Encoding Synchronous Interactions Using Labelled Petri Nets*

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

[1] Dipartimento di Matematica, Università di Padova, Padova, Italy
[2] LIP, ENS Lyon, Université de Lyon (UMR CNRS - INRIA 5668), Lyon, France
[3] Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Abstract. We present an encoding of (bound) CSP processes with replication into Petri nets with labelled transitions. Through the encoding, the firing semantics of Petri nets models the standard operational semantics of CSP processes, which is both preserved and reflected. This correspondence allows for describing by net semantics the standard CSP observational equivalences. Since the encoding is modular with respect to process syntax, the paper puts on a firm ground the technology transfer between the two formalisms, e.g. recasting into the CSP framework well-established results like decidability of coverability for nets. This work complements previous results concerning the encoding of asynchronous interactions, thus witnessing the expressiveness of (open) labelled nets in modelling process calculi with alternative communication patterns.

Keywords: Communicating sequential processes (CSP), labelled Petri nets, net encoding of processes, synchronous interaction.

1 Introduction

Petri nets [17] are among the most widely used formalisms for the visual specification of concurrent and distributed systems. Their appeal lies in the ease of use as well as in the 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 are plainly represented. Also, in a Petri net the behavioural relations between computational steps, such as causal dependencies and nondeterministic choices, are explicit and easier to analyse.

These characteristics also favoured Petri nets as the target for the encoding of many textual formalisms, such as different process calculi. This is partly due to the availability of many tools and techniques for the analysis of net behavioural properties, like coverability, boundedness, and deadlock-freedom, so that any suitable encoding might offer the possibility of a fruitful technology transfer. However, it is the same simple and immediate graphical presentation of nets that attracted the attention of researchers, in the hope of clarifying the nature of concurrency and distributivity in the formalism at hand. Indeed, there has

* Supported by EU FP7-ICT IP ASCENS, MIUR PRIN CINA and ANR PACE.

E. Kühn and R. Pugliese (Eds.): COORDINATION 2014, LNCS 8459, pp. 1–16, 2014.

been since a long time an interest for the encoding of synchronous calculi such as Milner's CCS. Intuitively, the handshaking communication pattern of CCS and π-calculus can be implemented via nets in such a way that the operational behaviour of a process is (at least) preserved by the encoding [10,7].

In a recent work we offered a further witness to the flexibility of nets by providing an encoding for asynchronous CCS [1]. More precisely, our encoding preserves the operational behaviour of processes as well as asynchronous bisimilarity, captured by standard net bisimilarity. In order to model the intrinsic reactivity of CCS processes, the encoding resorted to open Petri nets [2], i.e., nets extended with the possibility of interacting with the environment through an interface. Specifically, the interface consists of a set of places designated as open, where the environment can create and consume tokens. Interfaces were also essential to define composition operations on nets, thus allowing for a modular definition of the encoding. The need of considering reactive extensions of Petri nets in order to have a modular model, with compositional semantics, have been felt by several authors, leading to the Box Calculus [3], the Petri net components [13] and other open net models [15,4], just to mention a few.

This paper aims at further extending our results by moving back to synchronous processes, yet taking into account the broadcast communication pattern, as provided by Hoare's CSP [12]. More precisely, we identify an expressive fragment of CSP which can be mapped modularly into Petri nets via an encoding that is preserving as well as reflecting the operational semantics. Since most of CSP semantics are based on traces, the encoding is guaranteed to preserve and reflect also the common observational equivalences for the calculus. This allows some immediate technology transfer from nets to processes. For instance, coverability, the maximal degree of parallelism of a process (given by the number of its sub-processes occurring in parallel) and convergence (i.e., the possibility of termination) can be proved to be decidable in the CSP setting. Some of these decidability results seem to be the first of their kind for (bound) CSP processes.

The idea of mapping CSP processes into nets arose early on, see among others [9,16,6]. 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 [14]. 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 standard 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 the 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" [14, 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.

P	$::=$	$STOP$	inactive process
		$\oplus_{i=1}^{n} a_i.P_i$	guarded alternative
		$P + Q$	nondeterministic choice
		$P \mid_X Q$	parallel composition
		$P \backslash X$	hiding
		$!_a.P$	replication

Fig. 1. CSP processes

The paper is structured as follows. Section 2 recalls the syntax and the operational semantics of CSP, while Section 3 introduces labelled nets with interfaces, as well as a suitable algebra for them. The core of the paper is Section 4, which presents the modular encoding from (bound) CSP processes into labelled nets. In Section 5 the encoding is proved to preserve and reflect the operational semantics, and hence the standard observational equivalences of the calculus (such as trace equivalence). The encoding is exploited in Section 6, which provides some examples of its effects on the technology transfer between the two formalisms. Finally, Section 7 discusses some expressiveness issues for the considered models, taking advantage from the encoding, and it draws some conclusions while providing a few pointers to future works.

2 Communicating Sequential Processes

In this section we briefly review the calculus of Communicating Sequential Processes (CSP) [12], presenting its syntax and operational semantics. We actually focus on a fragment of the calculus, which will be used throughout the paper.

Definition 1 (CSP processes). *Let Σ be the alphabet of* communication events*, ranged over by $a, b, c \ldots$ The set of CSP* processes *$\mathcal{P}$, ranged over by $P, Q, R, \ldots$, is generated by the grammar in Fig. 1, where $X \subseteq \Sigma$ is a finite set of events.*

The process $STOP$ cannot perform any event, i.e., it is a deadlocked process. The guarded alternative $\oplus_{i=1}^{n} a_i.P_i$ can perform any event a_i, for $i \in \{1, \ldots, n\}$, and then behave as P_i. For the sake of simplicity, we assume that $\forall j, \forall z.\, a_j \neq a_z$. The nondeterministic choice $P + Q$ can behave as either P or Q. The operators $\oplus$ and $+$ differs for the fact that for $\oplus$ the choice is external, i.e., it is the environment that determines the branch to be chosen, while for $+$ the choice is internal to the process. The process $P \mid_X Q$ is the parallel composition of P and Q, where the events in X are forced to synchronise, while those in $\Sigma \setminus X$ 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, that is, they become internal to the process. Finally, the replication $!_a.P$ can indefinitely perform an event a and spawn a parallel copy of P.

$$(Alt)\quad \frac{j \in \{1,\ldots,n\}}{\oplus_{i=1}^{n} a_i.P_i \xrightarrow{a_j} P_j}$$

$$(Cho_1)\ \frac{}{P + Q \xrightarrow{\tau} P} \qquad (Cho_2)\ \frac{}{P + Q \xrightarrow{\tau} Q}$$

$$(Syn_1)\ \frac{P \xrightarrow{\mu} P' \quad \mu \notin X}{P \mid_X Q \xrightarrow{\mu} P' \mid_X Q} \qquad (Syn_2)\ \frac{Q \xrightarrow{\mu} Q' \quad \mu \notin X}{P \mid_X Q \xrightarrow{\mu} P \mid_X Q'}$$

$$(Syn_3)\ \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q' \quad a \in X}{P \mid_X Q \xrightarrow{a} P' \mid_X Q'}$$

$$(Hid_1)\ \frac{P \xrightarrow{\mu} P' \quad \mu \notin X}{P\backslash X \xrightarrow{\mu} P'\backslash X} \qquad (Hid_2)\ \frac{P \xrightarrow{a} P' \quad a \in X}{P\backslash X \xrightarrow{\tau} P'\backslash X}$$

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

Fig. 2. CSP operational semantics

The guarded alternative is a specialisation of the external choice operator. This restriction does not represent a serious limitation since, as explained in [18], it is rare to find a usage of the external choice which cannot be expressed as a guarded alternative. More interestingly, we consider guarded replication in place of recursion. This will be important for ensuring the existence of a finite Petri net encoding for the class of CSP processes considered (see Section 5).

The behaviour of CSP processes, intuitively described above, is formalised in terms of a set of syntax directed rules which axiomatise a transition relation.

Definition 2 (operational semantics of CSP). *The labelled transition system (LTS) for CSP processes is the relation* $\rightarrow \subseteq \mathcal{P} \times (\Sigma \uplus \{\tau\}) \times \mathcal{P}$ *inductively defined by the rules in Fig. 2, where we write* $P \xrightarrow{\mu} P'$ *for* $\langle P, \mu, P' \rangle \in \rightarrow$.

We write $P \xrightarrow{s} {}^* P'$ for a sequence $P = P_1 \xrightarrow{\mu_1} P_2 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_{n-1}} P'$ with $s = \mu_1\mu_2\ldots\mu_{n-1}$. Moreover, we write $P \stackrel{s}{\Rightarrow} P'$ when $P \xrightarrow{s'}{}^* P'$ for some s' such that s is obtained from s' by removing the τ's. We write simply $P \rightarrow P'$ and $P \rightarrow^* P'$ instead of $P \xrightarrow{\mu} P'$ and $P \xrightarrow{s}{}^* P'$, respectively, whenever we are not interested in identifying the labels.

Definition 3 (bound processes). *A CSP process* $P \in \mathcal{P}$ *is called* bound *if parallel compositions* $\_ \mid_X \_$ *occur under the scope of replications only with* $X = \emptyset$.

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

$!_a.(b.b.STOP \mid_{\{b\}} b.STOP)$. As discussed in Section 4, this fact will be essential for defining a finite encoding of bound CSP processes into Petri nets.

Relying on the LTS defined above several observational semantics can be defined over CSP processes. In this paper, we will focus on the one based on traces, i.e., sequences of visible transitions.

Definition 4 (traces). *Let $P \in \mathcal{P}$ be a CSP process. We define $traces(P) = \{s \in \Sigma^* : \exists Q.\, P \stackrel{s}{\Rightarrow} Q\}$.*

Traces are exploited to provide a behavioral equivalence for processes.

Definition 5 (trace equivalence). *Let $P, Q \in \mathcal{P}$ be two CSP processes. They are called* trace equivalent*, written $P =_T Q$, if $traces(P) = traces(Q)$.*

Example 1. Consider the processes $P = a.(d.b.STOP \backslash d) \oplus b.a.STOP$ and $Q = (c.a.STOP \mid_{\{c\}} c.b.STOP) \backslash c$. It is easy to see that $traces(P) = traces(Q) = \{\epsilon, a, ab, b, ba\}$. Hence they are trace equivalent.

3 Labelled Petri Nets with Interfaces

This section reviews *labelled Petri nets*, i.e., ordinary P/T nets with labelled transitions [17]. Nets are also enriched with interfaces and endowed with composition operators in order to allow for an inductive encoding of CSP processes.

3.1 Labelled Petri Nets

Let $X^\oplus$ be the free commutative monoid over a set X. An element $m \in X^\oplus$, a *multiset* over X, is often viewed as a function $m : X \to \mathbb{N}$ (the set of natural numbers) that associates a multiplicity with every element of X. We write $m_1 \subseteq m_2$ if $\forall x \in X$, $m_1(x) \leq m_2(x)$. The symbol 0 denotes the empty multiset. Given $f : X \to Y$ we denote its extension to multisets by $f^\oplus : X^\oplus \to Y^\oplus$.

Hereafter Σ denotes a fixed set over which all nets are labelled. In the encoding of processes into nets, Σ is the set of CSP communication events.

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

The state of a net is given by a *marking*, i.e., a multiset of places $m \in S^\oplus$. Hereafter, the components of a net N will be assumed S, T, ${}^\bullet(.)$, $(.)^\bullet$ and λ, possibly with subscripts. We will often write ${}^\bullet t$ and $t^\bullet$ instead of ${}^\bullet(t)$ and $(t)^\bullet$.

Definition 7 (net morphism). *Let N_1, N_2 be two labelled 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 $t \in T_1$:*

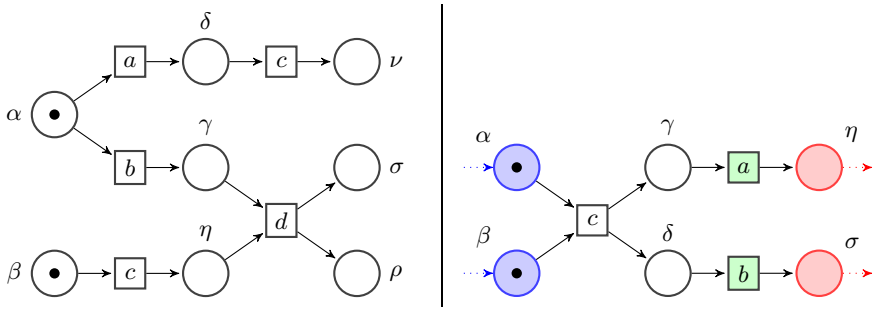

Fig. 3. Graphical representation of labelled nets, the rightmost with interfaces

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)*
3. $\lambda_1(t) = \lambda_2(f_T(t))$ *(preservation of labels).*

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.

Example 2. Fig. 3 (left) shows a labelled 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. Transition labels are placed inside the corresponding rectangle. For the sake of readability, also some places are provided with an identifier, yet positioned outside of the corresponding circle.

3.2 Petri Nets with Interfaces

In order to define the encoding of CSP processes into Petri nets inductively, we equip nets with "handles" for interacting with the environment and define operations for composing them.

Definition 8 (Petri net with interfaces). *A* Petri net with interfaces *is a tuple* $\mathbf{N} = \langle I, O, N, V \rangle$*, where* N *is a labelled net,* I *and* O *are subsets of places, the input and output places, and* V *is a subset of transitions, the visible transitions.*

Hereafter, the components of a net with interfaces $\mathbf{N}$ will be assumed to be I, N, O, and V, possibly with subscripts.

The standard operational semantics on Petri nets naturally induces a semantics for nets with interfaces, where the firing of a transition that is not visible is turned into a silent action τ. This is expressed by the rules in Fig. 4.

Graphically, a net with interfaces is depicted as a net with input interface on the left and output interface on the right, marked with incoming and outgoing dotted arrows, respectively. Places in the input and output interface are in blue and red, respectively (grey if in b&w), while internal places are white. Moreover, visible transitions are green (grey if in b&w) and hidden ones are white.

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

Fig. 4. Operational semantics of nets with interfaces

Example 3. An example of net with interfaces is shown in Fig. 3 (right). The input interface consists of the places α and β, while the output interface contains the places η and σ. The white places γ and δ are internal, i.e., they do not belong to the interfaces. The white transition labelled c is hidden, while those labelled a and b are visible. Finally, in the current marking m_0 of the net, the places α and β are marked. By applying the (HID) rule in Fig. 4 we obtain the firing $m_0 \xrightarrow{\tau} m_1 = \{\gamma, \delta\}$. By rule (VIS), we get $m_1 \xrightarrow{a} m_2 = \{\eta, \delta\}$ and $m_1 \xrightarrow{b} m_3 = \{\gamma, \sigma\}$. Finally, $m_2 \xrightarrow{b} m_4$ and $m_3 \xrightarrow{a} m_4$, with $m_4 = \{\eta, \sigma\}$.

As in the case of CSP processes, we write $m \xrightarrow{s}{}^* m'$ when there is a sequence $m = m_1 \xrightarrow{\mu_1} m_2 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_{n-1}} m'$ with $s = \mu_1 \mu_2 \ldots \mu_{n-1}$. We also write $m \stackrel{s}{\Rightarrow} m'$ when $m \xrightarrow{s'}{}^* m'$ for some s' such that s is obtained from s' by removing the τ's.

We next define suitable composition operators on nets with interfaces.

Definition 9 (sequential composition). *Let $\mathbf{N}_1$ and $\mathbf{N}_2$ be nets with interfaces such that $O_1 = I_2 = S_1 \cap S_2$ and $T_1 \cap T_2 = \emptyset$. Their sequential composition is the net with interfaces $\mathbf{N}_1 \circ \mathbf{N}_2 = \langle I_1, O_2, N, V_1 \cup V_2 \rangle$, where N is the pointwise union $N_1 \cup N_2$, with the obvious pre-set, post-set and labelling functions.*

Intuitively, the sequential composition $\mathbf{N}_1 \circ \mathbf{N}_2$ is obtained by taking the disjoint union of the nets underlying $\mathbf{N}_1$ and $\mathbf{N}_2$, and gluing the output places of N_1 with the corresponding input places of N_2. For the sake of presentation, it is convenient to assume that the two nets intersect only on the input/output interfaces and take the plain union. This could require some alpha-renaming.

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

Definition 10 (synchronised parallel composition). *Let $\mathbf{N}_1$ and $\mathbf{N}_2$ be nets with interfaces such that $S_1 \cap S_2 = \emptyset$ and $T_1 \cap T_2 = \emptyset$, and let $X \subseteq \Sigma$. Their synchronised parallel composition on X is the net with interfaces $\mathbf{N}_1 \otimes_X \mathbf{N}_2 = \langle I_1 \cup I_2, O_1 \cup O_2, N, V \rangle$, where the set of visible transitions is*

$$\begin{aligned} V = V_1 \otimes_X V_2 = \{&\langle t_1, t_2 \rangle : t_1 \in V_1^X \wedge\ t_2 \in V_2^X \wedge \lambda_1(t_1) = \lambda_2(t_2)\} \\ \cup \{&\langle t_1, * \rangle : t_1 \in V_1^X \wedge\ V_2^{\lambda_1(t_1)} = \emptyset\} \\ \cup \{&\langle *, t_2 \rangle : t_2 \in V_2^X \wedge\ V_1^{\lambda_2(t_2)} = \emptyset\} \\ \cup \{&t : t \in V_i \setminus V_i^X\} \end{aligned}$$

and $N = (S, T, {}^\bullet(.), (.)^\bullet, \lambda)$ defined as follows

- $S = S_1 \cup S_2 \uplus \{p\}$
- $T = (T_1 \setminus V_1) \cup (T_2 \setminus V_2) \cup V$

- $^\bullet t = \begin{cases} ^\bullet t & \text{if } t \in T_i \setminus V_i^X \\ ^\bullet t_1 \oplus {}^\bullet t_2 & \text{if } t = \langle t_1, t_2 \rangle \text{ with the convention } {}^\bullet * = p \end{cases}$
- $t^\bullet = \begin{cases} t^\bullet & \text{if } t \in T_i \setminus V_i^X \\ t_1{}^\bullet \oplus t_2{}^\bullet & \text{if } t = \langle t_1, t_2 \rangle, \text{with the convention } *^\bullet = 0 \end{cases}$
- $\lambda(t) = \begin{cases} \lambda_i(t_i) & \text{if } t \in T_i \setminus V_i^X \\ \lambda_1(t_1) & \text{if } t = \langle t_1, t_2 \rangle \wedge t_1 \neq * \\ \lambda_2(t_2) & \text{if } t = \langle t_1, t_2 \rangle \wedge t_2 \neq *. \end{cases}$

We write $\mathbf{N}_1 \otimes \mathbf{N}_2$ for $\mathbf{N}_1 \otimes_\emptyset \mathbf{N}_2$. Intuitively, the synchronised parallel composition $\mathbf{N}_1 \otimes_X \mathbf{N}_2$ is obtained by taking the disjoint union of the nets N_1 and N_2, except for those visible transitions labelled with a symbol $x \in X$, which are forced to fire synchronously. Concretely, 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 ($V_2^{\lambda_1(t_1)} = \emptyset$), it will not be executable in the synchronised product. This is obtained by turning transition t_1 into $\langle t_1, * \rangle$ and adding to its pre-set a new place p, which will never 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 will ensure 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 an operation for restricting the set of visible transitions of a net. It is called hiding as it has an obvious analogy with the corresponding operation of CSP processes.

Definition 11 (hiding). *Let* $\mathbf{N}$ *be a net with interfaces and let* $X \subseteq \Sigma$*. The hiding of* $\mathbf{N}$ *with respect to* X *is the net* $\mathbf{N} \backslash X = \langle I, O, N, V' \rangle$ *where* $V' = V \backslash V^X$*.*

Given a net $\mathbf{N}$, the restriction $\mathbf{N} \setminus X$ behaves exactly as $\mathbf{N}$, but transitions labelled in X, which were previously visible, are now hidden.

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

Definition 12 (marked nets). *A* marked net with interface $\mathbb{N}$ *is a pair* $\langle \mathbf{N}, m \rangle$*, where* $\mathbf{N}$ *is a net with interfaces and* $m \in S^\oplus$ *is the* initial *marking.*

For marked nets we can consider the language of traces starting from the initial marking and the corresponding equivalence.

Definition 13 (traces). *Let* $\mathbb{N}$ *be a marked net with interfaces. Its* set of traces *is* $traces(\mathbb{N}) = \{s \in \Sigma^* : \exists n.\, m \stackrel{s}{\Rightarrow} n\}$*. Two marked nets with interfaces* $\mathbb{N}_1$ *and* $\mathbb{N}_2$ *are* trace equivalent*, written* $\mathbb{N}_1 =_T \mathbb{N}_2$*, if* $traces(\mathbb{N}_1) = traces(\mathbb{N}_2)$*.*

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

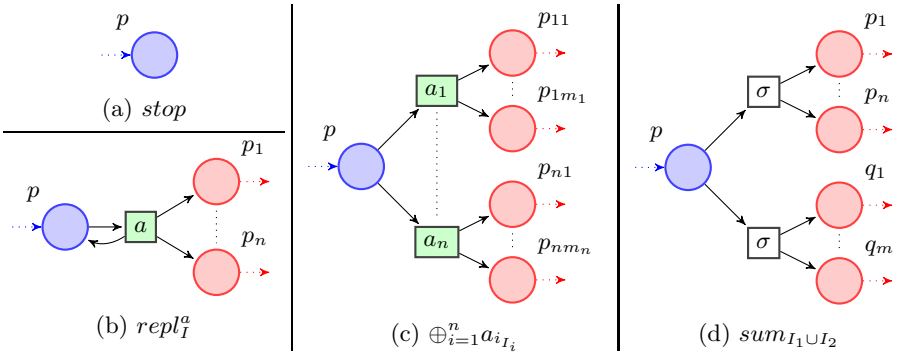

Fig. 5. The constant nets *stop*, $repl_I^a$, $\oplus_{i=1}^n a_{i_{I_i}}$ and $sum_{I_1 \cup I_2}$

Definition 14 (closing). *Let* $\mathbf{N}$ *be a net with interfaces. We denote by* $Cl(\mathbf{N})$ *the marked net with interfaces* $\langle\langle \emptyset, O, N, V\rangle, I\rangle$.

4 From Processes to Nets

This section introduces an encoding for CSP processes into nets with interfaces. It is defined inductively by exploiting the composition operators for nets introduced in Section 3. As anticipated, the encoding is restricted to bound processes.

The encoding relies on a set of constant nets, depicted in Fig. 5, which are combined using the composition operators on nets. The net *stop* in Fig. 5(a) consists of a single place. The net $repl_I^a$ in Fig. 5(b), where $a \in \Sigma$ and $I = \{p_1, \ldots, p_n\}$, 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. The set I is the set of input places of the encoding of the process under the replication operator. The net $\oplus_{i=1}^n a_{i_{I_i}}$ in Fig. 5(c), where $a_i \in \Sigma$ and each I_i is a set of places, is intended to provide a combinator for the guarded alternative. It consists of n transitions, labelled $a_1, \ldots, a_n$, all competing for the token in their common pre-set. Each transition a_i has $I_i = \{p_{i1}, \ldots, p_{im_i}\}$ as post-set, corresponding to the input places of the encoding of the continuation of a_i. Finally, the net $sum_{I_1 \cup I_2}$ in Fig. 5(d) is a combinator for nondeterministic (internal) choice. As above, $I_1 = \{p_1, \ldots, p_n\}$ and $I_2 = \{q_1, \ldots, q_n\}$ are sets of places which are the input places of the encodings of the processes involved in the choice. Note that the two transitions are hidden, so by definition of the operational semantics they will be turned into silent actions τ. Hence the label $\sigma \in \Sigma$, fixed for the hidden transitions of the internal choice, is totally irrelevant.

Definition 15 (encoding for processes). *Let* P *be a bound process. The encoding of* P*, denoted by* $[\![P]\!]$*, is defined as* $[\![P]\!] = Cl(|P|)$*, where* $|.|$ *is given inductively according to the rules in Fig. 6.*

The encoding of a process P is obtained by composing the encoding of its subprocesses and finally marking the input places. It therefore contains one place

$$
\begin{array}{lcl}
|STOP| & = & stop \\
|\oplus_{i=1}^{n} a_i.P_i| & = & \oplus_{i=1}^{n} a_{i I_{|P_i|}} \circ (\bigotimes_{i=1}^{n} |P_i|) \\
|P+Q| & = & sum_{I_{|P|} \cup I_{|Q|}} \circ (|P| \otimes |Q|) \\
|!_a.P| & = & repl^{a}_{I_{|P|}} \circ |P| \\
|P \backslash X| & = & |P| \backslash X \\
|P \mid_X Q| & = & |P| \otimes_X |Q|
\end{array}
$$

Fig. 6. Encoding for CSP processes

for each operator !, +, $\oplus$ and process $STOP$ of P. Some additional places are inserted by the synchronised parallel composition of nets in order to keep some components inactive (see Definition 10). Note that, in the following examples, we avoid to represent such places when they are isolated. 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.

Example 4 (prefix and parallel synchronised processes). Consider the process $P = (a.c.STOP \oplus b.d.STOP) \mid_{\{d\}} c.d.STOP$. Its encoding is depicted in Fig. 7 (right), where input and output interfaces are empty and all transitions are visible. It is obtained by closing the net $|P|$, the result of the parallel composition, synchronised on d, of the encodings $|a.c.STOP \oplus b.d.STOP|$ and $|c.d.STOP|$, in turn depicted in the left part of Fig. 7. More precisely, the net on the upper part illustrates $|a.c.STOP \oplus b.d.STOP|$. The places ν and σ represent the subnets encoding the $STOP$ processes (those reached after the events c and d, respectively). The subnet rooted at place δ is the encoding of the subprocess $c.STOP$. Analogously, the subnet rooted γ is the encoding of the subprocess $d.STOP$. The encoding of the subprocess $a.c.STOP \oplus b.d.STOP$ is obtained by sequentially composing the net $a.I_{\{\delta\}} \oplus b.I_{\{\gamma\}}$ with $|c.STOP| \otimes |d.STOP|$. The net in the lower part represents the encoding $|c.d.STOP|$ of $c.d.STOP$.

Example 5 (bound processes). Consider the process $Q = a.a.STOP \mid_{\{a\}} a.STOP$. The encodings $|a.a.STOP|$ and $|a.STOP|$ are depicted in Fig. 8(a) and (b). The encoding $|Q|$ is obtained as their parallel composition, synchronised on a, as shown in Fig. 8(c). Each transition labelled by a of $|a.a.STOP|$ is "combined" with any other transition labelled by a in $|a.STOP|$. 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 $Q \xrightarrow{a} a.STOP \mid_{\{a\}} STOP$, in such a way that the remaining occurrence of a cannot be executed since it has no counterpart in the parallel subprocess.

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

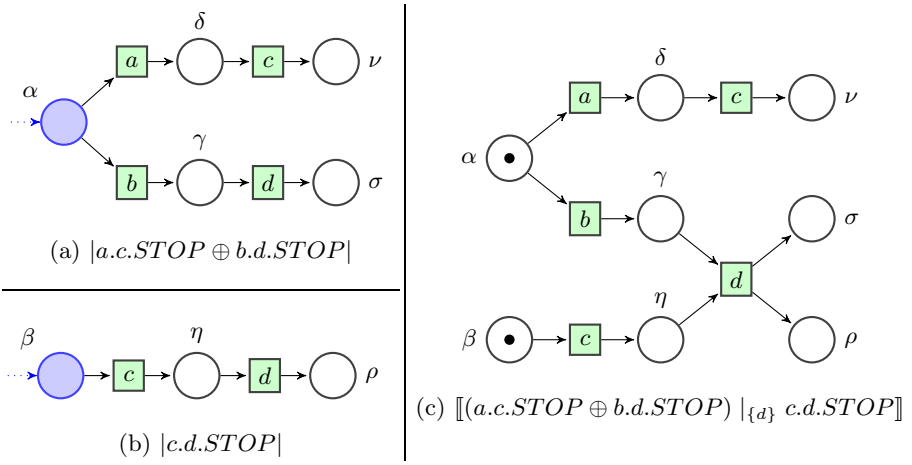

Fig. 7. Some process encodings

The net $[\![R]\!]$ in Fig. 8 (d) is obtained by closing the net $|R|$, which in turn is the sequential composition of $repl^{b}_{\{\alpha,\delta\}}$ with $|Q|$. Notice 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 disagree with the semantics of the CSP process.

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 non-bound processes.

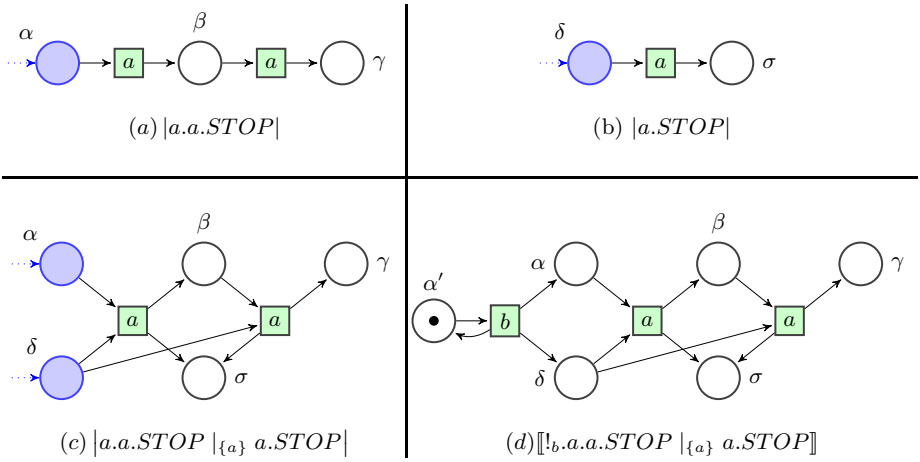

Fig. 8. Process encodings

5 Relating CSP and Labelled 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 processes preserves and reflects process transitions, and, consequently, the standard behavioural CSP equivalences such as, for instance, trace equivalence.

In order to state these results, we first need to establish a correspondence between the processes reachable from P, hereafter denoted by the set $reach(P) = \{Q : P \rightarrow^* Q\}$, and the markings of $[\![P]\!]$.

The encoding of a bound process P is inductively defined as the composition of the encoding of its subprocesses. Note that, by definition of the CSP operational semantics, whenever a process P performs a transition to P', the process P' is obtained from P by replacing a subprocess with its reduct. Then, it is easy to see that the encoding of those processes reachable from P can be mapped to subnets of $[\![P]\!]$.

Lemma 1 (reachable processes as subnets). *Let P be a bound process and let Q be a subprocess of P or a process reachable from P. Let N_P and N_Q be the labelled nets underlying the encodings $[\![P]\!]$ and $[\![Q]\!]$, respectively. Then, a net morphism $f_{Q,P} : N_Q \rightarrow N_P$ can by uniquely chosen.*

The proof relies on the fact that given a subprocess Q of a process P, a mapping between the net underlying the encoding $[\![Q]\!]$ into the one underlying $[\![P]\!]$ can be obtained by the inductive definition of the encoding. Hence, each subprocess of P corresponds to a subnet of $[\![P]\!]$. Using this fact, it is not difficult to prove that also the encoding of a process reachable from P can be mapped to a subnet of $[\![P]\!]$. 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 synchronise on any event, can be mapped to the same subnet.

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

Definition 16. *Let P be a bound process. The function $\mathbf{m}^P : reach(P) \rightarrow S^{\oplus}_{[\![P]\!]}$ maps any process $Q \in reach(P)$ into the marking $f^{\oplus}_{Q,P}(m_{[\![Q]\!]})$.*

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

Theorem 1. *Let P be a bound process and let $Q \in reach(P)$. 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 $Q \xrightarrow{\mu} R$ with $m = \mathbf{m}^P(R)$.*

The result establishes a bijection between the labelled transitions performed by any process $Q \in reach(P)$ and the transition firings in the net $[\![P]\!]$ from the marking $\mathbf{m}^P(Q)$.

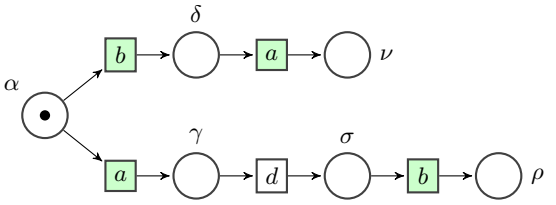

Fig. 9. Net encoding the process $(d.a.b.STOP)\backslash d \oplus b.a.STOP$

Such a bijection can then be lifted to a fundamental correspondence between the trace semantics of processes and of their encodings.

Theorem 2. *Let P, Q be bound processes. Then*

$$P =_T Q \text{ if and only if } [\![P]\!] =_T [\![Q]\!].$$

Example 6. Consider the processes $P = a.(d.b.STOP\backslash d) \oplus b.a.STOP$ and $Q = (c.a.STOP \mid_{\{c\}} c.b.STOP)\backslash c$ of Example 1. The encoding of P is in Fig. 9, while the encoding of Q is the net in Fig. 3 (right), once the interfaces are removed. Note that there is a correspondence between the labelled transitions of each process and those of its encoding. For instance, $P \xrightarrow{a} d.b.STOP\backslash d$ corresponds to $\mathbf{m}^P(P) = \{\alpha\} \xrightarrow{a} \{\gamma\} = \mathbf{m}^P(d.b.STOP\backslash d)$. The transition $d.b.STOP\backslash d \xrightarrow{\tau} b.STOP\backslash d$ corresponds to $\mathbf{m}^P(d.b.STOP\backslash d) = \{\gamma\} \xrightarrow{\tau} \{\sigma\} = \mathbf{m}^P(b.STOP\backslash d)$ and $b.STOP\backslash d \xrightarrow{b} STOP\backslash d$ to $\mathbf{m}^P(b.STOP\backslash d) = \{\sigma\} \xrightarrow{b} \{\rho\} = \mathbf{m}^P(STOP\backslash d)$. Moreover, $P \xrightarrow{b} a.STOP$ corresponds to $\mathbf{m}^P(P) = \{\alpha\} \xrightarrow{b} \{\delta\} = \mathbf{m}^P(a.STOP)$, and finally $a.STOP \xrightarrow{a} STOP$ to $\mathbf{m}^P(a.STOP) = \{\delta\} \xrightarrow{a} \{\nu\} = \mathbf{m}^P(STOP)$.

We have a correspondence also between the transitions of Q and those of its net encoding. Therefore, it is easy to conclude that the nets are trace equivalent.

6 Some Hints about Technology Transfer

The encoding of bound CSP processes into labelled nets enables to transfer results concerning expressiveness and tractability from one formalism to the other, as it was the case for the net encoding of CCS in [1].

Observe that trace equivalence is obviously undecidable for both bound CSP processes and Petri nets (since they include as a fragment the basic parallel processes for which trace equivalence is known to be undecidable [11]). Still, even though this does not give new insights, we note that by using the encoding, the undecidability of trace equivalence for Petri nets can be also deduced directly from the undecidability of trace equivalence for bound CSP.

Reachability, namely the possibility of reaching a given process Q via a sequence of transitions from a start process P, is not a particularly interesting property for CSP. Since during process evolution the number of parallel components can only increase, the property turns out to be decidable. Indeed, in order

to establish whether Q is reachable it suffices to consider the fragment of the LTS including the processes reachable from P having a number of parallel subprocesses bounded by that of Q. It is instead more interesting the reachability under the garbage collection of $STOP$: it breaks the monotonicity mentioned above, and the removal itself is not trivial. In fact, $P \mid_X STOP$ is equivalent to P only when X does not include channel names on which P can synchronise. E.g., think of the process $a.STOP$ which can perform an a-labelled transition while $a.STOP \mid_{\{a\}} STOP$ is deadlocked.

Alternatively one can consider control state reachability, i.e., the reachability of a configuration including a given subprocess. In this last case, it is sufficient to consider net coverability. It is folklore that the control state reachability problem is undecidable for full CSP, while the corresponding property of coverability is known to be decidable for Petri nets [8]. By exploiting the encoding, decidability of coverability can be transferred from Petri nets to bound processes.

Corollary 1 (reachability). *Let P, Q be bound processes. The problem of establishing whether there exists bound process R such that $P \rightarrow^* R$ and Q is a sub-process of R is decidable.*

Thanks to the correspondence between processes reachable from a process P and reachable markings in the net encoding of P, decidability of boundedness in Petri nets [8] implies that it is possible to determine whether a CSP process has a finite number of states.

Corollary 2 (finite state). *Let P be a bound process. It is decidable whether P has a finite number of reachable states.*

Again, the property of being finite state can be more interesting for CSP processes when working up to garbage collection of useless STOP parallel components. It can be seen that this property is naturally captured by the boundedness of the subset of places of $[\![P]\!]$ not corresponding to STOP processes.

Analogously, it is possible to identify an upper bound to the degree of parallelism of a bound CSP process, i.e., to the number of parallel subcomponents of a process during its evolution. More precisely, define the structural degree of a CSP process P as $sdeg(P \mid_X Q) = sdeg(P) + sdeg(Q)$ and $sdeg(P) = 1$ otherwise. Then the degree of P is $deg(P) = \sup\{sdeg(P') : P \rightarrow^* P'\}$. The close correspondence between $deg(P)$ and the maximal total number of tokens in the reachable markings of $[\![P]\!]$ immediately leads to the following result.

Corollary 3 (parallelism). *Let P be a bound process. The problem of determining whether $deg(P)$ is finite is decidable. Moreover, for any given $k \in \mathbb{N}$, it is decidable whether $deg(P) \leq k$.*

A classical property in the analysis of the expressiveness of process calculi is convergence, i.e., the existence of a terminating computation. We recall such notion below, according to [5].

Definition 17 (convergence). *A process P is called* convergent *if there exists Q such that $P \rightarrow^* Q \not\rightarrow$.*

Convergence of a bound process can be reduced to the existence of a deadlock in its encoding, a property which is known to be decidable for Petri nets [8].

Corollary 4 (convergence). *Convergence is decidable for bound processes.*

7 Conclusions and Further Works

In this work we have identified a fragment of CSP, consisting of what we called bound processes, that can be encoded into (labelled) Petri nets. The encoding preserves and reflects the (strong) transitions of the process calculus and, consequently, the whole spectrum of (both strong and weak) behavioural equivalences definable on the transition system of CSP processes. Furthermore, the encoding is syntax-driven, hence modular, mapping each process operator into a suitable one for labelled nets with interfaces. As far as we know, this is a main improvement with respect to former proposals.

Interfaces are in fact the key ingredient to achieve modularity: they are needed in defining the net operators upon which our encoding lays its foundations. Reactive extensions of Petri nets, endowed with means for interacting with the environment [3,13,2,4], naturally arise as extensions of nets allowing for compositional reasoning. This feature plays a key role for modelling various brands of process calculi. Indeed, in [1] they were pivotal in the encoding of an asynchronous fragment of CCS into open nets. Interestingly enough, asynchronous interactions are captured by a form of composition where net components interact only over places, while in this paper the synchronous interaction is realised by letting net components interact over an interface consisting of transitions. This should have been intuitively expected, since in Petri nets the token flow is eminently asynchronous, while transitions synchronise different token flows.

Therefore, our results confirm that (open) Petri nets can accommodate both asynchronous message passing and barrier synchronisation. In principle, this leaves space for a calculus combining both characteristics, endowed with a direct encoding into nets. This would take us back in full circle, since this calculus would trace its roots on some early proposals for net encoding of processes [16].

Acknowledgements. We are grateful to the referees for their insightful suggestions on the submitted version of the paper, and, in particular, for pointing out the relevance of control state reachability in the analysis of CSP processes.

References

1. Baldan, P., Bonchi, F., Gadducci, F.: Encoding asynchronous interactions using open Petri nets. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 99–114. Springer, Heidelberg (2009)
2. Baldan, P., Corradini, A., Ehrig, H., Heckel, R., König, B.: Bisimilarity and behaviour-preserving reconfigurations of open Petri nets. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 126–142. Springer, Heidelberg (2007)

3. Best, E., Devillers, R., Hall, J.G.: The Petri box calculus: a new causal algebra with multi-label communication. In: Rozenberg, G. (ed.) APN 1992. LNCS, vol. 609, pp. 21–69. Springer, Heidelberg (1992)
4. Bruni, R., Melgratti, H.C., Montanari, U., Sobocinski, P.: Connector algebras for C/E and P/T nets' interactions. Logical Methods in Computer Science 9(3), 1–65 (2013)
5. Busi, N., Gabbrielli, M., Zavattaro, G.: Comparing recursion, replication, and iteration in process calculi. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 307–319. Springer, Heidelberg (2004)
6. Degano, P., Gorrieri, R., Marchetti, S.: An exercise in concurrency: a CSP process as a condition/event system. In: Rozenberg, G. (ed.) APN 1988. LNCS, vol. 340, pp. 85–105. Springer, Heidelberg (1988)
7. Devillers, R., Klaudel, H., Koutny, M.: A compositional Petri net translation of general pi-calculus terms. Formal Aspects of Computing 20(4-5), 429–450 (2008)
8. Esparza, J., Nielsen, M.: Decidability issues for Petri nets - a survey. Elektronische Informationsverarbeitung und Kybernetik 30(3), 143–160 (1994)
9. Goltz, U., Reisig, W.: CSP-programs with individual tokens. In: Rozenberg, G. (ed.) APN 1984. LNCS, vol. 188, pp. 169–196. Springer, Heidelberg (1985)
10. Gorrieri, R., Montanari, U.: SCONE: A simple calculus of nets. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 2–30. Springer, Heidelberg (1990)
11. Hirshfeld, Y.: Petri nets and the equivalence problem. In: Börger, E., Gurevich, Y., Meinke, K. (eds.) CSL 1993. LNCS, vol. 832, pp. 165–174. Springer, Heidelberg (1994)
12. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Upper Saddle River (1985)
13. Kindler, E.: A compositional partial order semantics for Petri net components. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 235–252. Springer, Heidelberg (1997)
14. Llorens, M., Oliver, J., Silva, J., Tamarit, S.: Generating a Petri net from a CSP specification: A semantics-based method. Advances in Engineering Software 50, 110–130 (2012)
15. Nielsen, M., Priese, L., Sassone, V.: Characterizing behavioural congruences for Petri nets. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 175–189. Springer, Heidelberg (1995)
16. Olderog, E.R.: Operational Petri net semantics for CCSP. In: Rozenberg, G. (ed.) APN 1987. LNCS, vol. 266, pp. 196–223. Springer, Heidelberg (1987)
17. Reisig, W.: Understanding Petri Nets. Springer (2013)
18. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall, Upper Saddle River (1998)