Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Modular encoding of synchronous and asynchronous interactions using open Petri nets ☆

Paolo Baldan [a], Filippo Bonchi [b], Fabio Gadducci [c,*], Giacoma Valentina Monreale [c]

[a] Dipartimento di Matematica, Università di Padova, Italy
[b] ENS Lyon, Université de Lyon, LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA), France
[c] Dipartimento di Informatica, Università di Pisa, Italy

ARTICLE INFO

Article history:
Received 17 March 2014
Received in revised form 1 September 2014
Accepted 7 November 2014
Available online xxxx

Keywords:
Open Petri nets
Asynchronous CCS
CSP
Net encoding of processes
Synchronous and asynchronous interaction

ABSTRACT

The paper investigates the relationships between two well-known approaches to the modelling of concurrent and distributed systems, process calculi and Petri nets. A framework for the modular encoding of process calculi into Petri nets is proposed, which is based on a reactive variant of Petri nets. In particular, two exemplary calculi are considered: (asynchronous) CCS and CSP, representing alternative interaction paradigms, namely asynchronous and (broadcast) synchronous communication. The encoding is proved to preserve as well as to reflect the operational semantics. As a consequence, it is well-behaved with respect to the standard behavioural equivalences, a fact that is exploited to perform a "technology transfer" between the two formalisms, in terms of un/decidability results for classical properties such as reachability and deadlock-freedom.

The encoding highlights the expressiveness of the proposed reactive variant of nets, as well as paving the way for a fruitful integration of tools and techniques between the visual formalism of nets and the algebraic framework of processes.

1. Introduction

Synchronisation mechanisms, which allow for a proper interaction between parallel system components, obviously play a central role in concurrency theory.

A synchronisation paradigm is the heart of the design of any process calculus, around which the formalism is built. The interaction paradigm in Hoare's Communicating Sequential Processes (CSP) [1] is broadcast synchronisation, while processes of Milner's Calculus of Communicating Systems (CCS) [2] interact via synchronous two-party communication. Since the spread of massively distributed systems, much more attention has been devoted to 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* [3,4], many process calculi (e.g., [5–7]) have been proposed that embody some asynchronous communication mechanism, amongst which the asynchronous CCS [8] is the most relevant for our work.

☆ Partly supported by EU FP7-ICT IP 257414 ASCENS, MIUR PRIN 2010LHT4KM CINA, and ANR 121S02001 PACE.
* Corresponding author.
E-mail addresses: baldan@math.unipd.it (P. Baldan), filippo.bonchi@gmail.com (F. Bonchi), fabio@di.unipi.it (F. Gadducci), vale@di.unipi.it (G.V. Monreale).

http://dx.doi.org/10.1016/j.scico.2014.11.019
0167-6423/

Petri nets [9] are among the most widely used formalisms for the visual specification of concurrent systems. In a Petri net the behavioural relations between computational steps, such as causal dependencies and non-deterministic choices, are explicit and easy to analyse. The appeal of Petri nets lies in their ease of use as well as in their expressiveness. Indeed, their graphical presentation allows for a simple description of possibly complex interaction patterns, in such a way that both synchronous and asynchronous features can be represented.

Synchronisation mechanisms are somehow less explicit in Petri nets. Roughly, the state of a net consists of a set of tokens distributed among the places of the net, while its dynamics is expressed by the token flows that are determined by transition firings. Thus, transitions realise a sort of synchronous composition of different token flows: all the places in the pre-set must be filled by tokens in order to enable a transition. The interaction on places, instead, is eminently asynchronous: a token is produced in a place and later consumed, when needed.

In order to take advantage from the best of the two settings, the relation between process calculi and Petri nets has been often investigated. In particular, Petri nets have been used as the target for the encoding of many process 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 any suitable encoding 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 π-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 [10–13].

In this work we propose a modular and uniform approach to process encoding which relies on *open nets* [14–17], reactive extensions of the ordinary net model, and we show that it can naturally be applied to calculi adhering to either the synchronous or the asynchronous interaction paradigm.

Our open nets are ordinary P/T Petri nets 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 by exchanging tokens on open places or by synchronising on visible transitions. The choice of exposing both places and transitions to the environment emerges naturally from the need of encoding calculi featuring both synchronous and asynchronous communications since, as observed above, interactions on places and transitions have an asynchronous and synchronous flavor, respectively. Indeed, in our case studies, for encoding asynchronous communications we let net components to interact only on places while synchronous communications reduces to an interaction on transitions. Such a difference is reflected by the encoding of the restriction operator, which corresponds to closing an open place for asynchronous calculi and to hiding a visible transition for synchronous ones.

Concretely, as case studies we focus on two paradigmatic process calculi, namely CSP and CCS, and we encode two interaction mechanisms whose connection with nets has received less attention in the literature: the broadcast synchronisation pattern of CSP [1] and the asymmetric message passing of asynchronous CCS [8]. These are instances of the main alternatives concerning the communication pattern, namely, asynchronous message reception vs broadcast synchronisation, respectively. A third classical paradigm is synchronous two-party communication, where channels adopt a strict handshaking pattern, and messages are simultaneously sent and received as in the seminal CCS [2]. There is a large literature on the encoding of CCS into Petri nets (see, e.g., [10–13]) and the calculus naturally fits in our framework, by an easy adaptation of the ideas in these papers (especially the latter). As this case study would give no further insights, we decided not to treat it explicitly.

We identify fragments of both asynchronous CCS and CSP, hereafter referred to as *bound*, which can be mapped *modularly* into Petri nets via encodings that preserve as well as reflect the operational semantics. 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 (bisimilarity for asynchronous CCS 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. We explicitly prove this fact for trace equivalence on CSP and (strong and weak) bisimulation equivalence on asynchronous CCS. The latter correspondence is more surprising, given the asymmetry between sending and receiving in asynchronous equivalences.

Summarising, the main features of our encodings are

1. modularity: the encoding is built inductively from a set of basic net constants, by using few composition operators;
2. structural congruence of processes coincides with isomorphism of nets;
3. process transitions are in one-to-one correspondence with net firings;
4. behavioural equivalences of processes are preserved and reflected.

These features allow for a fruitful technology transfer amongst nets and the fragments of the two calculi.

For instance, by using the fact that reachability is decidable for Petri nets, through the encodings we prove that reachability and convergence are decidable for bound asynchronous CCS and CSP (which, thus, are not Turing powerful).

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

P	::=	nil	inactive process
		$\bigoplus_{i=1}^{n} \mu_i.P_i$	summation
		$\bar{a}$	output
		$P \mid Q$	parallel operator
		$(\nu a)P$	restriction operator
		$!_a.\, P$	replication

Fig. 1. ACCS processes.

In the other direction, we first show that for bound asynchronous CCS (strong and weak) bisimulation equivalence is undecidable, answering a question faced for the synchronous case in [18]. This is done by providing a suitable encoding of 2-counter machines into bound asynchronous CCS. Incidentally, this shows that the bound fragment is far from trivial as it is expressive enough to allow some form of encoding of a Turing complete formalism. Then, by exploiting the encoding, we deduce that these equivalences are undecidable also for open nets *without* visible transitions. This falls outside the known undecidability results for Petri nets [19] since in open nets without visible transitions all transitions are indistinguishable for strong equivalences and unobservable for weak equivalences (e.g., all open Petri nets without visible transitions and without open places are weakly bisimilar in our setting).

Synopsis The paper is structured as follows. In Section 2 we recall the syntax and the operational semantics of the calculi of interest in the paper, namely asynchronous CCS and CSP, as well as their behavioural equivalences. In Section 3 we present the open Petri nets model used for the encoding, introducing nets with interfaces and a suitable algebra for them. The core of the paper is represented by Sections 4 and 5, where we present the modular encodings into open nets of (bound) asynchronous CCS and CSP processes, respectively. We also prove that the encoding of each calculus preserves and reflects its operational semantics as well as some common behavioural equivalences. The encodings are exploited in Section 6 to provide some examples of their effects on the technology transfer between the formalisms we consider. In Section 7 we compare our proposal with others already presented in the literature, while in Section 8 we draw some conclusions and provide pointers to future works.

This paper extends the conference versions [20] and [21], where the encodings for asynchronous CCS and CSP into nets were originally introduced. In particular, the solution for asynchronous CCS presented here largely differs from the one in [20] for its use of the modular encoding technique proposed in [21], which in turn had to be generalised in order to allow for the modelling of both interaction patterns.

2. Process calculi

In this section we briefly recall the basic facts concerning two paradigmatic process calculi, asynchronous CCS and CSP. They are chosen as the target of the Petri net encoding in order to show the flexibility of our proposal.

2.1. Asynchronous CCS

The asynchronous CCS (ACCS) is characterised by the fact that message sending and reception are not synchronised. Instead, 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), while receiving is (processes must wait until a message becomes available). Observations reflects this asymmetry: since sending is non-blocking, receiving is unobservable.

We adopt the presentation in [22] that allows a non-deterministic choice for input prefixes and silent actions (a feature missing in [6,8]).

Definition 1 *(ACCS syntax).* Let $\mathcal{N}$ be a set of *names*, ranged over by $a, b, c, \ldots$ and $\tau \notin \mathcal{N}$. The set $\mathcal{P}_A$ of ACCS processes is generated by the syntax in Fig. 1, for μ ranging over $\mathcal{N} \cup \{\tau\}$. We let $P, Q, R, \ldots$ range over $\mathcal{P}_A$.

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, which disappears whenever it is received.

We assume the standard definitions for the set of free names of a process P, denoted by $\mathtt{fn}(P)$. Similarly for α-convertibility with respect to 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 Fig. 2, where $C[\_]$ denotes a process context such that the "hole" _ does not occur inside the scope of a replication $!_a$. The behaviour of a process P is described as a relation over processes up to $\equiv$.

Definition 2 *(Reduction semantics).* The *reduction relation* for ACCS processes is the relation $\rightarrow \subseteq \mathcal{P}_A \times \mathcal{P}_A$ inductively defined by the set of rules in Fig. 3 and closed under $\equiv$, where we write $P \rightarrow Q$ for $\langle P, Q \rangle \in \rightarrow$. As usual, we let $\Rightarrow$ denote the reflexive and transitive closure of $\rightarrow$.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

$$(\text{Alt}) \quad \frac{\rho \text{ permutation}}{\bigoplus_{i=1}^{n} \mu_i.P_i = \bigoplus_{i=1}^{n} \mu_{\rho(i)}.P_{\rho(i)}}$$

$$(\text{Par}_1) \quad \overline{P \mid Q = Q \mid P} \qquad (\text{Par}_2) \quad \overline{P \mid (Q \mid R) = (P \mid Q) \mid R}$$

$$(\text{Res}_1) \quad \frac{X \cap \mathtt{fn}(P) = \emptyset}{(\nu X)P = P} \qquad (\text{Res}_2) \quad \frac{X \cap \mathtt{fn}(C[nil]) = \emptyset}{C[(\nu X)P] = (\nu X)C[P]}$$

Fig. 2. ACCS structural axioms, for $C[\_]$ a process context with no occurrence of $!_a.\ -$.

$$(\text{Syn}) \quad \frac{j \in \{1, \ldots, n\} \quad \mu_j = a_j}{\bigoplus_{i=1}^{n} \mu_i.P_i \mid \bar{a}_j \to P_j} \qquad (\text{Tau}) \quad \frac{j \in \{1, \ldots, n\} \quad \mu_j = \tau}{\bigoplus_{i=1}^{n} \mu_i.P_i \to P_j}$$

$$(\text{Repl}) \quad \overline{!_a.\ P \mid \bar{a} \to\ !_a.\ P \mid P}$$

$$(\text{Par}) \quad \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad (\text{Res}) \quad \frac{P \to P'}{(\nu a)P \to (\nu a)P'}$$

Fig. 3. ACCS reduction semantics.

Rule (Syn) represents the reception of a message in a non-deterministic context: the process $a_j.P_j$ is ready to receive a message on the channel a_j; it then receives the message $\bar{a}_j$, which is consumed, and proceeds as P_j. Rule (Tau) represents an internal computation step. Rule (Repl) allows the spawning of a new parallel copy of process P whenever a message is received on a. Lastly, rules (Par) and (Res) state the closure of the reduction relation with respect to the operators of restriction and parallel composition.

The main difference with respect to the syntax of the calculus in [22] 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. Concerning the structural congruence, with respect to [22] we added an axiom schema for distributing the restriction under each operator different from replication, thus also under the sum and the prefix.

Definition 3 *(Bound ACCS processes).* An ACCS process is called *bound* if no restriction $(\nu a)\_$ occurs under replication.

Intuitively, the limitation to bound processes avoids 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 main difference with the synchronous calculus lies in the notion of observation. Since messages sending 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 4 *(Barb).* Let P be an ACCS process. We say that P *satisfies a 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 a weak barb* $\bar{a}$, denoted $P \Downarrow_{\bar{a}}$, if $P \Rightarrow Q$ and $Q \downarrow_{\bar{a}}$. When we are interested in the process Q, we will write $P \downarrow_{\bar{a}} Q$ and $P \Downarrow_{\bar{a}} Q$, respectively.

Now, strong and weak barbed bisimulation can be defined as in the synchronous case [23], but taking into account only output barbs.

Definition 5 *(Barbed bisimulation).* A symmetric relation $R \subseteq \mathcal{P}_A \times \mathcal{P}_A$ 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 $\sim_{bb}$ is the largest strong barbed bisimulation. *Weak barbed bisimulation* and *weak barbed bisimilarity* $\approx_{bb}$ are defined analogously by replacing $\downarrow_{\bar{a}}$ with $\Downarrow_{\bar{a}}$ and $\to$ with $\Rightarrow$.

Strong (weak) barbed bisimilarities are not congruences. For instance, $a.\bar{b} \sim_{bb} nil$ (and $a.\bar{b} \approx_{bb} nil$), since neither process can perform any transition, but whenever we insert them into the context $\_ \mid \bar{a}$, the former can perform a transition, the latter cannot. This fact suggests to define behavioural equivalence as follows.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

P	::=	nil	inactive process
		$\bigoplus_{i=1}^{n} a_i.P_i$	guarded alternative
		$P + Q$	non-deterministic choice
		$P \mid_X Q$	parallel operator
		$P \backslash X$	hiding operator
		$!_a.\, P$	replication

Fig. 4. CSP processes.

Definition 6 *(Barbed equivalence).* Two ACCS processes P, Q are *strongly barbed equivalent*, denoted $P \sim_{be} Q$, if $P \mid S \sim_{bb} Q \mid S$ for all processes S.

Similarly, they are *weakly barbed equivalent*, denoted $P \approx_{be} Q$, if $P \mid S \approx_{bb} Q \mid S$ for all processes S.

An alternative characterisation of barbed equivalence is proposed in [22]. Besides internal reductions, the bisimulation game considers the addition as well as the observation of output messages (the latter determining their deletion).

Definition 7 *(1-Bisimulation).* A symmetric relation $R \subseteq \mathcal{P}_A \times \mathcal{P}_A$ is a strong 1-bisimulation if whenever $(P, Q) \in R$ then

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

Strong 1-bisimilarity $\sim$ is the largest strong 1-bisimulation. Weak 1-bisimulation and weak 1-bisimilarity $\approx$ are defined analogously by replacing $\to$ with $\Rightarrow$ and $\downarrow_{\bar{a}}$ with $\Downarrow_{\bar{a}}$.

Proposition 1. *(See [22].)* $\sim_{be} = \sim$ *and* $\approx_{be} = \approx$.

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 $P \sim Q$. In fact, they both have a single possible reduction $P \to (\nu d)(!_d.\bar{e} \mid \bar{d} \mid d.\bar{c})$ and $Q \to (\nu d)(d.\bar{c} \mid d.\bar{e} \mid \bar{d})$. Now $(\nu d)(!_d.\bar{e} \mid \bar{d} \mid d.\bar{c}) \sim (\nu d)(d.\bar{c} \mid d.\bar{e} \mid \bar{d})$, since after one further reduction the replication operator is stuck and the two processes becomes essentially identical.

The process P can also receive on channel a. In the 1-bisimulation game this behaviour is revealed by plugging the processes into the context $\_ \mid \bar{a}$. The process $P \mid \bar{a}$ can choose one of the two branches of $\oplus$, but in any case it performs a reduction becoming $(\nu d)(!_d.\bar{e} \mid \bar{a} \mid \bar{d} \mid d.\bar{c})$. On the other hand, $Q \mid \bar{a}$ performs a reduction to $(\nu d)(d.\bar{c} \mid d.\bar{e} \mid \bar{d} \mid \bar{a})$, and 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 $a.\bar{a} \approx nil$ holds.

2.2. Communicating sequential processes

Together with CCS, CSP [1] is one of the earliest proposal of a calculus for modelling communicating and concurrent processes. Differently from the former, the latter adopts a broadcast communication pattern, so that messages are at once shared between all the processes participating in the synchronisation. In this section we briefly introduce some facts about the calculus, presenting its syntax and operational semantics.

Definition 8 *(CSP syntax).* Let Σ be the set of *communication events*, ranged over by $a, b, c \ldots$ and $\tau \notin \Sigma$. The set $\mathcal{P}_C$ of CSP processes is generated by the grammar in Fig. 4, for $X \subseteq \Sigma$ a finite set of events and μ ranging over $\Sigma \cup \{\tau\}$. We let $P, Q, R, \ldots$ range over $\mathcal{P}_C$.

The syntax has been slightly changed from the standard one in order to mimic as much as possible the one for ACCS, yet with equivalent operators. The deadlocked process *nil* stands e.g. for the standard *STOP*. The guarded alternative $\bigoplus_{i=1}^{n} a_i.P_i$ can perform any event a_i, for $i \in \{1, \ldots, n\}$, and then behaves as P_i. For the sake of simplicity, we assume that $\forall j, \forall z.\, a_j \neq a_z$. The non-deterministic choice $P + Q$ can behave as either P or Q. Observe that the operators $\oplus$ and $+$ differs for the fact that for the former the choice is external, i.e., the environment can determine the branch which is chosen, while for the latter the choice is internal to the process. The process $P \mid_X Q$ is a parallel composition of P and Q, where the events in X are forced to synchronise, while the others can be performed by P and Q independently. The hiding $P \backslash X$ behaves like P, except for the fact that the events in X are hidden to the environment. Finally, the replication $!_a.\, P$ can indefinitely perform an event a and spawn a copy of P.

The guarded alternative represents a specialisation of the external choice operator. This restriction does not represent a serious limitation since, as explained in [24], it is rare to find a usage of the external choice which cannot be expressed

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

$$\text{(Alt)}\quad \frac{j \in \{1,\ldots,n\}}{\bigoplus_{i=1}^{n} a_i.P_i \xrightarrow{a_j} P_j} \qquad \text{(Cho)}\quad \frac{}{P+Q \xrightarrow{\tau} P}$$

$$\text{(Repl)}\quad \frac{}{!_a.P \xrightarrow{a} !_a.P \mid_{\varnothing} P}$$

$$\text{(Asyn)}\quad \frac{P \xrightarrow{\mu} P' \quad \mu \notin X}{P \mid_X Q \xrightarrow{\mu} P' \mid_X Q} \qquad \text{(Syn)}\quad \frac{P \xrightarrow{\mu} P' \quad Q \xrightarrow{\mu} Q' \quad \mu \in X}{P \mid_X Q \xrightarrow{\mu} P' \mid_X Q'}$$

$$\text{(Hid}_1\text{)}\quad \frac{P \xrightarrow{\mu} P' \quad \mu \notin X}{P\backslash X \xrightarrow{\mu} P'\backslash X} \qquad \text{(Hid}_2\text{)}\quad \frac{P \xrightarrow{\mu} P' \quad \mu \in X}{P\backslash X \xrightarrow{\tau} P'\backslash X}$$

Fig. 5. CSP operational semantics.

$$\text{(Cho)}\quad \frac{}{P+Q = Q+P} \qquad \text{(Alt)}\quad \frac{\rho \text{ permutation}}{\bigoplus_{i=1}^{n} a_i.P_i = \bigoplus_{i=1}^{n} a_{\rho(i)}.P_{\rho(i)}}$$

$$\text{(Par}_1\text{)}\quad \frac{}{P \mid_X Q = Q \mid_X P} \qquad \text{(Par}_2\text{)}\quad \frac{}{P \mid_X (Q \mid_X R) = (P \mid_X Q) \mid_X R}$$

$$\text{(Par}_3\text{)}\quad \frac{X \cap \mathtt{fn}(P \mid_Y Q) = \emptyset}{P \mid_{X \cup Y} Q = P \mid_Y Q}$$

$$\text{(Res}_1\text{)}\quad \frac{X \cap \mathtt{fn}(P) = \emptyset}{P\backslash X = P} \qquad \text{(Res}_2\text{)}\quad \frac{X \cap \mathtt{fn}(C[nil]) = \emptyset}{C[P\backslash X] = C[P]\backslash X}$$

Fig. 6. CSP axioms.

as a guarded alternative. More interestingly, we consider guarded replication in place of recursion. This will simplify the definition of our Petri net encoding for the class of CSP processes considered (see Section 5.2).

We assume the standard definitions for the set of free names of a process P, denoted by $\mathtt{fn}(P)$. In particular, $\mathtt{fn}(P \mid_X Q) = X \cup \mathtt{fn}(P) \cup \mathtt{fn}(Q)$. We also consider processes up to α-convertibility with respect to the hiding operator $P\backslash a$: the name a is hidden in P and it can be freely α-converted.

Classical presentations of the calculus define the operational semantics by relying only on a set of syntax directed rules. Here, along the approach adopted also for ACCS, we consider a structural equivalence ($\equiv$) over CSP processes defined as the smallest congruence induced by the axioms in Fig. 6, where $C[\_]$ denotes a process context. The behaviour of a process P is then described as a relation over processes up to $\equiv$.

Definition 9 *(Transition semantics).* The *transition relation* for CSP processes is the relation $\rightarrow \subseteq \mathcal{P} \times (\Sigma \cup \{\tau\}) \times \mathcal{P}$ inductively defined by the rules in Fig. 5 and closed under $\equiv$, where $P \xrightarrow{\mu} P'$ means $\langle P, \mu, P' \rangle \in \rightarrow$.

We let $P \xrightarrow{\omega} {}^{*} P'$ denote a sequence $P = P_1 \xrightarrow{\mu_1} P_2 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_{n-1}} P_n = P'$ with $\omega = \mu_1\mu_2\ldots\mu_{n-1}$. We write $P \overset{\omega}{\Rightarrow}{}^{*} P'$ when $P \xrightarrow{\omega'} {}^{*} P'$ for some ω' such that ω is obtained from ω' by removing the τ's.

As in the case of CCS the encoding into Petri nets is feasible for a suitable fragment of the calculus with "finitary" features.

Definition 10 *(Bound CSP processes).* A CSP process is called *bound* if no synchronised parallel $\_ \mid_X \_$ occurs under replication unless $X = \emptyset$.

In words, in a bound process only pure parallel composition, without synchronisation, is allowed under the scope of replications. This avoids the possibility of having an unbounded number of parallel components synchronising on the same event. Additionally, a synchronisation under a replication would possibly lead to the generation of an unbounded number of conceptually different names as in $!_a.(b.b.nil \mid_{\{b\}} b.nil)$. As shown in Section 5, the restriction to bound processes is essential for defining an encoding of CSP into Petri nets.

Relying on the transition relation defined above several observational semantics can be defined over CSP processes. In this paper, we focus on the one based on traces, i.e., sequences of visible transitions. Two processes are deemed trace equivalent when they exhibit the same set of traces.

Definition 11 *(Traces).* Let P be a CSP process. We define $traces(P) = \{\omega : \omega \in \Sigma^{*} \ \wedge \ \exists Q \,.\, P \overset{\omega}{\Rightarrow}{}^{*} Q\}$. Two processes P, Q are called *trace equivalent*, denoted $P \approx_T Q$, if $traces(P) = traces(Q)$.

Example 2. Consider the processes $P = a.(d.b.nil\backslash d) \oplus b.a.nil$ and $Q = (c.a.nil \mid_{\{c\}} c.b.nil)\backslash c$. It is easy to see that $traces(P) = traces(Q) = \{\epsilon, a, ab, b, ba\}$. Hence they are trace equivalent.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

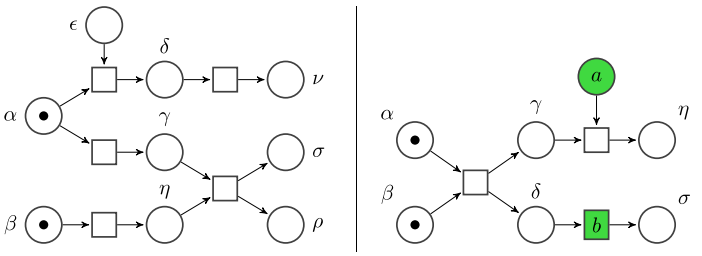

Fig. 7. Graphical representation of Petri nets, the rightmost open.

3. Open Petri nets with interfaces

This section reviews *open Petri nets*, i.e., ordinary P/T nets [9] equipped with distinguished sets of places and transitions. Open nets will be enriched also with interfaces, consisting of lists of places, and endowed with composition operators in order to allow for an inductive encoding of processes.

3.1. Open Petri nets

Let $X^{\oplus}$ be the free commutative monoid over a set X and 2^X the monoid of subsets over X, included in the obvious way in $X^{\oplus}$. An element $m \in X^{\oplus}$, called a *multiset* over X, is often viewed as a function $m : X \to \mathbb{N}$ (the set of natural numbers) associating a multiplicity with each $x \in X$. 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. Given $f : X \to Y$ we denote its extension to multisets by $f^{\oplus} : X^{\oplus} \to Y^{\oplus}$.

Definition 12 *(Petri net).* A *Petri net* is a tuple $N = (S, T, {}^\bullet(.), (.)^\bullet)$ where S is the set of places, T is the set of transitions, and ${}^\bullet(.), (.)^\bullet : T \to 2^{S_\perp}$ are functions mapping each transition to its pre- and post-set.

By $S_\perp$ we denote the lifting of S, i.e., $S_\perp = S \uplus \{\perp\}$, thus nets come equipped with a global "error" place $\perp$, as it will become clear in the presentation of the encoding. The state of a net is given by a *marking*, i.e., a multiset of places $m \in S^{\oplus}$. From now on we denote the components of a net N by S, T, ${}^\bullet(.)$ and $(.)^\bullet$, possibly with subscripts, and write ${}^\bullet t$ and $t^\bullet$ instead of ${}^\bullet(t)$ and $(t)^\bullet$.

Definition 13 *(Net morphism).* Let N_1, N_2 be two nets. A *net morphism* $f : N_1 \to N_2$ is a pair of functions $f = \langle f_S, f_T \rangle$, where $f_S : S_1 \to S_2$, $f_T : T_1 \to T_2$ satisfy for any $t \in T_1$

1. $f_S^{\oplus}({}^\bullet t) \subseteq {}^\bullet f_T(t)$ (reflection of pre-set);
2. $f_S^{\oplus}(t^\bullet) \subseteq f_T(t)^\bullet$ (reflection of post-set).

Net morphisms roughly represent the insertion of a net into a context. As a consequence, the pre- and post-set of transitions can be larger in the target net. In the following, given a net morphism $f : N_1 \to N_2$ we will often use f also to refer to its place and transition components, omitting the subscripts.

Example 3. Fig. 7(left) shows a net. 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 current marking m of the net. For the sake of readability, places are often provided with an identifier, yet positioned outside of the corresponding item.

In order to encode process calculi featuring synchronous and asynchronous communication into nets we consider a reactive generalisation of Petri nets, in the line of [14–17]. More precisely, nets are endowed with distinguished sets of *open* places and *visible* transitions. They represent, respectively, the places through which the environment interacts asynchronously with the net, by putting and removing tokens, and the events that are visible from the environment which can be used for synchronous interactions.

Open places and visible transitions of an open net carry a label. Hereafter we denote by $\mathcal{N}$ and Σ the corresponding fixed sets of labels. The choice is driven by our need of encoding CSP and ACCS, and it reflects the different frameworks: for ACCS channels become resources (hence tokens in places), while for CSP events become actions (hence firings of transitions).

Definition 14 *(Open net).* An *open (Petri) net* is a tuple $ON = \langle N, O, V, \lambda \rangle$, where N is a net, $O \subseteq S$ is a set of *open* places, $V \subseteq T$ a set of *visible* transitions, and $\lambda = \langle \lambda_S, \lambda_T \rangle$ a pair of labelling functions $\lambda_S : O \to \mathcal{N}$, $\lambda_T : V \to \Sigma$ for places and transitions, respectively, with λ_S injective.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

$$\text{(Vis)}\ \frac{m = {}^\bullet t \oplus m' \quad t \in V}{m \xrightarrow{\lambda(t)} t^\bullet \oplus m'} \qquad \text{(Hid)}\ \frac{m = {}^\bullet t \oplus m' \quad t \in T \setminus V}{m \xrightarrow{\tau} t^\bullet \oplus m'}$$

$$\text{(In)}\ \frac{s \in O}{m \xrightarrow{\lambda(s)^+} m \oplus s} \qquad \text{(Out)}\ \frac{s \in O}{m \xrightarrow{\lambda(s)^-} m \ominus s}$$

Fig. 8. Operational semantics of open nets.

When no confusion may arise, we often refer simply to λ as the labelling function, without specifying explicitly the net component it refers to. In the following, the components of an open net ON are assumed to be N, O, V, and λ, possibly with subscripts.

Definition 15 *(Open net morphism).* Let ON_1, ON_2 be two open nets. An *open net morphism* $f : ON_1 \to ON_2$ is a net morphism $f : N_1 \to N_2$ such that for any $t_1 \in T_1$ and $s_1 \in S_1$

1. if $f_T(t_1) \in V_2$ then $t_1 \in V_1$ and $\lambda_1(t_1) = \lambda_2(f_T(t_1))$;
2. if $f_S(s_1) \in O_2$ then $s_1 \in O_1$ and $\lambda_1(s_1) = \lambda_2(f_S(s_1))$.

An open net morphism is intended to represent the insertion of an open net into a larger context. Hence it is required to reflect visible transitions and places, as well as their labels.

The operational semantics of open nets is expressed by the rules in Fig. 8. Rule (Vis) and rule (Hid) are the standard rules of P/T nets (seen as multiset rewriting): the latter is the firing of a hidden transition, represented as a silent action τ. The remaining two rules model interaction with the environment. They state that in an open place at any moment the environment can generate (In) or remove (Out) a token.

Graphically, an open net is represented as an ordinary Petri net, with the open places and visible transitions in green (grey when viewed in b&w), while closed places and hidden transitions are white. Labels are inserted inside the corresponding circle/rectangle.

Example 4. An open net is shown in Fig. 7(right). Any open item has a label that is placed inside the corresponding circle/rectangle. In particular, there is one open place, which is labelled by a, and one visible transition, labelled by b. In the current marking m of the net, the places α and β are marked. Starting from it, some transitions are possible. For example, by applying rule (Hid) in Fig. 8, we obtain the firing $m \xrightarrow{\tau} m_1 = \{\gamma, \delta\}$. By rule (Vis), $m_1 \xrightarrow{b} m_2 = \{\gamma, \sigma\}$, while by rule (In) $m_1 \xrightarrow{a^+} m_3 = \{\gamma, \delta, a\}$. Moreover, by rule (Hid) $m_3 \xrightarrow{\tau} m_4 = \{\eta, \delta\}$ and by rule (Vis) $m_4 \xrightarrow{b} m_5 = \{\eta, \sigma\}$.

Labels are ranged over by l. Weak transitions $P \stackrel{l}{\Rightarrow} Q$ are defined as usual, i.e., $P \stackrel{l}{\Rightarrow} Q$ means $P \xrightarrow{\tau}{}^* \xrightarrow{l} \xrightarrow{\tau}{}^* Q$ for $l \neq \tau$ and $P \xrightarrow{\tau}{}^* Q$ otherwise.

Definition 16 *(Strong and weak bisimilarity).* Let ON_1, ON_2 be two open nets with $\lambda_1(O_1) = \lambda_2(O_2)$. A *strong bisimulation* between ON_1 and ON_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 ON_1 then $m_2 \xrightarrow{l} m_2'$ in ON_2 and $(m_1', m_2') \in \mathcal{R}$;
- if $m_2 \xrightarrow{l} m_2'$ in ON_2 then $m_1 \xrightarrow{l} m_1'$ in ON_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 strongly 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 $\stackrel{l}{\Rightarrow}$.

When a starting state is fixed, nets are called marked.

Definition 17 *(Marked nets).* A *marked open net* is a pair $\mathbf{N} = \langle ON_{\mathbf{N}}, m_{\mathbf{N}} \rangle$, where $ON_{\mathbf{N}}$ is an open net and $m_{\mathbf{N}} \in S^\oplus$ is the *initial* marking.

For marked nets we can consider the language of (weak) traces starting from the initial marking and the corresponding equivalence. As in the case of CSP processes, we let $m \xrightarrow{\omega}{}^* m'$ denote a sequence $m = m_1 \xrightarrow{l_1} m_2 \xrightarrow{l_2} \ldots \xrightarrow{l_{n-1}} m_n = m'$ with $\omega = l_1 l_2 \ldots l_{n-1}$. We also write $m \stackrel{\omega}{\Rightarrow}{}^* m'$ when $m \xrightarrow{\omega'}{}^* m'$ for some ω' such that ω is obtained from ω' by removing the τ's.

Definition 18 *(Traces).* Let $\mathbf{N}$ be a marked open net. We define the *set of traces* of $\mathbf{N}$ as $traces(\mathbf{N}) = \{\omega : \omega \in (\Sigma \cup \mathcal{N})^* \ \wedge \exists m. m_{\mathbf{N}} \stackrel{\omega}{\Rightarrow}{}^* m\}$. Two marked open nets $\mathbf{N}_1$, $\mathbf{N}_2$ are *trace equivalent*, denoted $\mathbf{N}_1 \approx_T \mathbf{N}_2$, if $traces(\mathbf{N}_1) = traces(\mathbf{N}_2)$.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

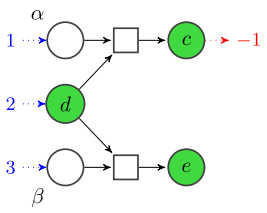

Fig. 9. Graphical representation of a net with interfaces.

3.2. Open Petri nets with interfaces

In order to provide a modular encoding for processes, it is necessary to define suitable composition operators on nets. However, the most obvious choice, namely pointwise disjoint union, fails to address all the nuances that are necessary for the encoding. To this end, we enrich open nets with two sets of interfaces, which represent "handles" through which net items can be singled out and manipulated during the composition. We then introduce two composition operations on nets that are relevant for building the encoding of a process.

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 19 *(Open nets with interfaces).* An open net with left interface l and right interface r is a triple $\mathbb{N} = \langle li, ON, ri \rangle$, where ON is an open net, $li : l^+ \to S$ and $ri : r^- \to S$ are the *left* and *right* 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 morphism as left places, and similarly for the places in the image of the right morphism. From now on we denote the components of an open net with interfaces by l^+, li, ON, ri, and r^-, possibly with subscripts.

Intuitively, thinking of open nets as terms, their left and right places represent the roots and the variables, respectively. This will become clearer when giving the definition of sequential composition (see Definition 21 below), mimicking the standard notion of term substitution (see e.g. the rules of the ACCS encoding in Definition 26).

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 the right places are red (grey when in b&w).

Example 5. A net with interfaces is shown in Fig. 9. The left interface consists of the places α, d and β, while the right interface contains only place c. The place labelled e does not belong to the interfaces.

We now define two suitable composition operators on nets with interfaces. Notation-wise, for each function $f : A \to B$ and $b \in B$ we denote by $f^{-1}(b)$ the counter image of b in A; with an abuse of notation, when f is injective we may also use $f^{-1}(b)$ to denote the unique element of the counter image.

Definition 20 *(Net quotient).* Given a net with interface $\mathbb{N} = l^+ \xrightarrow{li} ON \xleftarrow{ri} r^-$ and a label-preserving equivalence relation $\equiv$ on the set of places S, the *quotient net* $\mathbb{N}_{/\equiv}$ is the net $\mathbb{N}' = l^+ \xrightarrow{in \circ li'} ON' \xleftarrow{in \circ ri'} r^-$ where

- $S' = S_{/\equiv}$ and $in : S \to S'$ maps each place into its equivalence class;
- $T' = T$ with the obvious pre-sets and post-sets (a transition $t \in T'$ has $in({}^\bullet t)$ and $in(t^\bullet)$ as pre-set and post-set, respectively);
- $O' = O_{/\equiv}$ and $V' = V$.

Relying on the quotient we easily define some composition operators on nets.

Definition 21 *(Sequential composition).* Let $\mathbb{N}_1 = l_1^+ \xrightarrow{li_1} ON_1 \xleftarrow{ri_1} r_1^-$ and $\mathbb{N}_2 = l_2^+ \xrightarrow{li_2} ON_2 \xleftarrow{ri_2} r_2^-$ be (pointwise disjoint) nets with interfaces such that $r_1 = l_2$. Their *sequential composition* $\mathbb{N}_1 \circ \mathbb{N}_2$ is the net with interfaces arising as $(\mathbb{N}_1 \cup \mathbb{N}_2)_{/\equiv}$ where $\equiv$ is the equivalence on places induced by $ri_1(x) = li_2(-x)$ for $x \in r_1^-$ and $\lambda_1^{-1}(l) = \lambda_2^{-1}(l)$ for $l \in \lambda_1(O_1) \cap \lambda_2(O_2)$.

The sequential composition $\mathbb{N}_1 \circ \mathbb{N}_2$ is obtained by taking the disjoint union of the underlying open nets, and gluing the open and right places of N_1 respectively with the corresponding open and left places of N_2. Note that the equivalence $\equiv$ is well-defined, since λ_i is injective on O_i for $i \in \{1, 2\}$.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

In the following, given a net with interfaces $\mathbb{N}$ and a set $X \subseteq \Sigma$, we denote by V^X the subset of visible transitions in V labelled with an event in X, namely $V^X = \{t \in V : \lambda(t) \in X\}$.

Definition 22 *(Synchronous composition).* Let $\mathbb{N}_1 = l_1^+ \xrightarrow{li_1} ON_1 \xleftarrow{ri_1} r_1^-$ and $\mathbb{N}_2 = l_2^+ \xrightarrow{li_2} ON_2 \xleftarrow{ri_2} r_2^-$ be (pointwise disjoint) nets with interfaces and $X \subseteq \Sigma$. Their *synchronous composition* on X is the net with interfaces $\mathbb{N}_1 \bigotimes_X \mathbb{N}_2 = \mathbb{N}_{\equiv}$ where $\mathbb{N} = l^+ \xrightarrow{li} ON \xleftarrow{ri} r^-$, for $ON = \langle N, O, V, \lambda \rangle$ such that

- $S = S_1 \cup S_2$;
- $T = (T_1 \setminus W_1^X) \cup (T_2 \setminus W_2^X) \cup W^X$, with $W^X = \bigcup_{a \in X} \lambda_1^{-1}(a) \times \lambda_2^{-1}(a)$ and W_i^X's its projections
- ${}^\bullet t = \begin{cases} {}^\bullet t & \text{if } t \in T_i \setminus V_i^X \\ {}^\bullet t \cup \{\bot\} & \text{if } t \in V_i^X \setminus W_i^X \\ {}^\bullet t_1 \cup {}^\bullet t_2 & \text{if } t = \langle t_1, t_2 \rangle \in W^X \end{cases}$
- $t^\bullet = \begin{cases} t^\bullet & \text{if } t \in T_i \setminus W_i^X \\ t_1^\bullet \cup t_2^\bullet & \text{if } t = \langle t_1, t_2 \rangle \in W^X \end{cases}$
- $l = l_1 + l_2$, $li = li_1 + li_2$, and similarly for r and ri;
- $O = O_1 \cup O_2$ and $V = [(V_1 \setminus W_1^X) \cup (V_2 \setminus W_2^X)] \uplus W_X$, with the obvious labelling functions;

and where $\equiv$ is the equivalence on places induced by $\lambda_1^{-1}(l) = \lambda_2^{-1}(l)$ for $l \in \lambda_1(O_1) \cap \lambda_2(O_2)$.

We write $\mathbb{N}_1 \otimes \mathbb{N}_2$ for $\mathbb{N}_1 \bigotimes_\emptyset \mathbb{N}_2$. Intuitively, the synchronous composition $\mathbb{N}_1 \bigotimes_X \mathbb{N}_2$ is obtained by taking the disjoint union of the nets N_1 and N_2, except for the open places, which are merged, and for those visible transitions labelled with a symbol $a \in X$, which are forced to fire synchronously. Concretely, open places with the same label are glued and for each pair of transitions $t_1 \in V_1$ and $t_2 \in V_2$, with identical label in X, a new transition $\langle t_1, t_2 \rangle$ is inserted whose pre- and post-set is obtained as the union of the pre- and post-set of t_1 and t_2. If a transition t_1 in N_1 has no possibility of synchronising with a transition of N_2 since V_2 does not include transitions with the same label, it is not executable in the synchronised product. This is obtained by adding to its pre-set the "error" place $\bot$, which is never going to be marked. The same happens for transitions in N_2 that cannot synchronise with any transition in N_1. An alternative solution, equivalent from the point of view of the behaviour, would be the removal of the dead transitions. We preferred this solution since, when used for the encoding of CSP processes into nets, it ensures a closer structural correspondence between reducts of a process and the markings of the net encoding. Finally, transitions which are labelled outside X can fire asynchronously and thus are kept unchanged.

Lastly, we introduce two operations for restricting the set of open places and the set of visible transitions, respectively.

Definition 23 *(Restriction and hiding).* Let $\mathbb{N}$ be a net with interfaces. The restriction of $\mathbb{N}$ with respect to $a \in \mathcal{N}$ is the net $(\nu a)\mathbb{N} = \langle li, ON', ri \rangle$, where $ON' = \langle N, O \setminus \lambda^{-1}(a), V, \lambda' \rangle$ and λ' is the restriction of λ. The hiding of $\mathbb{N}$ with respect to $X \subseteq \Sigma$ is the net $\mathbb{N} \setminus X = \langle li, ON', ri \rangle$, where $ON' = \langle N, O, V \setminus V^X, \lambda' \rangle$ and λ' is the restriction of λ.

Given a net $\mathbb{N}$, the restricted net $(\nu a)\mathbb{N}$ is obtained by closing the open place labelled by a. Note that there could be no such place, in which case $(\nu a)\mathbb{N} = \mathbb{N}$. The restriction operator will be often used on sets of labels $X \subseteq \mathcal{N}$ and we will write $(\nu X)\mathbb{N}$ with the obvious meaning. The net with hiding $\mathbb{N} \setminus X$ behaves exactly as $\mathbb{N}$, but transitions labelled in X are no longer visible.

After building the net encoding of a process, we will need to mark its left places in order to fix the initial state. The following operation will then be used.

Definition 24 *(Marking).* Let $\mathbb{N}$ be a net with interfaces. The marking of $\mathbb{N}$ is the marked open net $init(\mathbb{N}) = \langle ON, m_{\mathbb{N}} \rangle$, where $m_{\mathbb{N}} = \biguplus_{n=1}^{l^+} li(n)$.

4. From ACCS processes to nets

In this section we introduce an encoding for ACCS processes into open nets and we prove that the encoding preserves and reflects the behaviour. As anticipated, the encoding will be restricted to *bound* processes, i.e., processes where restrictions never occur under replications.

4.1. Encoding ACCS processes as open nets

The encoding of ACCS processes into open nets is defined inductively by relying on a set of constant nets, depicted in Fig. 10, which are combined using the composition operators on nets introduced in Section 3. Nets with interfaces are exploited in the encoding for properly combining the different components. The net *nil* in Fig. 10(a), later used to represent the inactive process, consists of a single unmarked place. The net out_a in Fig. 10(b) models the output action on a channel

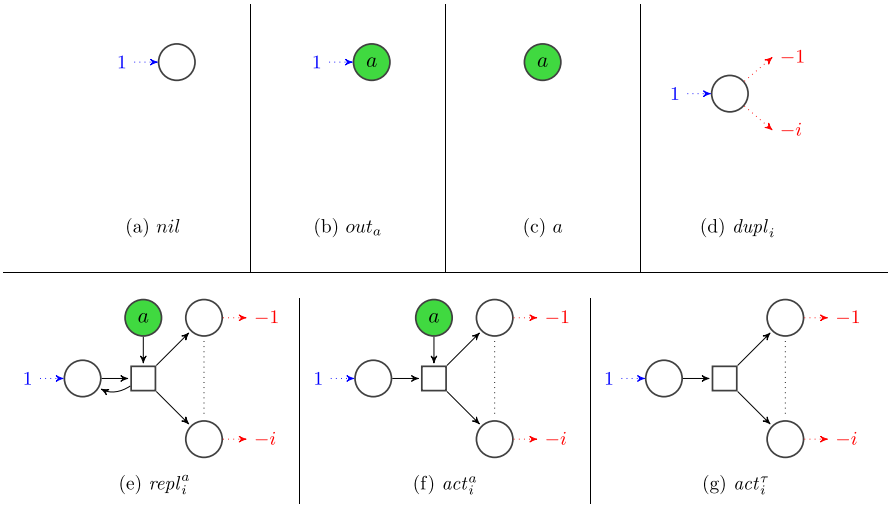

Fig. 10. The constant nets *stop*, $repl_i^a$, out_a, act_i^a, $dupl_i$ and act_i^τ.

$$
\begin{array}{lcl}
|nil| & = & nil \\
|\bar{a}| & = & out_a \\
\left|\bigoplus_{j=1}^{n} \mu_j.P_j\right| & = & dupl_n \circ \left\{\bigotimes_{j=1}^{n} \left(act_{sdeg(P_j)}^{\mu_j} \circ |P_j|\right)\right\} \\
|!_a.P| & = & repl_{sdeg(P)}^a \circ |P| \\
|(\nu a)P| & = & (\nu a)|P| \\
|P \mid Q| & = & |P| \otimes |Q|
\end{array}
$$

Fig. 11. Encoding for ACCS processes.

name a and it consists of a single left place which is also open. The net a in Fig. 10(c), where $a \in \mathcal{N}$, is very similar to the previous one but it has an empty left interface. It will be used to model additional free names in the encoding of a process (see Section 4.2). The net $dupl_i$ in Fig. 10(d) is a combinator for the summation of prefixes (input and τ actions) where i, the cardinality of the right interface, will match the number of prefixes involved in the sum. The net $repl_i^a$ in Fig. 10(e), where $a \in \mathcal{N}$, will 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. Again, 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. 10(f), where $a \in \mathcal{N}$, is intended to provide a combinator for the input action on a channel a. It consists of a hidden 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^τ in Fig. 10(g) models a τ prefix: the only difference with respect to act_i^a is the absence of an open place modelling the channel.

Note that transitions are hidden in all the constants of Fig. 10(g). Hence the nets built out of them will only offer transitions with silent actions τ (corresponding to reductions in corresponding ACCS processes). Labelled transitions will be determined by the interaction with the open places.

In order to define the encoding, we need to refer to the number of parallel sub-components for a process at the top level, as given by the definition below.

Definition 25 *(Structural degree).* The *structural degree* of a process P, denoted by *sdeg*(P), is inductively defined as $sdeg(P \mid Q) = sdeg(P) + sdeg(Q)$, $sdeg((\nu a)P) = sdeg(P)$, and $sdeg(P) = 1$ otherwise.

Finally, the definition below introduces the encoding of bound ACCS processes into nets with interfaces.

Definition 26 *(Net encoding for ACCS).* Let P be a bound ACCS process. The net encoding of P, denoted by $[\![P]\!]$, is defined as $[\![P]\!] = init(|P|)$, where $|.|$ is given by the inductive rules in Fig. 11.

The encoding of an ACCS process P is built inductively by composing the encodings of its subprocesses into nets with interfaces. Finally, in the resulting net the places in the left interface are marked. The encoding contains a place for each operator $!_a$, $\oplus$ and process *nil* of P and a place for each name of P. Places corresponding to free names are open. Transitions mimic the control flow of a process, passing the token between its sequential components.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

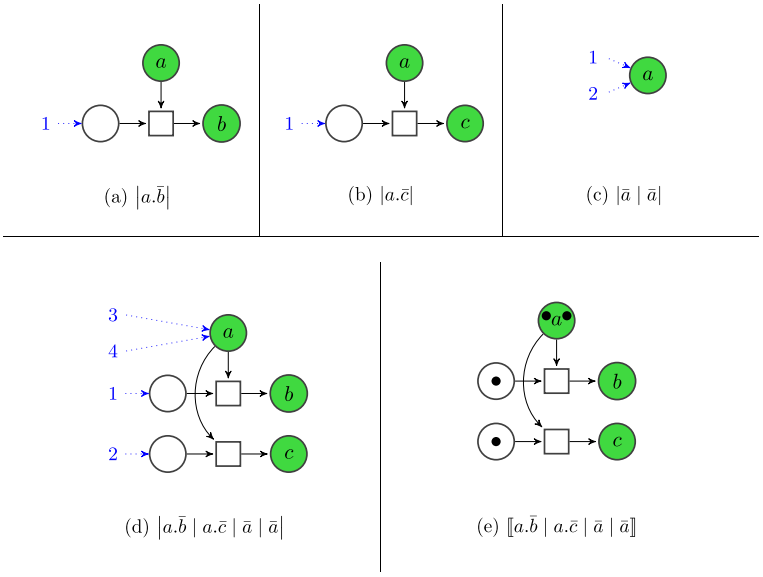

Fig. 12. The encoding of the process $a.\bar{b} \mid a.\bar{c} \mid \bar{a} \mid \bar{a}$.

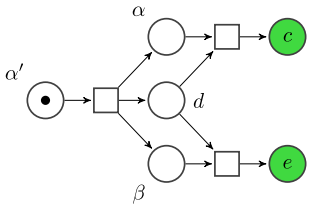

Fig. 13. Net encoding the process $(\nu d)(\tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d}))$.

Example 6 *(Inputs and outputs).* In Fig. 12 the encoding of the process $a.\bar{b} \mid a.\bar{c} \mid \bar{a} \mid \bar{a}$ is illustrated step by step. For the component $a.\bar{b}$ we have that $|a.\bar{b}|$ is $act_1^a \circ out_b$ which results in a net with a single transition. The unnamed left place (with an incoming pin) models the flow of control. When such a place will be marked, in order to progress by firing the transition, the net will require a token in the open place a, representing an output message on channel a. The firing of the transition consumes the message and produce a token in the output place b, representing an output message on channel b. The encoding of the component $a.\bar{c}$ is analogous. The component $\bar{a} \mid \bar{a}$ gives rise to the net $|\bar{a} \mid \bar{a}| = out_a \otimes out_a$ depicted in (c). Note that place a appears twice in the left interface, since the net represents two occurrences of the output message on channel a. The various components are then put in parallel (this requires some adjustments in the interfaces), merging the common open places (only a in this case) as depicted in (d). Finally, the encoding $[\![a.\bar{b} \mid a.\bar{c} \mid \bar{a} \mid \bar{a}]\!]$ is obtained by "activating" the left interface, namely by inserting tokens in the left places.

Example 7 *(Restricted and parallel processes).* Consider the process $Q = (\nu d)Q_1$, with $Q_1 = \tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d})$, previously introduced in Example 1. The net encoding $[\![Q]\!]$ is shown in Fig. 13. It is obtained by applying the $init(\cdot)$ operation to the net $|(\nu d)Q_1| = (\nu d)|Q_1|$. In turn, $|Q_1|$ results from the sequential composition of $dupl_1$ and $act_1^\tau \circ |Q_2|$, where $Q_2 = d.\bar{c} \mid d.\bar{e} \mid \bar{d}$. The net $|Q_2|$ coincides with the net in Fig. 9, but without output interface. Note that the place d is still 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.

Example 8 *(Bound processes).* Consider the process $Q = (\nu a)(a.a.nil \mid \bar{a})$. Its encoding is depicted in Fig. 14(left). Note that the second transition, corresponding to the second input on a in process Q, cannot fire since after the firing of the first transition place a is emptied and not filled again. This is consistent with ACCS reduction semantics: $Q \to (\nu a)a.nil$ and the remaining occurrence of a cannot be executed since there is no output action on a in parallel.

Now consider $R = !_b.Q \mid \bar{b} \mid \bar{b} = (!_b.(\nu a)(a.a.nil \mid \bar{a})) \mid \bar{b} \mid \bar{b}$, obtained by inserting process Q under a replication on b and then in a context offering two output messages $\bar{b}$, enabling the replication. Note that R is not bound because it contains a restriction operator under the replication.

The encoding of R, shown in Fig. 14(right), is obtained by marking the net $|R|$, which in turn is the parallel composition between $repl_2^b \circ |Q|$ and the net $b \otimes b$. The leftmost transition can fire twice, thus generating two tokens in both a and β. Also the rightmost transition modelling the second input action in Q can then be fired, in a way which disagrees with ACCS operational semantics.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

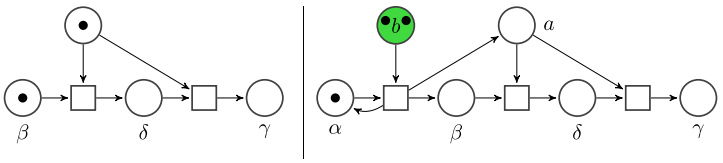

Fig. 14. Encodings for the processes $(\nu a)(a.a.nil \mid \bar{a})$ and $(!_b.(\nu a)(a.a.nil \mid \bar{a})) \mid \bar{b} \mid \bar{b}$.

Roughly speaking, the above problem arises since tokens corresponding to different occurrences of the replicated process are mixed in an improper way. Solving the problem by a different encoding, where each occurrence of a process involved in a replication corresponds to a different subnet in the encoding, would lead to an infinite net for processes that are not bound.

We conclude this section by observing that the encoding respects structural congruence, i.e., structural congruent processes are mapped into isomorphic nets and vice versa: the proof is laborious yet conceptually simple (see e.g. [25]).

Proposition 2. *Let P, Q be bound ACCS processes. Then, $P \equiv Q$ if and only if there is an isomorphism $f : [\![P]\!] \to [\![Q]\!]$ such that $f^{\oplus}(m_{[\![P]\!]}) = m_{[\![Q]\!]}$.*

4.2. Relating ACCS and open nets

This section shows that the encoding of ACCS processes preserves and reflects process reductions, as well as strong and weak bisimilarity. Hereafter we denote by $[\![P]\!]_\Gamma$ the encoding of a bound process P with respect to a set of names Γ, that is, $[\![P]\!]_\Gamma = init(|P| \otimes (\bigotimes_{a \in \Gamma} a))$. The addition of the component $\bigotimes_{a \in \Gamma} a$ to the encoding determines the presence of a place for each channel in Γ (which could possibly not occur free in P).

In order to state the correspondence results, we first need to establish a correspondence between the ACCS processes reachable from P, hereafter denoted by $reach(P) = \{Q : P \Rightarrow Q\}$, and the markings of the encoding $[\![P]\!]_\Gamma$.

The encoding of a process P is inductively defined as the composition of the encoding of its subprocesses. Since by definition whenever an ACCS process P performs a reduction to P', the process P' is obtained from P by replacing a subprocess with its reduct, it is not difficult to see that the encoding of processes reachable from P can be mapped to subnets of $[\![P]\!]_\Gamma$. We denote by $SN_{[\![P]\!]}$ the set of places of $[\![P]\!]$ which corresponds to names in P.

Lemma 1 *(Reachable processes as subnets). Let P be a bound ACCS process, $Q \in reach(P)$ and Γ a set of names. Then, an open net morphism $f_{Q \to P} : [\![Q]\!]_\Gamma \to [\![P]\!]_\Gamma$ can be uniquely chosen, which is injective on $SN_{[\![Q]\!]_\Gamma}$.*

The proof, detailed in Appendix A, relies on the fact that given a subprocess Q of a process P, a mapping of $[\![Q]\!]_\Gamma$ into $[\![P]\!]_\Gamma$ can be obtained by the inductive definition of the encoding. The open places representing the names Γ of $[\![Q]\!]_\Gamma$ are mapped in the corresponding places of $[\![P]\!]_\Gamma$. Then, by using the fact that each subprocess of P corresponds to a subnet of $[\![P]\!]_\Gamma$, it is not difficult to prove that also the encoding of a process reachable from P can be mapped to a subnet of $[\![P]\!]_\Gamma$. In fact, the processes in $reach(P)$ consist of compositions of reducts of subprocesses of P, where, due to replication, for some reducts we may have several parallel copies. The encodings of these copies, since by definition they do not contain restriction operators, can be mapped to the same subnet.

The injectivity condition ensures that the mapping $f_{Q \to P}$ does not collapse places of $[\![Q]\!]_\Gamma$ which correspond to different names. The property clearly holds for the places in $O_{[\![P]\!]}$ corresponding to the free names: it is implied by the fact that the labelling function is injective and preserved by net morphisms. As for the places in $SN_{[\![Q]\!]} \setminus O_{[\![Q]\!]}$, the property holds precisely thanks to the restriction to bound processes. Consider again the encoding in Fig. 14, with processes $P = (!_b.(\nu a)(a.a.nil \mid \bar{a})) \mid \bar{b} \mid \bar{b}$ and $Q = (!_b.(\nu a)(a.a.nil \mid \bar{a})) \mid (\nu a)(a.a.nil \mid \bar{a})) \mid \bar{b} \in reach(P)$. Now, the only possible morphism from $[\![Q]\!]$ to $[\![P]\!]$ would collapse the places corresponding to the two bound occurrences of a.

By using the lemma above, we can easily define a correspondence between the processes belonging to $reach(P)$ and the markings of $[\![P]\!]_\Gamma$.

Definition 27 *(Reachable processes as markings).* Let P be a bound ACCS process. The function $\mathbf{m}^P_\Gamma : reach(P) \to S^{\oplus}_{[\![P]\!]_\Gamma}$ maps any process $Q \in reach(P)$ into the marking $f^{\oplus}_{Q \to P}(m_{[\![Q]\!]_\Gamma})$.

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

Theorem 1 *(Process reductions as net firings). Let P be a bound ACCS process, $Q \in reach(P)$ and $\Gamma \subseteq \mathcal{N}$ a set of names. Then*

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

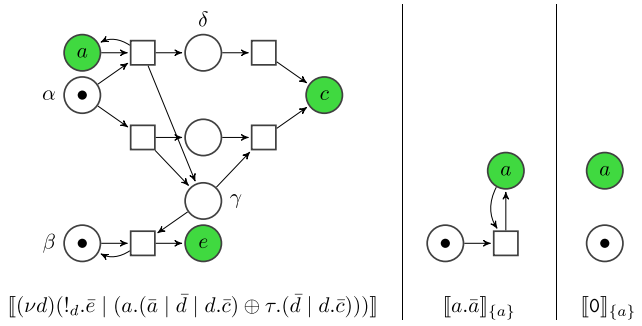

Fig. 15. Encodings for ACCS processes.

1. *if* $Q \to R$ *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* $Q \to R$ *for some* R *with* $m = \mathbf{m}^P_\Gamma(R)$.

Proof. See Appendix A. □

The result establishes a bijection between the reductions performed by any process $Q \in \mathit{reach}(P)$ and the firings in $[\![P]\!]_\Gamma$ from the marking $\mathbf{m}^P_\Gamma(Q)$.

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 ACCS processes and* Γ *a set of names such that* $\mathtt{fn}(P) \cup \mathtt{fn}(Q) \subseteq \Gamma$. *Then for* $\approxeq$ *either strong or weak bisimilarity*

$$P \approxeq Q \quad \textit{if and only if} \quad \mathbf{m}^P_\Gamma(P) \approxeq \mathbf{m}^Q_\Gamma(Q)$$

Proof. See Appendix A. □

Example 9. 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 $Q = (\nu d)(\tau.(d.\bar{c} \mid d.\bar{e} \mid \bar{d}))$ as in Example 1, and the set $\Gamma = \{a, c, e\}$ including the free names of both processes. The encoding of $[\![P]\!]_\Gamma$ is depicted in Fig. 15(left), while $[\![Q]\!]_\Gamma$ corresponds to the net in Fig. 13 with an additional isolated open place labelled a.

Since $P \sim Q$, according to Theorem 2 also the markings $\mathbf{m}^P_\Gamma(P) = \{\alpha, \beta\}$ in $[\![P]\!]_\Gamma$ and $\mathbf{m}^Q_\Gamma(Q) = \{\alpha'\}$ in $[\![Q]\!]_\Gamma$ are strongly bisimilar. Clearly, they exhibit the same behaviour as far as hidden transitions are concerned. Moreover, when the environment inserts a token into a, in $[\![P]\!]_\Gamma$ we obtain the marking $\{\alpha, \beta, a\}$ enabling the upper most transition and leading to $\{\beta, \delta, \gamma, a\}$. This is clearly equivalent to fire the other hidden transition.

Consider now the net encodings $[\![a.\bar{a}]\!]_{\{a\}}$ in Fig. 15(centre) and $[\![\mathit{nil}]\!]_{\{a\}}$ in Fig. 15(right). In both cases the only observable actions are either placing or removing a token on the open place a. In fact, in $[\![a.\bar{a}]\!]_{\{a\}}$ the execution of the hidden transition, consuming and producing a token in a, is unobservable when considering weak bisimilarity. The two nets are thus weakly bisimilar, consistently with the fact that $a.\bar{a} \approx \mathit{nil}$.

5. From CSP processes to nets

In this section we turn our attention to CSP and provide an open net encoding for CSP processes. As before, we prove that the encoding preserves and reflects the behaviour of processes. The encoding, also in the case of CSP, will be restricted to *bound* processes, where parallel composition can occur under the scope of replications only in the form $|_\emptyset$, that is, without synchronisation.

5.1. Encoding CSP processes as open nets

The encoding of CSP processes is based on some constant nets already introduced in Fig. 10, namely the net *nil* which encodes the process *nil*, the net dupl_i used as a combinator for the internal and external choice and the net act^τ_i which models the behaviour of the internal choice. In addition, we exploit the nets in Fig. 16, explicitly designed for CSP. The net ev^a_i (on the left), where $a \in \Sigma$, models an event a. It consists of a visible transition labelled a whose post-set is the right interface of the net, which will match the left interface of the encoding of the continuation of a. The net repl^a_i (on the right), with $a \in \Sigma$, by repeated firing of transition a allows for an arbitrary number of "parallel activations" of the net which follows. This will be used as a combinator for replication, where the cardinality of the right interface i will match that of the left interface of the encoding of the process under the replication operator.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

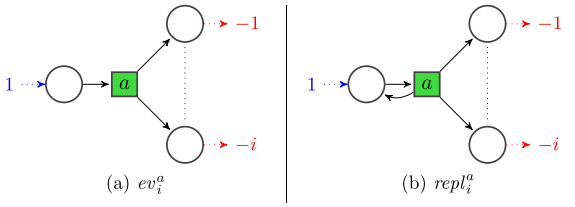

Fig. 16. The constant nets ev_i^a and $repl_i^a$.

$$
\begin{array}{lcl}
|nil| & = & nil \\
\left|\bigoplus_{j=1}^{n} a_i.P_i\right| & = & dupl_n \circ \left\{\bigotimes_{j=1}^{n} \left(ev^{a_j}_{sdeg(P_j)} \circ |P_j|\right)\right\} \\
|P_1 + P_2| & = & dupl_2 \circ \left\{\bigotimes_{j=1}^{2} \left(act^{\tau}_{sdeg(P_j)} \circ |P_j|\right)\right\} \\
|!_a.\, P| & = & repl^a_{sdeg(P)} \circ |P| \\
|P\backslash X| & = & |P|\backslash X \\
|P_1 \,|_X\, P_2| & = & |P| \otimes_X |Q|
\end{array}
$$

Fig. 17. Encoding for CSP processes.

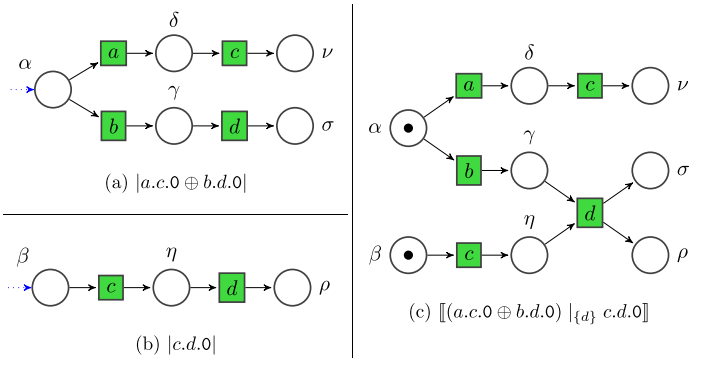

Fig. 18. Some process encodings.

The definition below introduces the encoding of CSP processes into nets. It uses the structural degree $sdeg(P)$ of a CSP process P, which is defined exactly as for ACCS (see Definition 25).

Definition 28 *(Net encoding for CSP).* Let P be a bound CSP process. The encoding of P, denoted by $[\![P]\!]$, is defined as $[\![P]\!] = init(|P|)$, where $|.|$ is given inductively according to the rules in Fig. 17.

The encoding of a process P is obtained by composing the encoding of its subprocesses and finally marking the left places. It therefore contains a place for each operator $!_a$, $+$, $\oplus$ and process nil of P. The error place can be used by the synchronised parallel composition of nets in order to keep some components inactive (see Definition 22). Transitions correspond to events. Recall that whenever two components are in a synchronised parallel composition a transition is inserted for each possible synchronisation, i.e., for each pair of events with the same name. Note that, differently from the ACCS encoding, places are all closed while visible events are modelled by visible transitions.

Example 10 *(Prefix and parallel synchronised processes).* Consider the process $P = (a.c.nil \oplus b.d.nil) \,|_{\{d\}}\, c.d.nil$. Its encoding is depicted in Fig. 18(right) where all transitions are visible. It is obtained by marking the left interface of the net $|P|$, the result of the parallel composition, synchronised on d, of the encodings $|a.c.nil \oplus b.d.nil|$ and $|c.d.nil|$, in turn depicted in the left side of Fig. 18. Let us focus on the net on the upper part, which illustrates $|a.c.nil \oplus b.d.nil|$. The places ν and σ represent the subnets encoding the nil processes (those reached after the events c and d, respectively). The subnet rooted at place δ is the encoding of the subprocess $c.nil$, namely $ev_1^a \circ nil$. Analogously, the subnet rooted at γ is the encoding of $d.nil$. The encoding of the subprocess $a.c.nil \oplus b.d.nil$ is obtained by sequentially composing the net $dupl_2$ with $(ev_1^a \circ |c.nil|) \otimes (ev_1^b \circ |d.nil|)$.

As in the case of ACCS the encoding is limited to bound processes. The following example provide an intuitive motivation of this restriction.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

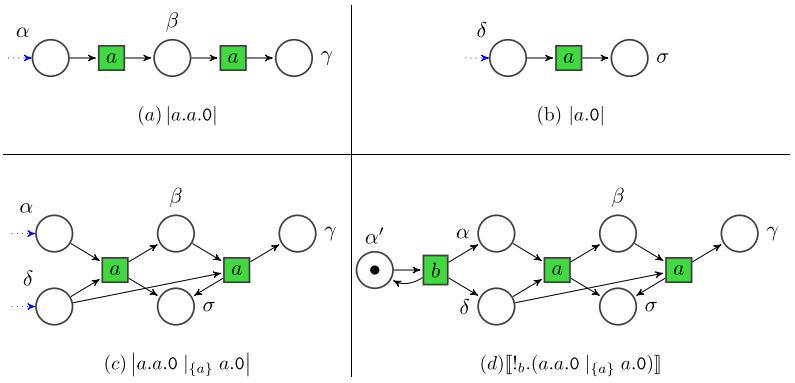

Fig. 19. Process encodings.

Example 11 *(Bound processes).* Consider the process $Q = a.a.nil \mid_{\{a\}} a.nil$. The encodings $|a.a.nil|$ and $|a.nil|$ are depicted in Fig. 19(a) and Fig. 19(b), respectively. The encoding $|Q|$ is obtained as the parallel composition, synchronised on a, between $|a.a.nil|$ and $|a.nil|$, as shown in Fig. 19(c). Each transition labelled by a of $|a.a.nil|$ is "combined" with any other transition labelled by a in $|a.nil|$. Observe that the second a-labelled transition in the encoding of Q cannot fire since after the firing of the first a-labelled transition, place δ is emptied and never filled again. This is consistent with the operational semantics of CSP where $Q \xrightarrow{a} a.nil \mid_{\{a\}} nil$, in such a way that the remaining occurrence of a cannot be executed since it has no counterpart in the parallel subprocess.

Now consider $R = !_b.Q = !_b.(a.a.nil \mid_{\{a\}} a.nil)$, that is the process Q inserted in a replication. Observe that R is not a bound process as it contains a non-trivial parallel synchronised product (where synchronisation is over a non-empty set of events) under the scope of a replication.

The net $[\![R]\!]$ in Fig. 19(d) is obtained by marking $|R|$, which in turn is the sequential composition of $repl_2^b$ with $|Q|$. Note that the b-labelled transition can fire any number of times, thus generating an unbounded number of tokens in α and δ. Hence, also the second a-labelled transition has the opportunity of being fired, in a way which disagrees with the operational semantics of CSP.

Once again the problem arises since tokens corresponding to different occurrences of a replicated process are improperly mixed. So, also for CSP solving the problem would lead to an infinite net for processes that are not bound.

The net encoding for CSP respects the structural congruence.

Proposition 3. *Let P, Q be bound CSP processes. Then, $P \equiv Q$ if and only if there is an isomorphism $f : [\![P]\!] \to [\![Q]\!]$ such that $f^{\oplus}(m_{[\![P]\!]}) = m_{[\![Q]\!]}$.*

5.2. Relating CSP and open nets

In this section we show that any bound CSP process and its net encoding behave essentially in the same way. More precisely, the net encoding of CSP processes preserves and reflects process transitions, and, consequently, the standard behavioural equivalences for CSP like, for instance, trace equivalence.

In order to state these results, as in the case of ACCS, we establish a correspondence between $reach(P) = \{Q : P \xrightarrow{\omega}{}^{*} Q\}$, the processes reachable from P, and the markings reachable in the net $[\![P]\!]$. We denote by $TS_{[\![P]\!]}$ the set of transitions of $[\![P]\!]$ which arise as synchronisations, namely which correspond to events a that occur under the scope of a synchronisation $|_X$ with $a \in X$. These are exactly those transitions whose pre-set has cardinality greater than one.

Lemma 2 *(Reachable processes as subnets). Let P be a bound CSP process and $Q \in reach(P)$. Then, an open net morphism $f_{Q \to P} : [\![Q]\!] \to [\![P]\!]$ can be uniquely chosen, which is injective on $TS_{[\![Q]\!]}$.*

As it happened for ACCS, also in the encoding of CSP processes parallel copies of the same subprocess are mapped to the same subnet. This does not lead to a violation of the injectivity requirement since, by definition, the replicated subprocesses do not synchronise on any event. The proof follows the same outline as the one for ACCS. More details are given in Appendix B.

By using the lemma above, we define a correspondence between the processes in $reach(P)$ and markings of $[\![P]\!]$.

Definition 29 *(Reachable processes as markings).* Let P be a bound CSP process. The function $\mathbf{m}^P : reach(P) \to S^{\oplus}_{[\![P]\!]}$ maps any process $Q \in reach(P)$ into the marking $f^{\oplus}_{Q \to P}(m_{[\![Q]\!]})$.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

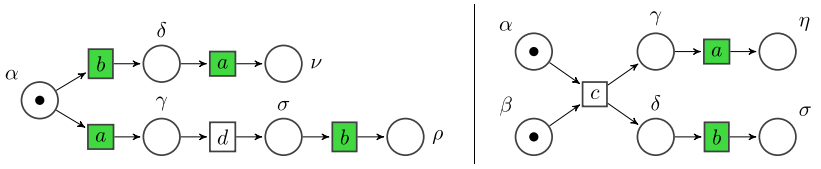

Fig. 20. Encoding for $a.(d.b.nil\backslash d) \oplus b.a.nil$ and $(c.a.nil \mid_{\{c\}} c.b.nil)\backslash c$.

Now, we can state the two main correspondence results for the CSP encoding.

Theorem 3 *(Process transitions as net firings). Let P be a bound CSP process and $Q \in reach(P)$. Then*

1. *if $Q \xrightarrow{\mu} R$ then $\mathbf{m}^P(Q) \xrightarrow{\mu} \mathbf{m}^P(R)$ in $[\![P]\!]$;*
2. *if $\mathbf{m}^P(Q) \xrightarrow{\mu} m$ in $[\![P]\!]$ then $Q \xrightarrow{\mu} R$ for some R such that $m = \mathbf{m}^P(R)$.*

Proof. See Appendix B. □

The result establishes a bijection between the labelled transitions performed by any process $Q \in reach(P)$ and the firings in $[\![P]\!]$ from the marking $\mathbf{m}^P(Q)$. Such a bijection can then be lifted to a fundamental correspondence between the trace semantics in the two formalisms.

Theorem 4. *Let P, Q be bound processes. Then*

$$P \approx_T Q \quad \text{if and only if} \quad [\![P]\!] \approx_T [\![Q]\!].$$

Example 12. Recall the processes of Example 2, namely $P = a.(d.b.nil\backslash d) \oplus b.a.nil$ and $Q = (c.a.nil \mid_{\{c\}} c.b.nil)\backslash c$. The encoding of P is the net in Fig. 20(left), while the encoding of Q is in Fig. 20(right). As prescribed by Theorem 3, there is a correspondence between the labelled transitions of each process and those of its encoding. For instance, $P \xrightarrow{a} d.b.nil\backslash d$ corresponds to $\mathbf{m}^P(P) = \{\alpha\} \xrightarrow{a} \{\gamma\} = \mathbf{m}^P(d.b.nil\backslash d)$. The transition $d.b.nil\backslash d \xrightarrow{\tau} b.nil\backslash d$ corresponds to $\mathbf{m}^P(d.b.nil\backslash d) = \{\gamma\} \xrightarrow{\tau} \{\sigma\} = \mathbf{m}^P(b.nil\backslash d)$ and $b.nil\backslash d \xrightarrow{b} nil\backslash d$ to $\mathbf{m}^P(b.nil\backslash d) = \{\sigma\} \xrightarrow{b} \{\rho\} = \mathbf{m}^P(nil\backslash d)$. Moreover, $P \xrightarrow{b} a.nil$ corresponds to $\mathbf{m}^P(P) = \{\alpha\} \xrightarrow{b} \{\delta\} = \mathbf{m}^P(a.nil)$, and finally $a.nil \xrightarrow{a} nil$ to $\mathbf{m}^P(a.nil) = \{\delta\} \xrightarrow{a} \{\nu\} = \mathbf{m}^P(nil)$. It is also easy to see that, as it follows from $P \approx_T Q$ and Theorem 4, the net encodings $[\![P]\!]$ and $[\![Q]\!]$ are trace bisimilar.

6. On the technology transfer

The proposed encodings of process calculi into (open) Petri nets, thanks to the tight correspondence they establish between the structure and behaviour of the original process and its net encoding, enable to transfer results concerning expressiveness and tractability from one formalism to the other. We next discuss this possibility for the two cases considered in the paper, namely ACCS and CSP.

6.1. Undecidability of bisimilarity for open nets

This section shows the undecidability of (strong and weak) bisimulation equivalence of open nets, taking advantage from the encodings we have proposed. Observe that while for general open nets these undecidability results immediately follows from the undecidability of trace and bisimulation equivalence for ordinary labelled Petri nets, for those open nets where all transitions are hidden (and thus are considered indistinguishable in the strong case and unobservable in the weak case) the results were previously unknown.

We first notice that trace equivalence is obviously undecidable for both bound CSP processes and labelled open nets (since they include as a fragment the basic parallel processes for which trace equivalence is known to be undecidable [26]). Even though this does not give new insights, it is worth noting that, by using the encoding, the undecidability of trace equivalence for Petri nets can be deduced directly from the undecidability of trace equivalence for bound CSP.

Concerning open nets where all transitions are hidden (and thus unlabelled), we first show that (strong and weak bisimilarity) for bound ACCS processes is undecidable. In particular, the result about bisimilarity answers a question faced for the synchronous case in [18]. Then, by using Theorem 2 we deduce that the same result holds for open nets, a previously unknown fact.

The undecidability of bisimilarity for bound ACCS processes is proved by reduction to the halting problem for Minsky's two-register machines, adapting a proof technique originally proposed in [19].

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

Definition 30 *(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 $\gamma(P)$, which do not correspond to a correct computation of the program P (due to the absence of zero-tests in ACCS). Still, this can be used to prove undecidability of (strong and weak) bisimulation equivalence. In fact, given P, a second process $\gamma'(P)$ can be built such that $\gamma(P)$ and $\gamma'(P)$ are behaviourally equivalent if and only if program P does not terminate. Therefore, deciding the corresponding equivalence for bound ACCS processes would allow to decide the termination of two-register machines. As two-register machines are Turing powerful, we can state the result below.

Theorem 5 *(Undecidability of ACCS bisimilarity). Strong and weak bisimilarity are undecidable for bound ACCS processes.*

Proof. See Appendix C. □

The properties of the encoding (Theorem 2) ensure that the same undecidability results hold for open nets, even if we restrict to nets with no visible transition.

Corollary 1 *(Undecidability of open net bisimilarity). Weak and strong bisimilarity are undecidable for open nets with no visible transitions.*

The results above can be easily adapted to other behavioural equivalences, such as trace equivalence [27] or failure equivalence [18].

6.2. Decidability of ACCS and CSP properties

In this section we use the encodings for transferring some decidability results from Petri nets to the bound fragments of ACCS and CSP. More precisely, by using the fact that properties like reachability, boundedness and presence of deadlocks are decidable for nets, we show that analogous properties are decidable for bound ACCS and CSP (thus, in particular, they are not Turing powerful).

It is worth remarking that the properties we consider on ACCS and CSP regards their operational behaviour (reduction semantics for ACCS and labelled transitions for CSP). This is captured in the encodings by transition firings only. Hence the presence of open places in the encodings is irrelevant for this section and the net encodings can be seen as ordinary (labelled, for CSP) Petri nets.

Let us start with reachability. Note that this property, seen as the possibility of reaching a given process Q via a sequence of transitions from a start process P, is not particularly meaningful in this context. Since during process evolution the number of parallel components can only increase, even though some components could terminate or get stuck, the property turns out to be decidable. Indeed, in order to establish whether Q is reachable it suffices to consider the fragment of the transition system including the processes reachable from P having a number of parallel subprocesses bounded by that of Q. A more interesting property is the reachability under the garbage collection of deadlock processes *nil*, since it breaks the monotonicity we mentioned above. However, for CSP the removal itself is not trivial: it is in general false that $P \mid_X nil$ is trace equivalent to P.

Alternatively, one can consider control state reachability, i.e., the reachability of a configuration including a given subprocess. Through the encoding, control state reachability in process calculi can be reduced to coverability in Petri nets. It is folklore that the control state reachability problem is undecidable for Turing complete process calculi like CCS and CSP, while the corresponding property of coverability is known to be decidable for Petri nets [28]. By exploiting the encoding, we can deduce decidability of control state reachability for bound ACCS and CSP processes. For a given process P, below we denote by $subreach(P)$ the set of subprocess of processes reachable from P, namely $subreach(P) = \{Q \mid \exists R \in reach(P).\ Q \text{ subprocess of } R\}$.

Proposition 4 *(Decidability of control state reachability). Let P, Q be bound ACCS or CSP processes. The problem of establishing whether $Q \in subreach(P)$ is decidable.*

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

Proof. In the case of CSP, by Theorem 3 we have that $Q \in subreach(P)$ if and only if there exists an open net morphism $f : [\![Q]\!] \to [\![P]\!]$ such that $f_S^{\oplus}(m_{[\![Q]\!]})$ is coverable in $[\![P]\!]$, which – for CSP – is a net not including open places.

By Theorem 1 the same holds also for ACCS, with the proviso that $f_S^{\oplus}(m_{[\![Q]\!]})$ should be coverable in $[\![P]\!]$ by using only hidden transitions, that is, $f_S^{\oplus}(m_{[\![Q]\!]})$ must be coverable in the net obtained by $[\![P]\!]$ by closing the open places.

Exploiting this fact, since the number of morphisms $f : [\![Q]\!] \to [\![P]\!]$ is finite, decidability of coverability for Petri nets immediately implies decidability of control state reachability for bound ACCS and CSP processes. □

The correspondence between ACCS reductions/CSP transitions and net firings can be exploited to obtain other results for bound processes. For example, since boundedness is decidable for Petri nets [28], it is possible to determine whether a bound process can reach only a finite number of states.

Proposition 5 *(Finite state). Let P be a bound ACCS or CSP process. The problem of establishing whether $reach(P)$ is finite is decidable.*

Proof. Let P a CSP process. The result immediately follows by observing that, thanks to Theorem 3, we have that $reach(P)$ is finite if and only if in $[\![P]\!]$ the set of reachable markings is finite. An analogous argument holds for ACCS processes. It suffices to observe that, since by Theorem 1 reductions of a process P corresponds to firings of hidden transitions in $[\![P]\!]$, we need to consider only those markings reachable by firing hidden transitions, namely markings reachable in the net obtained from $[\![P]\!]$ by closing the open places. □

Analogously, it is possible to identify an upper bound to the degree of parallelism of a bound ACCS or CSP process, i.e., to the number of parallel subcomponents of a process during its evolution. Define the degree of P as $deg(P) = \sup\{sdeg(P') : P' \in reach(P)\}$. For both calculi, the close correspondence between $deg(P)$ and the maximal total number of tokens in the reachable markings of $[\![P]\!]$ immediately leads to the following.

Proposition 6 *(Parallelism). Let P be a bound ACCS or CSP process. The problem of determining whether $deg(P)$ is finite is decidable. Moreover, for any given $k \in \mathbb{N}$, the problem of determining whether $deg(P) \leq k$ is decidable.*

Another property which is often considered when studying the expressiveness of process calculi is convergence, i.e., the existence of a terminating computation. A process P is called *convergent* (according to [29]) if there exists process $Q \in reach(P)$ such that Q is deadlocked (it cannot further progress). Again, as an immediate corollary of the tight correspondence between the operational semantics of a bound process and of its net encoding, as established by Theorems 1 and 3, convergence of a process can be reduced to the presence of deadlocks in a Petri net, a property which is known to be decidable [28].

Corollary 2 *(Convergence). Convergence is decidable for bound processes.*

In [18] it is shown that, in the synchronous case, adding priorities radically changes the situation: bound CCS becomes Turing complete and convergence is thus undecidable. It is almost immediate to show that the same applies to the asynchronous case. The fact that adding priorities makes bound ACCS Turing complete is proved by noting that, by using priorities, the encoding of two-register machines into bound ACCS (denoted by $\gamma(.)$ in Appendix C) can be made deterministic. This is not surprising as, on the net side, priorities are strictly connected to inhibitor arcs, which make Petri nets Turing powerful [30].

7. Related works

The open net model proposed in this paper is a mild variation of the one in [14], with the addition of visible transitions and explicit interfaces. In the literature a number of reactive extension of Petri nets, endowed with suitable composition operators, have been considered. Some approaches emphasise the algebraic view, identifying a set of operators which allow one to build complex nets starting from a set of predefined basic components. For instance, in the *Petri box calculus* [31–33], a special class of nets, called *plain boxes*, provides the basic components which are then combined by means of refinement-based operations. The *Petri nets with interface* [34,35] are based on an algebra of (labelled) Petri nets with interfaces. The interface consists of a set of public (output) places and (input) transitions and the operators allow one to add new transitions and places, to connect existing public transitions and places by new arcs or to hide items. The latter work has a natural evolution in the encoding of Petri nets into bigraphical reactive systems [15], which in turn has been the inspiration for open *CPR nets* [36,37]. Other approaches are more "component-oriented" and put the emphasis on the mechanisms which allow one to build larger systems by combining nets with clearly identified interfaces. The book [38] proposes a framework for net composition where a so-called daughter net can be inserted into a host net, by joining the two nets along a predefined set of places, playing the role of open places. Interestingly, the same book [38] also focuses on an alternative approach to net composition, based on an operation of synchronised parallel product in the style of [13], which joins two

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

nets by forcing the synchronisation of transitions with the same label. This is essentially the same mechanism that we used for synchronising transitions. *Petri net components* [39] again offer an interface with input and output places along with an operation for combining components by connecting the input places of a component to the output places of the other, and vice versa. There are close similarities with many of these models. Indeed, in designing our open net model we followed the intuition that encoding synchronous and asynchronous interactions would require a reactive extension exposing both places and transitions to the environment, and took from the literature the features needed for doing this in the simplest way.

The idea of mapping CSP processes into nets arose early on, see among others [40–42]. Conceptually, all these encodings are syntax-driven: each process is split into a family of sequential components, which represent the places of a net, and a (possibly concurrent) semantics for the calculus is thus obtained. As of more recent advances, we are aware of [43]. There, an on-the-fly algorithm is devised for building (and optimising) a net from a CSP process by exploiting its transition system. In our encoding we followed the spirit of the former proposals, striving for modularity: the encoding itself has a denotational flavor, mapping each operator of the calculus into an operator on nets, and as a consequence preservation and reflection of CSP operational semantics are easily stated and proved. We believe that such clarity is due to the identification of the right CSP fragment. Indeed, it is noteworthy that in all those papers mentioned above the recursion of nested parallel processes is not allowed "because the set of places of the generated Petri net would be infinite" [43, p. 111]. Our paper lifts such a constraint: our chosen CSP fragment is not finite state, but rather it bounds the number of parallel processes synchronising on the same channel.

Concerning CCS [2], as already mentioned, the encoding of its synchronous version into Petri nets has been widely studied (see, e.g., [10–13]). Although not explicitly treated, synchronous two-party communications, where channels adopt a strict handshaking pattern, would naturally fit in our framework, by an easy adaptation of other encodings proposed in the literature (see in particular [13]). This would require the addition to our framework a two-party synchronisation operator for open Petri nets.

Coming to asynchronous interactions, their relation with Petri nets has received much less attention than the synchronous case. In general terms, most proposals we are aware of put their emphasis on the preservation of the operational semantics, while behavioural equivalences are seldom studied. This happens, e.g., in the net encoding of the join calculus, where communication is asynchronous, presented in [44]. In particular, the fragment of the join calculus with no name passing and process generation is shown to correspond to ordinary P/T nets, while, in order to encode wider classes of join processes, high-level 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 [44,45] exploit the encoding to compare the expressiveness of Linda-like calculi with various communication primitives. In [46] 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 for KLAIM the techniques available for net verification. Concrete results in this direction are obtained in [47], where finite control π-calculus processes are encoded as safe Petri nets and verified using an unfolding-based technique.

The undecidability result that we obtained for asynchronous CCS is analogous to that in [18] for the synchronous case: building upon [29], that paper offers some results concerning the expressive power of restriction and its interplay with replication in synchronous CCS-like calculi.

8. Conclusions and further works

In this paper we presented a framework for the modular encoding of process calculi into open Petri nets, a reactive generalisation of ordinary nets with distinguished sets of open places and visible transitions, which are accessible from the environment. Interestingly enough, asynchronous interactions are captured by interaction over open places, while synchronous interactions are realised by letting net components interact over visible transitions. This translates at a formal level the intuition that in Petri nets the token flow is eminently asynchronous, while transitions synchronise different token flows. Our encodings are syntax-driven, hence modular, mapping each process operator into a suitable one for nets. Actually, in order to allow an inductive construction of open Petri nets from a set of constants, we considered open nets enriched with interfaces, which are used for net composition.

In particular, we have detailed the encodings of ACCS and CSP, representing paradigmatic instances of two main alternatives concerning the pattern of communications, namely, asynchronous message passing and broadcast synchronisation, respectively. For both calculi, we have identified an expressive fragment by proposing an encoding which, besides respecting the structural congruence of processes, preserves as well as reflects the operational semantics and, consequently, their behavioural equivalences.

Even though the study of asynchronous interaction has been tailored over ACCS, we feel confident that it can be generalised 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 [12,44,46], 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.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

We believe that the tight connection between open nets and synchronous and asynchronous calculi is quite enlightening. First of all, most of the encodings we are aware of focus on the preservation of variants of reachability or of the operational behaviour [44–46,48]. Instead, ours encoding allows us to establish a correspondence at the observational level. Most importantly, it allows for a fruitful "technology transfer". For ACCS, we proved the undecidability of strong and weak bisimilarity for bound processes, answering a question faced for the synchronous case in [18]. Through the encoding this result can be used to prove the undecidability of bisimilarity also for open nets without visible transitions, a previously unknown fact. Conversely, by using the fact that reachability/coverability is decidable for Petri nets, through the encoding we prove that various properties like (control state) reachability and convergence are decidable for bound ACCS and CSP.

Acknowledgements

We are grateful to the referees of the conference papers where we originally presented the results further developed here. In particular, we are indebted to the referee that pointed out the relevance of control state reachability in process calculi. Our gratitude goes also to the reviewers of this submission whose detailed and insightful comments helped improving our work.

Appendix A. Proofs for Section 4.2

In order to prove Theorem 1 we extend Lemma 1 and Definition 27, by considering not only the set of processes reachable from a given process P, but also all the corresponding subprocesses, hereafter denoted by $subreach(P)$.

We start with a simple technical observation.

Lemma 3 *(Subprocesses as subnets). Let P be an ACCS process, Γ a finite set of names and Q a subprocess of P. Then an injective open net morphism $f_{Q\to P} : [\![Q]\!]_\Gamma \to [\![P]\!]_\Gamma$ can be uniquely chosen.*

Proof. The result follows from the observation that the encoding $[\![P]\!]_\Gamma$ of process P is defined inductively relying on the encoding of its subprocesses, a fact that naturally provides a mapping $f_{Q\to P} : [\![Q]\!]_\Gamma \to [\![P]\!]_\Gamma$. When building the encoding of a larger process, the pre- and post-set of transitions can only be made larger, hence $f_{Q\to P}$ reflects pre- and post-sets, as required by the definition of net morphism (Definition 13). Moreover, $f_{Q\to P}$ is an open net morphism. In fact, no transition is visible, hence condition 1 of Definition 15 is trivially satisfied. Additionally, open places can be closed (as an effect of restrictions), but not the converse, hence also reflection of open places holds, as required to be an open net morphism (condition 2 in Definition 15). □

A generalisation of Lemma 1 holds, where also subprocesses of processes reachable from a process P are seen as markings of the net encoding $[\![P]\!]$. Recall that $SN_{[\![P]\!]}$ denotes the set of places of $[\![P]\!]$ that corresponds to names in P.

Lemma 4 *(Subprocesses of reachable processes as subnets). Let P be a bound ACCS process, $Q \in subreach(P)$ and Γ a finite set of names. Then an open net morphism $f_{Q\to P} : [\![Q]\!]_\Gamma \to [\![P]\!]_\Gamma$ can be uniquely chosen, which is injective on $SN_{[\![Q]\!]}$.*

Moreover, if $Q_1 \in subreach(Q) \cap subreach(P)$, then $f_{Q_1\to P} = f_{Q\to P} \circ f_{Q_1\to Q}$.

Proof. Consider a process $Q \in subreach(P)$. This means that there is a process Q' such that $P \Rightarrow Q'$ and Q is a subprocess of Q'. The existence of the open net morphism $f_{Q\to P} : [\![Q]\!]_\Gamma \to [\![P]\!]_\Gamma$ can be proved inductively on the length of the reduction sequence. When the length is 0, the result follows from Lemma 3. When the length is greater than 0, namely $P \Rightarrow P' \to Q'$, then we distinguish various cases according to the reduction rule used in the last step.

For rules (Syn) and (Tau), Q' is a subprocess of P', hence Q is a subprocess of P' and thus we conclude by inductive hypothesis.

For rule (Par), we have that $P' = P'_1 \mid P'_2$ and $Q' = P'_1 \mid Q'_2$ with $P'_2 \to Q'_2$. The inductive hypothesis provides a morphism $f_{Q_2\to P'_2} : [\![Q_2]\!]_\Gamma \to [\![P'_2]\!]_\Gamma$ for any subprocess Q_2 of Q'_2 and a morphism $f_{P'_2\to P} : [\![P'_2]\!]_\Gamma \to [\![P]\!]_\Gamma$. Hence we get an open net morphism $f_{Q_2\to P} : [\![Q_2]\!]_\Gamma \to [\![P]\!]_\Gamma$ for any subprocess Q_2 of Q'_2. Similarly, by inductive hypothesis we have a morphism $f_{P_1\to P} : [\![P_1]\!]_\Gamma \to [\![P]\!]_\Gamma$ for any subprocess P_1 of P'_1. Since the subprocesses of Q' are either subprocesses of Q'_1 or subprocesses of Q'_2 or Q' itself, we easily conclude.

For rule (Res), we have that $P' = (\nu a)P'_1$ and $Q' = (\nu a)Q'_1$ with $P'_1 \to Q'_1$. As above, the inductive hypothesis provides a morphism $f_{Q_1\to P'_1} : [\![Q_1]\!]_\Gamma \to [\![P'_1]\!]_\Gamma$ for any subprocess Q_1 of Q'_1 and a morphism $f_{Q'_1\to P} : [\![Q'_1]\!]_\Gamma \to [\![P]\!]_\Gamma$. Hence we get an open net morphism $f_{Q_1\to P} : [\![Q_1]\!]_\Gamma \to [\![P]\!]_\Gamma$ for any subprocess Q_1 of Q'.

For the case of rule (Repl) the boundness hypothesis plays a central role. In fact, $P' = !_a.P'_1 \mid \bar{a}$ and $Q' = !_a.P'_1 \mid P'_1$. Hence the inductive hypothesis, exactly as before, ensures the existence of open net morphisms $f_{P_1\to P} : [\![P_1]\!]_\Gamma \to [\![P]\!]_\Gamma$, satisfying the injectivity property, for any subprocess P_1 of $!_a.P'_1$ (and thus for any subprocess of P'_1). This naturally gives also an open net morphism for the entire $Q' = !_a.P'_1 \mid P'_1$, namely $f_{Q'\to P} : [\![Q']\!]_\Gamma \to [\![P]\!]_\Gamma$. Such a morphism maps the encoding of the two copies of P'_1 in Q' to the encoding of the unique copy of P'_1 in P'. However, this is still injective on

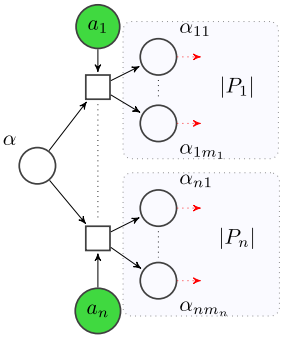

Fig. 21. Subnet corresponding to the process $\bigoplus_{i=1}^{n} a_i.P_i$.

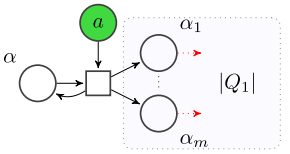

Fig. 22. Subnet corresponding to the process $!_a.\, Q_1 \mid \bar{a}$.

$SN_{[\![Q']\!]}$. In fact, by the boundness hypothesis, P_1' does not contains restrictions, hence places corresponding to names are open. Therefore in the encoding $[\![!_a.\, P_1' \mid P_1']\!]_\Gamma$ for any name there is a unique place. □

Definition 31 *(Subprocesses of reachable processes as markings).* Let P be a bound ACCS process. The function $\mathbf{m}_\Gamma^P : \mathit{subreach}(P) \to S^{\oplus}_{[\![P]\!]_\Gamma}$ maps any subprocess Q of a process reachable from P into the marking $f^{\oplus}_{Q\to P}(m_{[\![Q]\!]_\Gamma})$.

Now we are ready to provide a proof for Theorem 1, that we report below.

Theorem 1 *(Process reductions as net firings). Let P be a bound ACCS process, $Q \in \mathit{reach}(P)$ and $\Gamma \subseteq \mathcal{N}$ a set of names. Then*

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

Proof. (1) The proof is by induction on the depth of the derivation $Q \to R$.

- Assume that $Q \to R$ by applying rule (Syn). This means that $Q = \bigoplus_{i=1}^{n} \mu_i.P_i \mid \overline{a_j}$, $\mu_j = a_j$ with $j \in \{1,\ldots,n\}$, and $R = P_j$. By Lemma 4, in $[\![P]\!]_\Gamma$ there is the subnet of Fig. 21 and $\mathbf{m}_\Gamma^P(Q) = \{\alpha, a_j\}$. Thus $\mathbf{m}_\Gamma^P(Q) \xrightarrow{\tau} m'$ and $m' = \{\alpha_{j1},\ldots,\alpha_{jm_j}\} = f^{\oplus}_{R\to P}(m_{[\![R]\!]_\Gamma}) = \mathbf{m}_\Gamma^P(R)$.
- Assume that $Q \to R$ by applying rule (Repl). This means that $Q = !_a.\, Q_1 \mid \bar{a}$ and $R = !_a.\, Q_1 \mid Q_1$. By Lemma 4, in $[\![P]\!]_\Gamma$ there is the subnet of Fig. 22 and $\mathbf{m}_\Gamma^P(Q) = \{\alpha, a\}$. Thus $\mathbf{m}_\Gamma^P(Q) \xrightarrow{\tau} m'$ and $m' = \{\alpha, \alpha_1,\ldots,\alpha_m\} = f^{\oplus}_{R\to P}(m_{[\![R]\!]_\Gamma}) = \mathbf{m}_\Gamma^P(R)$.
- The cases of rules (Par) and (Res) are analogous. Hence we focus on (Par).
  Assume that $Q \to R$ by applying the (Par) rule. This means that $Q = Q_1 \mid Q_2$, $Q_1 \to Q_1'$, and $R = Q_1' \mid Q_2$.
  By definition we have $\mathbf{m}_\Gamma^P(Q) = f^{\oplus}_{Q\to P}(m_{[\![Q]\!]_\Gamma}) = f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]_\Gamma}) \oplus f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]_\Gamma})) = f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]_\Gamma})) \oplus f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]_\Gamma}))$. Now, since $Q_1 \to Q_1'$, by inductive hypothesis we know that $\mathbf{m}_\Gamma^{Q_1}(Q_1) \xrightarrow{\tau} \mathbf{m}_\Gamma^{Q_1}(Q_1') = f^{\oplus}_{Q_1'\to Q_1}(m_{[\![Q_1']\!]_\Gamma})$. Therefore, since Q_1 is a subprocess of Q, by Lemma 4 $f^{\oplus}_{Q_1\to Q}(\mathbf{m}_\Gamma^{Q_1}(Q_1)) \xrightarrow{\tau} f^{\oplus}_{Q_1\to Q}(f^{\oplus}_{Q_1'\to Q_1}(m_{[\![Q_1']\!]_\Gamma}))$.
  By definition, $f^{\oplus}_{Q_1\to Q}(\mathbf{m}_\Gamma^{Q_1}(Q_1)) = f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]_\Gamma})$ and by Lemma 4, since $Q_1' \in \mathit{subreach}(Q_1)$, $Q_1' \in \mathit{subreach}(Q)$ and $Q_1 \in \mathit{subreach}(Q)$, we also have $f^{\oplus}_{Q_1\to Q}(f^{\oplus}_{Q_1'\to Q_1}(m_{[\![Q_1']\!]_\Gamma})) = f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1']\!]_\Gamma})$. Moreover, we know that Q is reachable from P, Q_1 is a subprocess of it, and $Q_1' \in \mathit{subreach}(Q_1)$, therefore by Lemma 4 $f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]_\Gamma})) \xrightarrow{\tau} f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1'\to Q}(m_{[\![Q_1']\!]_\Gamma}))$.
  Now, by the operational semantics of open nets, $f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]_\Gamma})) \oplus f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]_\Gamma})) \xrightarrow{\tau} f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1'\to Q}(m_{[\![Q_1']\!]_\Gamma})) \oplus f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]_\Gamma})) = m$ and by using Lemma 4 it is possible to deduce $m = \mathbf{m}_\Gamma^P(R)$.
- Assume that $Q \to R$ because $Q \equiv Q_1$, $Q_1 \to R_1$, and $R_1 \equiv R$. Since $Q \equiv Q_1$, then by Proposition 2 we deduce that $[\![Q]\!]$ and $[\![Q_1]\!]$ are isomorphic and thus also $[\![Q]\!]_\Gamma$ and $[\![Q_1]\!]_\Gamma$ are. Analogously, since $R_1 \equiv R$, we have that $[\![R]\!]_\Gamma$ is

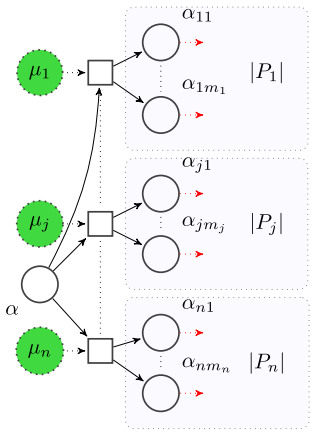

Fig. 23. Subnet corresponding to the process $\bigoplus_{i=1}^{n} \mu_i.P_i$.

isomorphic to $[\![R_1]\!]_\Gamma$. Moreover, since $Q_1 \to R_1$, we can apply the inductive hypothesis and obtain $\mathbf{m}^P_\Gamma(Q_1) \xrightarrow{\tau} \mathbf{m}^P_\Gamma(R_1)$. Therefore we can easily conclude that $\mathbf{m}^P_\Gamma(Q) \xrightarrow{\tau} \mathbf{m}^P_\Gamma(R)$.

(2) Assume $\mathbf{m}^P_\Gamma(Q) \xrightarrow{\tau} m$. The proof proceeds by structural induction on Q.

- Suppose $Q = nil$, $Q = \bar{a}$ or $Q = !_a.\, Q_1$. In all these cases the thesis trivially holds, because there are no transitions from $\mathbf{m}^P(Q)$.
- Suppose that $Q = \bigoplus_{i=1}^{n} \mu_i.P_i$. By Lemma 4 this means that the encoding of P includes a subnet like the one in Fig. 23, corresponding to the encoding of Q in P, where the places μ_i are dotted since they must be omitted for all i such that $\mu_i = \tau$. By construction, $\mathbf{m}^P(Q) = \{\alpha\}$ and, since $\mathbf{m}^P_\Gamma(Q) \xrightarrow{\tau} m$, there exists $j \in \{1, \ldots, n\}$ with $\mu_j = \tau$ (otherwise the corresponding transition could not fire). Moreover $m = \{\alpha_{j_1}, \ldots, \alpha_{j_{m_j}}\}$. By the operational semantics of ACCS, we immediately get that $Q \to P_j$ and, by Lemma 4, $\{\alpha_{j_1}, \ldots, \alpha_{j_n}\} = \mathbf{m}^P_\Gamma(P_j)$ as desired.
- Suppose that $Q = (\nu x)Q_1$. We know that $\mathbf{m}^P_\Gamma(Q) = f^{\oplus}_{Q \to P}(m_{[\![Q]\!]_\Gamma}) = f^{\oplus}_{Q \to P}(f^{\oplus}_{Q_1 \to Q}(m_{[\![Q_1]\!]_\Gamma}))$. By Lemma 4 we have $f^{\oplus}_{Q \to P}(f^{\oplus}_{Q_1 \to Q}(m_{[\![Q_1]\!]_\Gamma})) = f^{\oplus}_{Q_1 \to P}(m_{[\![Q_1]\!]_\Gamma}) = \mathbf{m}^P_\Gamma(Q_1)$. We thus have that $\mathbf{m}^P_\Gamma(Q_1) \xrightarrow{\tau} m$ and so $\mathbf{m}^{Q_1}_\Gamma(Q_1) \xrightarrow{\tau} m_1$ with $m = f_{Q_1 \to P}(m_1)$. Since $\mathbf{m}^{Q_1}_\Gamma(Q_1) \xrightarrow{\tau} m_1$, we can apply the inductive hypothesis and deduce $Q_1 \to Q_1'$ and $m_1 = \mathbf{m}^{Q_1}_\Gamma(Q_1')$. By the operational semantics $Q \to (\nu x)Q_1' = R$ and $m = f_{Q_1 \to P}(\mathbf{m}^{Q_1}_\Gamma(Q_1'))$, which by Lemma 4 is equal to $\mathbf{m}^P_\Gamma(Q_1') = \mathbf{m}^P_\Gamma(R)$.
- Suppose that $Q = Q_1 \mid Q_2$. By Lemma 4 we can show that $\mathbf{m}^P_\Gamma(Q) = \mathbf{m}^P_\Gamma(Q_1) \oplus \mathbf{m}^P_\Gamma(Q_2)$. We distinguish two cases. When, for some $i \in \{1, 2\}$ the fired transition is enabled by $\mathbf{m}^P_\Gamma(Q_1)$ or by $\mathbf{m}^P_\Gamma(Q_2)$ we can proceed by induction similarly to what we did for $Q = (\nu x)Q_1$.
  Otherwise, the firing of the transition requires the marking $\mathbf{m}^P_\Gamma(Q_1) \oplus \mathbf{m}^P_\Gamma(Q_2)$. This means that the encoding of one of the two processes in parallel, say Q_1, is performing a transition requiring a token in some place s and the other, say Q_2, is providing the token in s. By definition of the encoding the place s must be open. The transitions requiring a token in an open place originates either from the encoding of input prefixes, i.e., of processes of the shape $\bigoplus_{i=1}^{n} \mu_i.P_i$, or from a replication $!_a.\, P_1$. Accordingly, we distinguish two possibilities.
  - The first possibility is that the encoding of Q_1 contains the subnet shown in Fig. 21 and $\mathbf{m}^P_\Gamma(Q_1) = \{\alpha\} \oplus m_1$. Hence $Q_1 \equiv \bigoplus_{i=1}^{n} \mu_i.P_i \mid R_1$, with $m_1 = \mathbf{m}^P_\Gamma(R_1)$. The encoding of Q_2 must provide a token in some place μ_j. Since, by Lemma 4, the morphism $f_{Q \to P}$ is injective on places corresponding to names, we deduce that the encoding of Q_2 must include a place a_j which is marked, i.e., $\mathbf{m}^P_\Gamma(Q_2) = \{a_j\} \oplus m_2$, with $\mu_j = a_j$. Therefore $Q_2 \equiv \overline{\mu_j} \mid R_2$ with $m_2 = \mathbf{m}^P_\Gamma(R_2)$. By the operational semantics of ACCS, $Q \to P_j \mid R_1 \mid R_2$. Thus we conclude that the target marking is

$$\begin{aligned} m &= \{\alpha_{j1}, \ldots, \alpha_{jm_j}\} \oplus m_1 \oplus m_2 \\ &= \mathbf{m}^P_\Gamma(P_j) \oplus \mathbf{m}^P_\Gamma(R_1) \oplus \mathbf{m}^P_\Gamma(R_2) \\ &= \mathbf{m}^P_\Gamma(P_j \mid R_1 \mid R_2) \end{aligned}$$

    as desired.
  - The second possibility is that $[\![Q_1]\!]_\Gamma$ contains the subnet of Fig. 22 and $\mathbf{m}^P_\Gamma(Q_1) = \{\alpha\} \oplus m_1$, hence $Q_1 \equiv !_a.\, P_1 \mid R_1$ with $m_1 = \mathbf{m}^P_\Gamma(R_1)$. The encoding $[\![Q_2]\!]$ is providing the token in a. Since, by Lemma 4, the morphism $f_{Q \to P}$ is injective on places corresponding to names, we deduce that the encoding of Q_2 must include a place a, which

is marked, i.e., $\mathbf{m}^P_\Gamma(Q_2) = \{a\} \oplus m_2$. Hence $Q_2 \equiv \bar{a} \mid R_2$ with $m_2 = \mathbf{m}^P_\Gamma(R_2)$. By the operational semantics of ACCS $Q \to !_a.\, P_1 \mid P_1 \mid R_1 \mid R_2$. Thus the target marking is

$$\begin{aligned} m &= \{\alpha, \alpha_1, \ldots, \alpha_m\} \oplus m_1 \oplus m_2 \\ &= \mathbf{m}^P_\Gamma(!_a.\, P_1) \oplus \mathbf{m}^P_\Gamma(P_1) \oplus \mathbf{m}^P_\Gamma(R_1) \oplus \mathbf{m}^P_\Gamma(R_2) \\ &= \mathbf{m}^P_\Gamma(!_a.\, P_1 \mid P_1 \mid R_1 \mid R_2) \end{aligned}$$

as desired. □

In order to prove Theorem 2, we extend the domain of the function $\mathbf{m}^P_\Gamma$ by considering the set $reach(P)_\Gamma$, that is, the set of states reachable by freely adding in parallel output messages on the names in Γ. Given $P_2 \in reach(P)_\Gamma$ and $a_i \in \Gamma$ with $i \in \{1, \ldots, n\}$, we have $\mathbf{m}^P_\Gamma(P_2 \mid \overline{a_1} \mid \ldots \mid \overline{a_n}) = \mathbf{m}^P_\Gamma(P_2) \oplus a_1 \oplus \ldots \oplus a_n$.

Moreover, we also observe that Theorem 1 can be extended by considering processes $Q \in reach(P)_\Gamma$. The proof relies on the fact that if $Q \in reach(P)_\Gamma$, then there exists P' such that $Q \in reach(P')$ and $P' \equiv P \mid \overline{a_1} \mid \ldots \mid \overline{a_n}$, where $a_i \in \Gamma$ are the output messages added to reach from P the process Q.

Theorem 2. *Let P, Q be bound ACCS processes and Γ a set of names such that $\mathtt{fn}(P) \cup \mathtt{fn}(Q) \subseteq \Gamma$. Then for $\approxeq$ either strong or weak bisimilarity*

$$P \approxeq Q \quad \text{if and only if} \quad \mathbf{m}^P_\Gamma(P) \approxeq \mathbf{m}^Q_\Gamma(Q)$$

Proof. We focus on strong bisimilarity $\sim$. The proof for weak bisimilarity $\approx$ is completely analogous.

In order to prove that if $\mathbf{m}^P_\Gamma(P) \sim \mathbf{m}^Q_\Gamma(Q)$ then $P \sim Q$, we build the relation

$$R = \left\{ \left(P', Q'\right) \in reach(P)_\Gamma \times reach(Q)_\Gamma \;\middle|\; \mathbf{m}^P_\Gamma\left(P'\right) \sim \mathbf{m}^Q_\Gamma\left(Q'\right) \right\}$$

and we prove that it is a strong 1-bisimulation. Consider $(P', Q') \in R$. We have to prove that conditions (1)–(3) of Definition 7 hold.

1. if $P' \downarrow_{\bar{a}} P''$ then $Q' \downarrow_{\bar{a}} Q''$ and $(P'', Q'') \in R$
   By definition $P' \equiv \bar{a} \mid P''$, thus $\mathbf{m}^P_\Gamma(P') = a \oplus \mathbf{m}^P_\Gamma(P'')$. Note that $a \in \Gamma$, so a is an open place of $[\![P]\!]_\Gamma$ and $\mathbf{m}^P_\Gamma(P') \xrightarrow{a^-} \mathbf{m}^P_\Gamma(P'')$. Since $\mathbf{m}^P_\Gamma(P') \sim \mathbf{m}^Q_\Gamma(Q')$, then $\mathbf{m}^Q_\Gamma(Q') \xrightarrow{a^-} \mathbf{m}^Q_\Gamma(Q') \ominus a$ with $\mathbf{m}^P_\Gamma(P'') \sim \mathbf{m}^Q_\Gamma(Q') \ominus a$. This transition can be performed only if a belongs to $\mathbf{m}^Q_\Gamma(Q')$. Therefore $Q' \equiv \bar{a} \mid Q''$ and $\mathbf{m}^Q_\Gamma(Q'') = \mathbf{m}^Q_\Gamma(Q') \ominus a$. Thus $(P'', Q'') \in R$, as desired.
2. if $P' \to P''$ then $Q' \to Q''$ and $(P'', Q'') \in R$
   By Theorem 1, $P' \to P''$ implies that $\mathbf{m}^P_\Gamma(P') \xrightarrow{\tau} \mathbf{m}^P_\Gamma(P'')$ in $[\![P]\!]_\Gamma$. Since $\mathbf{m}^P_\Gamma(P') \sim \mathbf{m}^Q_\Gamma(Q')$, then $\mathbf{m}^P_\Gamma(Q') \xrightarrow{\tau} \mathbf{m}^Q_\Gamma(Q'')$ and $\mathbf{m}^P_\Gamma(P'') \sim \mathbf{m}^P_\Gamma(Q'')$. Again by Theorem 1 we have $Q' \to Q''$ and by construction $(P'', Q'') \in R$.
3. $\forall a \in \mathcal{N}.\, (P' \mid \bar{a}, Q' \mid \bar{a}) \in R$
   Let $a \in \mathcal{N}$. Without loss of generality we may assume that $a \in \Gamma$, since otherwise a would not belong to $\mathtt{fn}(P) \cup \mathtt{fn}(Q)$ and thus it could interact neither with P' nor with Q'. By construction, since $(P', Q') \in R$ we have that $\mathbf{m}^P_\Gamma(P') \sim \mathbf{m}^Q_\Gamma(Q')$. Now, since $a \in \Gamma$ is an open place, $\mathbf{m}^P_\Gamma(P') \xrightarrow{a^+} \mathbf{m}^P_\Gamma(P') \oplus a$. Hence we must have $\mathbf{m}^Q_\Gamma(Q') \xrightarrow{a^+} m'$ with $\mathbf{m}^P_\Gamma(P') \sim m'$. Necessarily, by the operational semantics of open nets, $m' = \mathbf{m}^Q_\Gamma(Q') \oplus a$ and thus $\mathbf{m}^P_\Gamma(P') \oplus a \sim \mathbf{m}^Q_\Gamma(Q') \oplus a$. Recalling that $\mathbf{m}^P_\Gamma(P' \mid \bar{a}) = \mathbf{m}^P_\Gamma(P') \oplus a$, and analogously $\mathbf{m}^Q_\Gamma(Q' \mid \bar{a}) = \mathbf{m}^Q_\Gamma(Q') \oplus a$ we immediately conclude that $(P' \mid \bar{a}, Q' \mid \bar{a}) \in R$.

Conversely, in order to prove that if $P \sim Q$ then $\mathbf{m}^P_\Gamma(P) \sim \mathbf{m}^Q_\Gamma(Q)$, one can easily show that the relation below is a strong bisimulation

$$\left\{ \left(\mathbf{m}^P_\Gamma\left(P'\right), \mathbf{m}^Q_\Gamma\left(Q'\right)\right) \mid \left(P', Q'\right) \in reach(P)_\Gamma \times reach(Q)_\Gamma \text{ and } P' \sim Q' \right\}. \qquad \square$$

Appendix B. Proofs for Section 5.2

As in the case of ACCS, in order to prove Theorem 3 we provide a technical lemma. The proof is omitted since it is completely analogous to that of Lemma 3.

Lemma 5 *(Subprocesses as subnets). Let P be a CSP process and Q a subprocess of P. Then an injective open net morphism $f_{Q \to P} : [\![Q]\!] \to [\![P]\!]$ can be uniquely chosen.*

Then we extend Lemma 2 and Definition 29 by considering the set $subreach(P)$, containing all the subprocesses of a process reachable from P.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

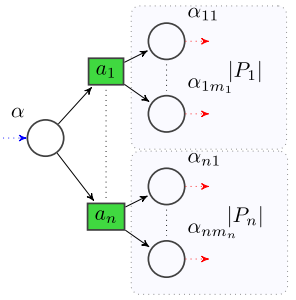

Fig. 24. Subnet corresponding to the process $\bigoplus_{i=1}^{n} a_i.P_i$.

Lemma 6 *(Subprocesses of reachable processes as subnets). Let P be a bound CSP process and $Q \in subreach(P)$. Then, an open net morphism $f_{Q\to P} : [\![Q]\!] \to [\![P]\!]$ can be uniquely chosen, which is injective on $TS_{[\![Q]\!]}$.*

Moreover, if $Q_1 \in subreach(Q) \cap subreach(P)$, then $f_{Q_1\to P} = f_{Q\to P} \circ f_{Q_1\to Q}$.

Proof sketch. The proof is analogous to that of Lemma 4. Consider a process $Q \in subreach(P)$. Then there is a process Q' such that $P \stackrel{\omega}{\Rightarrow} Q'$ and Q is a subprocess of Q'. The existence of the open net morphism $f_{Q\to P} : [\![Q]\!]_\Gamma \to [\![P]\!]_\Gamma$ can be proved inductively on the length of the derivation, distinguishing various cases according to the rule used in the last step.

This is a routine induction. The most interesting case, where the boundness hypothesis is fundamental, is when $P \stackrel{\omega}{\Rightarrow} P' \stackrel{\mu}{\longrightarrow} Q'$ and the last step uses rule (Repl). Then $P' = !_\mu.P'_1$ and $Q' = !_\mu.P'_1 \mid P'_1$. Using the inductive hypothesis we easily build a morphism $f_{Q'\to P} : [\![Q']\!] \to [\![P]\!]$ which maps the encoding of the two copies of P'_1 in Q' to the encoding of the unique copy of P'_1 in P'. Since, by the boundness hypothesis, P'_1 does not include synchronisations, we have that $TS_{[\![Q']\!]} = \emptyset$ and thus the injectivity requirement is trivially satisfied. □

Definition 32 *(Subprocesses of reachable processes as markings).* Let P be a bound CSP process. The function $\mathbf{m}^P : subreach(P) \to S^{\oplus}_{[\![P]\!]}$ maps any subprocess Q of a process reachable from P to the marking $f^{\oplus}_{Q\to P}(m_{[\![Q]\!]})$.

Theorem 3 *(Process transitions as net firings). Let P be a bound CSP process and $Q \in reach(P)$. Then*

1. *if $Q \stackrel{\mu}{\longrightarrow} R$ then $\mathbf{m}^P(Q) \stackrel{\mu}{\longrightarrow} \mathbf{m}^P(R)$ in $[\![P]\!]$;*
2. *if $\mathbf{m}^P(Q) \stackrel{\mu}{\longrightarrow} m$ in $[\![P]\!]$ then $Q \stackrel{\mu}{\longrightarrow} R$ for some R such that $m = \mathbf{m}^P(R)$.*

Proof. (1) The proof is by induction on the depth of the derivation $Q \stackrel{\mu}{\longrightarrow} R$.

- The cases for rules (Alt), (Cho) and (Repl) are similar. We focus on (Alt).
  Assume that $Q \stackrel{\mu}{\longrightarrow} R$ by applying rule (Alt). This means that $Q = \bigoplus_{i=1}^{n} a_i.P_i$, $\mu = a_j$ with $j \in \{1,\dots,n\}$, and $R = P_j$. Thanks to Lemma 6, in $[\![P]\!]$ there exists the subnet of Fig. 24 and $\mathbf{m}^P(Q) = \{\alpha\}$. Hence, $\mathbf{m}^P(Q) \stackrel{a_j}{\longrightarrow} m'$ and $m' = \{\alpha_{j1},\dots,\alpha_{jm_j}\} = f^{\oplus}_{R\to P}(m_{[\![R]\!]}) = \mathbf{m}^P(R)$.
- Assume that $Q \stackrel{\mu}{\longrightarrow} R$ by applying rule (Asyn). This means that $Q = Q_1 \mid_X Q_2$, $\mu \notin X$, $Q_1 \stackrel{\mu}{\longrightarrow} Q'_1$, and $R = Q'_1 \mid_X Q_2$. By definition, we have $\mathbf{m}^P(Q) = f^{\oplus}_{Q\to P}(m_{[\![Q]\!]}) = f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]}) \oplus f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]})) = f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]})) \oplus f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]}))$.
  Since $Q_1 \stackrel{\mu}{\longrightarrow} Q'_1$, by inductive hypothesis we get that $\mathbf{m}^{Q_1}(Q_1) \stackrel{\mu}{\longrightarrow} \mathbf{m}^{Q_1}(Q'_1)$. This means that $m_{[\![Q_1]\!]} \stackrel{\mu}{\longrightarrow} f^{\oplus}_{Q'_1\to Q_1}(m_{[\![Q'_1]\!]})$ and thus $f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]}) \stackrel{\mu}{\longrightarrow} f^{\oplus}_{Q_1\to Q}(f^{\oplus}_{Q'_1\to Q_1}(m_{[\![Q'_1]\!]}))$, since Q_1 is a subprocess of Q. Now, since $Q'_1 \in subreach(Q_1)$, $Q'_1 \in subreach(Q)$, and $Q_1 \in subreach(Q)$, by Lemma 6 we have $f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]}) \stackrel{\mu}{\longrightarrow} f^{\oplus}_{Q'_1\to Q}(m_{[\![Q'_1]\!]})$. Additionally $Q \in reach(P)$ and thus $f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]})) \stackrel{\mu}{\longrightarrow} f^{\oplus}_{Q\to P}(f^{\oplus}_{Q'_1\to Q}(m_{[\![Q'_1]\!]}))$.
  Now, by the definition of the operational semantics, $f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]})) \oplus f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]})) \stackrel{\mu}{\longrightarrow} f^{\oplus}_{Q\to P}(f^{\oplus}_{Q'_1\to Q}(m_{[\![Q'_1]\!]})) \oplus f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]}))$. Finally, by Lemma 6, we can easily deduce that $f^{\oplus}_{Q\to P}(f^{\oplus}_{Q'_1\to Q}(m_{[\![Q'_1]\!]})) \oplus f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]})) = \mathbf{m}^P(R)$.
- Assume that $Q \stackrel{\mu}{\longrightarrow} R$ by applying rule (Syn). This means that $Q = Q_1 \mid_X Q_2$, $\mu \in X$, $Q_1 \stackrel{\mu}{\longrightarrow} Q'_1$, $Q_2 \stackrel{\mu}{\longrightarrow} Q'_2$, and $R = Q'_1 \mid_X Q'_2$. By definition we have $\mathbf{m}^P(Q) = f^{\oplus}_{Q\to P}(m_{[\![Q]\!]})$. Since $Q_1 \stackrel{\mu}{\longrightarrow} Q'_1$, by inductive hypothesis $\mathbf{m}^{Q_1}(Q_1) \stackrel{\mu}{\longrightarrow} \mathbf{m}^{Q_1}(Q'_1)$, that is, $m_{[\![Q_1]\!]} \stackrel{\mu}{\longrightarrow} f^{\oplus}_{Q'_1\to Q_1}(m_{[\![Q'_1]\!]})$. Analogously, we have that $m_{[\![Q_2]\!]} \stackrel{\mu}{\longrightarrow} f^{\oplus}_{Q'_2\to Q_2}(m_{[\![Q'_2]\!]})$. Moreover, Q_1 and Q_2 are subprocesses of Q, therefore by Lemma 6 we have mappings of the encodings of Q_1 and Q_2 into $[\![Q]\!]$. By the definition of the encoding, since $\mu \in X$, both μ transitions of $[\![Q_1]\!]$ and $[\![Q_2]\!]$ are mapped to the same transition in

$[\![Q]\!]$, with pre-set $f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]}) \oplus f^{\oplus}_{Q_2\to Q}(m_{[\![Q_2]\!]})$ and post-set $f^{\oplus}_{Q_1\to Q}(f^{\oplus}_{Q'_1\to Q_1}(m_{[\![Q'_1]\!]})) \oplus f^{\oplus}_{Q_2\to Q}(f^{\oplus}_{Q'_2\to Q_2}(m_{[\![Q'_2]\!]}))$. This means that $m_{[\![Q]\!]} \xrightarrow{\mu} f^{\oplus}_{R\to Q}(m_{[\![R]\!]})$. So $f^{\oplus}_{Q\to P}(m_{[\![Q]\!]}) \xrightarrow{\mu} f^{\oplus}_{Q\to P}(f^{\oplus}_{R\to Q}(m_{[\![R]\!]}))$ since $Q \in reach(P)$, and thus, by Lemma 6, $\mathbf{m}^P(Q) \xrightarrow{\mu} \mathbf{m}^P(R)$.

- Assume that $Q \xrightarrow{\mu} R$ by applying rule (Hid_1). This means that $Q = Q_1\backslash X$, $\mu \notin X$, $Q_1 \xrightarrow{\mu} Q'_1$, and $R = Q'_1\backslash X$. We have that $\mathbf{m}^P(Q) = f^{\oplus}_{Q\to P}(m_{[\![Q]\!]}) = f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]}))$. By inductive hypothesis we have $\mathbf{m}^{Q_1}(Q_1) \xrightarrow{\mu} \mathbf{m}^{Q_1}(Q'_1)$. This means that $m_{[\![Q_1]\!]} \xrightarrow{\mu} f^{\oplus}_{Q'_1\to Q_1}(m_{[\![Q'_1]\!]})$. Since Q_1 is a subprocess of Q and $\mu \notin X$, that is, the corresponding transition is also visible in $[\![Q]\!]$, we deduce $f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]}) \xrightarrow{\mu} f^{\oplus}_{Q_1\to Q}(f^{\oplus}_{Q'_1\to Q_1}(m_{[\![Q'_1]\!]}))$. By Lemma 6, $f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]}) \xrightarrow{\mu} f^{\oplus}_{Q'_1\to Q}(m_{[\![Q'_1]\!]}))$, and since $Q \in reach(P)$ we have $f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]})) \xrightarrow{\mu} f^{\oplus}_{Q\to P}(f^{\oplus}_{Q'_1\to Q}(m_{[\![Q'_1]\!]}))$. Thanks again to Lemma 6, we infer that $f^{\oplus}_{Q\to P}(f^{\oplus}_{Q'_1\to Q}(m_{[\![Q'_1]\!]})) = \mathbf{m}^P(R)$, and therefore $\mathbf{m}^P(Q) \xrightarrow{\mu} \mathbf{m}^P(R)$.
- The case for the rule (Hid_2) is very similar to the previous one. The only difference is that the label of the transition of Q_1 belongs to X, so the corresponding transition in the encoding of Q becomes hidden.
- Assume that $Q \xrightarrow{\mu} R$ because $Q \equiv Q'$, $Q' \xrightarrow{\mu} R'$ and $R' \equiv R$. Since $Q \equiv Q'$, by Proposition 3 we know that $[\![Q]\!] = [\![Q']\!]$. Analogously, since $R' \equiv R$, then $[\![R]\!] = [\![R']\!]$. Moreover, since $Q' \xrightarrow{\mu} R'$, by inductive hypothesis $\mathbf{m}^P(Q') \xrightarrow{\mu} \mathbf{m}^P(R')$. It is then immediate to conclude $\mathbf{m}^P(Q) \xrightarrow{\mu} \mathbf{m}^P(R)$.

(2) Assume that $\mathbf{m}^P(Q) \xrightarrow{\mu} m$. The proof is by structural induction on Q.

- The cases for $Q = \bigoplus_{i=1}^n a_i.P_i$, $Q = Q_1 + Q_2$ and $Q = !_a.Q_1$ are similar. We focus on the first one. Suppose that $Q = \bigoplus_{i=1}^n a_i.P_i$. By Lemma 6, this means that the encoding of P has a subnet like the one in Fig. 24, corresponding to the encoding of Q in P. Therefore, $\mathbf{m}^P(Q) = \{\alpha\}$, $\mu = a_j$ for $j \in \{1,\ldots,n\}$ and $m = \{\alpha_{j_1},\ldots,\alpha_{j_{m_j}}\}$. As desired, by the operational semantics of CSP, $Q \xrightarrow{a_j} P_j$ and moreover, by Lemma 6, $\{\alpha_{j_1},\ldots,\alpha_{j_n}\} = \mathbf{m}^P(P_j)$.
- Suppose that $Q = Q_1 \mid_X Q_2$. This means that $\mathbf{m}^P(Q) = \mathbf{m}^P(Q_1) \oplus \mathbf{m}^P(Q_2)$. We distinguish two cases. The first case is when for some $i \in \{1,2\}$ the fired transition is enabled by $\mathbf{m}^P(Q_1)$ or by $\mathbf{m}^P(Q_2)$. Without loss of generality, we can assume that $\mathbf{m}^P(Q_1) \xrightarrow{\mu} m'_1$. This means that $\mu \notin X$ and $m = m'_1 \oplus \mathbf{m}^P(Q_2)$. By Lemma 6, we have that $\mathbf{m}^{Q_1}(Q_1) \xrightarrow{\mu} m_1$ with $m'_1 = f^{\oplus}_{Q_1\to P}(m_1)$. Now, by inductive hypothesis, $Q_1 \xrightarrow{\mu} Q'_1$ and $\mathbf{m}^{Q_1}(Q'_1) = m_1$. By the operational semantics of CSP, we immediately get $Q \xrightarrow{\mu} Q'_1 \mid_X Q_2 = R$ and, by Lemma 6, we deduce also that $m = \mathbf{m}^P(R)$.
  In the second case, the firing of the transition t with $\lambda(t) = \mu$ requires the marking $\mathbf{m}^P(Q_1) \oplus \mathbf{m}^P(Q_2)$. By definition of the encoding we have $\mu \in X$. Additionally since, by Lemma 6, the morphism $f_{Q\to P}$ is injective on transitions corresponding to synchronisation events, there is a single μ-labelled transition in $[\![Q]\!]$ which is mapped to the fired transition t in $[\![P]\!]$. Since $[\![Q]\!] = [\![Q_1]\!] \bigotimes_X [\![Q_2]\!]$, such a transition must arise as the synchronisation of μ-labelled transitions t_1 in $[\![Q_1]\!]$ and t_2 in $[\![Q_2]\!]$. Therefore, $\mathbf{m}^{Q_1}(Q_1) \xrightarrow{\mu} m_1$ and $\mathbf{m}^{Q_1}(Q_2) \xrightarrow{\mu} m_2$ and $m = f^{\oplus}_{Q_1\to P}(m_1) \oplus f^{\oplus}_{Q_2\to P}(m_2)$. By inductive hypothesis we have $Q_1 \xrightarrow{\mu} Q'_1$ and $m_1 = \mathbf{m}^{Q_1}(Q'_1)$ and $Q_2 \xrightarrow{\mu} Q'_2$ and $m_2 = \mathbf{m}^{Q_2}(Q'_2)$. So, we can conclude that $Q \xrightarrow{\mu} R = Q'_1 \mid_X Q'_2$ and $m = f^{\oplus}_{Q_1\to P}(\mathbf{m}^{Q_1}(Q'_1)) \oplus f^{\oplus}_{Q_2\to P}(\mathbf{m}^{Q_1}(Q'_2))$. Finally, by using Lemma 6, we can show that $m = \mathbf{m}^P(R)$.
- Suppose that $Q = Q_1\backslash X$. By definition, $\mathbf{m}^P(Q) = f^{\oplus}_{Q\to P}(m_{[\![Q]\!]}) = f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]}))$ and, by Lemma 6, $f^{\oplus}_{Q\to P}(f^{\oplus}_{Q_1\to Q}(m_{[\![Q_1]\!]})) = \mathbf{m}^P(Q_1)$. We have two cases: either $\mu \notin X$ or $\mu = \tau$. Assume that $\mu \notin X$. Since by hypothesis $\mathbf{m}^P(Q_1) \xrightarrow{\mu} m$ and $\mu \notin X$, also $\mathbf{m}^{Q_1}(Q_1) \xrightarrow{\mu} m_1$ with $m = f^{\oplus}_{Q_1\to P}(m_1)$. By inductive hypothesis $Q_1 \xrightarrow{\mu} Q'_1$ and $m_1 = \mathbf{m}^{Q_1}(Q'_1)$. Therefore, we can conclude that $Q \xrightarrow{\mu} Q'_1\backslash X = R$. Moreover, $m = f^{\oplus}_{Q_1\to P}(\mathbf{m}^{Q_1}(Q'_1))$ and, by Lemma 6, $m = \mathbf{m}^P(Q'_1) = \mathbf{m}^P(R)$.
  Assume that $\mu = \tau$. Since by hypothesis $\mathbf{m}^P(Q_1) \xrightarrow{\mu} m$, we have that $\mathbf{m}^{Q_1}(Q_1) \xrightarrow{\mu'} m_1$, with $m = f^{\oplus}_{Q_1\to P}(m_1)$ and $\mu' = \tau$ or $\mu' \neq \tau$. In both cases, by inductive hypothesis $Q_1 \xrightarrow{\mu'} Q'_1$ and $m_1 = \mathbf{m}^{Q_1}(Q'_1)$. Moreover, if $\mu' = \tau$, also $Q \xrightarrow{\tau} Q'_1\backslash X = R$. Otherwise, if $\mu' \neq \tau$, since $\mu = \tau$, the transition t of $[\![Q_1]\!]$ such that $\lambda(t) = \mu'$, must be visible and it becomes hidden in $[\![Q]\!]$. This means that $\mu' \in X$ and, by definition of the operational semantics for CSP, $Q \xrightarrow{\mu'} Q'_1\backslash X = R$. In both cases, $m = f^{\oplus}_{Q_1\to P}(\mathbf{m}^{Q_1}(Q'_1)) = \mathbf{m}^P(Q'_1) = \mathbf{m}^P(R)$. □

Appendix C. Proof of Theorem 5

Theorem 5 *(Undecidability of ACCS bisimilarity). Strong and weak bisimilarity are undecidable for bound ACCS processes.*

Proof. We first define, for any two-register machine program P, a bound process $\gamma(\mathsf{P})$ which "non-deterministically simulates" the computations of the program (starting with null registers). Some "wrongful" computations are possible in the process $\gamma(\mathsf{P})$, not corresponding to a correct computation of the program P (due to the absence of zero-tests in the con-

sidered fragment of ACCS). Still, adapting an idea of [19], this can be used to prove undecidability of trace and (strong and weak) bisimulation equivalence.

Hereafter, we use *process constants* which are not in the considered fragment of asynchronous CCS. However, observe that they can be represented by means of bound processes. Indeed any constant K defined by $K = P[K]$ can be represented as $(\nu a)\bar{a} \mid !_a.\, P\{^{\bar{a}}/_K\}$, where a is a fresh name for $P[K]$ and $P\{^{\bar{a}}/_K\}$ denotes the substitution of all the occurrences of K with $\bar{a}$. As an example the process $K = a.(K|\bar{c}) + b.(K \mid \bar{d})$ stands for $(\nu e)[\bar{e} \mid !_e.(a.(\bar{e}|\bar{c}) + b.(\bar{e} \mid \bar{d}))]$.

Observe that if a process P using constant definitions is restriction-free then the associated process using guarded replications is a bound process.

The idea is to view registers r_1 and r_2 as channel names. The presence of value n in a register x is represented by the presence of n basic processes $\bar{x}$ in parallel in the current state, and the decrement and increment operations over x are modelled by an input and an output over x, respectively.

For a program $\mathsf{P} = \mathsf{I}_1 \ldots \mathsf{I}_s$, different channel names $a_1, \ldots, a_s, a_{s+1}$ model the program counter. The processes encoding each single instruction are in parallel, but the process encoding instruction I_i is enabled by an input on channel a_i. More precisely, the encoding of I_i, denoted I_i, is defined by the clauses

- if $\mathsf{I}_i = s(x, j)$ then $I_i = a_i.\, s.\, (\bar{x} \mid \bar{a}_j \mid \bar{o}_i \mid I_i)$;
- if $\mathsf{I}_i = zd(x, j, k)$ then $I_i = a_i.\, (\tau.\, z.(\bar{a}_j \mid \bar{o}_i \mid I_i) + x.\, d.\, (\bar{a}_k \mid \bar{o}_i \mid I_i))$.

The presence of τ in the encoding of $zd(x, j, k)$ is needed for technical reasons which will be clarified below. Finally, there is a process corresponding to termination, namely, $I_{s+1} = a_{s+1}.\, \overline{out}$.

Any instruction contains an input operation which corresponds to the kind of instruction which has been executed: for instructions $s(x, j)$, an input on channel s is performed; for instructions $zd(x, j, k)$, an input on z (zero) or d (decrement) is executed, according to the branch followed. Thus, the fact that the instruction i has been executed is signalled with an output on channel o_i.

We can now define the process associated with a program P as

$$\gamma(\mathsf{P}) = (\nu L)\, Q$$

where $Q = \bar{a}_0 \mid I_1 \mid \ldots \mid I_{s+1}$ and $L = \{a_0, \ldots, a_{s+1}, r_1, r_2\}$.

Computations of $\gamma(\mathsf{P})$ are non-deterministic simulations of computations of the two-register machine, which may include false trails: when executing the process corresponding to an instruction $zd(x, j, k)$, the branch "z" corresponding to $x = 0$ can be chosen even though register x is strictly positive.

Following [19] we build a process $\gamma'(\mathsf{P})$ such that $\gamma(\mathsf{P}) \sim \gamma'(\mathsf{P})$ if and only if the program P terminates. Intuitively, $\gamma'(\mathsf{P})$ contains two copies of process Q previously defined. The second copy, denoted Q', has its own program counter and it does not include the output message on *out* at termination. One starts executing Q' and the only way to jump to the instructions of the first copy Q is to execute a cheating transition, i.e., to follow the "z" branch in the encoding of an instruction $zd(x, j, k)$, when $x > 0$. Observe that the presence of τ in the first "z" branch (the "innocent" one) ensures that this branch can be used to strongly simulate the second cheating "z" branch, as needed later in the proof.

Formally, the process $\gamma'(\mathsf{P})$ corresponding to a program P is defined as

$$\gamma'(\mathsf{P}) = \big(\nu L \cup L'\big)\big(Q \mid Q'\big)$$

where Q is as before and $Q' = \bar{b}_0 \mid I'_1 \mid \ldots \mid I'_{s+1}$ and $L' = \{b_0, \ldots, b_{s+1}, r_1, r_2\}$.

The instructions in Q' follow those of program P, according to the intuition above. More precisely

- if $\mathsf{I}_i = s(x, j)$ then $I'_i = b_i.\, s.\, (\bar{x} \mid \bar{b}_j \mid \bar{o}_i \mid I'_i)$;
- if $\mathsf{I}_i = zd(x, j, k)$ then

$$I'_i = b_i.\, \big(\tau.\, z.\big(\bar{b}_j \;\big|\; \bar{o}_i \;\big|\; I'_i\big) + x.\, z.\, \big(\bar{x} \;\big|\; \bar{a}_j \;\big|\; \bar{o}_i \;\big|\; I'_i\big) + x.\, d.\, \big(\bar{b}_k \;\big|\; \bar{o}_i \;\big|\; I'_i\big)\big);$$

where, in the second case, the summand $x.\, z.\, (\bar{x} \mid \bar{a}_j \mid \bar{o}_i \mid I'_i)$ adds the possibility of choosing explicitly the wrong branch of the jump instruction: knowing that $x > 0$, we deliberately assume it to be zero and choose the "z" branch. In this way, and only in this way, we jump to the other copy Q of the process.

Finally, the process corresponding to termination is $I_{s+1} = b_{s+1}.\, 0$. Note that it does not output on channel *out*.

We now prove the desired result. The idea is to show that for a program P

1. if the computation of P does terminate then $\gamma(\mathsf{P}) \not\approx \gamma'(\mathsf{P})$ (hence $\gamma(\mathsf{P})$ and $\gamma'(\mathsf{P})$ are neither strong bisimilar);
2. if the computation of P does not terminate then $\gamma(\mathsf{P}) \sim \gamma'(\mathsf{P})$ (hence $\gamma(\mathsf{P})$ and $\gamma'(\mathsf{P})$ are also weak bisimilar).

Having this, since termination for two-register machines is undecidable, it cannot be possible to decide if $\gamma(\mathsf{P}) \sim \gamma'(\mathsf{P})$ or if $\gamma(\mathsf{P}) \approx \gamma'(\mathsf{P})$.

We thus finally show that items (1) and (2) above actually hold:

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

1. If the machine terminates, then $\gamma(\mathsf{P})$ may terminate by executing only steps in which the machine is correctly simulated ("z" branches are followed only when the register is 0). The process $\gamma'(\mathsf{P})$ can only do the same thing, executing the subprocess Q' and terminating without making an output on *out*. In particular, it is never able to "jump" to the Q subprocess, as the only possible way of doing this would be to take the cheating "z" branch of a jump instruction (execute z branch when register is not zero).
   More formally, let $i_1, \ldots, i_k$ be the sequence of instructions executed by the program in order to terminate. Then the process $\gamma(\mathsf{P})$ is able to consume a corresponding sequence of output messages $\bar{t}_1 \ldots \bar{t}_k$, with $t_i \in \{s, d, z\}$ for $i \in \{1, \ldots, k\}$, namely there is a sequence of processes $P_0 = \gamma(\mathsf{P})$, P_1, …, P_k such that for any $i \in \{1, \ldots, k\}$

   $$P_{i-1} \mid \bar{t}_i \ \Downarrow_{\bar{o}_i} \ P_i$$

   and $P_k \Downarrow_{\overline{out}}$. This sequence cannot be weakly simulated by $\gamma'(\mathsf{P})$, since whenever $P'_0 = \gamma'(\mathsf{P})$, P'_1, …, P'_k such that for any $i \in \{1, \ldots, k\}$

   $$P'_{i-1} \mid \bar{t}_i \ \Downarrow_{\bar{o}_i} \ P'_i$$

   we necessarily have that $P'_k \Downarrow_{\overline{out}}$ does not hold (since the simulating sequence is "confined" to the Q' component). This violates the requirements of Definition 7 and thus $\gamma(\mathsf{P}) \not\approx \gamma'(\mathsf{P})$.
2. If the machine does not terminate, then $\gamma(\mathsf{P}) \sim \gamma'(\mathsf{P})$. In fact the two processes are essentially identical. The only difference is that $\gamma'(\mathsf{P})$ can deliberately choose the cheating z branch of a jump instruction (execute z branch when register is not zero), but this can be strongly simulated in $\gamma(\mathsf{P})$. After this, the two processes will be executing the same "instruction" of the sub-process Q and will never be distinguishable. Hence it is immediate to see that $\gamma(\mathsf{P}) \sim \gamma'(\mathsf{P})$. □

References

[1] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.
[2] R. Milner, A Calculus of Communicating Systems, LNCS, vol. 92, Springer, 1980.
[3] K. Honda, M. Tokoro, An object calculus for asynchronous communication, in: P. America (Ed.), ECOOP 1991, in: LNCS, vol. 512, Springer, 1991, pp. 133–147.
[4] G. Boudol, Asynchrony and the π-calculus, Tech. rep. 1702, INRIA, Sophia Antipolis, 1992.
[5] R. De Nicola, G. Ferrari, R. Pugliese, KLAIM: a kernel language for agents interaction and mobility, IEEE Trans. Softw. Eng. 24 (5) (1998) 315–330.
[6] I. Castellani, M. Hennessy, Testing theories for asynchronous languages, in: V. Arvind, R. Ramanujam (Eds.), FSTTCS 1998, in: LNCS, vol. 1530, Springer, 1998, pp. 90–101.
[7] G. Ferrari, R. Guanciale, D. Strollo, Event based service coordination over dynamic and heterogeneous networks, in: A. Dan, W. Lamersdorf (Eds.), ICSOC 2006, in: LNCS, vol. 4294, Springer, 2006, pp. 453–458.
[8] M. Boreale, R. De Nicola, R. Pugliese, Asynchronous observations of processes, in: M. Nivat (Ed.), FOSSACS 1998, in: LNCS, vol. 1378, Springer, 1998, pp. 95–109.
[9] W. Reisig, Understanding Petri Nets, Springer, 2013.
[10] R. Gorrieri, U. Montanari, SCONE: a simple calculus of nets, in: J.C.M. Baeten, J.W. Klop (Eds.), CONCUR 1990, in: LNCS, vol. 458, Springer, 1990, pp. 2–30.
[11] U. Goltz, CCS and Petri nets, in: I. Guessarian (Ed.), Semantics of Systems of Concurrent Processes, in: LNCS, vol. 469, Springer, 1990, pp. 334–357.
[12] R. Devillers, H. Klaudel, M. Koutny, A compositional Petri net translation of general pi-calculus terms, Form. Asp. Comput. 20 (4–5) (2008) 429–450.
[13] G. Winskel, A new definition of morphism on Petri nets, in: M. Fontet, K. Mehlhorn (Eds.), STACS 1984, in: LNCS, vol. 166, Springer, 1984, pp. 140–150.
[14] P. Baldan, A. Corradini, H. Ehrig, R. Heckel, Compositional semantics for open Petri nets based on deterministic processes, Math. Struct. Comput. Sci. 15 (1) (2004) 1–35.
[15] R. Milner, Bigraphs for Petri nets, in: J. Desel, W. Reisig, G. Rozenberg (Eds.), Lectures on Concurrency and Petri Nets, in: LNCS, vol. 3098, Springer, 2003, pp. 686–701.
[16] V. Sassone, P. Sobociński, A congruence for Petri nets, in: T. Mens, A. Schürr, G. Taentzer (Eds.), PNGT 2004, in: ENTCS, vol. 127, Elsevier, 2005, pp. 107–120.
[17] R. Bruni, H.C. Melgratti, U. Montanari, P. Sobociński, Connector algebras for C/E and P/T nets' interactions, Log. Methods Comput. Sci. 9 (3) (2013) 1–65.
[18] J. Aranda, F. Valencia, C. Versari, On the expressive power of restriction and priorities in CCS with replication, in: L. de Alfaro (Ed.), FOSSACS 2009, in: LNCS, vol. 5504, Springer, 2009, pp. 242–256.
[19] P. Jancar, Undecidability of bisimilarity for Petri nets and some related problems, Theor. Comput. Sci. 148 (2) (1995) 281–301.
[20] P. Baldan, F. Bonchi, F. Gadducci, Encoding asynchronous interactions using open Petri nets, in: M. Bravetti, L. Zavattaro (Eds.), CONCUR 2009, in: LNCS, vol. 5710, Springer, 2009, pp. 99–114.
[21] P. Baldan, F. Bonchi, F. Gadducci, G. Monreale, Encoding synchronous interactions using labelled Petri nets, in: E. Kühn, R. Pugliese (Eds.), COORDINATION 2014, in: LNCS, vol. 8459, Springer, 2014, pp. 1–16.
[22] R. Amadio, I. Castellani, D. Sangiorgi, On bisimulations for the asynchronous pi-calculus, Theor. Comput. Sci. 195 (2) (1998) 291–324.
[23] R. Milner, D. Sangiorgi, Barbed bisimulation, in: W. Kuich (Ed.), ICALP 1992, in: LNCS, vol. 623, Springer, 1992, pp. 685–695.
[24] A.W. Roscoe, The Theory and Practice of Concurrency, Prentice Hall, 1997.
[25] F. Gadducci, G.V. Monreale, A decentralised graphical implementation of mobile ambients, J. Log. Algebr. Program. 80 (2) (2011) 113–136.
[26] Y. Hirshfeld, Petri nets and the equivalence problem, in: E. Börger, Y. Gurevich, K. Meinke (Eds.), CSL 1993, in: LNCS, vol. 832, Springer, 1994, pp. 165–174.
[27] M. Boreale, R. De Nicola, R. Pugliese, Trace and testing equivalence on asynchronous processes, Inf. Comput. 172 (2002) 139–164.
[28] J. Esparza, M. Nielsen, Decidability issues for Petri nets – a survey, Elektron. Inf. verarb. Kybern. 30 (3) (1994) 143–160.
[29] N. Busi, M. Gabbrielli, G. Zavattaro, Comparing recursion, replication, and iteration in process calculi, in: J. Díaz, J. Karhumäki, A. Lepistö, D. Sannella (Eds.), ICALP 2004, in: LNCS, vol. 3142, Springer, 2004, pp. 307–319.
[30] T. Agerwala, M. Flynn, Comments on capabilities, limitations and "correctness" of Petri nets, Comput. Archit. News 4 (2) (1973) 81–86.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019

[31] E. Best, R. Devillers, J.G. Hall, The Petri box calculus: a new causal algebra with multi-label communication, in: G. Rozenberg (Ed.), APN 1992, in: LNCS, vol. 609, Springer, 1992, pp. 21–69.
[32] M. Koutny, J. Esparza, E. Best, Operational semantics for the Petri box calculus, in: B. Jonsson, J. Parrow (Eds.), CONCUR 1994, in: LNCS, vol. 836, Springer, 1994, pp. 210–225.
[33] M. Koutny, E. Best, Operational and denotational semantics for the box algebra, Theor. Comput. Sci. 211 (1–2) (1999) 1–83.
[34] M. Nielsen, L. Priese, V. Sassone, Characterizing behavioural congruences for Petri nets, in: I. Lee, S.A. Smolka (Eds.), CONCUR 1995, in: LNCS, vol. 962, Springer, 1995, pp. 175–189.
[35] L. Priese, H. Wimmel, A uniform approach to true-concurrency and interleaving semantics for Petri nets, Theor. Comput. Sci. 206 (1–2) (1998) 219–256.
[36] F. Bonchi, A. Brogi, S. Corfini, F. Gadducci, On the use of behavioural equivalences for web services' development, Fundam. Inform. 89 (4) (2008) 479–510.
[37] F. Bonchi, A. Brogi, S. Corfini, F. Gadducci, A net-based approach to web services publication and replaceability, Fundam. Inform. 94 (3–4) (2009) 305–330.
[38] W. Vogler, Modular Construction and Partial Order Semantics of Petri Nets, LNCS, vol. 625, Springer, 1992.
[39] E. Kindler, A compositional partial order semantics for Petri net components, in: P. Azéma, G. Balbo (Eds.), ICATPN 1997, in: LNCS, vol. 1248, Springer, 1997, pp. 235–252.
[40] U. Goltz, W. Reisig, CSP-programs with individual tokens, in: G. Rozenberg, H.J. Genrich, G. Roucairol (Eds.), APN 1984, in: LNCS, vol. 188, Springer, 1985, pp. 169–196.
[41] E.-R. Olderog, Operational Petri net semantics for CCSP, in: G. Rozenberg (Ed.), APN 1986, in: LNCS, vol. 266, Springer, 1987, pp. 196–223.
[42] P. Degano, R. Gorrieri, S. Marchetti, An exercise in concurrency: a CSP process as a condition/event system, in: G. Rozenberg (Ed.), APN 1987, in: LNCS, vol. 340, Springer, 1988, pp. 85–105.
[43] M. Llorens, J. Oliver, J. Silva, S. Tamarit, Generating a Petri net from a CSP specification: a semantics-based method, Adv. Eng. Softw. 50 (2012) 110–130.
[44] M. Buscemi, V. Sassone, High-level Petri nets as type theories in the join calculus, in: F. Honsell, M. Miculan (Eds.), FOSSACS 2001, in: LNCS, vol. 2030, Springer, 2001, pp. 104–120.
[45] N. Busi, G. Zavattaro, A process algebraic view of shared dataspace coordination, J. Log. Algebr. Program. 75 (1) (2008) 52–85.
[46] R. Devillers, H. Klaudel, M. Koutny, A Petri net semantics of a simple process algebra for mobility, in: C. Canal, M. Viroli (Eds.), EXPRESS 2005, in: ENTCS, vol. 154.3, Elsevier, 2006, pp. 71–94.
[47] R. Meyer, V. Khomenko, T. Strazny, A practical approach to verification of mobile systems using net unfoldings, Fundam. Inform. 94 (3–4) (2009) 439–471.
[48] N. Busi, G. Zavattaro, Expired data collection in shared dataspaces, Theor. Comput. Sci. 3 (298) (2003) 529–556.

Please cite this article in press as: P. Baldan et al., Modular encoding of synchronous and asynchronous interactions using open Petri nets, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2014.11.019