The Ins and Outs of Petri Net Composition

Elvio G. Amparore(✉) and Susanna Donatelli

Università degli Studi di Torino, Torino, Italy
{amparore,susi}@di.unito.it

Abstract. Composition is a key issue in Petri net modelling. It is a topic that has been studied for a long time, and that finds practical application in many Petri net frameworks and tools. Multiple approaches to composition exist, based on place or transition refinement, place superposition, transition synchronization, or sub-net substitution to cite a few. In this paper we revisit the peculiarities and technicalities (the ins and outs) of net composition based on the labelling of the net elements. We shall express general composition of nets through a combinatorial operator, that, instantiated with different policies, and completed with operators for parallel composition, label rewriting and restriction, allows us to define different forms of place-based and transition-based compositions. The use of this composition framework for model construction is also examined. For composition based on multisets of labels, we also provide an algorithm for the construction of the composed net that uses a modified version of the Farkas algorithm for the computation of semiflows.

Keywords: Petri net composition · Petri Box Calculus · CCS · CSP

1 Introduction

Composition has attracted the interest of Petri net researchers from the very early stages of the research in the field. Composition has been studied as an algebra for building nets from smaller "basic" blocks, as in the seminal work on Petri Box Calculus (PBC) [6,7] or as a way to compose existing models, independently from how they have been built (as in many tools). The definition of a "well-thought" algebra typically allows to exploit the composition also at the solution level, possibly at the price of some rigidity in the modelling process. On the other side, composing arbitrary Petri nets provides a lot of flexibility, but typically it is not as strong in terms of compositional properties and analyses. When a full algebra, with operators and associated properties is not available, it is left to the modeller to compose "reasonable" models in a "reasonable" way.

There exist multiple ways to perform composition. Models can be composed based on place or transition superposition, more rarely on both, and on place, transition or subnet substitution (also known as *refinement*). Composition rules can be based on place and transition names or on labels associated to the net elements. Especially for what concerns composition based on transitions, different

© Springer Nature Switzerland AG 2022

interpretations are present: we shall call *CCS-like* the CCS [26] inspired transition composition (like in PBC), and *CSP-like* the CSP [19] inspired transition superposition (like in [5]).

While net algebras and their operators have been studied in-depth [6,7,23], less attention has been devoted to the composition of arbitrary nets. The research questions that motivated our work were to understand the ins and outs of net composition, and how different forms of composition can be described, and later implemented, in a single framework, and whether the standard duality principle of place and transitions carries over in this context. As a result we have defined a framework for net composition that encompasses different composition rules, whether based on net elements' label, set of labels, or multisets of labels, and whether rooted on place superposition or transition synchronization, or both at the same time, and considering different form of transition-based synchronization.

In this framework, composition of nets is expressed through a *combinatorial operator*, that, instantiated with different *policies*, allows us to define different forms of place-based and transition-based compositions. Composition policy rules are defined over the labelling of the net elements using multisets of labels. The framework is completed by operators for parallel composition, label rewriting and restriction, all implemented in the GreatSPN [1] software. The proposed framework does not include explicit operators for recursion, nor for place, transition or subnet substitution/refinement.

We can summarize the paper's contributions as follows:

– A new framework for Petri net composition in which place- and transition-based composition are treated uniformly, with a new generic composition technique, controlled by an input composition *policy instance.*
– Two composition policies (unary conjugated and n-ary structured, inspired by CCS, CSP, and PBC) and the algorithms to compute the associated policy instances.
– Examples of how known operators of other languages and various modelling patterns can be defined in the proposed framework.

1.1 A Few Examples of Net Composition

Before proceeding to the main part of the paper, it is worth to set the ground by examining a few examples of net composition. We limit these first examples to nets in which at each element is associated at most one label.

Figure 1 shows an example of *CCS-like* parallel composition, which is based on actions and co-actions (*conjugate* actions). The co-action of a is named $\hat{a}$. Actions are the labels, and are depicted on top of the transition names. In CCS when two processes are put in parallel, each action of one process synchronizes with the co-actions of the other process and vice-versa. The joint action and co-action leads to a new transition labeled τ, and no further synchronization can occur. In the figure the label τ is omitted, assuming that each net element that does not have an associated label is labelled with τ. Moreover each action and co-action are still executable in the composed process, unless a *restriction*

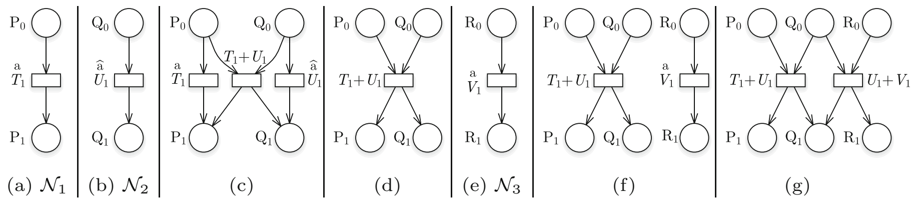


Fig. 1. CCS-like conjugate synchronization.

is specified. Figure 1(c) shows the composition of $\mathcal{N}_1$ with $\mathcal{N}_2$: it features a synchronized transition labelled τ, while still allowing both a and $\hat{a}$ to be executed. Figure 1(d) shows the composition of $\mathcal{N}_1$ with $\mathcal{N}_2$ with restriction over a: it features only the τ labelled synchronized transition. If the net in Fig. 1(d) is further composed with $\mathcal{N}_3$ no synchronization is possible, resulting in the net in Fig. 1(f).

If instead $\mathcal{N}_1$ and $\mathcal{N}_3$ are composed first, since all transitions are labelled with a, no synchronization occurs. If the resulting net is then composed with $\mathcal{N}_2$, it results in the net in Fig. 1(g), with U_1 composing with both T_1 and V_1. *From a modelling point of view* this can be seen as two processes ($\mathcal{N}_1$ and $\mathcal{N}_3$) that access at their will, but in mutual exclusion, the same resource, where label a can be interpreted as "providing" a resource and $\hat{a}$ as "requesting" it.

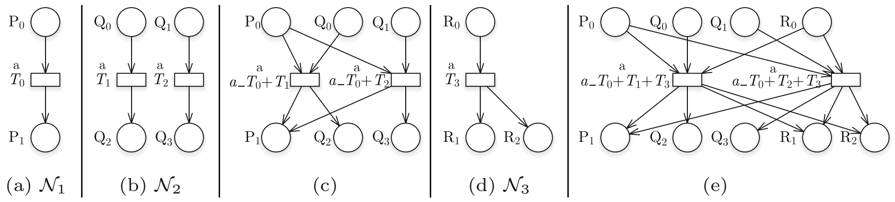


Fig. 2. CSP-like synchronization.

Figure 2 shows an example of a *CSP-like* parallel composition on a synchronization set S made of the single action a. In CSP there is no notion of co-actions. When two processes synchronize over S, each action $a \in S$ of one process synchronizes with every other action a in the other process. Actions that are not in S can still be freely executed. There is no restriction on the execution of actions that are not in S. The transition that represents the synchronization is also labelled with a, so that further synchronization can occur, permitting a straightforward implementation of the synchronization among any number of processes (multi-way synchronization).

Figure 2(c) is the result of the composition of $\mathcal{N}_1$ and $\mathcal{N}_2$ over the synchronization set $S = \{a\}$. All transitions labeled with a from $\mathcal{N}_1$ (i.e. T_0) are composed with every transition labeled with a from $\mathcal{N}_2$ (i.e. T_1 and T_2), resulting in two new transitions $a\_T_0{+}T_1$ and $a\_T_0{+}T_2$. Once synchronized, the merged transitions T_0, T_1 and T_2 are not preserved, and do not appear in (c). When (c) is composed with $\mathcal{N}_3$ resulting in net (e) that can be interpreted as a multi-way synchronization, modelling the case in which three processes need to reach a shared barrier, but may choose two different ways to do so.

From a modelling point of view also the use of CSP-like synchronization may require some cautions. With the same example of Fig. 2, if we interpret $\mathcal{N}_1$ as a resource, and $\mathcal{N}_2$ as two requests for that resource, than the net in Fig. 2(c) correctly represents the acquisition of the resource. If the obtained model is later composed with a net like $\mathcal{N}_3$, which feature *another* request of the resource, the resulting model, shown again in Fig. 2(e), does not correctly represent the resource acquisition. Indeed in this case the order of composition is important: to get the intended behaviour all resource requests have to be composed first (by composing $\mathcal{N}_2$ with $\mathcal{N}_3$ with an empty synchronization set) and then they may be composed with the resource model of $\mathcal{N}_1$.

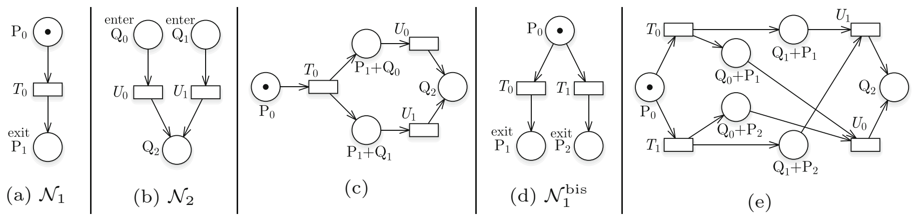


Fig. 3. Place-based composition.

Figure 3 shows an example of place composition. The net in Fig. 3(c) is the result of a simple sequential composition of $\mathcal{N}_1$ and $\mathcal{N}_2$: places are labelled as *enter*, *exit*, or τ, and exit places of net $\mathcal{N}_1$ are superposed to the enter places of net $\mathcal{N}_2$. The net in Fig. 3(e) is instead the result of the composition of net $\mathcal{N}_1^{\text{bis}}$ with net $\mathcal{N}_2$: the two exit places combined with the two enter places produce four places in the resulting net, which induces a synchronization over U_0 and U_1 among two "control flows" (the one coming through P_1 and the one from P_2) that are in mutual exclusion in $\mathcal{N}_1^{\text{bis}}$, leading to a deadlock. This is somehow counter-intuitive and shows that the PBC choice of having only two labels for places (*enter*, *exit*) may not be always adequate from a modelling point of view.

2 Previous Work on Composition

This section reviews some of the most relevant net algebras and examine which composition support is provided by a number of well-known Petri net tools. This review is certainly non-exhaustive, but it is meant to overview the large variety of composition rules present in the literature and provide pointers to them. Note that most of the literature presents nonuniform criteria to compose places and transitions. The framework proposed in this paper (Sect. 4) will follow instead the classical duality of Petri nets, and place- and transition-based composition will be treated using uniform rules. Since most net composition is based on *labels*, the type of labelling function is also a relevant aspect.

2.1 Net Algebras and Composition Frameworks

An early proposal for an algebra of Petri nets was provided in [23]. Starting from *simple nets* (a sequence of "head" place - transition - "tail" place), larger nets are generated through tail over head place-based composition operators. The list of operators includes superposition, merging, joining, exclusion, and other operations, with a focus on preserving structure in the composition formula. Transition-based refinement (a transition is substituted by an expression of nets) is also defined.

An early proposal for CSP-like transition synchronization can be found in [13].

Petri Box Calculus (PBC from now on) is a complete algebra that features operators inspired by those found in CCS, but operates on a specific class of Petri nets known as *Petri boxes* (nets where places are automatically labeled as *enter*, *exit* or internal). For what concerns basic composition mechanisms as for the scope of this paper, PBC performs various kinds of transition and place compositions. Transition synchronization is based on actions and co-actions. Transitions can be freely labeled with a multiset of labels. Places instead may only have a single label among $\{enter, exit\}$, as already mentioned.

Place and transition composition operators are separated. Places can be composed as *sequences* or *choices*. Transitions are composed by *synchronization*, which is a unary operator. Performing $\mathcal{N}\,\mathrm{sy}\{a\}$ synchronizes label a over net $\mathcal{N}$, which leads to the addition of a new transition for each pair of $a, \hat{a}$ transitions that can be "merged". Unary synchronization can also lead to unexpected consequences, see [7, p. 23]. Multiple nets can be composed by *parallel composition*, followed by a synchronization.

In PBC transition labels are *multisets*, which is needed to ensure that the synchronization of multiple labels is order-independent. This choice is relevant (see [7, p. 21]), and for this reason in this paper we also consider multisets of labels for each net element.

A limitation of PBC is that it does not have an explicit "multi-handshake" (or multi-way) synchronization. This type of synchronization occurs in practice [7, Sec. 2.8 and Chap. 9], for example every time a process needs to perform atomic operations on multiple variables. The *box algebra* [6, Sec. 4] is a generalization of PBC, a more abstract and general algebra that allows for multi-way synchronization, which facilitate the definition of a box algebra semantics of a process algebra like TCSP [18], as illustrated in [7, Sec. 8.2.2]. A second limitation of PBC is that its algorithm for the computation of the synchronized transitions may not terminate, generating infinite synchronizations [6, Sec. 4.5].

The work in [2] proposes a formal composition model for synchronizing multilabeled transitions in a similar way to PBC but, unlike PBC, this approach is guaranteed to always terminate. This synchronization is shown to be equivalent to the *semiflows* computation problem (i.e. finding the anullers of a matrix).

The use of semiflows, however, hides some technical details related to minimality, which result in generating only a subset of the possible interactions. We shall review this approach to overcome its limits in Sect. 5.1.

The Petri net standard PNML [29] provides support for *multi-page nets* (files containing multiple nets), but does not include a compositional specification. An examination of how to add modules in PNML, and how to construct nets from instances of such modules is given in [22], in an high level Petri net context.

Driving the modeller in the use of composition to build large models of computing systems was the objective of the PSR methodology [14], that organizes models into three layers (Processes, Services, and Resources). Each layer is defined in isolation, and then composed through CSP-like transition superposition. Transitions may have a set of associated labels, but in a well-defined manner: only single labels in the Resource layer (to model the "offer" of a resource), sets of labels for the Service layer (to model a service that acquires two or more resources at the same time) and single labels for the Process layer. Multisets of labels are not allowed, which means, for example, that a service cannot acquire two copies of the same resource at the same time.

Composition has also been considered for colored and high level nets: here the additional complexity is to appropriately define how to deal with all the extra information associated to places, transitions and arcs. High-level Petri net composition using the BPN^2 framework was introduced in [8]. Such composition is shown to be consistent with the unfolding and the operators of PBC. In CPN, a model [21] can be organised as a set of hierarchically related modules. "Substitution" transitions are replaced by subnets with well specified place-oriented input and output ports. Component aggregation of CPNs is described in [20], based on communicating modules. Hierarchical composition of Generalized Colored Stochastic Petri Nets (GCSPN) was first defined in [9], while transition superposition for the colored class of *well-formed nets* [10] was defined in [4]. A colored extension of the PSR methodology is given in [4].

2.2 Composition in Tools

Many tools supports some form of compositionality. Snoopy [17] implements hierarchical nets through P/T refinements. Nodes can be abstracted by a macro node, and a fine/coarse hierarchy can be visualized [16].

A CPN model in the CPN-Tools framework [27] can be defined hierarchically using pages and subpages [21]. Special substitution transitions connect superpages with subpages, and special tags (in/out) in the subpage allow to define the inner behaviour.

In Möebius [11] submodels are composed through superposition of places (shared state variables) [24]. It has two state-sharing formalisms: *Replicate/Join* composition and *Graph* composition. Fused places/transition have the same name (that must be unique in every composed model), hence it reduces to a simple merge of the same-name elements, and no complex combination of multiple net elements is possible.

In the ITS-tools framework [28] multiple GAL (*Guarded Action Language*) instances can be composed over synchronized events, but not over shared variables [25]. Events are labeled with symbols which guide the synchronization [3].

The GreatSPN [1] tool supports binary composition over labelled places and transitions, with some restricted form of multiset labelling and some support for colored net composition, following the rules defined in [4,5].

3 Definitions

Let Σ be a set of tags (also called symbols, actions, etc.). Tags will be used to label the places and transitions of a Petri net. Since we also label places, we prefer the use of the term *tag* instead of the more broadly used term *action*. Given a tag $a \in \Sigma$, let $\hat{a}$ be its *conjugate* (or complementary) tag. By convention $\hat{a} \neq a$ and $\hat{\hat{a}} = a$. Let $\hat{\Sigma} = \Sigma \cup \{\hat{a} \mid \forall a \in \Sigma\}$ be the set of all tags including their conjugated counterparts and $\mathcal{M}(\hat{\Sigma})$ be the set of all natural multisets of tags (including their conjugates). Elements of $\mathcal{M}(\hat{\Sigma})$ are indicated by formal sums and τ denotes the empty multiset, so, given the set of tags $\Sigma = \{a, b, c\}$, $a + 2 \cdot \hat{a} + 2b$, $\hat{a} + c$ and τ are examples of multisets of tags. A multiset of tags is *canonical* if it does not include both a tag and its conjugate (therefore $a+2\cdot\hat{a}+2b$ is not canonical).

Given $\sigma \in \mathcal{M}(\hat{\Sigma})$ and $A \subseteq \Sigma$, we indicate with $\sigma \setminus A$, the multiset obtained by removing all tags in A, and their conjugates. Notation $\sigma[a]$ denotes the multiplicity of a in σ.

Definition 1 (Labeled Petri net). *It is a tuple $\mathcal{N} = \langle P, T, I, O, \mathbf{m}_0, lab\rangle$, where P is the set of places, T is the set of transitions, $I : P \times T \to \mathbb{N}$ is the input function, $O : T \times P \to \mathbb{N}$ is the output function, $\mathbf{m}_0 : P \to \mathbb{N}$ is the initial marking, and $lab : (P \cup T) \to \mathcal{M}(\hat{\Sigma})$ is the net element labeling function.*

We use the term *net element* to identify elements in $(P \cup T)$ and *label* (of a net element) to indicate the multiset of tags associated to the net element by the labelling function *lab*. We consider only labels that are canonical. Let $\hat{\Sigma}_P$ and $\hat{\Sigma}_T$ be the subsets of $\hat{\Sigma}$ that appear on the labels of the place set P and on the transition set T, respectively. With $\mathcal{M}(P)$ we denote the set of the natural multisets of places, which can be represented as a weighted sum of elements of P, like $P_1 + 3{\cdot}P_4 + P_5$. Similarly $\mathcal{M}(T)$ is used for the transitions.

For notational convenience, we also use a matrix-oriented representation of a Petri net. Let $\mathbf{I} : |P| \times |T|$ and $\mathbf{O} : |T| \times |P|$ be the input and the output matrix of $\mathcal{N}$, respectively, with $\mathbf{I}[p, t] = I(p, t)$ and $\mathbf{O}[p, t] = O(p, t)$. Let $\mathbf{L}_{\mathrm{P}} : |P| \times |\Sigma_P|$ be the place labeling matrix, where $\mathbf{L}_{\mathrm{P}}[p, a]$ is the multiplicity of tag a in $lab(p)$ and it is negative if a appears conjugated, positive otherwise. Similarly, let $\mathbf{L}_{\mathrm{T}}$: $|T| \times |\Sigma_T|$ be the transition labeling matrix. A full example of the net matrices will be given at the end of Sect. 4.

Definition 2 (Semiflows). *Given an integer matrix $\mathbf{A}$, a flow $\mathbf{f}$ is an integer vector s.t. $\mathbf{f} \cdot \mathbf{A} = 0$, i.e. $\mathbf{f}$ is a* left anuller *of $\mathbf{A}$. A semiflow is a non-negative* *flow. The support* $[\![\mathbf{f}]\!]$ *of a flow* $\mathbf{f}$ *is the set of indices of the non-zero values, i.e.* $[\![\mathbf{f}]\!] = \{i \mid \mathbf{f}[i] \neq 0\}$*. A semiflow is canonical iff the g.c.d. of its non-zero entries is* 1*. A semiflow is minimal iff it is canonical and its support does not strictly contain the support of any other semiflow of* $\mathbf{A}$*. The set of all minimal semiflows is finite and unique, and let* $\mathbf{F}$ *be the matrix of the minimal semiflows [12, p. 82].*

4 A Framework for Net Composition

We proceed by defining a framework for composing labelled Petri nets that allows us to define, among others, the cases discussed in Sect. 1. We define the basic operations for net composition in terms of four basic operations. Three operations perform basic transformations on labels (tag rewriting, restriction) and merge multiple nets together without combining the elements (parallel composition). The fourth operation (combinatorial composition) combines net elements together by applying a policy.

Tag Rewriting. A *tag rewriting function* is a function $\lambda : \hat{\Sigma} \to \hat{\Sigma}$ that transforms tags. By extension, given a multiset of tags $\phi = w_1 \cdot a_1 + \ldots + w_n \cdot a_n$, let $\lambda(\phi)$ be the canonical multiset resulting from the application of λ to every tag, i.e. the canonical form of $w_1 \cdot \lambda(a_1) + \ldots + w_n \cdot \lambda(a_n)$. We define the *tag rewriting* operation on a net $\mathcal{N}$, denoted as $\lambda(\mathcal{N})$, as an operation that builds a new net $\mathcal{N}'$ where labels have been rewritten, i.e. $lab' = \lambda \circ lab$.

Parallel Composition. This operation juxtaposes multiple independent nets together into a single net. Given $\mathcal{N}_1 \ldots \mathcal{N}_n$ nets, let $\mathcal{N}_1 \parallel \ldots \parallel \mathcal{N}_n$ be a new net $\mathcal{N}'$ defined as:

- $P' = \cup_{i=1}^{n} P_i$ and $T' = \cup_{i=1}^{n} T_i$;
- $I'(p', t') = I_{\theta(p')}(p', t')$ if $\theta(p') = \theta(t')$, and 0 otherwise;
- $O'(t', p') = I_{\theta(p')}(t', p')$ if $\theta(p') = \theta(t')$, and 0 otherwise;
- $\mathbf{m}'_0(p') = (\mathbf{m}_0)_{\theta(p')}(p')$;
- $lab'(p') = lab_{\theta(p')}(p')$ and $lab'(t') = lab_{\theta(t')}(t')$;

where the function $\theta : P \times T \to \mathbb{N}$ is defined to associate each P/T elements of $\mathcal{N}$ to the index of the original net $\mathcal{N}_i$.

Restriction. This operation removes from a net $\mathcal{N}$ all elements whose label includes any of the tags in the set of *restriction tags* $A \subseteq \Sigma$ or their conjugates. The new net is indicated as $\mathcal{N}' = \mathcal{N} \setminus A$ and it is defined by:

- $P' = \big\{p \in P \mid \forall\, a \in A : lab(p)[a] = lab(p)[\hat{a}] = 0\big\}$;
- $T' = \big\{t \in T \mid \forall\, a \in A : lab(t)[a] = lab(t)[\hat{a}] = 0\big\}$;
- $I'(p, t) = I(p, t)$ and $O'(t, p) = O(t, p)$, for all $p \in P', t \in T'$;
- $\mathbf{m}'_0(p) = \mathbf{m}_0(p)$, for all $p \in P'$.
- $lab'(x) = lab(x)$, for all $x \in P' \cup T'$.

Combinatorial Composition. This operation alters the behaviour of a net $\mathcal{N}$ by defining a new set of places and transitions made as combinations of the net elements of the original net. Each new place (resp. transition) that is being composed is identified by a multiset of *composing places (resp. transitions)* from the original net. We divide the net composition into two tasks:

1. Identifying which *multisets* of places (transitions) will be composed together to form each new place (transition). These are described by a *composition instance* π, which can be generated by *composition policy* (defined in the next section).
2. Defining a new net with the new net elements, connected according to a composition of the original input and output functions.

A *composition instance* π is a pair $\pi = \langle C_P, C_T \rangle$, with $C_P \subseteq \mathcal{M}(P) \times \mathcal{M}(\hat{\Sigma}_P)$ and $C_T \subseteq \mathcal{M}(T) \times \mathcal{M}(\hat{\Sigma}_T)$. We use the notation $\langle \phi, \sigma \rangle$ to denote tuples in C_P, and $\langle \psi, \varsigma \rangle$ to denote tuples in C_T.

Given a net $\mathcal{N}$ and a composition instance π, the *combined net* $\mathcal{N}' = \mathcal{N} * \pi$ is obtained in the following way. Each tuple $\langle \phi, \sigma \rangle \in C_P$ defines a new place p' of $\mathcal{N}'$, s.t. the multiset ϕ tells the weighted combination of places of $\mathcal{N}$ that are combined together to form p', while σ is the label of p' Transitions follow a similar schema from C_T.

The combined net $\mathcal{N}'$ is defined as

– $P' = \{\text{new place } p' \text{ for each } \langle \phi, \sigma \rangle \in C_P\}$;
– $T' = \{\text{new transition } t' \text{ for each } \langle \psi, \varsigma \rangle \in C_T\}$;
– $I'(p', t') = \sum_{p \in P} \sum_{t \in T} \phi_{p'}[p] \cdot \psi_{t'}[t] \cdot I(p, t)$;
– $O'(p', t') = \sum_{p \in P} \sum_{t \in T} \phi_{p'}[p] \cdot \psi_{t'}[t] \cdot O(p, t)$;
– $\mathbf{m}'_0(p') = \sum_{p \in P} \phi_{p'}[p] \cdot \mathbf{m}_0(p)$;
– $lab(p') = \sigma_{p'}$ and $lab(t') = \varsigma_{t'}$.

with $\langle \phi_{p'}, \sigma_{p'} \rangle$ and $\langle \psi_{t'}, \varsigma_{t'} \rangle$ the tuples that originated p' and t', respectively.

Figure 4 shows an example of a composed net, where π is

$$C_P = \left\{ \begin{array}{c} \langle P_0, \tau \rangle, \\ \langle P_1, \tau \rangle, \\ \langle P_2, e \rangle, \\ \langle P_3, 2\hat{e} \rangle, \\ \langle \{2 \cdot P_2 + P_3\}, \tau \rangle \end{array} \right\}, \quad C_T = \left\{ \begin{array}{c} \langle T_0, a + b + c \rangle, \\ \langle T_1, \hat{a} + \hat{b} + 2d \rangle, \\ \langle T_0 + T_1, c + 2d \rangle \end{array} \right\}$$

i.e. two new net elements are added, $2 \cdot P_2 + P_3$ and $T_0 + T_1$, and all the other net elements are preserved. In the figure, drawn with the GreatSPN tool, multisets of tags are represented as tags separated by bars, so for instance $\hat{a} + \hat{b} + 2d$ is depicted as $\widehat{a}|\widehat{b}|2d$. Observe that the new arcs connecting the new nodes have the sum of the multiplicities. For instance, $O(T_0{+}T_1, 2 \cdot P_2{+}P_3) = 3$ because it is $2 \cdot O(T_0, P_2) + O(T_1, P3)$.

Alternatively, we can view the combinatorial composition as a matrix operation over the net elements. Let $\mathbf{F}_\mathrm{P}$ be a $|P'| \times |P|$ matrix that encodes the places

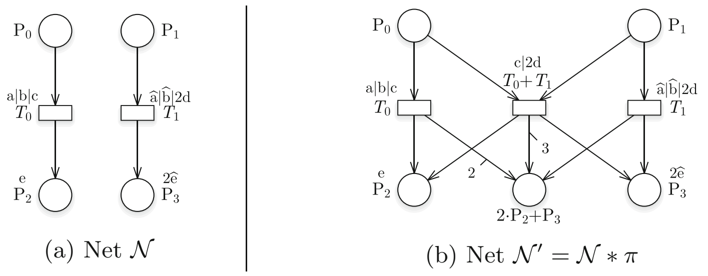


Fig. 4. Example of combinatorial composition for both places and transitions.

in C_P and $\mathbf{L}'_\mathrm{P}$ be a $|P'| \times |\Sigma_P|$ matrix that encodes the place labels, where conjugated labels are represented as negative values. Thus $[\mathbf{F}_\mathrm{P}|\mathbf{L}'_\mathrm{P}]$ encodes C_P in matrix form. Similarly, $\mathbf{F}_\mathrm{T} : |T'| \times |T|$ and $\mathbf{L}'_\mathrm{T} : |T'| \times |\Sigma_T|$ are used for C_T. For the example of Fig. 4, we have:

$$[\mathbf{F}_\mathrm{P}|\mathbf{L}'_\mathrm{P}] = \begin{array}{c} \begin{array}{ccccc} P_0 & P_1 & P_2 & P_3 & e \end{array} \\ \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -2 \\ 0 & 0 & 2 & 1 & 0 \end{array}\right] \end{array}, \qquad [\mathbf{F}_\mathrm{T}|\mathbf{L}'_\mathrm{T}] = \begin{array}{c} \begin{array}{cccccc} T_0 & T_1 & a & b & c & d \end{array} \\ \left[\begin{array}{cc|cccc} 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 & 0 & 2 \\ 1 & 1 & 0 & 0 & 1 & 2 \end{array}\right] \end{array}$$

Then, we can write the net composition in terms of matrix operations:

– The input matrix is $\mathbf{I}' = \mathbf{F}_\mathrm{P} \times \mathbf{I} \times \mathbf{F}_\mathrm{T}^T$;
– The output matrix is $\mathbf{O}' = \mathbf{F}_\mathrm{T} \times \mathbf{O} \times \mathbf{F}_\mathrm{P}^T$;
– The initial marking $\mathbf{m}'_0 = \mathbf{F}_\mathrm{P} \times \mathbf{m}_0$.

If we indicate with $\mathbf{Id}$ the identity matrix, whenever $[\mathbf{Id}|\mathbf{L}_\mathrm{P}]$ is a submatrix of $[\mathbf{F}_\mathrm{P}|\mathbf{L}'_\mathrm{P}]$ under some row permutation, the transformation is a *place extension*, since all places of $\mathcal{N}$ are preserved in $\mathcal{N}'$. A similar notion of *transition extension* can be defined on $[\mathbf{F}_\mathrm{T}|\mathbf{L}'_\mathrm{T}]$.

5 Composition Policies

We now define two *composition policies* to generate composition instances according to two paradigms inspired by PBC and of CSP.

5.1 Unary Conjugated Composition

We start by considering a composition policy for places and transitions which is based on the merging of conjugated tags, as in PBC transition synchronization. Let $A \subseteq \Sigma$ be the set of tags considered for the operation. The PBC transition synchronization is summarized by the following intuitive principle (adapted from [6, sec. 4.5]), by which, for every $a \in A$:

$$\begin{array}{c}\text{Repeatedly choose } a, \hat{a}\text{–pairs of labeled net elements, and} \\ \text{each time create a new composed net element from them.} \\ \text{Label this new element with the sum of the labels.}\end{array} \qquad (1)$$

The idea is to add to the original net new elements until all combinations that reduce the $a, \hat{a}$–pairs have been enumerated. This implies that (1) enumerates both the combinations that reach a τ-label, as well as all intermediate steps which may still have tags of Σ.

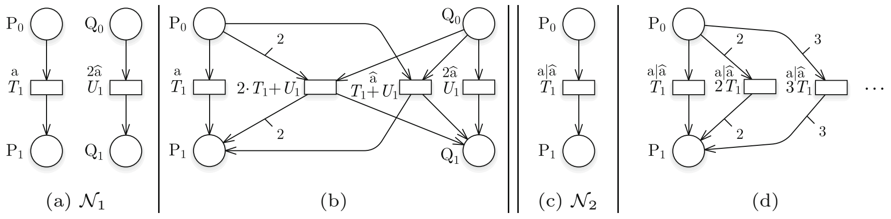


Fig. 5. Multitag conjugate composition and infinite PBC synchronization.

Figure 5(b) shows an example of the application of (1) to $\mathcal{N}_1$: T_1 and U_1 are retained, synchronization of T_1 and U_1 leads to transition $T_1 + U_1$, labelled with $\hat{a}$, which can be synchronized with T_1, leading to transition $2 \cdot T_1 + U_1$, labelled τ. There are no other composition of transitions that satisfies (1).

Note that this approach, which is the one employed by PBC for transition composition [6, Sec. 4.5], may repeatedly choose the same transition as a pair if it is labelled with both a tag a and its conjugate $\hat{a}$. Figure 5(d) shows an example of such PBC synchronization, applied to $\mathcal{N}_2$. In that case, transition T_1 is composed infinitely many times with itself, since each composition adds the labels $a + \hat{a}$ and then removes a single pair of $a, \hat{a}$ tags. Such scenario may happen because PBC labels may be multisets of tags that are not canonical. That's why we restrict our work to canonical labels.

The computation of all transitions pairs to be added can be non trivial, and Anisimov proposes in [2] an algorithm that is based on the computation of *minimal P/T-semiflows*. The Anisimov algorithm works on transitions with canonical labels, and it is based on the Farkas algorithm [12,15] for computing the *minimal P/T-semiflows*: therefore it is proved to always terminate. The intuition is that the goal of finding which combination of net elements reduces the sum of their labels to τ is equivalent to finding a linear combinations $\mathbf{f}$ of labels that cancel the tags in A, i.e. $\mathbf{f} \cdot \mathbf{L} = 0$. Vector $\mathbf{f}$ is, by Definition 2, a semiflow of $\mathbf{L}$. The Anisimov algorithm although, by considering only minimal semiflows, does not correspond exactly to (1) because:

– compositions that result in a transition with a label different from τ, like transition $T_1 + U_1$ in Fig. 5(b), are not generated;
– the algorithm does not either generates all composed transitions labelled with τ. Indeed the algorithm only considers minimal semi-flows, while these transitions corresponds to semi-flows that are non-minimal, although canonical (as explained later with reference to the example in Fig. 6).

Algorithm 1. Modified Farkas algorithm for unary conjugated composition.

```
 1: procedure ConjugateCompSet($\mathbf{L}, A$) // $\mathbf{L}$ is a $N \times M$ matrix
 2:     $[\mathbf{D}|\mathbf{A}] \leftarrow [\mathbf{Id}|\mathbf{L}]$
 3:     for $j$ between 1 and $M$ do
 4:         if $j$ corresponds to a column of $A$ then
 5:             for each $r_1 \neq r_2$ with $\mathbf{A}[r_1, j] > 0 \land \mathbf{A}[r_2, j] < 0$ do
 6:                 $[\mathbf{d}|\mathbf{a}] = [\mathbf{D}|\mathbf{A}][r_1, \cdot] + [\mathbf{D}|\mathbf{A}][r_2, \cdot]$
 7:                 $[\mathbf{d}|\mathbf{a}] \leftarrow [\mathbf{d}|\mathbf{a}] \;/\; gcd([\mathbf{d}|\mathbf{a}])$
 8:                 if $[\mathbf{d}|\mathbf{a}]$ does not appear in $[\mathbf{D}|\mathbf{A}]$ then
 9:                     $[\mathbf{D}|\mathbf{A}] \leftarrow$ AppendRows($[\mathbf{D}|\mathbf{A}], [\mathbf{d}|\mathbf{a}]$)
10:                 end if
11:             end for
12:         end if
13:     end for
14:     return $[\mathbf{D}|\mathbf{A}]$
15: end procedure
```

We therefore propose a modified Farkas algorithm for determining the instance of a composition policy that follows (1) and that overcomes the limitations of the Anisimov algorithm listed above. The pseudocode is shown in Algorithm 1. To compute the composition instance π, the unary conjugated composition policy extends both places and transitions simultaneously. To do so, the method is used twice, once for the places and once for the transitions, i.e.

$$[\mathbf{F}_{\mathrm{P}}|\mathbf{L}'_{\mathrm{P}}] \leftarrow \textsc{ConjugateCompSet}\mathbf{L}_{\mathrm{P}}, A$$
$$[\mathbf{F}_{\mathrm{T}}|\mathbf{L}'_{\mathrm{T}}] \leftarrow \textsc{ConjugateCompSet}\mathbf{L}_{\mathrm{T}}, A$$

Assume that we want to compute the composition of transitions. The method takes in input a $|T| \times |\Sigma_T|$ matrix $\mathbf{L}$, where $\mathbf{L}[t, s]$ is the multiplicity of tag s in label $lab(t)$, and Σ_T is the set of tags appearing on transitions. Conjugated tags appear as negative numbers in $\mathbf{L}$. The objective of Algorithm 1 is to find all linear combinations of labels that combine a–$\hat{a}$ pairs, until all τ combinations are generated. The loop at lines 3–13 considers one tag at a time. The tags in A are used to generate the combination rows. The inner loop 5–11 identifies all candidate combinations of net elements with a–$\hat{a}$ pairs in their labels. A combination is obtained by summing row r_1 with r_2, assuming that r_1 has tag a and r_2 has tag $\hat{a}$. The algorithm generates all such combinations until a fixed point is reached. Convergence is guaranteed since each new row $[\mathbf{d}|\mathbf{a}]$ generated at line 6 is such that $|\mathbf{a}[j]| < |\mathbf{A}[r_1, j]|$ and $|\mathbf{a}[j]| < |\mathbf{A}[r_2, j]|$, and the algorithm stops when all possible rows with $\mathbf{a} = 0$ are generated. However, Algorithm 1 may require an exponential number of steps to terminate.

While generating all the combinations that reduce the tag pairs, the algorithm may reach a point where all A–tags are zeroed for a row. In that case, the vector $\mathbf{d}$ for that row is a semiflow of the initial system. Unlike the Farkas algorithm, there is no check of *minimality* of such semiflows. Therefore all canonical semiflows are found. A second difference from the Farkas algorithm is the selection of the candidate vector at line 6. For semiflow computation, the vector $[\mathbf{d}|\mathbf{a}]$ would be computed as:

$$m_2 \cdot [\mathbf{D}|\mathbf{A}][r_1, \cdot] \; + \; m_1 \cdot [\mathbf{D}|\mathbf{A}][r_2, \cdot], \quad m_1 = |\mathbf{A}[r_1, j]|, \; m_2 = |\mathbf{A}[r_2, j]|$$

which would zero the value of $\mathbf{a}[j]$. By not multiplying by m_1 and m_2, all intermediate steps to reach the zero for $\mathbf{a}[j]$ are stored as rows in $[\mathbf{D}|\mathbf{A}]$. Each intermediate step can be seen as a new pair of $a, \hat{a}$ tags being cancelled from two groups of net elements, therefore implementing the principle (1).

Consider the net in Fig. 6(a) and the set $A = \{a, b\}$. The initial $[\mathbf{Id}|\mathbf{L}]$ matrix for transitions is shown in (2a). Row operations combine progressively row pairs, until a fixpoint is reached. All rows are kept.

$$\begin{array}{r|cccc|cc|}
 & T_1 & T_2 & T_3 & T_4 & a & b \\
row_1: & 1 & 0 & 0 & 0 & 2 & 0 \\
row_2: & 0 & 1 & 0 & 0 & 0 & -1 \\
row_3: & 0 & 0 & 1 & 0 & -1 & 1 \\
row_4: & 0 & 0 & 0 & 1 & -1 & 0
\end{array}$$

(2a) Matrix $[\mathbf{Id}|\mathbf{L}]$

$$\begin{array}{r|cccc|cc|l}
 & T_1 & T_2 & T_3 & T_4 & a & b & \\
row_5: & 1 & 0 & 1 & 0 & 1 & 1 & \\
row_6: & 1 & 0 & 0 & 1 & 1 & 0 & \\
row_7: & 1 & 0 & 2 & 0 & 0 & 2 & \\
row_8: & 1 & 0 & 1 & 1 & 0 & 1 & \\
row_9: & 1 & 0 & 0 & 2 & 0 & 0 & \text{minimal s.f.} \\
row_{10}: & 0 & 1 & 1 & 0 & -1 & 0 & \\
row_{11}: & 1 & 1 & 1 & 0 & 1 & 0 & \\
row_{12}: & 1 & 1 & 2 & 0 & 0 & 1 & \\
row_{13}: & 1 & 1 & 1 & 1 & 0 & 0 & \text{s.f.} \\
row_{14}: & 1 & 2 & 2 & 0 & 0 & 0 & \text{minimal s.f.}
\end{array} \qquad (2)$$

(2b) Rows appended to matrix $[\mathbf{D}|\mathbf{A}]$

The final matrix $[\mathbf{D}|\mathbf{A}]$ is made by all the initial rows of (2a) together with the rows in (2b), which contain the canonical semiflows (minimal and not), if they exists, and all the intermediate pairwise combinations.

The Anisimov algorithm generates the composition elements based exclusively on the minimal semiflows, i.e. Fig. 6(b).[1] The full unary conjugated composition policy, denoted as $\mathcal{N} *_C A$ with $A = \{a, b\}$, corresponds to generating the 14 transitions resulting from the rows of $\mathbf{D}$ in (2b). Each row $[\mathbf{d}|\mathbf{a}]$ results

[1] In (2b) row_{13} is not minimal because its support (all four transitions) contains the supports of both row_9 and row_{14}.

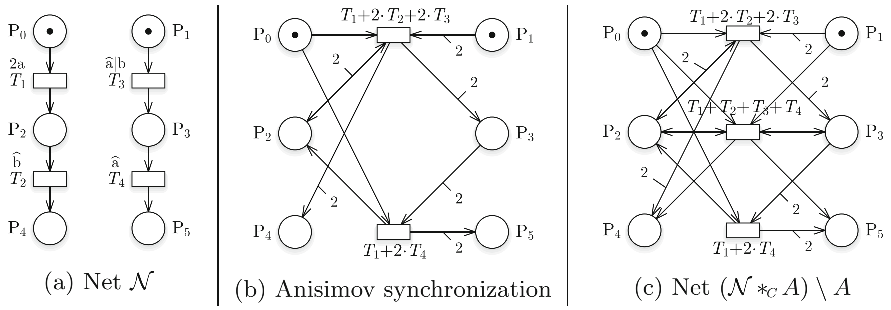


Fig. 6. Unary conjugated composition of $\mathcal{N}$, with $A = \{a, b\}$.

in a tuple $\langle \psi, \varsigma \rangle \in C_T$. Note that every initial transition is also preserved, since they appear as rows in $\mathbf{D}$. If we further restrict to A, for sake of readability, we obtain the net $(\mathcal{N} *_C A) \setminus A$ depicted in Fig. 6(c), where only the elements corresponding to semiflows are added.

The unary conjugated composition policy is a *place/transition extension*, since new net elements are added and no net element is removed.

5.2 N-Ary Structured Composition

The second composition policy that we consider is defined over a parallel composition of $n > 1$ nets $\mathcal{N} = (\mathcal{N}_1 \parallel \ldots \parallel \mathcal{N}_n)$. Again, a set of tags $A \subseteq \Sigma$ is defined to guide the policy. For each tag $a \in A$, new net elements result from composing one net element from every subnet $\mathcal{N}_1 \ldots \mathcal{N}_n$ that is labeled with a. For this policy there is no notion of conjugated tags. Moreover, the resulting multiplicity is 1 independently of the input tag multiplicities. When focusing on transitions only, this composition is similar to the parallel composition of CSP [19].

The sets C_P and C_T are computed independently. Consider the problem of identifying the places that will be composed together. For each tag $a \in A$, for every tuple of places $\langle p_1, \ldots p_n \rangle$ with $\forall\, i \geq n : \theta(p_i) = i \wedge a \in lab(p_i)$, then the tuple $\langle \phi, \sigma \rangle$ belongs to C_P, with:

- $\phi = p_1 + \ldots + p_n$, with all weights being one;
- $\sigma = \{a\} + \sum_{i=1}^{n} \big(lab(p_i) \setminus A\big)$.

An equivalent definition applies for the transitions.

Algorithm 2 shows the pseudo-code of the structured composition policy. To compute C_P and C_T, the algorithm is applied twice, i.e.

$$[\mathbf{F}_{\mathrm{P}} | \mathbf{L}'_{\mathrm{P}}] \leftarrow \textsc{StructuredCompSet}\mathbf{L}_{\mathrm{P}}, A, \theta$$
$$[\mathbf{F}_{\mathrm{T}} | \mathbf{L}'_{\mathrm{T}}] \leftarrow \textsc{StructuredCompSet}\mathbf{L}_{\mathrm{T}}, A, \theta$$

We define two variations of the structured composition:

Algorithm 2. N-Ary structured composition matrix.

```
1: procedure StructuredCompSet($\mathbf{L}, A, \theta$) // $\mathbf{L}$ is a $N \times M$ matrix
2:     $[\mathbf{F}|\mathbf{L}'] \leftarrow [\mathbf{Id}|\mathbf{L}]$
3:     for each tag $a \in A$ do
4:         for each tuple $\langle p_1, \ldots, p_n \rangle$ with $\theta(p_i) = i$ do
5:             $\phi = p_1 + \ldots + p_n$
6:             $\sigma \leftarrow \{a\} + \sum_{i=1}^{n} \big(lab(p_i) \setminus A\big)$
7:             $[\mathbf{F}|\mathbf{L}'] \leftarrow$ AppendRows($[\mathbf{F}|\mathbf{L}']$, vector form of $[\phi|\sigma]$)
8:         end for
9:     end for
10:    return $[\mathbf{F}|\mathbf{L}']$
11: end procedure
```

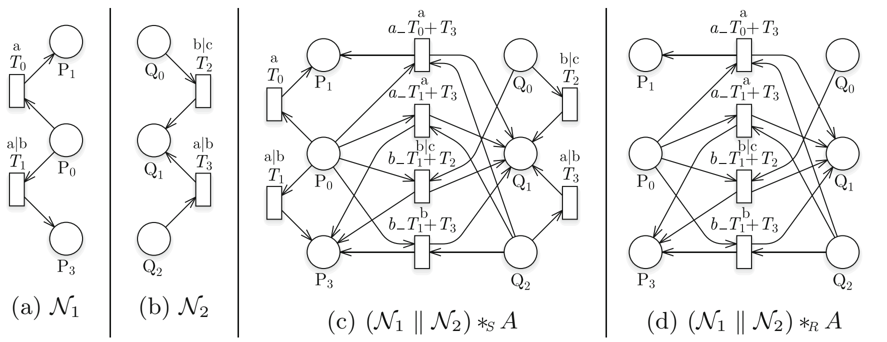


Fig. 7. Structured composition example, with $A = \{a, b\}$.

- *Structured extension:* $(\mathcal{N}_1 \parallel \ldots \parallel \mathcal{N}_n) *_S A$ extends $\mathcal{N}$ with the new elements. All existing net elements are kept.
- *Restricted structured composition:* $(\mathcal{N}_1 \parallel \ldots \parallel \mathcal{N}_n) *_R A$ first extends $\mathcal{N}$ with the new elements, and then removes all the elements of $(\mathcal{N}_1 \parallel \ldots \parallel \mathcal{N}_n)$ that were used to generate the new elements.

Figure 7 shows an example of both a structured and a restricted extensions of a parallel composition of two nets, with $A = \{a, b\}$. Only transitions are composed in this example. Four new transitions are added to C_T, corresponding to $\langle T_0 + T_3, a \rangle$, $\langle T_1 + T_3, a \rangle$, $\langle T_1 + T_2, b + c \rangle$, and $\langle T_1 + T_3, b \rangle$. Observe that $T_1 + T_3$ is composed twice, once for tag a and once for tag b. The *restricted* structured composition results in Fig. 7(d).

Notes on Operations. Tag rewriting and *restriction* could also be defined as composition policies.

6 Modelling Using the Composition Framework

We now focus on the usefulness of the framework for generating new models from existing ones.

6.1 Place-Based Composition

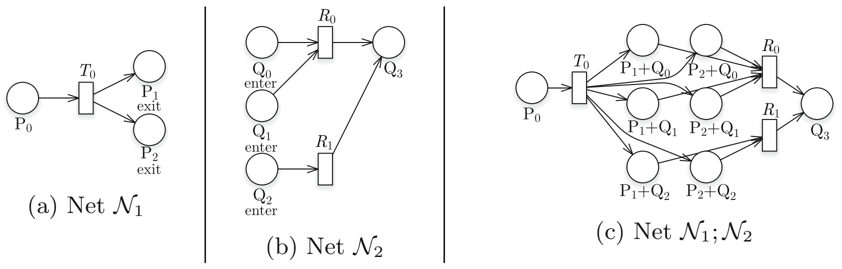


Fig. 8. Sequence as: $\big(\mathcal{N}_1 \parallel \lambda_{enter \to \widehat{exit}}(\mathcal{N}_2)\big) *_C A \setminus A$, with $A = \{exit\}$.

Place Sequence. Sequential composition is a typical composition pattern found in several algebras (CCS, CSP, PBC, and others). In the following we show how to use the framework to provide the sequence operator of PBC, that is based on the notion of entry and exit places. Assume a net $\mathcal{N}_1$ has some places labeled with a tag *exit* and a net $\mathcal{N}_2$ has some places labeled as *enter*. We can connect the exit places of $\mathcal{N}_1$ with the enter places of $\mathcal{N}_2$ by means of tag rewriting and combinatorial composition. Both the unary conjugated composition and the n-ary structured composition can be adopted. Figure 8 shows an example of sequential composition on places performed using unary conjugated composition, that leads to the formula: $\big(\mathcal{N}_1 \parallel \lambda_{enter \to \widehat{exit}}(\mathcal{N}_2)\big) *_C A \setminus A$, with $A = \{exit\}$. Tag rewriting is particularly useful in these situations, since it allows to identify pairs of conjugated tags that do not need to have the same name in the operand nets.

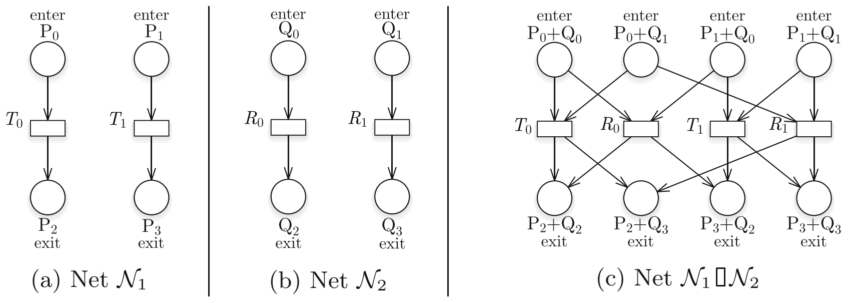


Fig. 9. Choice as $(\mathcal{N}_1 \parallel \mathcal{N}_2) *_R A$, with $A = \{enter, exit\}$.

Place Choice. Another common compositional pattern is *choice*. Again, our example consider the choice operator of PBC. In this setting, *enter* places of the two nets are multiplied together, the same for *exit* places, to split and then merge the control flows of the nets. Figure 9 shows an example of choice composition, taken from [6, Fig. 4]. The resulting net is obtained using restricted n-ary structured composition on the two tags $\{enter, exit\}$ leading to $(\mathcal{N}_1 \parallel \mathcal{N}_2) *_R A$, with $A = \{enter, exit\}$. Note that each place in the final net results from a composition of two places from each of the operand nets. A similar behaviour could also have been obtained using the unary conjugated composition, provided that tags are appropriately conjugated.

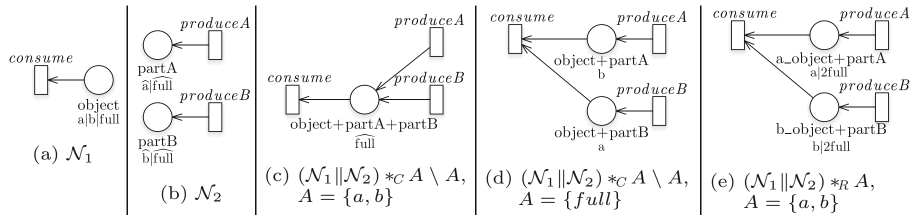


Fig. 10. Examples of place composition.

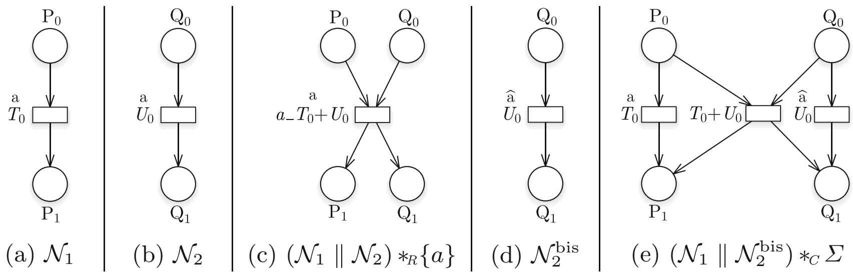
General Place Composition. Figure 10 shows different types of place-based composition for two nets $\mathcal{N}_1$ and $\mathcal{N}_2$, that induce a different interpretation of the multiset of tags associated to places. Indeed using different type of tags and different composition policies we can achieve rather diverse interpretations: this can be an advantage, but it requires a certain modelling expertise to appropriately master the composition process. Nets $\mathcal{N}_1$ and $\mathcal{N}_2$ in Fig. 10(a) and Fig. 10(b) can be interpreted as a very simple consumer and producer models. The net in Fig. 10(c) is obtained through unary conjugated composition, followed by restriction, on the set $A = \{a, b\}$. In formulae: $(\mathcal{N}_1 \| \mathcal{N}_2) *_C A \setminus A$. In this case the modelling objective was that the *object* place can contain elements coming from the places *partA* or *partB* and the composition ensures that any consumed object is actually consuming one part, either A or B. After composition, the individual identity (part A or part B) is lost. The tag $a + b$ of the place *object* of $\mathcal{N}_1$ can then be interpreted in a *or*-logic (either a or b).

If conjugated composition is performed instead on the *full* tag, which is included in the labels of the *partA* and *partB* places of $\mathcal{N}_2$, the identity of the two parts is kept in the composed net as two distinct places *object+partA* and *object+partB*, shown in Fig. 10(d). Consuming an object will now require consuming both a part A and a part B. In this way, the tag *full* interprets the composition using an *and*-logic (one token from every *full* place).

When place composition is realized through the n-ary structured composition policy, the result is similar to the *and*-logic. If composition is performed on tags $\{a, b\}$, each tag will result in an individual place, as in Fig. 10(e). Similarly, if composition is performed on $\{full\}$, this will again lead to two places. To have a 3-way composition resulting in a single place, the net $\mathcal{N}_2$ has to be separated further into two subnets, one for part A and one for part B, such that each subnet has only a single place tagged with *full*.

6.2 Transition-Based Composition

Composition of concurrent events as a composition of the transitions sharing the same tags (*synchronization*) is an important feature of any net algebra. We shall first consider how the proposed framework can express the CCS-like and CSP-like parallel operators, through which synchronization can be achieved, to then show how to express multi-way synchronization in two different forms.



P0
a
T_0
P1
(a) $\mathcal{N}_1$
Q0
a
U_0
Q1
(b) $\mathcal{N}_2$
P0
Q0
a
$a\_T_0{+}U_0$
P1
Q1
(c) $(\mathcal{N}_1 \parallel \mathcal{N}_2) *_R \{a\}$
Q0
$\hat{a}$
U_0
Q1
(d) $\mathcal{N}_2^{\text{bis}}$
P0
Q0
a
T_0
$T_0{+}U_0$
$\hat{a}$
U_0
P1
Q1
(e) $(\mathcal{N}_1 \parallel \mathcal{N}_2^{\text{bis}}) *_C \Sigma$

Fig. 11. CSP and CCS parallel operators.

CSP and CCS Parallel Operators. Nets $\mathcal{N}_1$ and $\mathcal{N}_2$ in Fig. 11 show two simple processes, let's say P and Q, that can both execute a. The net equivalent to the CSP process $P \parallel_{\{a\}} Q$ (parallel composition of P and Q with synchronization over action a) can be obtained as $(\mathcal{N}_1 \parallel \mathcal{N}_2) *_R \{a\}$, and it is depicted in Fig. 11(c).

If we now consider for process Q the net $\mathcal{N}_2^{\text{bis}}$ in Fig. 11(d), the net equivalent to the CCS process $P \parallel Q$ (parallel composition of P and Q over conjugate actions) can be obtained as $(\mathcal{N}_1 \parallel \mathcal{N}_2) *_C \{\Sigma\}$, and it is depicted in Fig. 11(e). Note that the composition is over the whole set Σ of actions (tags) as in CCS (in its original form): there is no way to limit the set of actions on which synchronization takes place and the resulting net can correctly execute independently also action a and $\hat{a}$, while the synchronized action is labelled τ. Moreover the two actions a and $\hat{a}$ are still executable.

Multi-way Synchronization requires a different approach, depending on whether we have a single common tag or conjugate tags. In the former case we can use structured composition, while in the latter one we need to use multiple tags.

Figure 12 shows how to realize a three-way synchronization with a single common tag. The three operand nets $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_3$ all have a transition with tag a. Restricted structured composition merges these transitions into a single transition, that is also connected to all input and output places.

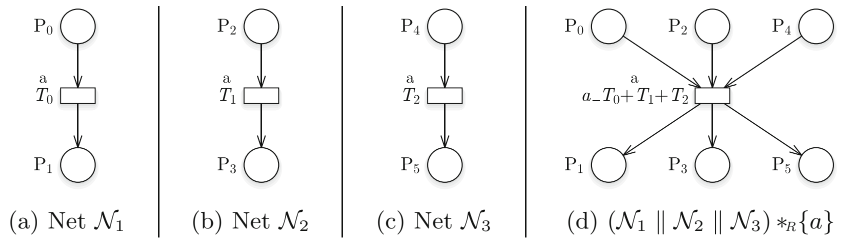


Fig. 12. Three-way synchronization using structured composition.

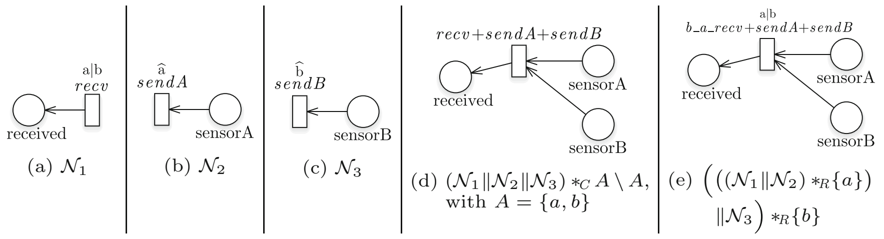
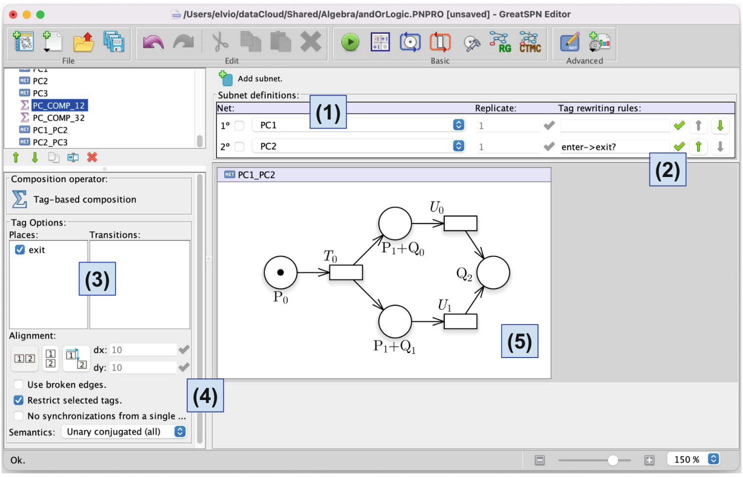


Fig. 13. Three-way synchronization using conjugated tags.

Figure 13 shows a three-way synchronization realized using conjugate tags. Nets $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ depicts the component of a system in which a token is placed into place *received* only when the two sensors' values are read in a single moment. The synchronization can be achieved through unary conjugated composition (merging all tags), as in Fig. 13(d), or by subsequently merging one tag after the other as multiple nested restricted structured composition, as in Fig. 13(e).

Multitag Synchronization. The other important aspect that is covered by the composition over multitags is that all transitions that fully complement the synchronized tags are generated (if possible). Figure 6(c) is an example of this behaviour. In this way, complex dependencies among the tags can be expressed. An example of application is a transition that needs to acquire n resources of type a, that are provided and locked by another transition with a $\hat{a}$ transition. The resulting synchronization consists in a single acquisition and n lock events into a single transition. When complementarity of the tags is implicit in the structure of the net, structured composition is also an option, as in Fig. 7(c).

Prototype. The proposed framework has been implemented as a prototype inside the GreatSPN software (https://github.com/greatspn/SOURCES). All figures used in the paper were generated using the proposed composition framework, with the exception of Fig. 5(d, e), that was generated manually. The framework is developed inside the graphical editor, where multiple nets can be composed together using the unary conjugated composition, or the n-ary structured composition. Restriction is optional and can be applied after each composition.



Subnet definitions:

Net:		Replicate:	Tag rewriting rules:
1°	PC1	1	
2°	PC2	1	enter->exit?

Composition operator: Tag-based composition

Tag Options:
Places: [✓ exit]
Transitions:

Alignment: dx: 10 dy: 10

Use broken edges.
[✓ Restrict selected tags.]
No synchronizations from a single ...
Semantics: Unary conjugated (all)

Fig. 14. Prototype implemented inside the GreatSPN graphical interface.

Figure 14 shows how the composition interface looks like in the tool. Composition pages are special subnets, and are defined following this schema. In (1) the composed nets are selected, and optional tag rewriting rules are specified (2). The composition tags for places and transitions are then selected (3), together with the composition policy, the optional tag restriction and other parameters (4). The composed net is then shown in the central pane (5).

7 Conclusions

In this paper we propose a novel framework for net composition that is focused on the simultaneous joint combination of places and transitions into a single policy-based combinatorial operator. Multiple policies can be defined for composition, and we provided a CCS-like unary conjugated composition, as well as a CSP-like n-ary structured composition. Conjugated composition is based on the intuitions of [2], but it is modified to follow the synchronization rules of the Petri box calculus. We have reviewed several common cases for Petri net composition and various modeling patterns, showing the effectiveness of the proposed approach in modeling terms. The operators have been defined in net syntactical terms. While this definition is enough to proceed, for example, to an implementation, it lacks a formal semantic interpretation of the composed net behaviour in terms of the possible executions of the composed nets. While for transition composition defined by the policies of Sect. 5.1 and 5.2 this may be an attainable goal, it is less clear how this can be achieved when the policies are applied to place-based composition, a topic that has received less attention in the literature and that certainly deserves more investigation.

References

1. Amparore, E.G., Donatelli, S.: GreatTeach: a tool for teaching (stochastic) petri nets. In: Khomenko, V., Roux, O.H. (eds.) PETRI NETS 2018. LNCS, vol. 10877, pp. 416–425. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91268-4_24
2. Anisimov, N.A., Golenkov, E.A., Kharitonov, D.I.: Compositional petri net approach to the development of concurrent and distributed systems. Program. Comput. Softw. **27**, 309–319 (2001). https://doi.org/10.1023/A:1012758417962
3. Arnold, A.: Nivat's processes and their synchronization. Theoret. Comput. Sci. **281**(1–2), 31–36 (2002)
4. Ballarini, P., Donatelli, S., Franceschinis, G.: Parametric stochastic well-formed nets and compositional modelling. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 43–62. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44988-4_5
5. Bernardi, S., Donatelli, S., Horvath, A.: Implementing compositionality for stochastic petri nets. Int. J. Softw. Tools Technol. Transf. **3**, 417–430 (2001)
6. Best, E., Devillers, R., Hall, J.G.: The box calculus: a new causal algebra with multi-label communication. In: Rozenberg, G. (ed.) Advances in Petri Nets 1992. LNCS, vol. 609, pp. 21–69. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55610-9_167
7. Best, E., Devillers, R., Koutny, M.: Petri Net Algebra. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-662-04457-5
8. Best, E., Fleischhack, H., Fraczak, W., Hopkins, R.P., Klaudel, H., Pelz, E.: A class of composable high level Petri nets. In: De Michelis, G., Diaz, M. (eds.) ICATPN 1995. LNCS, vol. 935, pp. 103–120. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60029-9_36
9. Buchholz, P.: Hierarchies in colored GSPNs. In: Ajmone Marsan, M. (ed.) ICATPN 1993. LNCS, vol. 691, pp. 106–125. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-56863-8_43
10. Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: On well-formed coloured nets and their symbolic reachability graph. In: Jensen, K., Rozenberg, G. (eds.) High-level Petri Nets, pp. 373–396. Springer, Heidelberg (1991). https://doi.org/10.1007/978-3-642-84524-6_13
11. Clark, G., et al.: The Möbius modeling tool. In: 9th International Workshop on Petri Nets and Performance Models, pp. 241–250 (2001)
12. Colom, J.M., Silva, M.: Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal p-semiflows. In: Rozenberg, G. (ed.) ICATPN 1989. LNCS, vol. 483, pp. 79–112. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-53863-1_22
13. De Cindio, F., et al.: A Petri net model for CSP. In: Proceedings of Convención Informática Latina (CIL 1981), Barcelona, vol. 81, pp. 392–406 (1981)
14. Donatelli, S., Franceschinis, G.: The PSR methodology: integrating hardware and software models. In: Billington, J., Reisig, W. (eds.) ICATPN 1996. LNCS, vol. 1091, pp. 133–152. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61363-3_8
15. Farkas, J.: Theorie der einfachen Ungleichungen. Journal für die reine und angewandte Mathematik (Crelles Journal) **1902**(124), 1–27 (1902)
16. Heiner, M.: How to draw a hierarchical Petri net? https://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/FAQ#QQuestions5. Accessed Jan 2022

17. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy – a unifying petri net tool. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 398–407. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31131-4_22
18. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Hoboken (1985)
19. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8), 666–677 (1978)
20. Jensen, K.: Coloured Petri nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) Petri Nets: Central Models and Their Properties. LNCS, vol. 254, pp. 248–299. Springer, Heidelberg (1987). https://doi.org/10.1007/BFb0046842
21. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. Int. J. Software Tools Tech. Transf. **9**(3), 213–254 (2007)
22. Kindler, E.: Modular PNML revisited: some ideas for strict typing. Arbeitsberichte aus dem Arbeitsberichte aus dem Fachbereich Informatik 20 (2007)
23. Kotov, V.E.: An algebra for parallelism based on petri nets. In: Winkowski, J. (ed.) MFCS 1978. LNCS, vol. 64, pp. 39–55. Springer, Heidelberg (1978). https://doi.org/10.1007/3-540-08921-7_55
24. Lampka, K., Siegle, M.: Symbolic composition within the Möbius framework. In: Proceedings of the 2nd MMB Workshop, pp. 63–74 (2002)
25. Mieg, Y.T.: From Symbolic Verification To Domain Specific Languages. Ph.D. thesis. Sorbonne Université, UPMC; Laboratoire d'informatique de Paris 6 [LIP6] (2016)
26. Milner, R. (ed.): A Calculus of Communicating Systems. LNCS, vol. 92. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10235-3
27. Ratzer, A.V., et al.: CPN tools for editing, simulating, and analysing coloured petri nets. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 450–462. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44919-1_28
28. Thierry-Mieg, Y.: Symbolic model-checking using ITS-tools. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 231–237. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_20
29. Weber, M., Kindler, E.: The petri net markup language. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 124–144. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40022-6_7