# Asynchronous Traces and Open Petri Nets

Paolo Baldan[1], Filippo Bonchi[2], Fabio Gadducci[3](✉), and Giacoma V. Monreale[3]

[1] Dipartimento di Matematica, Università di Padova, Padova, Italy
[2] ENS Lyon, Université de Lyon, LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA), Lyon, France
[3] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
gadducci@di.unipi.its

**Abstract.** The relation between process calculi and Petri nets, two fundamental models of concurrency, has been widely investigated. Many proposals exist for encoding process calculi into Petri nets while preserving some behavioural features of interest. We recently introduced a framework where a net encoding can be defined uniformly for calculi with different communication patterns, including synchronous two-party, multi-party, and asynchronous communication. The encoding preserves and reflects several behavioural semantics, notably bisimulation equivalence. The situation is less immediate for asynchronous calculi and trace semantics: considering traces that arise when viewing asynchronous calculi as a fragment of the synchronous ones, trace equivalence is not reflected by the encoding. Focusing on CCS, we argue that this phenomenon is related to the imperfect match between trace inclusion and may testing preorder. We consider an alternative labelled transition systems where the latter issue is solved, and we show that, indeed, the corresponding trace semantics is preserved and reflected by the net encoding.

**Keywords:** Asynchronous CCS · (Open) Petri nets · Modular encoding · May testing · Trace semantics

*"Ci sono più reti di Petri in terra di quanti baci abbia dato Catullo."*
*"There are more Petri nets in earth than kisses given by Catullo"*

PD, circa 1989

## 1 Introduction

The theory of concurrency and distribution contains several studies on the relation between two fundamental models, process calculi and Petri nets. In particular, Petri nets have been used as the target for the encoding of many process

Research partly supported by the MIUR PRIN 2010LHT4KM CINA, the ANR 121S02001 PACE and the University of Padua ANCORE.

C. Bodei et al. (Eds.): Degano Festschrift, LNCS 9465, pp. 86–102, 2015.
DOI: 10.1007/978-3-319-25527-9_8

calculi (and other textual formalisms). On the one hand, thanks to the simple and immediate visual presentation of nets, a suitable encoding can clarify the nature of concurrency and distribution in the formalism at hand. At the same time, it can highlight if and how the different synchronisation mechanisms can be represented in the net setting. On the other hand, the availability of many tools and techniques for the analysis of net behavioural properties, like reachability, boundedness, and deadlock-freedom, suggests that suitable encodings might offer the possibility of a fruitful technology transfer. Indeed, there has been since a long time an interest for the net encoding of calculi. Special attention has been devoted to CCS. There are several papers which show how the handshaking communication pattern of CCS (and $\pi$-calculus) can be implemented in the Petri net setting in such a way that the operational behaviour of a process is (at least) preserved by the encoding [15–17,26]. Pierpaolo was one of the initiators of this line of research [12,13], also devoting some attention [14] to a less explored paradigm, the multi-party communication pattern of e.g. CSP [18].

Most of those works exploit C/E systems, and are wired towards synchronous communication patterns. In recent works [2,3] we showed how resorting to the P/T paradigm, these ideas can be generalised in order to include asynchronous communication. This has been exemplified in the asynchronous CCS (ACCS) [7]. The encodings rely on *open nets* [4,8,22,24], a reactive extensions of the ordinary net model equipped with *open places* and *visible transitions*, i.e., distinguished sets of places and transitions which are accessible to the environment: a net may then interact with its environment either asynchronously, by exchanging tokens on open places, or by synchronising on visible transitions. We identified fragments of CSP and ACCS, hereafter referred to as *bound*, which can be mapped *in a modular way* into Petri nets via encodings that preserve as well as reflect the standard operational semantics of the two calculi. Modularity here means that we identify suitable operators on nets which exactly correspond to operators on processes, such that the encoding is built inductively from a set of basic net constants, and at the same time it preserves structural congruence. The term bound refers to limitations that are imposed to the use of recursion/replication which will be made precise later. The fragments are not Turing powerful (e.g., reachability is decidable), but expressive enough to model infinite state systems where standard behavioural equivalences (barbed bisimilarity for ACCS and trace equivalence for CSP) are undecidable.

Since most behavioural semantics for process calculi are based on their transition system, this correspondence at the operational level translates to a correspondence between virtually any observational equivalence.

The situation is less clearly cut with trace semantics for ACCS, which is in fact paradigmatic of a general problem of labelled operational semantics for asynchronous calculi. This is witnessed by the relationship of may testing with trace semantics, as explored in [7,11]: differently from the synchronous case, trace inclusion does not correspond directly to the may preorder, but some adjustment (working modulo some preorder on traces) is needed in order to take into account the unobservability of message reception. This fact also causes a mismatch with the notion of trace for open Petri nets, which instead directly describes all the possible interactions of a system with its environment. Indeed, it can be easily observed that labelled transitions, and thus trace inclusion, are only preserved but not reflected by our net encoding of ACCS processes. The problem can be solved by resorting to a different LTS, that we call saturated LTS, proposed in [11] (in turn inspired by [19]). The saturated LTS induces a notion of trace that directly captures all the possible interactions with the environment and has an immediate correspondence with the may preorder, namely the may preorder and trace inclusion coincide. For these reasons it fits nicely with the aforementioned encoding into open nets: the operational semantics via the saturated LTS is now preserved and reflected, and thus also trace semantics.

Among other technology transfers, our work opens the way to the study of testing semantics for Petri nets, so far scarcely investigated in the literature [20].

*Synopsis.* The paper is structured as follows. In Sect. 2 we recall the syntax, operational semantics and may testing theory of ACCS. In Sect. 3 we present the saturated LTS for ACCS, and we show that the corresponding notion of trace semantics exactly corresponds to may preorder. In Sect. 4 we describe open Petri nets with interfaces, and we define the modular encoding of (bound) ACCS processes into open nets. Finally, in Sect. 5 we prove that the net encoding of ACCS preserves and reflects saturated trace semantics. In Sect. 6 we then draw some conclusions and provide pointers to future works.

## 2 Asynchronous CCS

Asynchronous process calculi are characterised by the fact that message sending and reception are not synchronised. Rather, messages are sent and travel through some media until they reach destination. Thus sending is non-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). One can think that output messages are buffered [25] or stored in some shared workspace [9].

Asynchronous $\pi$-calculus was originally introduced in [1,19]. Here we consider a restriction – not featuring name passing – called asynchronous CCS (ACCS) [7, 11]. Besides the absence of name passing, the main difference with respect to the syntax of the calculus in [1] is the presence of a guarded input replication $!_a.P$, instead of the pure replication of a summation. Indeed, unguarded replication can have (unrealistic) infinitely branching behaviour, especially when considering a concurrent semantics. Just think of process $!\tau.\bar{a}$, which can concurrently generate an unbounded number of messages on channel $a$.

**Definition 1 (ACCS processes).** *Let $\mathcal{N}$ be a set of* names*, ranged over by $a, b, c, \ldots$ and let $\tau \notin \mathcal{N}$ be the silent action. We let $\gamma, \gamma_1, \ldots$ range over the set of* guards *$\mathcal{N} \cup \{\tau\}$, $\upsilon, \upsilon_1, \ldots$ over the set of* visible *actions $\mathcal{V} = \mathcal{N} \cup \overline{\mathcal{N}}$, and $\mu, \mu_1, \ldots$ over the set of all actions $\mathcal{A} = \mathcal{V} \cup \{\tau\}$. A* process *is a term generated by the syntax in Fig. 1. We let $P, Q, R, \ldots$ range over the set of processes $\mathcal{P}$.*

| | | | |
|---|---|---|---|
| $P$ | $::=$ | $0$ | inactive process |
| | | $\oplus_{i=1}^{n}\gamma_i.P_i$ | summation |
| | | $\bar{a}$ | output |
| | | $P \mid Q$ | parallel |
| | | $(\nu a)P$ | restriction |
| | | $!_a.P$ | replication |

**Fig. 1.** ACCS processes.

The main difference with standard CCS is the absence of output prefixes. The occurrence of an unguarded $\bar{a}$ indicates a message that is available on some communication media named $a$. It will disappear whenever it is received.

We assume the standard definition for the set of free names of a process $P$, which is denoted by $\texttt{fn}(P)$. Similarly, we assume that $\alpha$-convertibility holds with respect to the *restriction* operators $(\nu a)P$: the name $a$ is restricted in $P$, and thus it can be freely $\alpha$-converted.

$$(\mathsf{Alt})\ \frac{\rho \text{ permutation}}{\oplus_{i=1}^{n}\gamma_i.P_i = \oplus_{i=1}^{n}\gamma_{\rho(i)}.P_{\rho(i)}}$$

$$(\mathsf{Par}_1)\frac{}{P \mid Q = Q \mid P} \qquad (\mathsf{Par}_2)\frac{}{P \mid (Q \mid R) = (P \mid Q) \mid R}$$

$$(\mathsf{Res}_1)\frac{X \cap fn(P) = \emptyset}{(\nu X)P = P} \qquad (\mathsf{Res}_2)\frac{X \cap fn(C[0]) = \emptyset}{C[(\nu X)P] = (\nu X)C[P]}$$

**Fig. 2.** ACCS structural axioms: $C[-]$ is a process context with no occurrence of $!_a.-$.

*Structural equivalence* ($\equiv$) is the smallest congruence induced by the axioms in Fig. 2, where $C[-]$ denotes a process context such that the "hole" $-$ does not occur inside the scope of a replication $!_a$. With respect to [1] we added an axiom schema for distributing the restriction under each operator different from replication, thus also under the sum and the prefix.

The operational rules in Fig. 3, taken from [7], arise as a direct rephrasing of the rules of synchronous CCS restricted to the asynchronous fragment (whence the subscript "s"). The behaviour of a process $P$ is then described as a relation over processes up to $\equiv$, obtained by closing the rules under structural congruence.

**Definition 2 (Labeled semantics).** *The* labelled transition system *for ACCS processes is the relation $S \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ inductively defined by the set of rules in Fig. 3, where $P \xrightarrow{\mu}_s Q$ means that $\langle P, \mu, Q\rangle \in S$. Weak transitions $P \overset{w}{\Rightarrow}_s Q$ are defined by the following rules, where $w \in \mathcal{V}^*$ and $\epsilon$ denotes the empty trace.*

$$\frac{P (\xrightarrow{\tau}_s)^{\star} Q}{P \overset{\epsilon}{\Rightarrow}_s Q} \qquad \frac{P \xrightarrow{v}_s Q}{P \overset{v}{\Rightarrow}_s Q} \qquad \frac{P \overset{w_1}{\Rightarrow}_s Q \overset{w_2}{\Rightarrow}_s R}{P \overset{w_1 w_2}{\Rightarrow}_s R}$$

*We write $P \overset{w}{\Rightarrow}_s$ if there exists some $Q$ such that $P \overset{w}{\Rightarrow}_s Q$ and we define the set of traces of a process $P$ as $traces_s(P) = \{w \mid P \overset{w}{\Rightarrow}_s\}$.*

$$(\mathsf{Act})\ \frac{j \in \{1,\dots,n\}}{\oplus_{i=1}^{n} \gamma_i.P_i \xrightarrow{\gamma_j}_s P_j} \qquad (\mathsf{Repl})\frac{}{!_a.P \xrightarrow{a}_s\ !_a.P \mid P}$$

$$(\mathsf{Par})\ \frac{P \xrightarrow{\mu}_s P'}{P \mid Q \xrightarrow{\mu}_s P' \mid Q} \qquad (\mathsf{Syn})\ \frac{P \xrightarrow{a}_s P',\, Q \xrightarrow{\overline{a}}_s Q'}{P \mid Q \xrightarrow{\tau}_s P' \mid Q'}$$

$$(\mathsf{Res})\ \frac{P \xrightarrow{\mu}_s P' \quad \mu \notin \{a, \overline{a}\}}{(\nu a)P \xrightarrow{\mu}_s (\nu a)P'} \qquad (\mathsf{Out})\frac{}{\overline{a} \mid P \xrightarrow{\overline{a}}_s P}$$

$$(\mathsf{Con})\frac{P \equiv P',\ P' \xrightarrow{\mu}_s Q',\ Q' \equiv Q}{P \xrightarrow{\mu}_s Q}$$

**Fig. 3.** ACCS labelled semantics.

Testing semantics equates processes that cannot be taken apart by the interaction with external observers. This is formalised via a notion of test.

**Definition 3 (May testing preorder).** *An* observer *is an ACCS process that can perform a distinguished output action $\overline{\checkmark}$ (the success action), with $\checkmark \notin \mathcal{N}$. For process $P$ and observer $O$, $P$* may *$O$ if there exists a successful computation of $P \mid O$, namely $P \mid O \overset{\overline{\checkmark}}{\Rightarrow}_s$. For processes $P$ and $Q$, we write $P \sqsubseteq_m Q$ ($P \equiv_m Q$) if $P$ may $O$ implies $Q$ may $O$ (and vice versa) for all observers $O$.*

The above definition can be (and usually is) hard to verify, since it requires to take into account all possible observers. For synchronous languages like CCS and $\pi$-calculus, this problem can be easily avoided by observing that $\sqsubseteq_m$ coincides with the standard trace inclusion. Unfortunately, this is no longer true for asynchronous calculi. For instance, it is easy to see that $a.b.P \sqsubseteq_m b.a.P$ and $a.\overline{a} \sqsubseteq_m \mathsf{0}$, but clearly neither in the former nor in the latter case the traces of the first process are included in those of the second one.

A solution is devised in [7], by relying on the following order on traces.

**Definition 4 (Trace order).** *The* trace order *for processes is the reflexive and transitive relation $\leq_A \subseteq \mathcal{V}^\star \times \mathcal{V}^\star$ inductively defined by the set of rules in Fig. 4 and closed under pre- and post-composition.*

*For processes $P$ and $Q$, we write $P \leq_m Q$ if whenever $w \in traces_s(P)$ then $w' \in traces_s(Q)$ for some $w' \leq_A w$.*

The trace order takes into account the asynchronous nature of communications. The intuition is that, given a process $P$ and a trace $s$, if $P$ may offer $\overline{s}$ (for a trace $s$, its dual $\overline{s}$ is defined in the obvious way), then it may also offer $\overline{t}$ for all

$t \leq_A s$. The inequality $\epsilon \leq_A a$ is motivated by the fact that whenever a process can exhibit a trace including an output message $\bar{a}$, it can offer the same trace where $\bar{a}$ has been removed, since output is non-blocking, hence any transition that follows can be performed independently of the output. For a quite similar reason, an output can be deferred as much as desired, whence the inequality $va \leq_A av$. Finally, if a process can emit an output on $a$ and later input on the same channel, then it can input its own message, leading to an internal move. This motivates the last inequality $\epsilon \leq_A a\bar{a}$.

$$\epsilon \leq_A a \qquad va \leq_A av \qquad \epsilon \leq_A a\bar{a}$$

**Fig. 4.** Trace ordering laws.

As shown in [7], the relevant fact concerning the relation $\leq_m$ is that t coincides with the may preorder.

**Theorem 1 (Alternative may testing).** *Let $P$, $Q$ be ACCS processes. Then $P \sqsubseteq_m Q$ iff $P \leq_m Q$.*

*Example 1.* Consider the processes $P = (\nu d)(!_d.\bar{e} \mid (a.(\bar{a} \mid \bar{d} \mid d.\bar{c}) \oplus \tau.(\bar{d} \mid d.\bar{c})))$ and $Q = (\nu d)(\tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d}))$. It is not difficult to see that $P \equiv_m Q$. For instance, consider the trace $\bar{e}$, which can be obtained in $P$ via the sequence $P \xrightarrow{\tau}_s (\nu d)(!_d.\bar{e} \mid \bar{d} \mid d.\bar{c}) \xrightarrow{\tau}_s (\nu d)(!_d.\bar{e} \mid \bar{e} \mid d.\bar{c}) \xrightarrow{\bar{e}}_s (\nu d)(!_d.\bar{e} \mid d.\bar{c})$, and the trace is terminated as the replication is stuck. For $Q$ we have the same trace via $Q \xrightarrow{\tau} (\nu d)(d.\bar{c} \mid d.\bar{e} \mid \bar{d}) \xrightarrow{\tau}_s (\nu d)(d.\bar{c} \mid \bar{e}) \xrightarrow{\bar{e}}_s (\nu d)(d.\bar{c})$.

A different execution is $P \xrightarrow{a}_s (\nu d)(!_d.\bar{e} \mid \bar{a} \mid \bar{d} \mid d.\bar{c}) \xrightarrow{\bar{a}}_s (\nu d)(!_d.\bar{e} \mid \bar{d} \mid d.\bar{c}) \xrightarrow{\tau}_s (\nu d)(!_d.\bar{e} \mid \bar{e} \mid d.\bar{c}) \xrightarrow{\bar{e}}_s (\nu d)(!_d.\bar{e} \mid d.\bar{c})$. The corresponding traces $a$, $a\bar{a}$ and $a\bar{a}\bar{e}$, can be matched in $Q$ by $\epsilon \leq_m a$, $\epsilon \leq_m a\bar{a}$ and $\bar{e} \leq_m a\bar{a}\bar{e}$.

Similar considerations lead to show that process $a.\bar{a} \equiv_m \mathsf{0}$, one of the idiosyncratic features of asynchronous communication.

## 3 May Testing via Saturated Traces

In this section we show that the may preorder can be characterised in terms of trace inclusion by resorting to traces defined on a different LTS for ACCS processes, which originates from [11], in turn similar to [19]. We will see later, in Sect. 4, that with this notion of trace there is a perfect match between trace semantics for ACCS processes and for their net encodings.

**Definition 5 (Saturated LTS).** *The* saturated *LTS for ACCS processes is the relation $R \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ inductively defined by the set of rules in Fig. 5, where $P \xrightarrow{\mu} Q$ means that $\langle P, \mu, Q\rangle \in R$. For a process $P$, weak transitions (denoted by $P \overset{w}{\Rightarrow}$) and the set of traces (denoted traces$(P)$) are defined as before.*

$$(\mathsf{Syn})\ \frac{\gamma_1 = a}{\oplus_{i=1}^{n}\gamma_i.P_i \mid \overline{a} \xrightarrow{\tau} P_1} \qquad (\mathsf{Tau})\ \frac{\gamma_1 = \tau}{\oplus_{i=1}^{n}\gamma_i.P_i \xrightarrow{\tau} P_1}$$

$$(\mathsf{Repl})\ \frac{}{!_a.P \mid \overline{a} \xrightarrow{\tau} !_a.P \mid P} \qquad (\mathsf{Par})\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}$$

$$(\mathsf{Res})\ \frac{P \xrightarrow{\mu} P' \quad \mu \notin \{a, \overline{a}\}}{(\nu a)P \xrightarrow{\mu} (\nu a)P'} \qquad (\mathsf{Con})\ \frac{P \equiv P' \quad P' \xrightarrow{\mu} Q' \quad Q' \equiv Q}{P \xrightarrow{\mu} Q}$$

$$(\mathsf{Out})\ \frac{}{\overline{a} \mid P \xrightarrow{\overline{a}} P} \qquad (\mathsf{In})\ \frac{}{P \xrightarrow{a} \overline{a} \mid P}$$

**Fig. 5.** Saturated labelled semantics.

The main novelty with respect to the previous set of rules is the presence of rule (In), stating that the environment can freely provide output messages. Dually, rule (Out) can be interpreted as the environment receiving (and thus consuming) a message. Rules (Syn) and (Repl) now model internal reductions. It is easy to see (indeed, this is the definition proposed in [11]) that $\rightarrow$ can be alternatively defined as the least relation on $\mathcal{P} \times \mathcal{A} \times \mathcal{P}$ such that

- $\xrightarrow{\mu}_s \subseteq \xrightarrow{\mu}$ and
- for all $a \in \mathcal{N}$, $P \xrightarrow{a} P \mid \overline{a}$.

The relation between the two LTSs is summarized by the following lemma.

**Lemma 1 (Non-saturated vs saturated).** *Let $P$, $Q$ be ACCS processes. Then*

1. $P \xrightarrow{\tau}_s Q$ *iff* $P \xrightarrow{\tau} Q$;
2. $P \xrightarrow{\overline{a}}_s Q$ *iff* $P \xrightarrow{\overline{a}} Q$;
3. *if* $P \xrightarrow{a}_s Q$ *then* $P \xrightarrow{a}\xrightarrow{\tau} Q$.

*Proof.* It is easy to show that for any name $a \in \mathcal{N}$, process $P$ performs an input $P \xrightarrow{a}_s Q$ iff $P \equiv (\nu a_1)\ldots(\nu a_m)(\oplus_{i=1}^{n}\gamma_i.P_i \mid P')$, with $\gamma_1 = a \neq a_k$ for $k \in \{1,\ldots,m\}$ and $Q \equiv (\nu a_1)\ldots(\nu a_m)(P_1 \mid P')$. Dually, process $P$ performs an output $P \xrightarrow{\bar{a}}_s Q$ iff $P \equiv P' \mid \bar{a}$. From these facts items (1)-(3) follow. $\square$

*Example 2.* Consider again the processes $P = (\nu d)(!_d.\bar{e} \mid (a.(\bar{a} \mid \bar{d} \mid d.\bar{c}) \oplus \tau.(\bar{d} \mid d.\bar{c})))$ and $Q = (\nu d)(\tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d}))$ from Example 1. It can be seen that $P \approx_T Q$. For instance, consider the execution $P \xrightarrow{a} (\nu d)(!_d.\bar{e} \mid (a.(\bar{a} \mid \bar{d} \mid d.\bar{c}) \oplus \tau.(\bar{d} \mid d.\bar{c}))) \mid \bar{a} \xrightarrow{\tau} (\nu d)(!_d.\bar{e} \mid \bar{a} \mid \bar{d} \mid d.\bar{c}) \xrightarrow{\tau} \bar{a}(\nu d)(!_d.\bar{e} \mid \bar{d} \mid d.\bar{c}) \xrightarrow{\tau} \tau(\nu d)(!_d.\bar{e} \mid \bar{e} \mid d.\bar{c}) \xrightarrow{\tau} \bar{e}(\nu d)(!_d.\bar{e} \mid \bar{e} \mid d.\bar{c})$ that generates the trace $a\bar{a}\bar{e}$, which in the previous LTS could only be simulated up to $\leq_m$.

In the saturated LTS we have that $Q \xrightarrow{a} (\nu d)(\tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d})) \mid \bar{a} \xrightarrow{\bar{a}} (\nu d)(\tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d})) \xrightarrow{\tau} (\nu d)(d.\bar{c} \mid d.\bar{e} \mid \bar{d}) \xrightarrow{\tau} (\nu d)(d.\bar{c} \mid \bar{e}) \xrightarrow{\bar{e}} (\nu d)(d.\bar{c} \mid \bar{e})$, which gives exactly the same trace.

We can now prove that trace inclusion in the saturated LTS is a further characterisation of the may preorder. In order to show this fact, we prove that trace inclusion in the saturated LTS coincide with $\leq_m$ in the LTS of Definition 2.

**Proposition 1 (Soundness).** *Let $P$ be an ACCS process and $w \in traces(P)$. If $w' \geq_A w$ then $w' \in traces(P)$.*

*Proof.* The proof proceeds by induction on $\leq_A$, but we need to strengthen the inductive hypothesis. For a set of ACCS processes $S$, we define its closure $\mathcal{C}(S)$ as the least set of processes such that

$$\mathcal{C}(S) = S \cup \{P \,|\, \overline{a} \ :\ P \in \mathcal{C}(S) \wedge\ a \in \mathcal{N}\}.$$

Observe that (†) $\mathcal{C}(\mathcal{C}(S)) = \mathcal{C}(S)$ and (‡) if $P \overset{w}{\Rightarrow} Q$ and $P' \in \mathcal{C}(\{P\})$, then $P' \overset{w}{\Rightarrow} Q'$ with $Q' \in \mathcal{C}(\{Q\})$. Now, the proof that

$$\text{if } P \overset{w}{\Rightarrow} Q \text{ and } w \leq_A w', \text{ then } P \overset{w'}{\Rightarrow} Q' \text{ and } Q' \in \mathcal{C}(\{Q\})$$

is easily carried out and it immediately implies our statement. □

**Proposition 2 (Completeness).** *Let $P$ be an ACCS process and $w \in traces(P)$. Then there exists $w' \leq_A w$ such that $w' \in traces_s(P)$.*

*Proof.* The statement is proved by induction on $w$. For the base case, if $w = \epsilon$, then by Lemma 1(1) $P \overset{\epsilon}{\Rightarrow} Q$ iff $P \overset{\epsilon}{\Rightarrow}_s Q$. For the inductive case, we distinguish two sub-cases according to the first action in the trace.

- If $w = \overline{a}w'$, then $P \overset{\overline{a}}{\Rightarrow} Q$ and $w' \in traces(Q)$. By Lemma 1(1–2), $P \overset{\overline{a}}{\Rightarrow}_s Q$. By induction hypothesis, there exists $w'' \leq_A w'$ such that $Q \overset{w''}{\Rightarrow}_s$. Therefore $P \overset{\overline{a}w''}{\Rightarrow}_s$ and $\overline{a}w'' \leq_A w$.
- If $w = aw'$, then $P \overset{\epsilon}{\Rightarrow} P' \xrightarrow{a} Q$ and $w' \in traces(Q)$. By definition of $\xrightarrow{a}$ we have that $Q = P' \mid \overline{a}$. Relying on the fact that the set of traces of $P \mid \overline{a}$ can be characterised as

$$traces(P) \cup \{w_1w_2 \mid w_1aw_2 \in traces(P)\} \cup \{w_1\overline{a}w_2 \mid w_1w_2 \in traces(P)\}$$

  from $w' \in traces(P' \mid \overline{a})$ we have that
  - If $w' \in traces(P')$, $w = aw' \geq_A w'$. By induction hypothesis, there exists $w'' \leq_A w'$ such that $P' \overset{w''}{\Rightarrow}_s$. Therefore $P \overset{w''}{\Rightarrow}_s$ and $w'' \leq_A w$.
  - If $w' \in \{w_1w_2 \mid w_1aw_2 \in traces(P)\}$, $w = aw' = aw_1w_2 \geq w_1aw_2 \in traces(P')$. By induction hypothesis, there exists $w'' \leq_A w_1aw_2$ such that $P' \overset{w''}{\Rightarrow}_s$. Therefore $P \overset{w''}{\Rightarrow}_s$ and $w'' \leq_A w$.
  - If $w' \in \{w_1\overline{a}w_2 \mid w_1w_2 \in traces(P)\}$, $w = aw' = aw_1\overline{a}w_2 \geq_A w_1a\overline{a}w_2 \geq_A w_1w_2 \in traces(P')$. By induction hypothesis, there exists $w'' \leq_A w_1w_2$ such that $P' \overset{w''}{\Rightarrow}_s$. Therefore $P \overset{w''}{\Rightarrow}_s$ and $w'' \leq_A w$. □

Now, the desired result immediately follows.

**Theorem 2 (May testing via traces inclusion).** *Let $P$, $Q$ be ACCS processes. Then $traces(P) \subseteq traces(Q)$ iff $P \leq_m Q$.*

The result above is similar to [11, Theorem 1], which states that $traces_s(P) \subseteq traces(Q)$ iff $P \sqsubseteq_m Q$. However, it is worth remarking that the latter theorem does not imply ours.

Indeed, we believe that our work provides some interesting, despite preliminary, insights: soundness and completeness (Propositions 1 and 2) state exactly that $traces(P)$ is the upward closure of $traces_s(P)$ with respect to the ordering $\leq_A$. The preorder $\leq_m$ is one of the standard ways to lift an ordering to its powerset, and it is well-known that such a lifting coincides with the inclusion of upward closure.

## 4 Open Petri Nets

Let $X^\oplus$ be the free commutative monoid over a set $X$ and let $2^X$ be the powerset of $X$. An element $m \in X^\oplus$ is referred to as a *multisets* over $X$, since it can be viewed as a function $m : X \to \mathbb{N}$ (the set of natural numbers) associating a multiplicity with each $x \in X$. A subset $Y \subseteq X$ is often confused with the multiset $\bigoplus_{y \in Y} y$. 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. Given $f : X \to Y$ we denote its extension to multisets by $f^\oplus : X^\oplus \to Y^\oplus$.

**Definition 6 (Petri nets).** *A* Petri net *is a tuple $N = (S, T, {}^\bullet(.), (.)^\bullet)$ where $S$ is the set of places, $T$ is the set of transitions, ${}^\bullet(.), (.)^\bullet : T \to 2^S$ are functions mapping each transition to its pre- and post-set.*

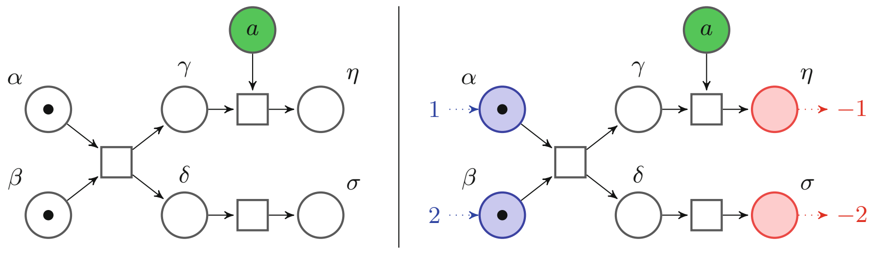

**Fig. 6.** Graphical representation of open Petri nets, on the right with interfaces.

In order to encode a process calculus into Petri nets we consider a reactive generalisation of Petri nets, in the line of [4,8,22,24]. More precisely, nets are endowed with distinguished sets of *open* places. They represent the places through which the environment interacts with the net, by putting and removing tokens visible from the environment. Open places carry a label, and hereafter we let $\mathcal{N}$ be the corresponding set of labels. The choice is driven by our need of encoding ACCS channels: messages on channels would correspond to tokens in places.

**Definition 7 (Open Petri net).** *An* open net *is a triple* $ON = \langle N, O, \lambda \rangle$, *where* $N$ *is a net,* $O \subseteq S$ *is a set of* open *places, and* $\lambda : O \to \mathcal{N}$ *is an injective* labelling *function.*

*A* marked open net $\mathbf{N}$ *is a pair* $\langle ON, m_0 \rangle$, *where* $ON$ *is an open net and* $m_0 \in S^{\oplus}$ *is the* initial *marking.*

The operational semantics of open nets is presented in Fig. 7. Rule (Step) is the standard rule of P/T nets (seen as multiset rewriting), represented as a silent action $\tau$. The remaining rules model interaction with the environment. They state that in an open place at any moment the environment can generate (In) or remove a token (Out). Note that interactions based on exchanging tokens is naturally asynchronous. For a word $w \in \mathcal{V}^{\star}$, weak transitions $m \overset{w}{\Rightarrow} m'$ are defined as in ACCS. Similarly, for the traces *traces*($\mathbf{N}$) of a marked open net $\mathbf{N}$.

$$(\mathsf{Step})\ \frac{m = {}^{\bullet}t \oplus m' \quad t \in T}{m \xrightarrow{\tau} t^{\bullet} \oplus m'} \qquad (\mathsf{In})\ \frac{s \in O}{m \xrightarrow{\lambda(s)} m \oplus s} \qquad (\mathsf{Out})\ \frac{s \in O}{m \xrightarrow{\overline{\lambda(s)}} m \ominus s}$$

**Fig. 7.** Operational semantics of open nets.

*Example 3.* A marked open net is shown in Fig. 6 (left). As usual, circles represent places and rectangles transitions. Arrows represent pre- and post-sets of transitions. Bullets in places, referred to as tokens, represent the initial marking $m_0$ of the net. For the sake of readability, places are often provided with an identifier, yet positioned outside of the corresponding item.

Any open place has a name which is placed inside the corresponding circle. In particular, there is one open place, the green one, which is labelled by $a$. Finally, in the initial marking $m_0$ of the net, the places $\alpha$ and $\beta$ are marked. For example, by applying the (Step) rule in Fig. 7, we obtain the firing $m_0 \xrightarrow{\tau} m_1 = \{\gamma, \delta\}$. By applying again the same rule $m_1 \xrightarrow{\tau} m_2 = \{\gamma, \sigma\}$, while by the (In) rule $m_1 \xrightarrow{a} m_3 = \{\gamma, \delta, a\}$. Moreover, by applying twice the (Step) rule, $m_3 \xrightarrow{\tau} m_4 = \{\eta, \delta\}$ and $m_4 \xrightarrow{\tau} m_5 = \{\eta, \sigma\}$.

It is easy to see that the set of traces of this net consists of all and only the traces $w \in \mathcal{V}^{\star}$ such that (1) only $a$ and $\overline{a}$ occur in $w$; and (2) in every prefix of $w$, the number of occurrences of $a$ is larger than the number of occurrences of $\overline{a}$.

## 4.1 Open Petri Nets with Interfaces

In order to allow for an inductive construction of open Petri nets from a set of basic components, we enrich open nets with interfaces and suitable operators for net composition along the interfaces.

In the following, for each $n \in \mathbb{N}$ we denote by $n^+$ the set $\{1, \ldots, n\}$, by $n^-$ the set $\{-1, \ldots, -n\}$ and by 0 the empty set $\emptyset$. Also, for $f : n^+ \to S$ and $g : m^+ \to S$, we denote $f + g : (n+m)^+ \to S$ the function $(f+g)(x) = f(x)$ if $x \leq n$ and $g(x-n)$ otherwise (and similarly for $n^-$ and $m^-$).

**Definition 8 (Open nets with interfaces).** *Let $l, r \in \mathbb{N}$. An open net with left interface $l$ and right interface $r$ is a triple $IN = \langle li, ON, ri \rangle$, where $ON$ is an open net, $li : l^+ \to S$ and $ri : r^- \to S$ are the* left *and* right *interface functions, respectively.*

We denote by $l^+ \xrightarrow{li} ON \xleftarrow{ri} r^-$ a net with left interface $l^+$ and right interface $r^-$. With an abuse of notation, in the following we refer to the places belonging to the image of the left interface function as left places, and similarly for the places in the image of the right one. From now on we will denote the components of an open net with interfaces by $l^+, li, ON, ri$, and $r^-$, possibly with subscript.

Graphically, a net with interfaces is represented as an open net, with the left interface on the left and the right interface on the right, marked with incoming and outgoing dotted arrows, respectively. Arrows of the left places are blue while those of right places are red (grey when in b & w).

*Example 4.* A net with interfaces is shown in Fig. 6 (right). The left interface consists of the places $\alpha$, and $\beta$, while the right one contains $\eta$ and $\sigma$. The places labelled $\gamma$ and $\delta$ are internal, i.e., they do not belong to the interfaces.

Relying on the notion of interface, we can define two suitable composition operators on nets. Here we just provide an informal description: The reader is referred to [2,3] for a detailed definition.

**Definition 9 (Composition operations).** *Let $IN_1 = l_1^+ \xrightarrow{li_1} ON_1 \xleftarrow{ri_1} r_1^-$ and $IN_2 = l_2^+ \xrightarrow{li_2} ON_2 \xleftarrow{ri_2} r_2^-$ be (point-wise disjoint) nets with interfaces.*

- *When $r_1 = l_2$, their* sequential composition *$IN_1 \circ IN_2$ is the net with interfaces $l_1^+$ and $r_2^-$ obtained by taking the disjoint union of the nets $N_1$ and $N_2$ and merging the open (right) places of $N_1$ with the corresponding open (left) places of $N_2$.*
- *Their* parallel composition *$IN_1 \otimes IN_2$ is the net with interfaces $(l_1 + l_2)^+$ and $(r_1 + r_2)^-$ obtained by taking the disjoint union of the nets $N_1$ and $N_2$, and merging the open places of $N_1$ with the corresponding open places of $N_2$.*
- *The restriction $(\nu a)IN_1$ of $IN_1$ with respect to $a \in \mathcal{N}$ is the net with interfaces $l_1^+$ and $r_1^-$ obtained by closing the open places labelled by $a$. We often generalise the operator to any $X \subseteq \mathcal{N}$.*

After building the encoding of a process, we also need to fix its initial state. This is accomplished by marking the the left places of the resulting open net. To this end, the following operation will then be used.

**Definition 10 (Marking).** *Let $IN$ be a net with interfaces. The marking of $IN$ is the marked open net $init(IN) = \langle ON, m_0 \rangle$, where $m_0 = \biguplus_{n=1}^{l^+} li(n)$.*

## 4.2 From ACCS Processes to Nets

Exploiting the algebra of open nets outlined in the previous section, we introduce an encoding for ACCS processes into open nets that preserves and reflects the behaviour. The encoding will be restricted to *bound* processes, i.e., processes where restrictions never occurs under replications.

**Definition 11 (Bound ACCS processes).** *An ACCS process is called* bound *if no restriction* $(\nu a)-$ *occurs under replication.*

Intuitively, by restricting to bound processes we avoid the generation of an unbounded number of restricted (and thus conceptually different) names. This will be essential to guarantee the finiteness of the Petri net encoding.

The encoding of a process is defined inductively starting from a set of constant nets, those depicted in Fig. 8, which are then combined using the composition operators on nets in Sect. 4.1. The net *nil* in Fig. 8(a), which is later used to represent the inactive process, consists of a single unmarked place. The net $out_a$ in Fig. 8(b) models the output action on a channel name $a$ and it consists of a single left place, which is also open. The net $a$ in Fig. 8(c), where $a \in \mathcal{N}$, is very similar to the previous but it has an empty left interface. It is going to be used to model additional free names in the encoding of a process. The net $dupl_i$ in Fig. 8(d) is a combinator for the summation of prefixes (input and $\tau$ actions) where $i$, the cardinality of the right interface, matches the number of prefixes involved in the sum. The net $repl_i^a$ in Fig. 8(e), where $a \in \mathcal{N}$, is going to be used as a combinator for replication. It allows for a new "parallel activation" of the net which follows, each time a token is inserted in the open place $a$. Once more, $i$ is the cardinality of the right interface which will match that of the left interface of the encoding of the process under the replication operator. The net $act_i^a$ in Fig. 8(f), where $a \in \mathcal{N}$, provides a combinator for the input action on a channel $a$. It consists of a transition with two places in the pre-set, a left place for the flow of control and the open place $a$ modelling the channel on which the input is required. Again, $i$ is the cardinality of the right interface matching the left interface of the encoding of the continuation of $a$. Finally, the net $act_i^\tau$ in Fig. 8(g) models a $\tau$ prefix: the only difference with respect to $act_i^a$ is the absence of an open place modelling the channel.

The definition below introduces the net encoding of bound ACCS processes.

**Definition 12 (Encoding for processes).** *Let $P$ be an ACCS bound process. The encoding of $P$, denoted by $[\![P]\!]$, is defined as $[\![P]\!] = init(|P|)$, where $|.|$ is given by the inductive rules in Fig. 9, where $l_{|P|}$ and $l_{|P_j|}$ denote the left interfaces of the corresponding encodings.*

The encoding of an ACCS process $P$ is built inductively by composing those of its sub-processes, and by marking the places in the left interface of the resulting net. The encoding contains one place for each operator $!_a$, $\oplus$ and process $\mathtt{0}$ of $P$ and a place for each name of $P$, which are open just for free names. Transitions mimic the control flow of a process, passing the token between its sequential

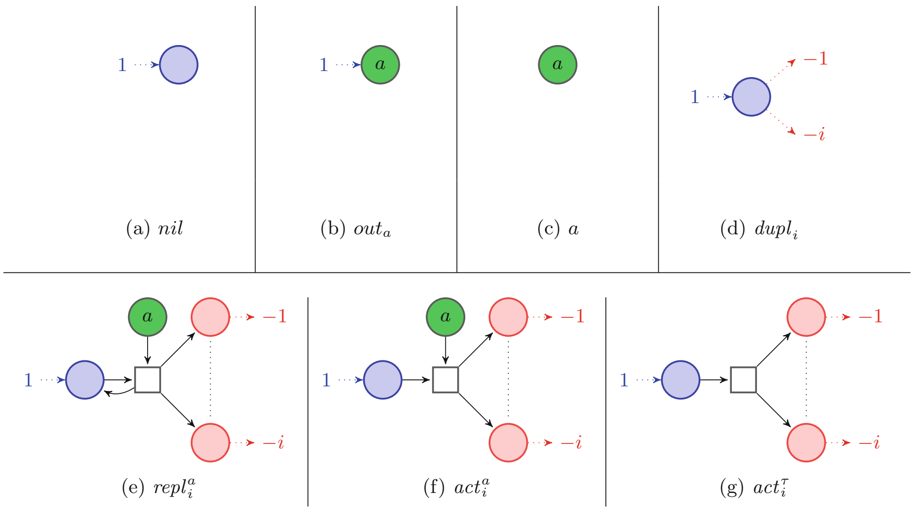

**Fig. 8.** The constant nets *stop*, $repl^a_i$, $out_a$, $act^a_i$, $dupl_i$ and $act^\tau_i$.

$$
\begin{array}{lcl}
|0| & = & nil \\
|\bar{a}| & = & out_a \\
\left|\oplus_{j=1}^{n}\gamma_j.P_j\right| & = & dupl_n \circ \{\otimes_{j=1}^{n}(act^{\gamma_j}_{l_{|P_j|}} \circ |P_j|)\} \\
|!_a.P| & = & repl^a_{l_{|P|}} \circ |P| \\
|(\nu a)P| & = & (\nu a)\,|P| \\
|P \mid Q| & = & |P| \otimes |Q|
\end{array}
$$

**Fig. 9.** Encoding for ACCS processes.

components. It can be shown that the encoding respects structural congruence: structurally equivalent processes are mapped into isomorphic nets and vice versa.

*Example 5. ([Restricted and parallel processes])* Consider again the process $Q = (\nu d)Q_1$, with $Q_1 = \tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d})$, which was introduced in Example 1. The encoding $|Q|$ is shown in Fig. 10. It is obtained by applying the $init(\cdot)$ operation to the net $(\nu d)\,|Q_1|$. In particular, $|Q|_1$ is the result of the sequential composition between $dupl_1$ and $act^\tau_1 \circ |Q_2|$, where $Q_2 = d.\bar{c} \mid d.\bar{e} \mid \bar{d}$. In turn, the net $|Q_2|$ is obtained by the parallel composition between $d.\bar{c} \mid d.\bar{e}$ and $\bar{d}$, where the former is obtained via the parallel and sequential compositions of constant nets. The places labelled by $c$, $d$, and $e$ correspond to the output actions $\bar{c}$, $\bar{d}$, and $\bar{e}$ of $Q_2$. They are all open in $|Q_2|$, meaning that they represent free names of $Q_2$. The sub-net rooted at $\alpha$ is the encoding of the sub-process $d.\bar{c}$, while the one rooted at $\beta$ encodes the sub-process $d.\bar{e}$. The place $d$ is open in $|Q_1|$, but since $d$ is restricted in $Q$, it is removed from the set of open places of $|Q_1|$ by applying the restriction operation of nets.

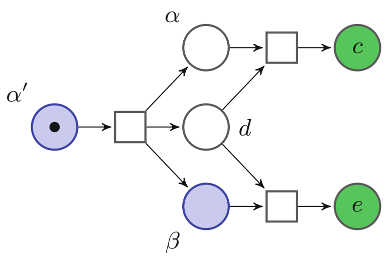

**Fig. 10.** Net encoding the process $(\nu d)(\tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d}))$.

We denote by $[\![P]\!]_\Gamma$, the encoding of a bound process $P$ with respect to a set of names $\Gamma$, that is, $[\![P]\!]_\Gamma = \mathit{init}(|P| \otimes (\otimes_{a\in\Gamma} a))$. The addition of the component $\otimes_{a\in\Gamma} a$ determines the presence in the encoding of a place for each channel in $\Gamma$ (which could possibly not occur free in $P$). One can establish a correspondence between the ACCS processes reachable from $P$, hereafter denoted by the set $\mathit{reach}(P) = \{Q : \exists w \in \mathcal{V}^\star, P \overset{w}{\Rightarrow}_s Q\}$, and the markings of $[\![P]\!]_\Gamma$, through which we can relate internal reductions in ACCS processes and their encodings.

**Theorem 3 (Process reductions as net firings).** *Let $P$ be an ACCS bound process and $\Gamma$ a set of names. Then there is a function $\mathbf{m}^P_\Gamma : \mathit{reach}(P) \to S^\oplus_{[\![P]\!]_\Gamma}$, mapping any process $Q \in \mathit{reach}(P)$ into a marking of $[\![P]\!]_\Gamma$, such that*

1. *if $Q \xrightarrow{\tau}_s R$ then $\mathbf{m}^P_\Gamma(Q) \xrightarrow{\tau} \mathbf{m}^P_\Gamma(R)$ in $[\![P]\!]_\Gamma$;*
2. *if $\mathbf{m}^P_\Gamma(Q) \xrightarrow{\tau} m$ in $[\![P]\!]_\Gamma$ then $Q \xrightarrow{\tau}_s R$ with $m = \mathbf{m}^P_\Gamma(R)$.*

*Proof.* Since by Lemma 1(1) silent transitions in $\to$ and $\to_s$ are exactly the same, the result follows from a straightforward adaption of [2, Theorem1]. $\square$

## 5 Traces in ACCS and in Open Nets

The correspondence between the reduction-based operational semantics of ACCS processes and their net encodings immediately lifts to a preservation and reflection of various behavioural equivalence, notably weak and strong barbed bisimilarity (see [2,3]). In this section we show that the correspondence holds also for trace equivalence, as it is easily proved if the saturated LTS is considered.

We first observe a mismatch between the notion of trace for ACCS processes in Sect. 2, which captures the interactive behaviour of a process only up to the $\leq_m$ preorder on traces, and that for Petri nets, which instead fully describe all the possible interactions of a system with its environment. Indeed, for such notion of trace, trace inclusion is not reflected by the net encoding of processes.

*Example 6.* Consider again the processes $P = (\nu d)(!_d.\bar{e} \mid (a.(\bar{a} \mid \bar{d} \mid d.\bar{c}) \oplus \tau.(\bar{d} \mid d.\bar{c})))$ and $Q = (\nu d)(\tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d}))$ as introduced in Example 1. We already observed there that $a\bar{a}\bar{e} \in \mathit{traces}_s(P)$, while $a\bar{a}\bar{e} \notin \mathit{traces}_s(Q)$. On the contrary, the encodings of $P$ and $Q$ have exactly the same traces: indeed, this also happens in the saturated LTS, as shown in Example 2.

We next prove that by considering traces for ACCS processes on the saturated LTS of Sect. 3 there is a perfect match between trace semantics for ACCS processes and for their encodings. More precisely, it is possible to prove that in the saturated LTS the correspondence between transitions of an ACCS process and those of its encoding, is not limited to internal reductions (as expressed by Theorem 3) but extends to labelled transitions.

**Theorem 4 (Labelled transitions as net firings).** *Let $P$ be a bound ACCS process, $\Gamma$ a set of names and $Q \in reach(P)$. Then*

1. *if $Q \xrightarrow{v} R$ and $v \in \Gamma \cup \overline{\Gamma}$ then $\mathbf{m}^P_\Gamma(Q) \xrightarrow{v} \mathbf{m}^P_\Gamma(R)$ in $[\![P]\!]_\Gamma$;*
2. *if $\mathbf{m}^P_\Gamma(Q) \xrightarrow{v} m$ in $[\![P]\!]_\Gamma$ then $Q \xrightarrow{v} R$ with $m = \mathbf{m}^P_\Gamma(R)$.*

*Proof.* 1. Assume that $Q \xrightarrow{v} R$ and $v \in \Gamma \cup \overline{\Gamma}$. We distinguish two cases. If $v = a \in \Gamma$ then by definition of the saturated semantics $R = Q \mid \bar{a}$. By definition of the encoding $a$ is an open place in $[\![P]\!]_\Gamma$, hence $\mathbf{m}^P_\Gamma(Q) \xrightarrow{a} \mathbf{m}^P_\Gamma(Q) \oplus a = \mathbf{m}^P_\Gamma(R)$.

If instead, $v = \bar{a} \in \overline{\Gamma}$, then by definition of the saturated semantics it must be $Q \equiv R \mid \bar{a}$. By definition of the encoding $a$ is an open place in $[\![P]\!]_\Gamma$, hence $\mathbf{m}^P_\Gamma(Q) \xrightarrow{\bar{a}} \mathbf{m}^P_\Gamma(Q) \ominus a = \mathbf{m}^P_\Gamma(R)$.

2. Analogous. $\square$

Finally, by using the above result and by recalling that, by Lemma 1(1), silent transitions in $\rightarrow$ and $\rightarrow_s$ coincide, we can conclude the following.

**Corollary 1 (Preservation and reflection of trace semantics).** *Let $P$, $Q$ be ACCS bound processes and $\Gamma$ a set of names such that $fn((P)) \cup fn((Q)) \subseteq \Gamma$. Then $traces(P) \subseteq traces(Q)$ iff $traces(\mathbf{m}^P_\Gamma(P)) \subseteq traces(\mathbf{m}^Q_\Gamma(Q))$.*

## 6 Conclusions and Further Works

In this paper we investigated trace semantics and its may testing characterisation for asynchronous calculi, focusing on asynchronous CCS, and their encodings based on open Petri nets. By considering the LTS of [11,19], we proved that the trace semantics is preserved and reflected by the encoding.

It has to be noted that Theorems 3 and 4 are reminiscent of the similar results in [2,3], and they are actually made easier by the simpler net encoding for ACCS, with respect to CSP, which is presented here. Also noteworthy is the possible connection between the testing semantics and the minimal context semantics, as originally proposed in [21]. We already explored the connection with weak and strong barbed bisimilarities for ACCS in [6], and we do hope that the present work will help cast further lights on other observational equivalences.

Indeed, observe that the results in the paper naturally suggests also a notion of (may) testing semantics for Petri nets, where an observer is any other net including some success transition. Few studies exist in the literature (see, e.g., [20]) and it seems non-trivial to understand whether may testing for nets would coincide with trace equivalence. Differently from processes, the notion of context for nets, intended as a expression built out of constants and sequential and parallel composition, seems too powerful, since it allows for reusing the same transition several times and to merge open places.

In [12], Pierpaolo and coauthors pointed out that a good encoding of a (synchronous) calculus into nets should also preserve the intended degree of concurrency. Our proposal seems to move away from this requirement in the encoding of the replication $!_a.P$ (see Fig. 8(e)). Each unfolding step causes not only its continuation $P$ but also the following occurrences of $a.P$ while, intuitively, these should be considered independent as $!_a.P$ is a finite shorthand for $a.P \mid a.P \mid \ldots$ A solution to this problem – as suggested in a different context in [10, Section 7.2] – could be found by using contextual nets [23] and replacing the feedback edges in Fig. 8(e) with a single read arc. We did not adopt this solution in order to keep our model as simple as possible, and because we decided to leave out of the scope of this paper any analysis of concurrency. The validity of our choice is motivated by a general analysis concerning the concurrent features of systems communicating by means of asynchronous interactions: as we showed in [5], concurrency cannot be observed in such systems, and they include those specified by ACCS and open Petri nets.

**Acknowledgements.** We are indebted in many ways to Pierpaolo Degano. Indeed, the earliest exposure of the third author to Petri nets was in a remote cycle of seminars, whose initial lesson was introduced by the quotation in the first page. A scary moment, if there ever was one. Along the years, we all –either as Ph.D. students or later on as co-authors/colleagues/partners in projects– benefited from the insights and availability of Pierpaolo. More technically, we already mentioned his early contributions on net encoding for calculi. In general terms, the insistence on the proof structure of a computation in order to distill a suitable (concurrent) semantics for a calculus, which is typical of the work of Pierpaolo since the early Eighties, has been a fixed star: the modularity of our net encoding spills out of this "commandment".

We are most grateful to the anonymous reviewers whose suggestions and remarks helped us to improve the paper.

## References

1. Amadio, R., Castellani, I., Sangiorgi, D.: On bisimulations for the asynchronous $\pi$-calculus. Theoret. Comput. Sci. **195**(2), 291–324 (1998)
2. Baldan, P., Bonchi, F., Gadducci, F., Monreale, G.: Modular encoding of synchronous and asynchronous interactions using open Petri nets. Sci. Comput. Program. **109**, 96–124 (2015)
3. Baldan, P., Bonchi, F., Gadducci, F., Monreale, G.V.: Encoding synchronous interactions using labelled Petri nets. In: Kühn, E., Pugliese, R. (eds.) COORDINATION 2014. LNCS, vol. 8459, pp. 1–16. Springer, Heidelberg (2014)
4. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional semantics for open Petri nets based on deterministic processes. Math. Struct. Comput. Sci. **15**(1), 1–35 (2004)

5. Baldan, P., Bonchi, F., Gadducci, F., Monreale, G.V.: Concurrency cannot be observed, asynchronously. Math. Struct. Comput. Sci. **25**(4), 978–1004 (2015)
6. Bonchi, F., Gadducci, F., Monreale, G.V.: A general theory of barbs, contexts, and labels. ACM Trans. Comput. Logic **15**(4), 35:1–35:27 (2014)
7. Boreale, M., De Nicola, R., Pugliese, R.: Trace and testing equivalence on asynchronous processes. Inf. Comput. **172**(2), 139–164 (2002)
8. Bruni, R., Melgratti, H.C., Montanari, U., Sobocinski, P.: Connector algebras for C/E and P/T nets' interactions. Log. Methods Comput. Sci. **9**(3), 1–65 (2013)
9. Busi, N., Gorrieri, R., Zavattaro, G.: Comparing three semantics for Linda-like languages. Theoret. Comput. Sci. **240**(1), 49–90 (2000)
10. Busi, N., Gorrieri, R.: Distributed semantics for the $\pi$-calculus based on Petri nets with inhibitor arcs. Logic Algebraic Program. **78**(3), 138–162 (2009)
11. Castellani, I., Hennessy, M.: Testing theories for asynchronous languages. In: Sarukkai, S., Arvind, V. (eds.) FST TCS 1998. LNCS, vol. 1530, pp. 90–102. Springer, Heidelberg (1998)
12. Degano, P., De Nicola, R., Montanari, U.: CCS is an (augmented) contact free C/E system. In: Zilli, M.V. (ed.) Mathematical Models for the Semantics of Parallelism. LNCS, vol. 280, pp. 144–165. Springer, Heidelberg (1986)
13. Degano, P., De Nicola, R., Montanari, U.: A distributed operational semantics for CCS based on condition/event systems. Acta Informatica **26**(1/2), 59–91 (1988)
14. Degano, P., Gorrieri, R., Marchetti, S.: An exercise in concurrency: a CSP process as a condition/event system. In: Rozenberg, G. (ed.) APN 1998. LNCS, vol. 340, pp. 85–105. Springer, Heidelberg (1987)
15. Devillers, R., Klaudel, H., Koutny, M.: A compositional Petri net translation of general $\pi$-calculus terms. Formal Aspects Comput. **20**(4–5), 429–450 (2008)
16. Goltz, U.: CCS and Petri nets. In: Guessarian, I. (ed.) Semantics of Systems of Concurrent Processes. LNCS, vol. 469, pp. 334–357. Springer, Heidelberg (1990)
17. Gorrieri, G., Montanari, U.: SCONE: A simple calculus of nets. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 2–31. Springer, Heidelberg (1990)
18. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Upper Saddle River (1985)
19. Honda, K., Tokoro, M.: An object calculus for asynchronous communication. In: Tokoro, M., Nierstrasz, O., Wegner, P. (eds.) ECOOP 1991. LNCS, vol. 612, pp. 21–51. Springer, Heidelberg (1991)
20. Jenner, L., Vogler, W.: Fast asynchronous systems in dense time. Theoret. Comput. Sci. **254**(1–2), 379–422 (2001)
21. Leifer, J.J., Milner, R.: Deriving bisimulation congruences for reactive systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, p. 243. Springer, Heidelberg (2000)
22. Milner, R.: Bigraphs for Petri nets. In: Reisig, W., Desel, J., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 686–701. Springer, Heidelberg (2004)
23. Montanari, U., Rossi, F.: Contextual nets. Acta Informatica **32**(6), 545–596 (1995)
24. Sassone, V., Sobociński, P.: A congruence for Petri nets. In: Mens, T., Schürr, A., Taentzer, G. (eds.) PNGT 2004. ENTCS, vol. 127, pp. 107–120. Elsevier (2005)
25. Selinger, P.: Categorical structure of asynchrony. In: Brookes, S., Jung, A., Mislove, M., Scedrov, A. (eds.) MFPS 1999. ENTCS, vol. 20. Elsevier (1999)
26. Winskel, G.: A new definition of morphism on Petri nets. In: Fontet, M., Mehlhorn, K. (eds.) STACS 1984. LNCS, vol. 166, pp. 140–150. Springer, Heidelberg (1984)