reduction semantics of asynchronous CCS, further presenting the strong and weak (barbed) bisimilarities for the calculus. Section 3 recalls open nets and their equivalences. Section 4 is the core of the paper: it presents the encoding from bound processes of asynchronous CCS into open nets, proving that the encoding preserves and reflects the operational as well as the observational semantics of the calculus. Section 5 discusses some expressiveness issues for the considered models, taking advantage from the encoding. Finally, Section 6 draws some conclusions and provides pointers to future works.

2 Asynchronous CCS

Differently from synchronous calculi, where messages are simultaneously sent and received, in asynchronous communication the messages are sent and travel through some media until they reach destination. Thus sending is not blocking (i.e., a process may send even if the receiver is not ready to receive), while receiving is (processes must wait until a message becomes available). Observations reflect the asymmetry: since sending is non-blocking, receiving is unobservable.

This section introduces asynchronous CCS as a fragment of asynchronous π-calculus (with no name passing). We adopt the presentation in [6] that allows non-deterministic choice for input prefixes (a feature missing in [4, 7]).

Definition 1 (processes). *Let $\mathcal{N}$ be a set of* names*, ranged over by $a, b, c, \ldots$, and $\tau \notin \mathcal{N}$. A* process *P is a term generated by the (mutually recursive) syntax*

$$P ::= M,\ \ \bar{a},\ \ (\nu a)P,\ \ P_1 \mid P_2,\ \ !_a.P \qquad M ::= \mathtt{0},\ \ \mu.P,\ \ M_1 + M_2$$

for μ ranging over $\{\tau\} \cup \mathcal{N}$. We let $P, Q, R, \ldots$ range over the set Proc of processes, and $M, N, O \ldots$ range over the set Sum of summations*.*

The main difference with standard CCS [34] is the absence of output prefixes. The occurrence of an unguarded $\bar{a}$ indicates a message that is available on some communication media named a, and it disappears whenever it is received.

We assume the standard definitions for the set of free names of a process P, denoted by $fn(P)$. Similarly for α-convertibility w.r.t. the *restriction* operators $(\nu a)P$: the name a is restricted in P, and it can be freely α-converted. *Structural equivalence* ($\equiv$) is the smallest congruence induced by the axioms in Figure 1. The behaviour of a process P is then described as a relation over processes up to $\equiv$, obtained by closing a set of rules under structural congruence.

Definition 2 (reduction semantics). *The* reduction relation *for processes is the relation $R_A \subseteq Proc \times Proc$ inductively defined by the following set of rules*

$$a.P + M \mid \bar{a} \to P \qquad \tau.P + M \to P \qquad !_a.P \mid \bar{a} \to P \mid !_a.P$$

$$P \mid Q = Q \mid P \qquad P \mid (Q \mid R) = (P \mid Q) \mid R \qquad P \mid \mathtt{0} = P$$
$$M{+}N = N{+}M \qquad M{+}(N{+}O) = (M{+}N){+}O \qquad M{+}\mathtt{0} = M \qquad N{+}N = N$$
$$(\nu a)(\nu b)P = (\nu b)(\nu a)P \qquad (\nu a)(P \mid Q) = P \mid (\nu a)Q \quad \text{for } a \notin fn(P) \qquad (\nu a)\mathtt{0} = \mathtt{0}$$
$$(\nu a)(M + \mu.P) = M + \mu.(\nu a)P \quad \text{for } a \notin fn(M + \mu.\mathtt{0})$$

Fig. 1. The set of structural axioms.

$$\frac{P \to Q}{(\nu a)P \to (\nu a)Q} \qquad \frac{P \to Q}{P \mid R \to Q \mid R}$$

and closed under $\equiv$*, where* $P \to Q$ *means that* $\langle P, Q\rangle \in R_A$*. As usual, we let* $\Rightarrow$ *denote the reflexive and transitive closure of* $\to$*.*

The first rule denotes the reception of a message, possibly occurring inside a non-deterministic context: the process $a.P$ is ready to receive a message along channel a; it then receives message $\bar{a}$ and proceeds as P. The second rule represents an internal computation, while on the third rule the replication of a process P occurs after a message is received on a. The latter rules state the closure of the reduction relation w.r.t. the operators of restriction and parallel composition.

A difference w.r.t. the standard syntax of the asynchronous calculus proposed in [6] is the use of guarded input replication $!_a.P$ instead of pure replication $!M$ (see, e.g., [35] which shows that for the synchronous π-calculus, this restriction does not affect the expressiveness of the calculus). Since we plan to later use our encoding to study the concurrency properties of asynchronous interactions, this choice appears more reasonable. Indeed, unguarded replication has an (unrealistic) infinitely branching behaviour when considering concurrent semantics: just think of process $!\tau.\bar{a}$. We also remark that, at the price of a slight complication of the presentation, the results in the paper could be easily extended to a calculus with replication for guarded sums, i.e., allowing for terms of the kind $!\Sigma_{i\in I}a_i.P_i$.

As for the structural axioms, we added sum idempotency and an axiom schema for distributing the restriction under the sum: neither of them is changing the reduction relation, whilst they simplify our encoding of processes into nets.

2.1 Behavioural equivalences

The main difference with the synchronous calculus lies in the notion of *observation*. Since sending messages is non-blocking, an external observer can just send messages to a system without knowing if they will be received or not. For this reason receiving should not be observable and thus *barbs*, i.e., basic observations on processes, take into account only outputs.

Definition 3 (barb). *Let* P *be a process. We say that* P satisfies the strong barb $\bar{a}$*, denoted* $P \downarrow \bar{a}$*, if there exists a process* Q *such that* $P \equiv \bar{a} \mid Q$*.*

Similarly, P satisfies the weak barb $\bar{a}$*, denoted* $P \Downarrow \bar{a}$*, if* $P \Rightarrow Q$ *and* $Q \downarrow \bar{a}$*.*

Now, strong and weak barbed bisimulation can be defined as in the synchronous case [20], but taking into account only output barbs.

Definition 4 (barbed bisimulation). *A symmetric relation $R \subseteq Proc \times Proc$ is a strong barbed bisimulation if whenever $(P, Q) \in R$ then*

1. *if $P \downarrow \bar{a}$ then $Q \downarrow \bar{a}$,*
2. *if $P \to P'$ then $Q \to Q'$ and $(P', Q') \in R$.*

Strong barbed bisimilarity $\sim$ is the largest strong barbed bisimulation.

Weak barbed bisimulation and weak barbed bisimilarity $\approx$ are defined analogously by replacing $\downarrow \bar{a}$ with $\Downarrow \bar{a}$ and $\to$ with $\Rightarrow$.

Strong (weak) barbed bisimilarities are not congruences. Indeed, $a.\bar{b} \sim \mathtt{0}$ (and $a.\bar{b} \approx \mathtt{0}$), since neither process can perform any transition, but when inserted into the context $- \mid \bar{a}$, the former can perform a transition, while the latter cannot. Behavioural equivalences which are congruences are obtained as follows.

Definition 5 (barbed equivalence). *Let P, Q be processes. They are* strongly barbed equivalent*, denoted $P \sim_b Q$, if $P \mid S \sim Q \mid S$ for all processes S.*

Similarly, they are weakly barbed equivalent*, denoted $P \approx_b Q$, if $P \mid S \approx Q \mid S$ for all processes S.*

An alternative characterization of barbed equivalence considers *output transitions* and the closure w.r.t. the parallel composition with outputs in the bisimulation game. *Strong* and *weak output transitions* are defined as follows: we respectively write $P \xrightarrow{\bar{a}} Q$ if $P \equiv \bar{a} \mid Q$, and $P \stackrel{\bar{a}}{\Rightarrow} Q$ if $P \Rightarrow P' \xrightarrow{\bar{a}} Q' \Rightarrow Q$.

Definition 6 (1-bisimulation). *A symmetric relation $R \subseteq Proc \times Proc$ is a strong 1-bisimulation if whenever $(P, Q) \in R$ then*

1. *$\forall a \in \mathcal{N}. (P \mid \bar{a}, Q \mid \bar{a}) \in R$,*
2. *if $P \to P'$ then $Q \to Q'$ and $(P', Q') \in R$,*
3. *if $P \xrightarrow{\bar{a}} P'$ then $Q \xrightarrow{\bar{a}} Q'$ and $(P', Q') \in R$.*

Strong 1-bisimilarity $\sim_1$ is the largest strong 1-bisimulation.

Weak 1-bisimulation and weak 1-bisimilarity $\approx_1$ are defined analogously by replacing $\to$ with $\Rightarrow$ and $\xrightarrow{\bar{a}}$ with $\stackrel{\bar{a}}{\Rightarrow}$.

Proposition 1 ([6]). *$\sim_b = \sim_1$ and $\approx_b = \approx_1$.*

Example 1. Consider the processes $P = (\nu d)(!_d.\bar{e} \mid (a.(\bar{a} \mid \bar{d} \mid d.\bar{c}) + \tau.(\bar{d} \mid d.\bar{c})))$ and $Q = (\nu d)(\tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d}))$. It is not difficult to see that $P \sim_1 Q$. First consider their internal steps: $P \to (\nu d)(!_d.\bar{e} \mid \bar{d} \mid d.\bar{c})$, while $Q \to (\nu d)(d.\bar{c} \mid d.\bar{e} \mid \bar{d})$. Notice that $(\nu d)(!_d.\bar{e} \mid \bar{d} \mid d.\bar{c}) \sim_1 (\nu d)(d.\bar{c} \mid d.\bar{e} \mid \bar{d})$, since after one execution the replication operator is stuck.

The process P can also receive on the channel a. Instead of observing the input transitions $\xrightarrow{a}$, in the 1-bisimulation game this behaviour is revealed plugging the processes into $- \mid \bar{a}$. The process $P \mid \bar{a}$ can choose one of the two branches of $+$, but in any case it performs an internal transition becoming

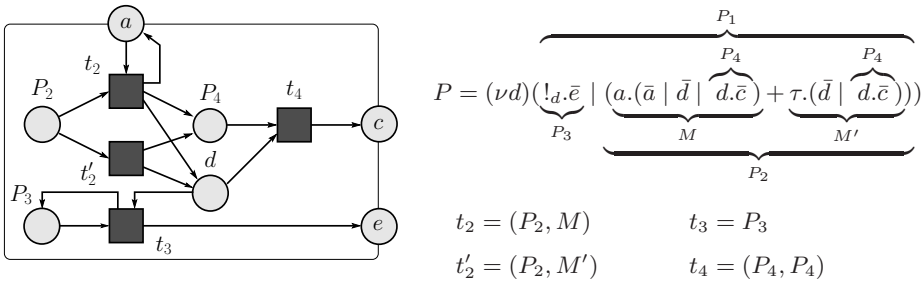

Fig. 2. An open net encoding a process P.

$(\nu d)(!_d.\bar{e} \mid \bar{a} \mid \bar{d} \mid d.\bar{c})$. On the other hand, $Q \mid \bar{a}$ performs an internal transition to $(\nu d)(d.\bar{c} \mid d.\bar{e} \mid \bar{d} \mid \bar{a})$, and clearly the resulting states are 1-bisimilar.

Furthermore, consider the process $a.\bar{a}$: it is one of the idiosyncratic features of the asynchronous communication that the equivalence $a.\bar{a} \approx_1 \mathtt{0}$ holds.

3 Open nets

Differently from process calculi, standard Petri nets do not exhibit an interactive behaviour, i.e., they are intended to model concurrent systems considered as *standalone*. This section reviews *open nets*, i.e., ordinary P/T nets with a distinguished set of *open places*: they represent the interfaces through which the environment interacts with a net, by putting and removing tokens (see [21–23]).[4]

Given a set X, let $X^{\oplus}$ denote the free commutative monoid over X. An element $m \in X^{\oplus}$, called a *multiset* over X, is often viewed as a function from X to $\mathbb{N}$ (the set of natural numbers) that associates a multiplicity with every element of X. We write $m_1 \subseteq m_2$ if $\forall x \in X$, $m_1(x) \leq m_2(x)$. If $m_1 \subseteq m_2$, the multiset $m_2 \ominus m_1$ is defined as $\forall x \in X\ m_2 \ominus m_1(x) = m_2(x) - m_1(x)$. The symbol 0 denotes the empty multiset.

Definition 7 (Open P/T Petri net). *An* open (P/T Petri) net *is a tuple* $N = (S, T, {}^\bullet(.), (.)^\bullet, O)$ *where* S *is the set of places,* T *is the set of transitions,* ${}^\bullet(.), (.)^\bullet : T \to S^{\oplus}$ *are functions mapping each transition to its pre- and post-set, and* $O \subseteq S$ *is the set of* open places.

Example 2. Figure 2 shows an open net: as usual, circles represent places and rectangles transitions. Arrows from places to transitions represent function ${}^\bullet(.)$, while arrows from transitions to places represent $(.)^\bullet$. An open net is enclosed in a box and open places are on the border of such a box. Additionally, any open place has a name which is placed inside the corresponding circle: in this example these are chosen from the set $\mathcal{N}$. Also transitions and closed places are provided

[4] Differently from [21], yet with no loss of generality, we do not distinguish between input and output open places, tailoring equivalences accordingly.

$$(\text{TR})\ \frac{t \in T}{{}^\bullet t \oplus c \xrightarrow{\tau} t^\bullet \oplus c} \qquad (\text{IN})\ \frac{s \in O}{m \xrightarrow{s^+} m \oplus s} \qquad (\text{OUT})\ \frac{s \in O}{m \xrightarrow{s^-} m \ominus s}$$

Table 1. Operational Semantics of open nets.

with an identifier, yet positioned outside of the corresponding circle or square: their precise meaning, as well as an explanation of the process on the right, is provided later on. The open place identified by a and the closed place identified by P_2 form e.g. the pre-set of transition t_2; its post-set is formed by a, P_4 and d.

Given an open net N, we consider the set of *interactions* (ranged over by i) $\mathcal{I}_N = \{s^+, s^- \mid s \in O\}$. The set of *labels* (ranged over by l) consists in $\{\tau\} \uplus \mathcal{I}_N$. The operational semantics of open nets is expressed by the rules on Table 1, where we write ${}^\bullet t$ and $t^\bullet$ instead of ${}^\bullet(t)$ and $(t)^\bullet$. The rule (TR) is the standard rule of P/T nets (seen as multiset rewriting) modelling internal transitions. The other two rules model interactions with the environment: in any moment a token can be inserted in (rule (IN)) or removed from (rule (OUT)) an open place.

Weak transitions are defined as usual, i.e., $\stackrel{\tau}{\Rightarrow}$ denotes the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$ and $\stackrel{i}{\Rightarrow}$ denotes $\stackrel{\tau}{\Rightarrow}\stackrel{i}{\longrightarrow}\stackrel{\tau}{\Rightarrow}$.

Definition 8 (Strong and weak bisimilarity). *Let N_1, N_2 be open nets with the same interface i.e., $O_1 = O_2$. A* strong bisimulation *between N_1 and N_2 is a relation over markings $\mathcal{R} \subseteq S_1^\oplus \times S_2^\oplus$ such that if $(m_1, m_2) \in \mathcal{R}$ then*

- *if $m_1 \xrightarrow{l} m_1'$ in N_1 then $m_2 \xrightarrow{l} m_2'$ in N_2 and $(m_1', m_2') \in \mathcal{R}$;*
- *if $m_2 \xrightarrow{l} m_2'$ in N_2 then $m_1 \xrightarrow{l} m_1'$ in N_1 and $(m_1', m_2') \in \mathcal{R}$.*

Two markings $m_1 \in S_1^\oplus$ and $m_2 \in S_2^\oplus$ are bisimilar, written $m_1 \sim m_2$, if $(m_1, m_2) \in \mathcal{R}$ for some strong bisimulation $\mathcal{R}$.

Weak bisimilarity $\approx$ *is defined analogously by replacing strong transitions $\xrightarrow{l}$ by weak transitions $\stackrel{l}{\Rightarrow}$.*

In order to ease the intuition behind net bisimilarity, nets must be thought of as black boxes, where only the interfaces (i.e., the open places) are visible. Two nets are weak bisimilar if they cannot be distinguished by an external observer that may only insert and remove tokens in open places. For strong bisimilarity the observer can also see the occurrence of internal transitions.

4 From processes to nets

Our encoding is restricted to *bound processes*, i.e., processes where restrictions never occur under replications. Any bound process is structurally equivalent to a process of the shape $(\nu X)P$, for P a restriction-free process and $X \subseteq fn(P)$. In the following, we implicitly assume that bound processes are always in this

$$
\begin{array}{lll}
Pl(P_1 \mid P_2) = Pl(P_1) \cup Pl(P_2) & & Pl(\bar{a}) = \{a\} \\
Pl(!_a.P) = \{a\} \cup \{[!_a.P]\} \cup Pl(P) & & Pl_s(a.P) = \{a\} \cup Pl(P) \\
Pl(M) = \begin{cases} \emptyset & \text{if } M \equiv 0 \\ \{[M]\} \cup Pl_s(M) & \text{otherwise} \end{cases} & & Pl_s(\tau.P) = Pl(P) \\
 & & Pl_s(M_1 + M_2) = Pl_s(M_1) \cup Pl_s(M_2)
\end{array}
$$

Fig. 3. The place functions.

shape. Moreover, we call *basic processes* those bound processes described by the following syntax (where P must be restriction-free)

$$B ::= \bar{a},\ \ !_a.P,\ \ M$$

Observe that any restriction-free process is just the parallel composition of several basic processes. In our net encoding, basic processes become places, marked with a number of tokens equal to the number of occurrences of the corresponding basic process in the parallel composition. Hereafter, we use $[P]$ to denote the equivalence class of process P w.r.t. structural equivalence.

Definition 9 (places). *The (mutually recursive) functions $Pl(P)$ and $Pl_s(M)$, associating to each restriction-free process and summation, respectively, a set of places, are defined by structural induction according to the rules in Figure 3.*

The set of places associated with a restriction-free process P thus consists in the set of names of P together with the set of equivalence classes of basic sub-processes of P of shape $!_a.P$ and M, for $M \not\equiv 0$. Notice that a basic process $\bar{a}$ is simply encoded by a token in the place corresponding to name a. We use $\mathbf{m}(P)$ to denote the *marking* associated with a restriction-free process P, which, assuming $\mathbf{m}(0)$ to be the empty marking, is defined by

$$\mathbf{m}(P_1 \mid P_2) = \mathbf{m}(P_1) \oplus \mathbf{m}(P_2) \qquad \mathbf{m}(\bar{a}) = a \qquad \mathbf{m}(M) = [M] \qquad \mathbf{m}(!_a.P) = [!_a.P]$$

Definition 10 (transitions, pre- and post-conditions). *Let M be a summation. The set of atoms of M, denoted $Atom(M)$, is inductively defined as:*

$$Atom(0) = \emptyset,\ Atom(\mu.P) = \{[\mu.P]\},\ Atom(M_1 + M_2) = Atom(M_1) \cup Atom(M_2)$$

for $\mu \in \mathcal{N} \cup \{\tau\}$. Now, let P be a restriction-free process. The set of transitions of P, denoted $T(P)$, is inductively defined as:

$$
\begin{array}{cc}
T(0) = \emptyset & T(P_1 \mid P_2) = T(P_1) \cup T(P_2) \\
T(\bar{a}) = \emptyset & T(!_a.P) = \{[!_a.P]\} \cup T(P) \\
\multicolumn{2}{c}{T(M) = (\{[M]\} \times Atom(M)) \cup \bigcup_{[\mu.P] \in Atom(M)} T(P)}
\end{array}
$$

The pre- and post-conditions ${}^\bullet(.)_P, (.)^\bullet_P : T(P) \to Pl(P)^\oplus$ are defined as follows.

$${}^\bullet t_P = \begin{cases} \{[M]\} \oplus fn(\mu.0) & \text{if } t = \langle [M], [\mu.P] \rangle \\ \{[!_a.P]\} \oplus \{a\} & \text{if } t = [!_a.P] \end{cases} \qquad t^\bullet_P = \begin{cases} \mathbf{m}(P) & \text{if } t = \langle [M], [\mu.P] \rangle \\ \{[!_a.P]\} \oplus \mathbf{m}(P) & \text{if } t = [!_a.P] \end{cases}$$

Intuitively, transitions mimic the control flow of a process, passing the token between its sequential components (its basic processes).

We next introduce the net encoding for bound processes. In order to get a full correspondence between behavioural equivalences, we need to encode our processes parametrically w.r.t. a set of names Γ, as usually necessary in graphical encodings of process calculi (based e.g. on DPO rewriting [36] or bigraphs [37]).

Definition 11. *Let P be a restriction-free process and X, Γ disjoint sets of names such that $fn((\nu X)P) \subseteq \Gamma$. Then, the open net $[\![(\nu X)P]\!]_\Gamma$ is the tuple*

$$(Pl(P) \cup \Gamma, T(P), {}^\bullet(.)_P, (.)^\bullet_P, \Gamma)$$

First we take the net associated with P, consisting of those places and transitions as given in Definitions 9 and 10. Then the set of places is extended with those names belonging to Γ (the assumption $\Gamma \cap X = \emptyset$ avoids that by chance a restricted name in X is overlapped). Finally, we take as open those places corresponding to a name in Γ (by hypothesis this includes the free names of $(\nu X)P$).

Example 3. The net in Example 2 is the encoding $[\![P]\!]_\Gamma$ of the process P in Example 1, with $\Gamma = \{a, c, e\}$. The places identified by a, c, and e correspond to the free names of the process, while place d corresponds to the restricted name d. The remaining places correspond to (equivalence classes of) the basic sub-processes of P: for example, the place P_2 corresponds to the sub-process $[a.(\bar{a} \mid \bar{d} \mid d.\bar{c}) + \tau.(\bar{d} \mid d.\bar{c})]$, while the place P_4 to $[d.\bar{c}]$. The transition t_2 encodes the pair (P_2, M), for M the atom $[a.(\bar{a} \mid \bar{d} \mid d.\bar{c})]$. It may fire in presence of a token in a and P_2: it roughly represents the reduction $P_2 \mid \bar{a} \to \bar{a} \mid \bar{d} \mid d.\bar{c}$.

We close this section by establishing a first correspondence result.

Proposition 2. *Let P, Q be bound processes and Γ a set of names such that $fn(P) \cup fn(Q) \subseteq \Gamma$. Then $P \equiv Q$ iff the open nets $[\![P]\!]_\Gamma$ and $[\![Q]\!]_\Gamma$ are isomorphic.*

4.1 Relating asynchronous CCS and open nets

This section shows that our encoding preserves and reflects process reductions, as well as strong and weak behavioural equivalences. In order to state these results, we must define a correspondence between the set of processes reachable from P, hereafter denoted by $reach(P)$, and markings over the net $[\![P]\!]_\Gamma$. For this, we need a technical lemma concerning reductions for bound processes.

Lemma 1. *Let P be a restriction-free process and X a set of names. Then,*

1. *$(\nu X)P \to Q$ iff $Q \equiv (\nu X)Q_1$ and $P \to Q_1$;*
2. *if $P \to Q$ then $Q \equiv B_1 \mid \ldots \mid B_n$, for B_i's basic sub-processes of P.*

The above lemma tells us that each process Q reachable from a bound process $(\nu X)P$ can be seen as a (possibly empty) marking over the net $[\![(\nu X)P]\!]_\Gamma$. In fact, the set of places of $[\![(\nu X)P]\!]_\Gamma$ contains all the basic sub-processes of P.

Definition 12. *Let P be a bound process and X, Γ disjoint sets of names such that $P \equiv (\nu X)P_1$ (for P_1 restriction-free) and $fn(P) \subseteq \Gamma$. The function $\mathbf{m}_\Gamma^{X,P_1} : reach(P) \to (Pl(P_1) \cup \Gamma)^\oplus$ maps any process $Q \equiv (\nu X)Q_1$ (for Q_1 restriction-free) reachable from P into the marking $\mathbf{m}(Q_1)$ over the net $[\![P]\!]_\Gamma$.*

Example 4. Recall $P = (\nu d)P_1$, for $P_1 = !_d.\bar{e} \mid (a.(\bar{a} \mid \bar{d} \mid d.\bar{c}) + \tau.(\bar{d} \mid d.\bar{c}))$, from Example 1. Let $X = \{d\}$ and $\Gamma = \{a, c, e\}$. The function $\mathbf{m}_\Gamma^{X,P_1}$ maps the processes reachable from P into markings of $[\![P]\!]_\Gamma$, that is, the net in Figure 2. E.g., P is mapped to $P_2 \oplus P_3$; $(\nu d)(!_d.\bar{e} \mid \bar{d} \mid d.\bar{c})$ is mapped to $P_3 \oplus d \oplus P_4$; $(\nu d)(!_d.\bar{e} \mid \bar{c})$ is mapped to $P_3 \oplus c$ and $(\nu d)(\bar{e} \mid d.\bar{c})$ to $e \oplus P_4$.

Once established that any process reachable from a bound process P identifies a marking in the net $[\![P]\!]_\Gamma$, we can state the main correspondence results.

Theorem 1. *Let P be a bound process, and X, Γ disjoint sets of names such that $P \equiv (\nu X)P_1$ (for P_1 restriction-free) and $fn(P) \subseteq \Gamma$. Moreover, let Q be a process reachable from P. Then,*

1. *if $Q \to R$ then $\mathbf{m}_\Gamma^{X,P_1}(Q) \xrightarrow{\tau} \mathbf{m}_\Gamma^{X,P_1}(R)$ in $[\![P]\!]_\Gamma$;*
2. *if $\mathbf{m}_\Gamma^{X,P_1}(Q) \xrightarrow{\tau} m$ in $[\![P]\!]_\Gamma$, then $Q \to R$ for $m = \mathbf{m}_\Gamma^{X,P_1}(R)$.*

The result establishes a bijection between the reductions performed by any process Q reachable from P, and the firings in $[\![P]\!]_\Gamma$ from the marking $\mathbf{m}_\Gamma^{X,P_1}(Q)$, for any restriction-free P_1 such that $P \equiv (\nu X)P_1$.

Such a bijection can then be lifted to a fundamental correspondence between the observational semantics in the two formalisms.

Theorem 2. *Let P, Q be bound processes and X, Y, Γ sets of names (with $X \cap \Gamma = Y \cap \Gamma = \emptyset$) such that $P \equiv (\nu X)P_1$ and $Q \equiv (\nu Y)Q_1$ (for P_1, Q_1 restriction-free) and $fn(P) \cup fn(Q) \subseteq \Gamma$. Then,*

1. *$P \sim_1 Q$ iff $\mathbf{m}_\Gamma^{X,P_1}(P) \sim \mathbf{m}_\Gamma^{Y,Q_1}(Q)$;*
2. *$P \approx_1 Q$ iff $\mathbf{m}_\Gamma^{X,P_1}(P) \approx \mathbf{m}_\Gamma^{Y,Q_1}(Q)$.*

Please note that the markings $\mathbf{m}_\Gamma^{X,P_1}(P)$ and $\mathbf{m}_\Gamma^{X,Q_1}(Q)$ live in the open nets $[\![P]\!]_\Gamma$ and $[\![Q]\!]_\Gamma$, respectively.

Example 5. Consider the net on the left of Figure 4: it corresponds to $[\![Q]\!]_\Gamma$, for $\Gamma = \{a, c, e\}$, $Q = (\nu d)Q_1$ and $Q_1 = \tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d})$ as in Example 1. Note the presence of the isolated place a: it says that name a does not occur in Q. Now, the left-most place represents the (equivalence class of the) sub-process Q_1; the left-most transition, identified by t, the pair (Q_1, Q_1); and its firing is the execution of the internal transition $Q_1 \to d.\bar{c} \mid d.\bar{e} \mid \bar{d}$, putting a token on the three places representing the restricted name d and the basic sub-processes $Q_2 = d.\bar{c}$ and $Q_3 = d.\bar{e}$.

It is easy to see that $\mathbf{m}_\Gamma^{X,P_1}(P) = P_2 \oplus P_3$ in $[\![P]\!]_\Gamma$ is strongly bisimilar to the marking $\mathbf{m}_\Gamma^{X,Q_1}(Q) = Q_1$ in $[\![Q]\!]_\Gamma$: they clearly induce the same internal behaviour; moreover when the environment inserts a token into a, $P_2 \oplus P_3 \oplus a$

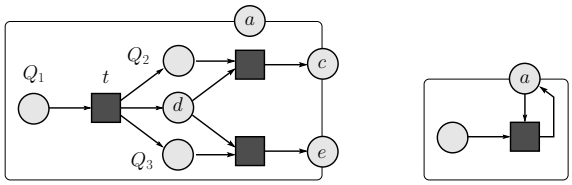

Fig. 4. The net encodings $[\![Q]\!]_\Gamma$ for $\Gamma = \{a, c, e\}$ (left) and $[\![a.\bar{a}]\!]_{\{a\}}$ (right).

may fire also transition t_2 producing $a \oplus P_4 \oplus d \oplus P_3$, but this is equivalent to the firing of the other internal transition t_2'. From this, it follows that $P \sim_b Q$.

Consider now the net in the right of Figure 4: it corresponds to the encoding of $[\![a.\bar{a}]\!]_{\{a\}}$. It is weakly bisimilar to the encoding $[\![\mathtt{0}]\!]_{\{a\}}$, represented by a net having only an open place, identified by a. In fact, in both cases the only observable action is either placing or removing a token on the open place a, while the execution of the transition $(a.\bar{a}, a.\bar{a})$, consuming and producing a, is unobservable from the environment. It thus holds that $a.\bar{a} \approx_b \mathtt{0}$.

5 Technology transfer on expressiveness

In this section we discuss some expressiveness issues for asynchronous CCS and open nets, taking advantage from the encoding presented before.

More specifically, we first show that strong and weak bisimilarity of bound processes of asynchronous CCS are undecidable, thus answering a question faced for the synchronous case in the recent [16]. By using Theorem 2, we can deduce that the same result holds for open nets, a fact which was previously unknown.

On the other hand, using the fact that reachability and convergence are decidable for Petri nets, through the encoding we can prove that the same holds for bound asynchronous CCS (which, thus, is not Turing powerful).

5.1 Undecidability of bisimilarity

The undecidability of bisimilarity for bound CCS processes is proved by reduction to the halting problem for Minsky's two-register machines, adapting a proof technique originally proposed in [29].

Definition 13 (two-register machine). *A* two-register machine *is a triple* $\langle r_1, r_2, P\rangle$ *where* r_1 *and* r_2 *are two registers which can hold any natural number, and the program* $P = I_1 \dots I_s$ *consists of a sequence of instructions. An* instruction I_i *can be one of the following: for* $x \in \{r_1, r_2\}$, $j, k \in \{1, \dots, s+1\}$

- $s(x, j)$*: increment the value of register* x *and jump to instruction* I_j
- $zd(x, j, k)$*: if* x *is zero, then jump to* I_j *else decrement* x *and jump to* I_k

The execution of the program starts from instruction I_1*, with* $r_1 = 0$, $r_2 = 0$ *and possibly terminate when the* $(s+1)$*st instruction is executed.*

The idea consists in defining, for any two-register machine program P, a bound process $\gamma(P)$ which "non-deterministically simulates" the computations of the program (starting with null registers). Some "wrongful" computations are possible in the process $\gamma(P)$, not corresponding to a correct computation of the program P (due to the absence of zero-tests in the considered fragment of CCS). Still, this can be used to prove undecidability of (strong and weak) bisimilarity. In fact, given P, a second process $\gamma'(P)$ can be built such that $\gamma(P) \sim \gamma'(P)$ iff program P does not terminate. Therefore, deciding bisimilarity for bound CCS processes would allow to decide the termination of two-register machines. As two-register machines are Turing powerful, we conclude.

Theorem 3. *Strong (weak) 1-bisimilarity of bound processes is undecidable.*

From the properties of the encoding (Theorem 2) we can deduce the same undecidability results for open nets.

Corollary 1. *Strong (weak) bisimilarity is undecidable for open nets.*

In the same way, one can easily prove that various other behavioural equivalences, including failure equivalence (the notion of equivalence considered in [16]) and language equivalence, are undecidable.

5.2 Decidability of reachability and convergence

It is immediate to see that for open nets the reachability problem, i.e., the problem of determining whether a given marking is reachable by means of a firing sequence starting from the initial marking, is decidable. In fact, reachability in an open net N can be reduced to reachability in a standard P/T net obtained from N by adding, for any open place s, two transitions t_s^- and t_s^+ which, respectively, freely remove and add tokens from s, i.e., ${}^\bullet t_s^- = t_s^{+\bullet} = s$ and $t_s^{-\bullet} = {}^\bullet t_s^+ = 0$.

By exploiting the encoding, we may transfer the result to bound asynchronous CCS, showing that reachability in an open environment, providing any needed message, is decidable.

Proposition 3. *Let P, Q be bound processes. Then the problem of establishing whether there exists an environment R, consisting of the (possibly empty) parallel composition of output messages, such that $P|R \Rightarrow Q$ is decidable.*

In particular, the problem $P \Rightarrow Q$ is decidable. In fact, if $X = fn(P) \cup fn(Q)$, it is easy to see that $P \Rightarrow Q$ iff $(\nu X)P \Rightarrow (\nu X)Q$, and this is turn equivalent to the existence of a suitable process R such that $R|(\nu X)P \Rightarrow (\nu X)Q$.

Another property which is often considered when studying the expressiveness of process calculi is convergence, i.e., the existence of a terminating computation. We recall such notion below, according to [28].

Definition 14 (convergence). *A process P is called* convergent *if there exists Q such that $P \Rightarrow Q \not\rightarrow$.*

Convergence is clearly decidable for an open net N, as it can be reduced to the existence of a deadlock in the standard P/T net obtained from N by closing all the open places, and this property is known to be decidable for P/T nets [38].

As a consequence the same holds for bound asynchronous CCS.

Proposition 4. *Convergence is decidable for bound processes.*

The paper [16] shows that, in the synchronous case, adding priorities to the language radically changes the situation: bound CCS becomes Turing complete and convergence is thus undecidable. It is easy to show that the same applies to the asynchronous case. The fact that adding priorities makes bound asynchronous CCS Turing complete can be proved by noting that using priorities the encoding of two-counter machines into bound asynchronous CCS can be made deterministic. This is not surprising as, on the Petri net side, priorities are strictly connected to inhibitor arcs, which make Petri nets Turing powerful [39].

6 Conclusions and further work

We believe that the relevance of our paper lies in establishing the fundamental correspondence between asynchronous calculi and open nets, as stated by the theorems of Section 4. Indeed, even if our presentation has been tailored over a variant of standard CCS, we feel confident that it can be generalized to other asynchronous calculi as well, at least to those based on a primitive notion of communication, i.e., without either value or name passing. As suggested by the work in [15, 30, 32], the generalisation to calculi with value or name passing looks feasible if one considers more expressive variants of Petri nets, ranging from high-level to reconfigurable/dynamic Petri nets.

We consider such a correspondence quite enlightening, since most of the encodings we are aware of focus on the preservation of some variants of reachability or of the operational behaviour [30–32, 40], while ours allow to establish a correspondence at the observational level.

As a remark, note that the encoding of CCS processes into open nets could be defined in a compositional way, either via a more syntactical presentation (thus losing the preservation of structural congruence) or by the exploiting the composition operation available for open nets. The latter would require to view open nets as cospans (with complex interfaces) in a suitable category [23, 36]. In order to keep the presentation simpler we adopted a direct definition.

We believe that the tight connection between Petri nets and asynchronous calculi allows for a fruitful "technology transfer". We started by showing the undecidability of bisimilarity for bound processes which, through the encoding, is used to prove undecidability of bisimilarity for open nets (where all internal transitions are considered indistinguishable in the strong case and unobservable in the weak case), a previously unknown fact. Analogously, decidability of reachability and convergence for open nets is transferred, through the encoding, to bound asynchronous CCS processes.

We are currently investigating the concurrent semantics for asynchronous CCS. The idea is to consider the *step* semantics for our nets, i.e., where many transitions may fire simultaneously, and then try to distill an adequate equivalence for bound process. Indeed, our initial results are quite encouraging. On the longer run, our hope would then be to lift these equivalences to richer calculi, such as the paradigmatic asynchronous π-calculus [6] and to different behavioural equivalences, including, e.g., failure and testing equivalences [7].

Acknowledgments. We are grateful to Barbara König for enlightening discussions on a preliminary version of this paper and to the anonymous reviewers for their inspiring comments.

References

1. Honda, K., Tokoro, M.: An object calculus for asynchronous communication. In: Proc. ECOOP'91. Volume 512 of LNCS. 133–147
2. G.Boudol: Asynchrony and the π-calculus. Technical Report 1702, INRIA, Sophia Antipolis (1992)
3. De Nicola, R., Ferrari, G., Pugliese, R.: KLAIM: A kernel language for agents interaction and mobility. IEEE Trans. Software Eng. **24**(5) (1998) 315–330
4. Castellani, I., Hennessy, M.: Testing theories for asynchronous languages. In: Proc. FSTTCS'98. Volume 1530 of LNCS., Springer (1998) 90–101
5. Ferrari, G., Guanciale, R., Strollo, D.: Event based service coordination over dynamic and heterogeneous networks. In: Proc. ICSOC'06. Volume 4294 of LNCS., Springer (2006) 453–458
6. Amadio, R., Castellani, I., Sangiorgi, D.: On bisimulations for the asynchronous pi-calculus. TCS **195**(2) (1998) 291–324
7. Boreale, M., De Nicola, R., Pugliese, R.: Asynchronous observations of processes. In: Proc. FOSSACS'98. Volume 1378 of LNCS., Springer (1998) 95–109
8. Boreale, M., De Nicola, R., Pugliese, R.: A theory of "may" testing for asynchronous languages. In: Proc. FOSSACS'99. Volume 1578 of LNCS., Springer (1999) 165–179
9. Rathke, J., Sobocinski, P.: Making the unobservable, unobservable. In: Proc. ICE'08. ENTCS, Elsevier (to appear).
10. Reisig, W.: Petri Nets: An Introduction. EATCS Monographs on Theoretical Computer Science. Springer (1985)
11. van der Aalst, W.: Pi calculus versus Petri nets: Let us eat "humble pie" rather than further inflate the "Pi hype". BPTrends **3**(5) (2005) 1–11
12. Goltz, U.: CCS and Petri nets. In: Semantics of Systems of Concurrent Processes. Volume 469 of LNCS., Springer (1990) 334–357
13. Gorrieri, R., Montanari, U.: SCONE: A simple calculus of nets. In: Proc. CONCUR'90. Volume 458 of LNCS., Springer (1990) 2–31
14. Busi, N., Gorrieri, R.: A Petri net semantics for pi-calculus. In: Proc. CONCUR'95. Volume 962 of LNCS., Springer (1995) 145–159
15. Devillers, R., Klaudel, H., Koutny, M.: A compositional Petri net translation of general pi-calculus terms. Formal Asp. Comput. **20**(4-5) (2008) 429–450
16. Aranda, J., Valencia, F., Versari, C.: On the expressive power of restriction and priorities in CCS with replication. In: Proc. FOSSACS'09. Volume 5504 of LNCS., Springer (2009) 242–256

17. Olderog, E.: Nets, terms and formulas. Cambridge University Press (1991)
18. Busi, N., Gorrieri, R.: Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs. Journal of Logic and Algebraic Programming **78** (2009) 138–162
19. Berry, G., Boudol, G.: The chemical abstract machine. TCS **96** (1992) 217–248
20. Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Proc. ICALP'92. Volume 623 of LNCS., Springer (1992) 685–695
21. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional semantics for open Petri nets based on deterministic processes. Mathematical Structures in Computer Science **15**(1) (2004) 1–35
22. Milner, R.: Bigraphs for Petri nets. In: Lectures on Concurrency and Petri Nets. Volume 3098 of LNCS., Springer (2003) 686–701
23. Sassone, V., Sobociński, P.: A congruence for Petri nets. In: Proc. PNGT'04. Volume 127 of ENTCS., Elsevier (2005) 107–120
24. Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets. Volume 625 of LNCS. Springer (1992)
25. Nielsen, M., Priese, L., Sassone, V.: Characterizing behavioural congruences for Petri nets. In: Proc. CONCUR'95. Volume 962 of LNCS., Springer (1995) 175–189
26. Koutny, M., Esparza, J., Best, E.: Operational semantics for the Petri box calculus. In: Proc. CONCUR'94. Volume 836 of LNCS., Springer (1994) 210–225
27. Kindler, E.: A compositional partial order semantics for Petri net components. In: Proc. ICATPN'97. Volume 1248 of LNCS., Springer (1997) 235–252
28. Busi, N., Gabbrielli, M., Zavattaro, G.: Comparing recursion, replication, and iteration in process calculi. In: Proc. ICALP'04. Volume 3142 of LNCS., Springer (2004) 307–319
29. Jancar, P.: Undecidability of bisimilarity for Petri nets and some related problems. TCS **148**(2) (1995) 281–301
30. Buscemi, M., Sassone, V.: High-level Petri nets as type theories in the join calculus. In: Proc. FOSSACS'01. Volume 2030 of LNCS., Springer (2001) 104–120
31. Busi, N., Zavattaro, G.: A process algebraic view of shared dataspace coordination. J. Log. Algebr. Program. **75**(1) (2008) 52–85
32. Devillers, R., Klaudel, H., Koutny, M.: A Petri net semantics of a simple process algebra for mobility. In: Proc. EXPRESS'05. Volume 154.3. (2006) 71–94
33. Meyer, R., Khomenko, V., Strazny, T.: A practical approach to verification of mobile systems using net unfoldings. In: Proc. Petri Nets 2008. Volume 5062 of LNCS., Springer (2008) 327–347
34. Milner, R.: A Calculus of Communicating Systems. Volume 92 of LNCS. Springer (1980)
35. Sangiorgi, D.: On the bisimulation proof method. Mathematical Structures in Computer Science **8**(5) (1998) 447–479
36. Gadducci, F.: Graph rewriting and the π-calculus. Mathematical Structures in Computer Science **17** (2007) 1–31
37. Milner, R.: Pure bigraphs: Structure and dynamics. Information and Computation **204** (2006) 60–122
38. Esparza, J., Nielsen, M.: Decidability issues for Petri nets - a survey. Journal Inform. Process. Cybernet. EIK **30**(3) (1994) 143–160
39. Agerwala, T., Flynn, M.: Comments on capabilities, limitations and "correctness" of Petri nets. Computer Architecture News **4**(2) (1973) 81–86
40. Busi, N., Zavattaro, G.: Expired data collection in shared dataspaces. TCS **3**(298) (2003) 529–556