Nested-unit Petri nets

Hubert Garavel

Nested-Unit Petri Nets

Hubert Garavel[a,b]

[a] *Univ. Grenoble Alpes,* Inria, Cnrs, *Grenoble* Inp, Lig, *38000 Grenoble, France*
[b] *Saarland University, Saarbrücken, Germany*

Abstract

Petri nets can express concurrency and nondeterminism but neither locality nor hierarchy. This article presents an extension of Petri nets, in which places can be grouped into so-called "units" expressing sequential components. Units can be recursively nested to reflect both the concurrent and hierarchical nature of complex systems. This model called NUPN (Nested-Unit Petri Nets) was originally developed for translating process calculi to Petri nets, but later found also useful beyond this setting. It allows significant savings in the memory representation of markings for both explicit-state and symbolic verification. Thirteen software tools already implement the NUPN model, which has also been adopted for the benchmarks of the Model Checking Contest (MCC) and the parallel problems of the Rigorous Examination of Reactive Systems (RERS) challenges.

Keywords: CADP, concurrency theory, decision diagram, formal method, formal semantics, model checking, Petri net, process algebra, process calculus, verification

1. Introduction

In a 1987 article [1], Rob van Glabbeek and Frits Vaandrager highlighted the two most heavily debated equations in concurrency theory.

- The first of these two equations is: for all atomic actions a, b, and c, do we have "$a \cdot (b + c) \stackrel{?}{=} a \cdot b + a \cdot c$", where "$\cdot$" denotes sequential composition[1] and "+" denotes nondeterministic choice. If the answer is yes, one stays in the framework of *linear-time semantics* or, if the answer is no, in the framework of *branching-time semantics*.

- The second of these two equations is: for all atomic actions a and b, do we have "$a \,||\, b \stackrel{?}{=} a \cdot b + b \cdot a$", where "||" denotes parallel composition. If the answer is yes, one stays in the framework of *interleaving semantics* or,

Email address: hubert.garavel@inria.fr (Hubert Garavel)
URL: http://convecs.inria.fr/people/Hubert.Garavel (Hubert Garavel)

[1] As usual, operator "$\cdot$" has higher precedence than all the other binary operators.

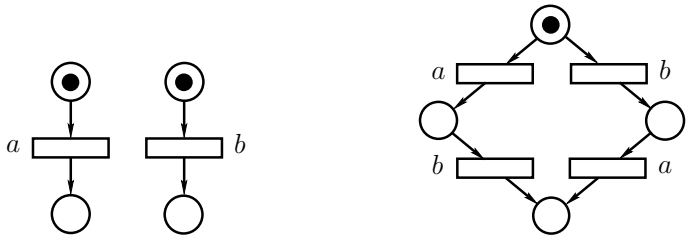

Figure 1: Two Petri nets that implement "$a \,||\, b$" (left) and "$a \cdot b + b \cdot a$" (right).

if the answer is no, in the framework of *true concurrency*. For instance, mainstream process calculi (such as CCS [2], ACP [3], CSP [4], and LOTOS [5]) rely on the interleaving semantics, while Petri nets — as well as Mazurkiewicz traces [6] and Winskel event structures [7] — can distinguish between parallel composition (see Fig. 1, left) and its approximation expressed in terms of sequential composition and nondeterministic choice (see Fig. 1, right); in general, a Petri net implementing parallel composition possesses as many tokens as there are operands composed together in parallel, whereas a Petri net implementing sequential composition and nondeterministic choice is one-safe, i.e., possesses a single token.

Now, let us consider a third (much less famous) equation, which can be traced back (at least) to [8, page 169]: for all atomic actions a, b, and c, do we have "$a \cdot b \,||_b\, b \cdot c \stackrel{?}{=} a \cdot b \cdot c \,||_b\, b$", where "$||_b$" denotes the parallel composition operator that only synchronizes atomic actions b, forcing them to execute simultaneously, and lets all other atomic actions interleave freely. From the point of view of mainstream process calculi, the answer is yes, as both sides of the equation are strongly equivalent to the term $a \cdot b \cdot c$.[2] From the point of view of Petri nets, the answer is also yes, since the two nets shown in Fig. 2 are identical, which can be seen by simply permuting the two arcs going out of transition b. In this example, neither process calculi nor Petri nets can express the information that the atomic action c takes place in a specific parallel operand, as this action can arbitrarily occur in the left or right parallel operand without changing the semantics of the corresponding term or net.

This issue is further illustrated by a fourth equation: for all atomic actions a, a', b, c, and c', do we have "$a \cdot b \cdot c \,||_b\, a' \cdot b \cdot c' \stackrel{?}{=} a' \cdot b \cdot c \,||_b\, a \cdot b \cdot c'$". With process calculi and Petri net, the answer is yes, meaning that the atomic actions a and a' (respectively, c and c', taking into account that "$||_b$" is commutative) can be freely permuted over the parallel composition operator without changing the overall semantics. In practice, however, when formally specifying

[2] Actually, $a \cdot \tau \cdot c \cdot NIL$ in the case of CCS, since mandatory synchronization on action b must be ensured by means of the restriction operator, which renames b into the silent action τ.

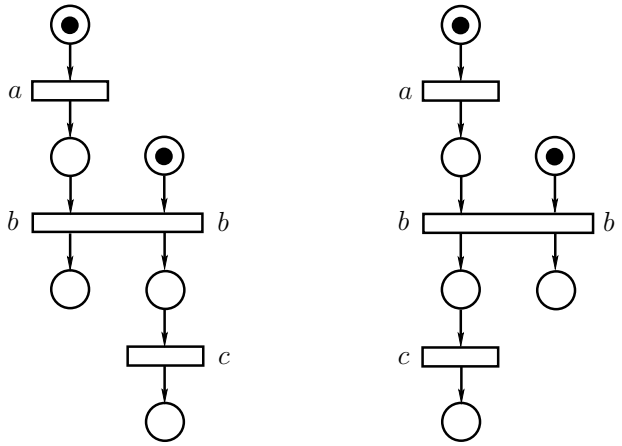

Figure 2: Two Petri nets that implement "$a \cdot b \,||_b\, b \cdot c$" (left) and "$a \cdot b \cdot c \,||_b\, b$" (right).

real-life communication protocols or distributed systems, it is not indifferent to know in which parallel component a given action occurs, e.g., on the server side or on the client side. Said otherwise, mainstream process calculi and Petri nets consider as *behaviourally* equivalent certain terms or nets that are *architecturally* different; although these formalisms capture many useful aspects, including concurrency and nondeterminism, they cannot express the concept of *locality*.

To address this issue in the setting of process calculi, proposals have been made to extend CCS with various notions of locality [9, 10, 11, 12, 8, 13] [14] [15] (see [16] for a survey). The approach proposed in the present article also addresses the locality issue, but in the context of Petri nets. Additionally, this approach remedies another shortcoming of Petri nets, by providing means to formally describe the hierarchical structure of complex systems, a modelling feature that exists in process calculi and that traditional (i.e., low-level) Petri nets are lacking.

In a nutshell, the present article proposes to extend Petri nets with hierarchical structuring capabilities inspired from process calculi. Technically, this is done by grouping together all the places that are local to the same sequential component; these groups of places are called *units*. When equipped with such extra information, Petri nets become able to distinguish between nets that are architecturally different but behaviourally equivalent, e.g., for the third equation above, the two nets shown in Fig. 3. Such nets directly reflect the hierarchical structure of the corresponding algebraic terms, as each operand of a parallel composition operator is mapped to a separate unit. Because, in most process calculi, the parallel composition operators can be freely combined, units can be recursively nested at an arbitrary depth, as shown in Fig. 4, to describe components containing sub-components that execute concurrently.

Our proposal is rooted in a longstanding effort to develop the comprehensive

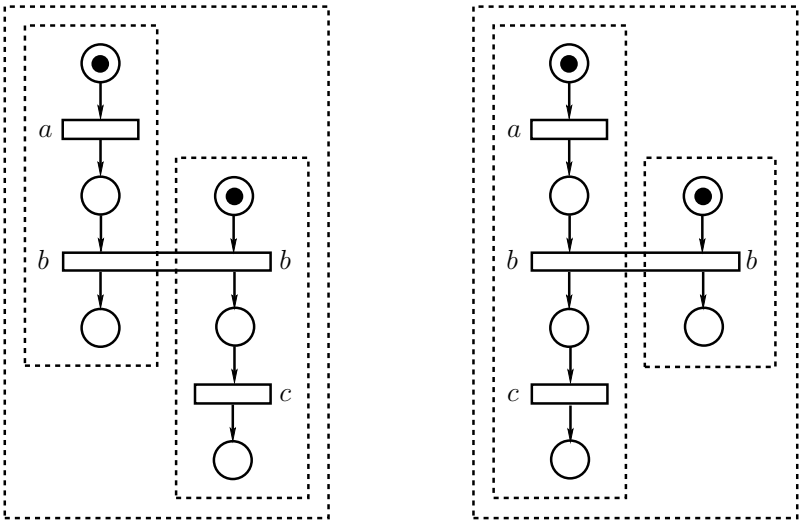

Figure 3: Units that help distinguishing "$a \cdot b \,||_b\, b \cdot c$" (left) from "$a \cdot b \cdot c \,||_b\, b$" (right).

CADP toolbox [17] for the design and verification of concurrent systems. The toolbox includes CÆSAR [18, 19, 20], an efficient compiler for the value-passing process calculus LOTOS standardized by ISO [5]. This compiler translates LOTOS to labelled transition systems using, as an intermediate step, interpreted Petri nets extended with units that reflect the hierarchical/concurrent structure of the source LOTOS specifications. Actually, the suggestion that the Petri nets generated by CÆSAR could retain structural information from the LOTOS source was formulated in 1988 by Eric Madelaine during a meeting; following this remark, the concept of nested units described in this article was progressively identified and refined as the most useful kind of information to be preserved.

For thirty years, this concept has been in use, but internally to the CADP toolbox only. Specifically, CÆSAR uses two different types of hierarchically structured nets: an interpreted Petri net, which comprises variables, expressions, assignments, guards, etc. (Fig. 5 shows an example of such a net, taken from [19]) and an elementary net, which is a data-less abstraction of the former by removing all value-passing information. The present article is about this latter model, which was initially named BPN (*Basic Petri Net*), but, since this acronym has been heavily overloaded[3], was renamed to NUPN[4] (*Nested-Unit Petri Net*) in 2013, when it found a new application field in the framework of

[3]BPN is used elsewhere as an acronym for *Backward Petri Net*, *Basic Petri Net* (as opposed to Colored Petri Net), *Batch Petri Net*, *Behavioural Petri Net*, *Biochemical Petri Net*, *Bounded Petri Net*, *Business Process Net*, B(PN)^2, etc.

[4]To be pronounced: "*new PN*".

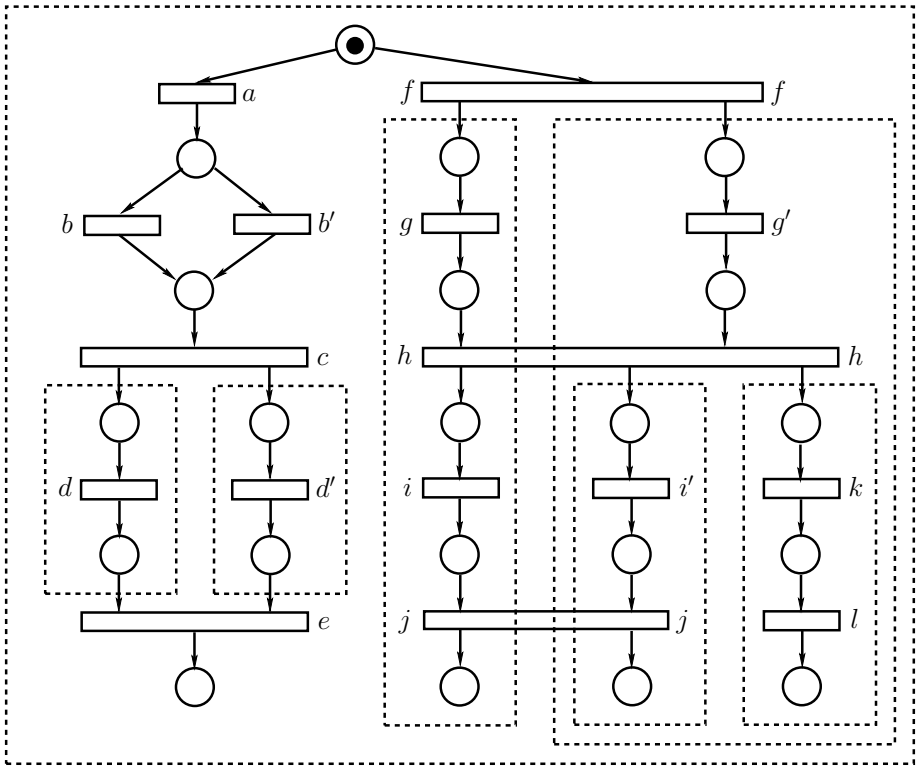

Figure 4: A Petri net with units that implements "$a \cdot (b + b') \cdot c \cdot (d \parallel d') \cdot e + (f \cdot g \cdot h \cdot i \cdot j \parallel_{f,h,j} f \cdot g' \cdot h \cdot (i' \cdot j \parallel k \cdot l))$".

the Model Checking Contest[5] [21, 22].

The present article is organized as follows. Section 2 defines the NUPN model in both its structural and behavioural aspects. Section 3 introduces the unit-safeness property for NUPNs and gives associated results. Section 4 discusses the expressiveness of the NUPN model with respect to P/T nets and communicating automata. Section 5 introduces a useful abstraction that naturally fits with the NUPN model. Section 6 indicates how the representation of markings can be optimized for NUPNs in both explicit-state and symbolic verification settings. Section 7 provides an overview of implementation efforts to equip the NUPN model with file formats, software tools, and collections of benchmarks. Section 8 does an extensive review of the state of the art to position the NUPN model with respect to related work. Finally, Section 9 gives concluding remarks and draws open perspectives for future work.

[5] See `http://mcc.lip6.fr`

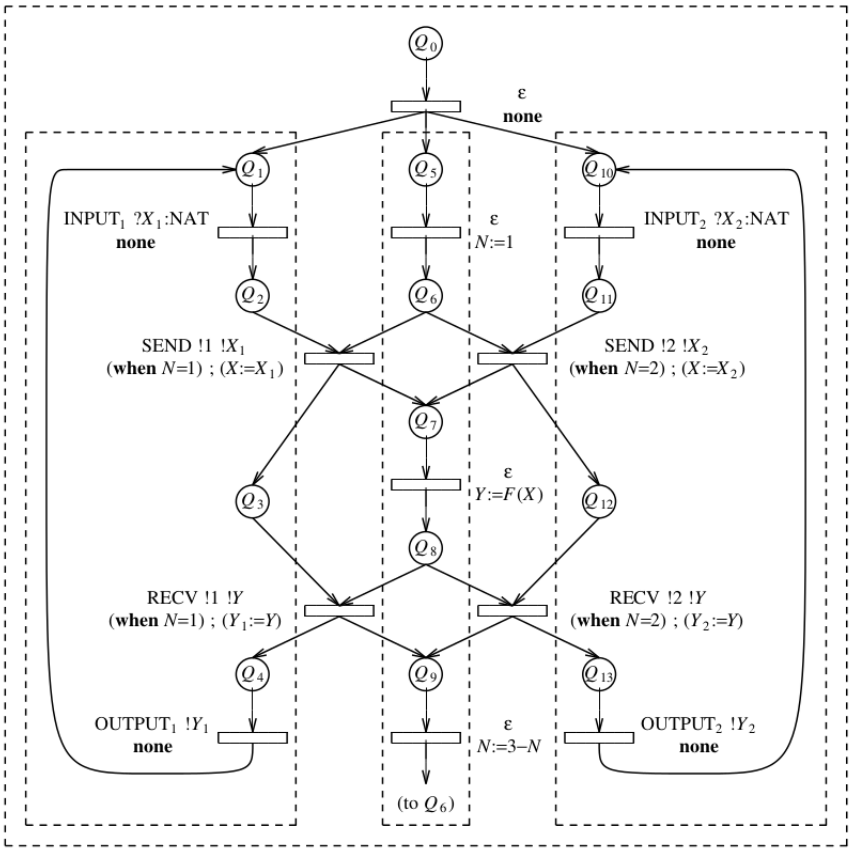

Figure 5: An Interpreted Petri net with units generated by the LOTOS compiler of CADP.

2. Nested-Unit Petri Nets

In the present article, we mostly consider Petri nets that are *ordinary* (i.e., all arc weights are equal to one) and *safe* (i.e., all place capacities are equal to one, meaning that each place can contain at most one token). According to a thorough survey [23, Sect. 1–4], such nets are called *net models of level one* (namely, nets in which places are marked by at most one unstructured token) and are covered by three main types of net models published in the scientific literature: *condition/event systems* (C/E, for short), *elementary nets*, and *one-safe P/T nets*. The survey gives a clear comparison of the differences between these three models. For the present article, condition/event systems are unsuitable, since we are interested in forward reachability rather than backward-and-forward reachability (i.e., our Petri-net transitions never fire in the reverse direction); we also prefer using the terms *places*, *transitions*, *markings*, etc. rather than their counterparts *conditions*, *events*, *cases* and *constellations*, etc.

in C/E terminology. Instead, we base our definitions on a combination of elementary nets (from which we retain the idea that markings are sets of places) and one-safe P/T nets (from which we drop all integer numbers, since all arc weights and all place capacities are equal to one, and from which we reuse the firing rules that allow self-loop transitions[6] to be fireable, whereas in elementary nets such transitions are either forbidden [24, Def. 1, 4th item] or always dead [23, Sect. 2.2]). Notice that firing rules similar to those adopted in the present article have also made their way in the C/E setting, where they are known as *augmented condition/event systems* [25].

2.1. Structure

This subsection formally defines the "structural" aspects of the NUPN model, i.e., the syntax and static semantics of this model.

Definition 1. *A (marked)* Nested-Unit Petri Net *(acronym: NUPN) is a 8-tuple* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *where:*

1. P *is a finite, non-empty set; the elements of* P *are called* places.
2. T *is a finite set such that* $P \cap T = \varnothing$*; the elements of* T *are called* transitions.
3. F *is a subset of* $(P \times T) \cup (T \times P)$*; the elements of* F *are called* arcs.
4. M_0 *is a subset of* P*;* M_0 *is called the* initial marking.
5. U *is a finite, non-empty set such that* $U \cap T = U \cap P = \varnothing$*; the elements of* U *are called* units.
6. u_0 *is an element of* U*;* u_0 *is called the* root unit.
7. $\sqsubseteq$ *is a binary relation over* U *such that* $(U, \sqsupseteq)$ *is a tree with a single root* u_0*, where* $(\forall u_1, u_2 \in U)\ u_1 \sqsupseteq u_2 \stackrel{\text{def}}{=} u_2 \sqsubseteq u_1$*; thus,* $\sqsubseteq$ *is reflexive, antisymmetric, transitive, and* u_0 *is the greatest element of* U *for this relation*[7]*; intuitively,* $u_1 \sqsubseteq u_2$ *expresses that unit* u_1 *is transitively nested in or equal to unit* u_2*.*
8. unit *is a function* $P \to U$ *such that* $(\forall u \in U \setminus \{u_0\})\ (\exists p \in P)\ \mathsf{unit}(p) = u$*; intuitively,* $\mathsf{unit}(p) = u$ *expresses that unit* u *directly contains place* p*.*

Notice that, despite the fact that NUPNs have been originally designed for process calculi, no particular assumption is made about place or transition labelling. The next definition provides useful notations derived from Def. 1.

Definition 2. *Let* $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN, and let* u*,* u_1*, and* u_2 *be any three units of* U*:*

- $u_1 \sqsubset u_2 \stackrel{\text{def}}{=} (u_1 \sqsubseteq u_2) \wedge (u_1 \neq u_2)$ *is the strict nesting partial order.*

[6] These are called *events with side conditions* in C/E terminology and *impure transitions* in elementary nets.

[7] The unit tree is defined as $(U, \sqsupseteq)$ rather than $(U, \sqsubseteq)$ so that the root u_0 of the tree (i.e., the minimal element for $\sqsupseteq$) is the maximal element for $\sqsubseteq$, i.e., the unit that contains all the other units.

- $\mathsf{disjoint}\,(u_1, u_2) \stackrel{\text{def}}{=} (u_1 \not\sqsubseteq u_2) \wedge (u_2 \not\sqsubseteq u_1)$ *characterizes pairs of units neither equal nor nested one in the other.*

- $\mathsf{subunits}^*(u) \stackrel{\text{def}}{=} \{u' \in U \mid (u' \sqsubset u)\}$ *gives all units transitively nested in* u*.*

- $\mathsf{subunits}\,(u) \stackrel{\text{def}}{=} \{u' \in U \mid (u' \sqsubset u) \wedge (\nexists u'' \in U)\ (u' \sqsubset u'') \wedge (u'' \sqsubset u)\}$ *gives all units directly nested in* u*.*

- $\mathsf{leaf}\,(u) \stackrel{\text{def}}{=} (\mathsf{subunits}\,(u) = \varnothing)$ *characterizes the units having no nested sub-unit, i.e., the minimal elements of* $(U, \sqsubseteq)$*.*

- $\mathsf{redundant}\,(u) \stackrel{\text{def}}{=} (\mathit{card}(\mathsf{subunits}\,(u)) = 1)$ *characterizes the units having exactly one directly nested sub-unit.*

- $\mathsf{places}\,(u) \stackrel{\text{def}}{=} \{p \in P \mid \mathsf{unit}\,(p) = u\}$ *gives all places directly contained in* u*; these are called the* local places *(or* proper places*) of* u*.*

- $\mathsf{places}^*(u) \stackrel{\text{def}}{=} \{p \in P \mid (\exists u' \in U)\ (u' \sqsubseteq u) \wedge (\mathsf{unit}\,(p) = u')\}$ *gives all places transitively contained in* u *or its sub-units.*

- $\mathsf{void}\,(u) \stackrel{\text{def}}{=} (\mathsf{places}\,(u) = \varnothing)$ *characterizes the units having no local place.*

- $\widetilde{U} \stackrel{\text{def}}{=} \{u \in U \mid \neg\mathsf{void}\,(u)\}$ *is the set of all units having local places.*

- $\mathsf{height}\,(N) \stackrel{\text{def}}{=} h(u_0)$*, where the auxiliary function* $h : U \to \mathbb{N}$ *is such that* $h(u) \stackrel{\text{def}}{=} 1$ *if* $\mathsf{leaf}\,(u)$*,* $h(u) \stackrel{\text{def}}{=} \max(\{h(u') \mid u' \in \mathsf{subunits}\,(u)\})$ *if* $\mathsf{void}\,(u)$*, or* $h(u) \stackrel{\text{def}}{=} 1 + \max(\{h(u') \mid u' \in \mathsf{subunits}\,(u)\})$ *otherwise;* $\mathsf{height}\,(N)$ *is akin to the height of the unit tree* $(U, \sqsupseteq)$*, with the difference that only non-void units are counted.*

- $\mathsf{width}\,(N) \stackrel{\text{def}}{=} \mathit{card}\,(\{u \in U \mid \mathsf{leaf}\,(u)\})$ *is akin to the width of the unit tree* $(U, \sqsupseteq)$*, with the difference that all leaf units are counted.*

- $\mathsf{trivial}\,(N) \stackrel{\text{def}}{=} (\mathsf{width}\,(N) = \mathit{card}\,(P))$ *characterizes the nets having as many leaf units as places.*

Notice that, in set theory, the width of the unit tree $(U, \sqsupseteq)$ is defined as $\max(\{\mathrm{card}\,(U^d_n) \mid n \in \mathbb{N}\})$, where $U^d_n = \{u \in U \mid d(u) = n\}$ and $d(u)$ is the depth of unit u. In our definition of $\mathsf{width}\,(N)$, $d(u)$ is replaced by $h(u)$, i.e., the height of unit u, and U^d_n is replaced by $U^h_n = \{u \in U \mid h(u) = n\}$, so that $\mathsf{width}\,(N) = \max(\{\mathrm{card}\,(U^h_n) \mid n \in \mathbb{N}\})$ possibly returns larger values than the tree width of $(U, \sqsupseteq)$. For instance, if $U = \{u_0, u_1, u_2, u_3, u_4\}$ with $u_1 \sqsubseteq u_0$, $u_2 \sqsubseteq u_0$, $u_3 \sqsubseteq u_2$, and $u_4 \sqsubseteq u_2$, then $\mathsf{width}\,(N) = 3$ (as there are three leaf units, each at height one, i.e., $U^h_1 = \{u_1, u_3, u_4\}$), whereas the tree width of $(U, \sqsupseteq)$ is equal to 2 (as there are two units at depth one, and also two at depth two, i.e., $U^d_1 = \{u_1, u_2\}$ and $U^d_2 = \{u_3, u_4\}$).

NUPNs present similarities with widespread data structures of computer science. There is an analogy between $\langle$ places, units $\rangle$ in NUPNs and $\langle$ files, directories $\rangle$ in file systems: a unit may contain places and/or sub-units in the same way a directory may contains files and/or sub-directories. There is another analogy between $\langle$ places, units $\rangle$ in NUPNs and $\langle$ attributes, elements $\rangle$ in XML: an element may have attributes and/or contain sub-elements; yet, these sub-elements are ordered and these attributes are not, whereas neither the sub-units nor the local places of each unit are ordered. Moreover, the following proposition establishes that NUPNs have particular properties that neither filesystems nor XML documents have.

Proposition 1. *Let* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN.*

1. *The number of units is such that* $1 \leq card(U) \leq card(P) + 1$.
2. *The number of non-void units is such that* $1 \leq card(\widetilde{U}) \leq card(P)$.
3. *There is at most one void unit (namely: the root unit), i.e.,* $U - \widetilde{U} \subseteq \{u_0\}$.
4. *The root unit, if void, has at least one sub-unit, i.e.,* $\mathsf{void}(u_0) \Rightarrow \neg\mathsf{leaf}(u_0)$.
5. *A leaf unit is never void, i.e.,* $(\forall u \in U)\ \mathsf{void}(u) \Rightarrow \neg\mathsf{leaf}(u)$.

Proof. Item 1: the left inequality directly follows from item 5 of Def. 1, which states that U is not empty; the right inequality follows from item 8 of Def. 1, which defines function unit as a surjection from P to $U \setminus \{u_0\}$, so that $\mathrm{card}(P) \geq \mathrm{card}(U \setminus \{u_0\}) = \mathrm{card}(U) - 1$. Item 2: the left inequality follows from items 1 and 8 of Def. 1, which state that there exists at least one place p and that each place is mapped to a unit; so there must exist a non-void unit containing p; the right inequality follows from the fact that the local places of each unit are not shared with any other unit ($u_1 \neq u_2 \Rightarrow \mathsf{places}(u_1) \cap \mathsf{places}(u_2) = \varnothing$) and that each non-void unit contains at least one local place. Item 3 follows from item 8 of Def. 1, which requires any unit different from the root unit to contain at least one local place. Item 4 follows from item 1 of the present proposition: there exists at least a non-void unit, which is necessarily (possibly transitively) nested in the root unit u_0 — notice that item 4 ensures that the three premises in the definition of the auxiliary function h in Def. 2 are mutually exclusive. Item 5: from item 3 of the present proposition, $\mathsf{void}(u) \Rightarrow u = u_0$ and, from item 4, $\mathsf{void}(u_0) \Rightarrow \neg\mathsf{leaf}(u_0)$. □

Thus, item 8 of Def. 1 plays a crucial role by forbidding the existence of void, non-root units. Without this restriction, the structure of NUPNs would be more complex, with an arbitrarily large number of units for a given number of places (e.g., void units that would either be disconnected, or nested at arbitrary depth), so that both measures $\mathsf{height}(N)$ and $\mathsf{width}(N)$ would be meaningless. In practice, all void units different from the root unit can be eliminated without loss of generality. Notice that, contrary to void units, redundant units are not prohibited, but tolerated.

Proposition 2. *Let* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. The family of sets* $\mathsf{places}(u)$*, where* $u \in \widetilde{U}$*, is a partition of* P.

Proof. It follows from item 8 of Def. 1 that all sets in the family are not empty. It follows from the definitions of places and unit that all sets in the family are pairwise disjoint. From these same definitions and the fact that unit is totally defined, it follows that the union of all sets in the family is equal to P. □

Proposition 3. *Let* $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN.*

1. $card(P) = 1 \Rightarrow \mathsf{trivial}(N)$.
2. $\mathsf{trivial}(N) \Rightarrow (\forall u \in U)\ (\neg\mathsf{leaf}(u) \Rightarrow \mathsf{void}(u))$.
3. $\mathsf{trivial}(N) \Rightarrow \widetilde{U} = \{u \in U \mid \mathsf{leaf}(u)\}$.
4. $\mathsf{trivial}(N) \Rightarrow \mathsf{height}(N) = 1$.
5. $\mathsf{trivial}(N) \Rightarrow (\forall u \in U)\ card(\mathsf{places}(u)) \leq 1$.

Proof. Item 1: if $\mathrm{card}(P) = 1$ then, from item 1 of Prop. 1, $\mathrm{card}(U) \in \{1, 2\}$; so, either N has a single unit, or N has a void root unit containing another unit; in both cases, there is only one leaf unit, so that $\mathsf{width}(N) = 1$. Item 2: by contradiction: assume the existence of a unit u' such that $\neg\mathsf{leaf}(u') \wedge \neg\mathsf{void}(u')$; then u' contains at least one local place noted p'; consider the set of places Q defined as $Q = \{p \in P \mid (\exists u \in U)\ \mathsf{leaf}(u) \wedge (\mathsf{unit}(p) = u)\}$; clearly $p' \in P$ and $p' \notin Q$; following item 5 of Prop. 1, for any unit u, $\mathsf{leaf}(u) \Rightarrow \neg\mathsf{void}(u)$, thus $\mathrm{card}(\mathsf{places}(u)) \geq 1$; therefore $\mathrm{card}(Q) = \mathrm{card}(\{p \in \mathsf{places}(u) \mid \mathsf{leaf}(u)\}) \geq \mathrm{card}(\{u \in U \mid \mathsf{leaf}(u)\}) = \mathsf{width}(N)$; if N is trivial, $\mathrm{card}(Q) \geq \mathrm{card}(P)$, then $P = Q$, given that $Q \subseteq P$; so place p' cannot exist, nor unit u' as well. Item 3: from item 5 of Prop. 1 and, if N is trivial, from item 2 of the present proposition, $(\forall u \in U)\ \mathsf{void}(u) \iff \neg\mathsf{leaf}(u)$; thus $\widetilde{U} = \{u \in U \mid \neg\mathsf{void}(u)\} = \{u \in U \mid \mathsf{leaf}(u)\}$. Item 4: by definition, $\mathsf{height}(N) = h(u_0)$; if $\mathsf{leaf}(u_0)$ then $h(u_0) = 1$; if $\neg\mathsf{leaf}(u_0)$ then, from items 2 and 3 of the present proposition, it follows that $\mathsf{void}(u_0)$ and $(\forall u \in U)\ u \neq u_0 \Rightarrow \mathsf{leaf}(u)$, so that $h(u_0) = \max(\{h(u) \mid u \in \mathsf{subunits}(u_0)\}) = \max(\{h(u) \mid \mathsf{leaf}(u)\}) = 1$. Item 5: from item 3 of the present proposition, it follows that $\mathrm{card}(\widetilde{U}) = \mathrm{card}(\{u \in U \mid \mathsf{leaf}(u)\}) = \mathsf{width}(N) = \mathrm{card}(P)$; from Prop. 2, $\{\mathsf{places}(u) \mid u \in \widetilde{U}\}$ is a partition of P; thus, each set $\mathsf{places}(u)$ must be a singleton; finally, from item 2 of the present proposition, any unit u not in $\widetilde{U}$ must be void, i.e., $\mathsf{places}(u) = \varnothing$. □

Notice that trivial NUPNs reach the upper bounds stated by the first two items of Prop. 1, i.e., $\mathrm{card}(\widetilde{U}) = \mathrm{card}(P)$ and $\mathrm{card}(U) = \mathrm{card}(P) + 1$ (if $\mathrm{card}(P) > 1$); however, some non-trivial NUPNs also satisfy such equalities, e.g., a NUPN having three units u_0, u_1, and u_2 ($u_2 \sqsubseteq u_1 \sqsubseteq u_0$) such that u_0 is void while u_1 and u_2 contain one local place each.

Notice also that none of the converse implications of the five items of Prop. 3 hold; for instance, the converse of items 2, 3, and 4 is false for a NUPN having a single unit containing two places; the converse of item 5 is false for a NUPN having two units u_0 and u_1 ($u_1 \sqsubseteq u_0$) containing one local place each.

2.2. Behaviour

This subsection defines the dynamic semantics of the NUPN model, namely the "token game" rules for computing markings and firing transitions. In a nutshell, the rules for a NUPN $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ are exactly the same

as those for an elementary net (P, T, F, M_0); that is, the unit-related part $(U, u_0, \sqsubseteq, \mathsf{unit})$ does not influence the execution of the NUPN. To take into account the existence of non-safe elementary nets, namely, the situations in which firing some transition t would add another token to a place p that already has a token, we adopt the usual distinction between *weak firing rules*, which allow to fire t and merge the two tokens of p into a single one, and *strict firing rules*, which forbid to fire t in such case.

Definition 3. *Let $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. Let t be a transition:*

- *The* pre-set *of t is the set of places defined as ${}^\bullet t \stackrel{\text{def}}{=} \{p \in P \mid (p, t) \in F\}$.*
- *The* post-set *of t is the set of places defined as $t^\bullet \stackrel{\text{def}}{=} \{p \in P \mid (t, p) \in F\}$.*
- *To avoid ambiguities, ${}^\bullet t$ and $t^\bullet$ will also be respectively noted ${}^{\bullet_F} t$ and $t^{\bullet_F}$ in contexts where several arc relations F are present.*
- *A* marking *M is defined as a set of places ($M \subseteq P$). Each place belonging to a marking M is said to be* marked *or, also, to possess a* token*.*
- *A transition t is* enabled *in some marking M iff it satisfies the predicate $\mathsf{enabled}\,(M, t) \stackrel{\text{def}}{=} ({}^\bullet t \subseteq M)$.*
- *A transition t can* safely fire *from some marking M iff it satisfies the predicate $\mathsf{safe\text{-}fire}\,(M, t) \stackrel{\text{def}}{=} \mathsf{enabled}\,(M, t) \wedge ((M \setminus {}^\bullet t) \cap t^\bullet = \varnothing)$*
- *A transition t can* weakly fire *from some marking M_1 to another marking M_2 iff $\mathsf{enabled}\,(M_1, t) \wedge (M_2 = (M_1 \setminus {}^\bullet t) \cup t^\bullet)$, which we note $M_1 \xrightarrow{t} M_2$.*
- *A transition t can* strictly fire *from some marking M_1 to another marking M_2 iff $\mathsf{safe\text{-}fire}\,(M_1, t) \wedge (M_2 = (M_1 \setminus {}^\bullet t) \cup t^\bullet)$.*
- *A marking M is* reachable *from the initial marking M_0 iff $M = M_0$ or there exist $n \geq 1$ transitions $t_1, t_2, ..., t_n$ and $(n-1)$ markings $M_1, M_2, ..., M_{n-1}$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \ ... \ M_{n-1} \xrightarrow{t_n} M$.*
- *A transition t is* quasi-live *is there exists some reachable marking M such that $\mathsf{enabled}\,(M, t)$. A transition that is not quasi-live is said to be* dead*.*
- *The NUPN is* safe *(or* one-safe*, or* contact-free*) iff for each reachable marking M and transition t, $\mathsf{enabled}\,(M, t) \Rightarrow \mathsf{safe\text{-}fire}\,(M, t)$. In such case, the weak firing rules and the strict firing rules coincide.*

Notice that Def. 1 is general enough and tolerates: (i) transitions with empty pre-sets (i.e., ${}^\bullet t = \varnothing$), keeping in mind that nets containing such transitions are not one-safe; (ii) transitions with empty post-sets (i.e., $t^\bullet = \varnothing$); (iii) transitions that are not pure (i.e., ${}^\bullet t \cap t^\bullet \neq \varnothing$); and (iv) transitions that are not simple (i.e., ${}^\bullet t = {}^\bullet t' \wedge t^\bullet = t'^\bullet \not\Rightarrow t = t'$).

Definition 4. *Let* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. Given a marking* M *and a unit* u*, let the* projection *of* M *on* u *be defined as* $M \rhd u \stackrel{\text{def}}{=} M \cap \mathsf{places}(u)$*.*

Proposition 4. *Let* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. Any marking* M *can be expressed as* $M = (M \rhd u_1) \uplus ... \uplus (M \rhd u_n)$*, where* $u_1, ..., u_n$ *are the units of* $\widetilde{U}$*, and where* $\uplus$ *denotes the disjoint set union.*

Proof. This directly follows from Prop. 2, given that the family $\mathsf{places}(u_1)$, ..., $\mathsf{places}(u_n)$ is a partition of P. □

3. Unit Safeness

This section introduces the so-called *unit-safeness* property, which does not exist in "classical" Petri nets but plays a central role in the NUPN model.

Definition 5. *Let* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. A marking* $M \subseteq P$ *is said to be* unit safe *iff it satisfies the predicate defined as follows:* $\mathsf{unit\text{-}safe}(M) \stackrel{\text{def}}{=} (\forall p_1, p_2 \in M)\ (p_1 \neq p_2) \Rightarrow \mathsf{disjoint}(\mathsf{unit}(p_1), \mathsf{unit}(p_2))$*; that is, all places of a unit-safe marking are contained in disjoint units.*

Proposition 5. *Let* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. For each marking* M*, for each unit* u*,* $\mathsf{unit\text{-}safe}(M) \Rightarrow \mathit{card}(M \rhd u) \leq 1$*; that is, a unit-safe marking cannot contain two different local places of the same unit.*

Proof. By contradiction. If $\mathrm{card}(M \rhd u) > 1$, there exist at least two different places p_1 and p_2 in $M \cap \mathsf{places}(u)$. Because p_1 and p_2 both belong to $\mathsf{places}(u)$, it follows that $\mathsf{unit}(p_1) = \mathsf{unit}(p_2)$, then $\neg\mathsf{disjoint}(\mathsf{unit}(p_1), \mathsf{unit}(p_2))$, and finally $\neg\mathsf{unit\text{-}safe}(M)$. □

Proposition 6. *Let* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. For each marking* M*, for all units* u *and* u'*,* $\mathsf{unit\text{-}safe}(M) \wedge (M \rhd u \neq \varnothing) \wedge (u' \sqsubset u \vee u \sqsubset u') \Rightarrow (M \rhd u' = \varnothing)$*; that is, if a unit-safe marking contains a local place of some unit* u*, it contains no local place of any ancestor or descendent unit* u' *of* u*.*

Proof. By contradiction. If $M \rhd u' \neq \varnothing$ then M contains at least one place $p \in \mathsf{places}(u)$ and at least one place $p' \in \mathsf{places}(u')$. If $u' \sqsubset u$ or $u \sqsubset u'$ then $\neg\mathsf{disjoint}(u, u')$, hence $\neg\mathsf{unit\text{-}safe}(M)$. □

Notice, still assuming that $\mathsf{unit\text{-}safe}(M) \wedge (u' \sqsubset u \vee u \sqsubset u')$, that the reverse implication $(M \rhd u = \varnothing) \Rightarrow (M \rhd u' \neq \varnothing)$ does not hold, as tokens can be absent from both u and u'.

Prop. 6 can be given an intuitive explanation in a process calculus setting. Consider a process term of the form $\boxed{B_1 \cdot (\ \boxed{B_2}\ ||\ \boxed{B_3}\ ) \cdot B_4}$ where B_1, B_2, B_3, and B_4 are sequential process terms, and where square boxes denotes the units enclosing the places corresponding to these terms. The above proposition states that: (i) while B_1 or B_4 execute, neither B_2 nor B_3 can execute, because they are in descendent units of the unit containing B_1 and B_4; and (ii) while

B_2 and/or B_3 execute, neither B_1 nor B_4 can execute, because they are in an ascendent unit of the units containing B_2 and B_3. Reasoning on "forks" and "joins" is another way to grasp the intuitive meaning of nested units.

Definition 6. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. N is said to be* unit safe *iff it is safe and all its reachable markings are unit safe.*

Thus, a unit-safe NUPN is also safe. The reverse implication does not hold; consider e.g., a safe NUPN with a single unit u_0 and two places p_1 and p_2 contained in u_0; let M_0 be $\{p_1, p_2\}$: this initial marking is safe but not unit safe.

Notice that, if NUPN definitions would be based on (ordinary) P/T nets rather than elementary nets, with markings defined as place multisets (i.e., functions $P \to \mathbb{N}$) rather than place subsets, unit safeness could be simply defined by requiring all reachable markings to be unit safe, without also asking for the net to be safe; indeed, in such case, unit safeness would imply safeness as a particular case (each reachable marking is unit safe and has, from Prop. 5, at most a token among the local places of each unit; thus, no place can contain more than one token).

Proposition 7. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. If N is trivial and safe, then N is unit safe.*

Proof. Assume $\mathsf{trivial}(N)$. Let $M \subseteq P$ be any marking (reachable or not). Let $n = \mathrm{card}(U) - 1$. If $n = 0$, then $\mathrm{card}(P) = 1$ and $\mathsf{unit\text{-}safe}(M)$ holds. If $n > 0$, let $u_1, ..., u_n$ denote the n leaf units such that $U = \{u_0, u_1, ..., u_n\}$; let $p_1, ..., p_n$ denote the n places of P such that $(\forall i \in \{1, ..., n\})\ \mathsf{unit}(p_i) = u_i$; for any two distinct places p_i and p_j in M, $\mathsf{disjoint}(\mathsf{unit}(p_i), \mathsf{unit}(p_j)) = \mathsf{disjoint}(u_i, u_j) =$ *true* since $i > 0$, $j > 0$, and $i \neq j$; thus $\mathsf{unit\text{-}safe}(M)$ holds. Therefore, N is unit safe. □

An important issue is an efficient decision procedure to determine whether a "syntactically well-formed" NUPN (according to Def. 1) is unit safe or not. This issue will be further discussed in Sect. 9. The following conditions give preliminary, yet useful checks that can be easily performed.

Proposition 8. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. Let t be a transition.*

1. *If $\neg\mathsf{unit\text{-}safe}(M_0)$ then N is not unit safe.*
2. *If $\neg\mathsf{unit\text{-}safe}({}^\bullet t)$ then either N is not unit safe or t is not quasi-live.*
3. *If $\neg\mathsf{unit\text{-}safe}(t^\bullet)$ then either N is not unit safe or t is not quasi-live.*

Proof. Item 1 directly follows from Def. 6 given that M_0 is a reachable marking. Items 2 and 3: by contradiction. Assuming both that N is unit safe and t is quasi-live, it follows from the latter condition that there exist two reachable markings M_1 and M_2 such that $M_1 \xrightarrow{t} M_2$; consequently, ${}^\bullet t \subseteq M_1$ and $t^\bullet \subseteq M_2$. If either $\neg\mathsf{unit\text{-}safe}({}^\bullet t)$ or $\neg\mathsf{unit\text{-}safe}(t^\bullet)$ then either $\neg\mathsf{unit\text{-}safe}(M_1)$ or $\neg\mathsf{unit\text{-}safe}(M_2)$; thus, N is not unit safe. □

It is well known that a safe net (P, T, F, M_0) is k-bounded, with an upper bound $k \leq \mathrm{card}\,(P)$. The unit-safeness property provides a tighter bound.

Proposition 9. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a unit-safe NUPN. Any reachable marking M satisfies $\mathrm{card}\,(M) \leq \mathsf{width}\,(N)$. In other words, the underlying net (P, T, F, M_0) is k-bounded, with an upper bound $k \leq \mathsf{width}\,(N)$.*

Proof. Because N is unit safe, any reachable marking M satisfies $\mathsf{unit\text{-}safe}\,(M)$. Prop. 5 ensures that each unit has at most one token in M, so that $\mathrm{card}\,(M) \leq \mathrm{card}\,(\widetilde{U})$, which is already, from item 2 of Prop. 1, a tighter bound than $\mathrm{card}\,(P)$. Moreover, Prop. 6 ensures that, on each path of the unit tree between the root unit and any leaf unit, at most one unit has a token in M; the largest number of tokens is obtained when all of them are located in leaf units, so that $\mathrm{card}\,(M) \leq \mathrm{card}\,(\{u \in U \mid \mathsf{leaf}\,(u)\}) = \mathsf{width}\,(N)$. □

The unit-safeness property can be reformulated as a system of linear inequalities over the tokens present in reachable markings. Notice that such constraints differ from the traditional S-invariants [26, 27], which are linear equations.

Proposition 10. *Let $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a safe NUPN. N is unit safe iff any reachable marking M satisfies the following system of inequalities:*

$$(\forall u \in \widetilde{U})\ (\forall u' \in \widetilde{U} \mid u \sqsubseteq u')\ \sum_{p \in \mathsf{places}\,(u) \cup \mathsf{places}\,(u')} x_p \leq 1 \qquad (I_{u,u'})$$

where each variable x_p gives the number of tokens in place p (i.e., x_p is equal to 1 if place p belongs to M, or 0 otherwise).

Proof. By double implication: (Direct) If N is unit safe, then $\mathsf{unit\text{-}safe}\,(M)$ is true. Prop. 5 ensures all inequalities $(I_{u,u'})$ with $u = u'$, since $\sum_{p \in \mathsf{places}\,(u)} x_p = \mathrm{card}\,(M \rhd u)$. Prop. 6 ensures all inequalities $(I_{u,u'})$ with $u \sqsubset u'$, knowing that $u \neq u' \Rightarrow \sum_{p \in \mathsf{places}\,(u) \cup \mathsf{places}\,(u')} x_p = \mathrm{card}\,(M \rhd u) + \mathrm{card}\,(M \rhd u')$ and, from Prop. 6, $(M \rhd u \neq \varnothing) \Rightarrow (M \rhd u' = \varnothing)$ and $(M \rhd u' \neq \varnothing) \Rightarrow (M \rhd u = \varnothing)$, i.e., $(M \rhd u = \varnothing) \vee (M \rhd u' = \varnothing)$, which leads, after applying Prop. 5 twice, to $\mathrm{card}\,(M \rhd u) + \mathrm{card}\,(M \rhd u') \leq 1$. (Reverse) If N is not unit safe, but safe, there exists some reachable marking M such that $\neg\mathsf{unit\text{-}safe}\,(M)$. Thus, there exist two distinct places p_1 and p_2, and two units $u_1 = \mathsf{unit}\,(p_1)$ and $u_2 = \mathsf{unit}\,(p_2)$ such that $\neg\mathsf{disjoint}\,(u_1, u_2)$, i.e., $u_1 \sqsubseteq u_2$ or $u_2 \sqsubseteq u_1$. In both cases, $\sum_{p \in \mathsf{places}\,(u_1) \cup \mathsf{places}\,(u_2)} x_p \geq x_{p_1} + x_{p_2} = 2$, so that M violates inequality (I_{u_1,u_2}) if $u_1 \sqsubseteq u_2$, and/or violates inequality (I_{u_2,u_1}) if $u_2 \sqsubseteq u_1$. □

4. Expressiveness

This section discusses the expressiveness of the NUPN model by showing its ability to encode mainstream forms of Petri nets.

As mentioned above, a unit-safe NUPN is safe, which implies that its underlying elementary net is also safe. The following proposition establishes the reverse implication.

Proposition 11. *Let (P, T, F, M_0) be any ordinary, safe P/T net (i.e., a safe elementary net). There exists at least one 4-tuple $(U, u_0, \sqsubseteq, \mathsf{unit})$ such that $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ is a unit-safe NUPN.*

Proof. Let $p_1, ..., p_n$ be the places of P, where $n = \mathrm{card}(P) \geq 1$. Let $u_0, u_1, ..., u_n$ be $(n+1)$ units and let $U = \{u_0, u_1, ..., u_n\}$. Let $\sqsubseteq$ be the relation defined by $(\forall u \in U)\ (u \sqsubseteq u) \wedge (u \sqsubseteq u_0)$; $(U, \sqsupseteq)$ is clearly a tree with a single root u_0. Let unit be the function $P \to U$ such that $(\forall i \in \{1, ..., n\})\ \mathsf{unit}(p_i) = u_i$, meaning that only the root unit u_0 has no local place. Therefore, the NUPN $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ satisfies all the structural conditions of Def. 1. This NUPN is trivial, by construction, and safe, because (P, T, F, M_0) is safe. From Prop. 7, it is also unit safe. □

Notice that the trivial NUPN (each place in a distinct unit) exhibited by this proof is not necessarily the only one: there may exist other unit-safe NUPNs with fewer units, such units thus having more local places; this issue will be discussed in Sect. 9. Also, this proof justifies why item 8 of Def. 1 allows the root unit to be void, whereas all other units cannot. In the particular case where P is a singleton, the trivial NUPN constructed by the proof could be slightly simplified by generating only one unit and avoiding the creation of a void, redundant root unit.

The next proposition establishes that NUPNs subsume *communicating automata*, i.e., sequential state machines that execute in parallel and possibly synchronize on (some of) their transitions. In the Petri net framework, communicating automata are easily expressed using so-called *state-machine components* (see, e.g., [28, p. 557]). There are various definitions of state-machine components (a survey is given in [23, Sect. 5–8]); we adopt the following one:

Definition 7. *Let $N = (P, T, F, M_0)$ be any ordinary P/T net.*

- *Let $P' \subseteq P$ be a set of places. The* subnet of N generated by P' [28, p. 557] *is defined to be the net $N' = (P', T', F', M'_0)$, where $M'_0 = M_0 \cap P'$ is the initial marking restricted to P', where $T' = \{t \in T \mid ({}^\bullet t \cup t^\bullet) \cap P' \neq \varnothing\}$ is the set of all the input and output transitions of the places of P', and where $F' = F \cap ((P' \times T') \cup (T' \times P'))$ is the set of arcs connecting the places of P' and the transitions of T'.*

- *The net N possesses a* state-machine decomposition *iff there exists a partition $P_1, ..., P_n$ of P such that, for each $i \in \{1, ..., n\}$, each subnet $N_i = (P_i, T_i, F_i, M_{0i})$ of N generated by P_i is a state machine* [28, p. 554] *with one token, i.e., $\mathrm{card}(M_{0i}) = 1$ and $(\forall t \in T_i)\ \mathrm{card}({}^{\bullet_{F_i}} t) = \mathrm{card}(t^{\bullet_{F_i}}) = 1$.*

Notice that our definition of state-machine components assumes that the sets of places $P_1, ..., P_n$ are pairwise disjoint, which is not required by all authors — see, e.g., in [23] the difference between *synchronized state machines*, whose components can share places, and *superposed automata nets* or *basic modular nets*, whose components cannot. Our definition is actually equivalent to that of superposed automata net systems [23, Def. 29], i.e., $(\forall i \in \{1, ..., n\})$ $(\forall t \in T)\ 0 \leq \mathrm{card}({}^\bullet t \cap P_i) = \mathrm{card}(t^\bullet \cap P_i) \leq 1$ and $\mathrm{card}(M_0 \cap P_i) = 1$.

Proposition 12. *Let (P, T, F, M_0) be any ordinary P/T net possessing a state-machine decomposition. There exists at least one 4-tuple $(U, u_0, \sqsubseteq, \mathsf{unit})$ such that $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ is a unit-safe NUPN.*

Proof. Let $P_1, ..., P_n$ be the sets of places resulting from Def. 7. Due to the state-machine decomposition, it is easy to prove, by induction, that any reachable marking M has the form $\{p_1, ..., p_n\}$, where each p_i belongs to P_i. Thus, the net (P, T, F, M_0) is safe. From Prop. 11, it exists a unit-safe NUPN $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$. However, in the present case, the proof of Prop. 11 leads to a trivial NUPN that is not optimal, since it has $\mathrm{card}\,(P) + 1$ units. We show below that a "better" NUPN, with only $n + 1$ units, can be constructed by exploiting state-machine decomposition.

Let $u_0, u_1, ..., u_n$ be $(n + 1)$ units and let $U = \{u_0, u_1, ..., u_n\}$. Let $\sqsubseteq$ be the relation defined by $(\forall u \in U)\ (u \sqsubseteq u) \wedge (u \sqsubseteq u_0)$; $(U, \sqsupseteq)$ is clearly a tree with a single root u_0. Let unit be the function $P \to U$ totally defined as follows: $(\forall i \in \{1, ..., n\})\ (\forall p \in P_i)\ \mathsf{unit}\,(p) = u_i$, meaning that only the root unit u_0 has no local place. The NUPN $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ satisfies all the structural conditions of Def. 1. This NUPN is safe because (P, T, F, M_0) is safe. For any reachable marking M having the aforementioned form $\{p_1, ..., p_n\}$, where $(p_1, ..., p_n) \in P_1 \times ... \times P_n$, for any two distinct places p_i and p_j in M, $\mathsf{disjoint}\,(\mathsf{unit}\,(p_i), \mathsf{unit}\,(p_j)) = \mathsf{disjoint}\,(u_i, u_j) = \mathit{true}$ since $i > 0$, $j > 0$, and $i \neq j$; therefore, the NUPN is unit safe. □

Unlike for Prop. 11, the units of the NUPN constructed by the proof of Prop. 12 reflect the parallel composition between state machines. Also, in the particular case where $n = 1$, this NUPN could be simplified by generating a single unit and avoiding the creation of a void, redundant root unit.

Nested units have the same theoretical expressiveness as communicating automata/state machines, but are more convenient for at least two reasons:

1. They add the notion of hierarchy to the concepts of concurrency and nondeterminism already present in elementary nets. This is similar to the escalation from communicating state machines to Statecharts [29] and hierarchical communicating state machines [30, 31].
2. For each state machine P_i, there exists an S-invariant, which states that $\sum_{p \in \mathsf{places}\,(P_i)} x_p = 1$, where M is any reachable marking and x_p the number of tokens M has in place p. This S-invariant is a consequence of the constraint that each transition of the subnet N_i must have exactly one input place and one output place in P_i. On the contrary, each unit u_i is not ruled by an S-invariant but a boundedness inequality of the form $\sum_{p \in \mathsf{places}\,(u_i)} x_p \leq 1$ (cf. Prop. 10). The possibility of having no token in a unit has proven useful when expressing safe nets as NUPNs (cf. proof of Prop. 11); in practice, it also provides greater modelling flexibility in at least three situations:
   - It enables a unit not to have a token in the initial marking and to get a token later (e.g., when the unit is launched by a "fork" transition).

- It enables a unit to lose its token (or its tokens, if that unit has sub-units running in parallel), either when the unit normally completes with a "join" transition, or when it is abruptly terminated by a transition implementing, e.g., the LOTOS "disable" operator [5] or the raise of an exception [32].
- It allows a transition to have an input place in a given unit but no output place in this unit, or even no output place at all. The latter is useful to model processes ending with deadlock, such as the CCS term $a \cdot NIL$, which can be implemented in two different ways: either (i) by a net N_1 having one transition with one input place p and one output place p' (leading to the inequality $x_p + x_{p'} \leq 1$, actually $x_p + x_{p'} = 1$), or (ii) by a net N_2 having one transition with one input place p and no output place (leading to the inequality $x_p \leq 1$). Contrary to S-invariants, which only support translation (i), the NUPN model supports both translations (i) and (ii), and generates two possible nets N_1 and N_2, both of which are unit safe.

5. Place-Fusion Abstraction

We now study an abstraction mentioned in [20, Sect. 5] that takes advantage of the units to turn a NUPN into a simpler net (i.e., with fewer places and a much smaller graph of reachable markings), still preserving some (but not all) useful properties of the original NUPN. This abstraction is somehow related to the concept of "place fusion" proposed in [33, Chap. 3] to ease the modular drawing of Coloured Petri Nets, especially in light of a remark found on the CPN tools web site[8]: "*if all the members of a fusion set are on the same page, we could replace the set with a single place and connect to it all the arcs that are connected to any member of the set*", in which a "*fusion set*" could stand for a NUPN unit. However, the behavioural semantics of our abstraction clearly differs from that given in [33], which states: "*when a token is added/removed at one of the places [in a fusion set], an identical token will be added/removed at all the others*".

5.1. Definitions

Definition 8. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. Let $\sim$ be the equivalence relation over P that puts in the same class all places having the same unit, i.e., $p_1 \sim p_2 \iff \mathsf{unit}(p_1) = \mathsf{unit}(p_2)$. Let P' be the quotient set of P by this relation $\sim$, hence $card(P') = card(\widetilde{U})$. For convenience, we assume that $P' \subseteq P$, meaning that each equivalence class is represented by a distinguished place chosen in P.*

- *We define the abstraction function α as the function $P \to P'$ that maps each place of P to the distinguished representative of its equivalence class.*

[8] See http://cpntools.org/documentation/concepts/hierarchy/fusion_places

- *We extend such abstraction to sets of places and markings by overloading the abstraction function α to also denote the function $2^P \to 2^{P'}$ such that $(\forall M \subseteq P)\ \alpha(M) \stackrel{\text{def}}{=} \{\alpha(p) \mid p \in M\}$.*

Definition 9. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN and let α be its abstraction function. We define N_α as the 8-tuple $(P', T, F', M_0', U, u_0, \sqsubseteq, \mathsf{unit}')$ derived from N by merging, in each unit u, all the local places of u into a single place, also local to u. Formally:*

- *$P' \stackrel{\text{def}}{=} \alpha(P)$ denotes the set of abstracted places.*
- *$F' \subseteq (P' \times T) \cup (T \times P')$ is the finest[9] arc relation that satisfies: $(\forall p \in P)\ (\forall t \in T)\ ((p,t) \in F \Rightarrow (\alpha(p), t) \in F') \wedge ((t,p) \in F \Rightarrow (t, \alpha(p)) \in F')$.*
- *$M_0' \stackrel{\text{def}}{=} \alpha(M_0)$ denotes the initial marking after abstraction.*
- *unit' is the function $P' \to U$ such that $(\forall p \in P)\ \mathsf{unit}'(\alpha(p)) = \mathsf{unit}(p)$.*

5.2. Preservation Results

Proposition 13. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN and let N_α be the 8-tuple $(P', T, F', M_0', U, u_0, \sqsubseteq, \mathsf{unit}')$ as defined in Def. 9. Then:*

1. *N_α is also a NUPN.*
2. *The pre-set in N_α of each transition t is ${}^{\bullet_{F'}}t = \alpha({}^{\bullet_F}t)$.*
3. *The post-set in N_α of each transition t is $t^{\bullet_{F'}} = \alpha(t^{\bullet_F})$.*

Proof. Item 1: because N satisfies all the conditions of Def. 1, it is easy to see that N_α also satisfies these conditions. Item 2 by double inclusion: (Reverse) $\alpha({}^{\bullet_F}t) = \alpha(\{p \in P \mid (p,t) \in F\}) = \{\alpha(p) \mid p \in P \wedge (p,t) \in F\}$, which, from the definition of F' in Def. 9, is included in $\{\alpha(p) \mid p \in P \wedge (\alpha(p), t) \in F'\}$, which, given that $P' = \alpha(P)$, is itself equal to $\{p' \in P' \mid (p', t) \in F'\}$, i.e., ${}^{\bullet_{F'}}t$. (Direct) Because F' is defined as the finest relation, $(\forall p' \in P)\ (p', t) \in F' \Rightarrow (\exists p \in P)\ (p' = \alpha(p) \wedge (p,t) \in F)$; thus, ${}^{\bullet_{F'}}t = \{p' \in P' \mid (p', t) \in F'\}$ is included in $\{\alpha(p) \mid p \in P \wedge (p,t) \in F\}$, i.e., $\alpha({}^{\bullet_F}t)$. Item 3: dual proof of item 2. □

Proposition 14. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN and let $N_\alpha = (P', T, F', M_0', U, u_0, \sqsubseteq, \mathsf{unit}')$ be its abstracted NUPN. For any marking $M \subseteq P$, for any transition t:*

1. *If $\mathsf{enabled}(M, t)$ in N, then $\mathsf{enabled}(\alpha(M), t)$ in N_α.*
2. *If $\mathsf{safe\text{-}fire}(M, t)$ in N, then $\mathsf{safe\text{-}fire}(\alpha(M), t)$ in N_α.*

Proof. Item 1: $\mathsf{enabled}(M,t)$ in $N \iff ({}^{\bullet_F}t \subseteq M) \Rightarrow (\alpha({}^{\bullet_F}t) \subseteq \alpha(M)) \iff ({}^{\bullet_{F'}}t \subseteq \alpha(M)) \iff \mathsf{enabled}(\alpha(M), t)$ in N_α; the reverse implication is false, e.g., if t has two input places p_1 and p_2 in the same unit and $M = \{p_1\}$. Item 2: $(\alpha(M) \setminus {}^{\bullet_{F'}}t) \cap t^{\bullet_{F'}} = (\alpha(M) \setminus \alpha({}^{\bullet_F}t)) \cap \alpha(t^{\bullet_F}) \subseteq \alpha(M \setminus {}^{\bullet_F}t) \cap \alpha(t^{\bullet_F}) = \alpha((M \setminus {}^{\bullet_F}t) \cap t^{\bullet_F})$; therefore, $\mathsf{safe\text{-}fire}(M,t)$ in N, i.e., $(M \setminus {}^{\bullet_F}t) \cap t^{\bullet_F} = \varnothing$,

[9] A relation R_1 is *finer* than a relation R_2 iff $(\forall x, y)\ xR_1y \Rightarrow xR_2y$.

implies $(\alpha(M) \setminus {}^{\bullet_{F'}}t) \cap t^{\bullet_{F'}} \subseteq \alpha(\varnothing) = \varnothing$, i.e., $\mathsf{safe\text{-}fire}\,(\alpha(M), t)$ in N_α; the reverse implication is false, e.g., if t has one input place p_1, one input place p_2, both places are in the same unit, and $M = \{p_1, p_2\}$. □

Proposition 15. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN and let N_α be its abstracted NUPN. For any marking M (reachable or not) of N:*

1. *If M is unit safe in N, then $\alpha(M)$ is unit safe in N_α.*
2. *If M is unit safe in N, then $card\,(\alpha(M)) = card\,(M)$.*
3. *For any unit u, $\alpha(M \rhd u) = \alpha(M) \rhd u$.*

Proof. Item 1: assume $M \subseteq P$ such that $\mathsf{unit\text{-}safe}\,(M)$; for any two distinct places p'_1 and p'_2 in $\alpha(M)$, there exist two places p_1 and p_2 in M such that $\alpha(p_1) = p'_1$ and $\alpha(p_2) = p'_2$; therefore, $\mathsf{unit}\,(p'_1) = \mathsf{unit}\,(p_1)$ and $\mathsf{unit}\,(p'_2) = \mathsf{unit}\,(p_2)$, hence $p'_1 \neq p'_2 \Rightarrow p_1 \neq p_2$; thus, $\mathsf{disjoint}\,(\mathsf{unit}\,(p'_1), \mathsf{unit}\,(p'_2)) = \mathsf{disjoint}\,(\mathsf{unit}\,(p_1), \mathsf{unit}\,(p_2))$ is true because M is a unit-safe marking. Item 2: assume $M \subseteq P$ such that $\mathsf{unit\text{-}safe}\,(M)$; from the definition of function α on sets of places, and because function α defined on places is not necessarily injective, one gets: $\mathrm{card}\,(\alpha(M)) \leq \mathrm{card}\,(M)$. Both terms can be proven equal by contradiction: if $\mathrm{card}\,(\alpha(M)) < \mathrm{card}\,(M)$, there must exist at least two distinct places p_1 and p_2 in M such that $\alpha(p_1) = \alpha(p_2)$, i.e., $\mathsf{unit}\,(p_1) = \mathsf{unit}\,(p_2)$, and thus $\neg\mathsf{disjoint}\,(\mathsf{unit}\,(p_1), \mathsf{unit}\,(p_2))$, which contradicts $\mathsf{unit\text{-}safe}\,(M)$. Item 3: From Def. 8, $\alpha(\mathsf{places}\,(u)) \subseteq \mathsf{places}\,(u)$ and $\alpha(M) \cap \alpha(\mathsf{places}\,(u)) = \alpha(M) \cap \mathsf{places}\,(u)$. Thus, $\alpha(M \rhd u) = \alpha(M \cap \mathsf{places}\,(u)) = \alpha(M) \cap \alpha(\mathsf{places}\,(u)) = \alpha(M) \cap \mathsf{places}\,(u) = \alpha(M) \rhd u$. □

Proposition 16. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN and let N_α be its abstracted NUPN. If N is unit safe, then for any quasi-live transition t: $card\,(\alpha({}^\bullet t)) = card\,({}^\bullet t)$ and $card\,(\alpha(t^\bullet)) = card\,(t^\bullet)$.*

Proof. Because N is unit safe and t is quasi-live, items 2 and 3 of Prop. 8 imply $\mathsf{unit\text{-}safe}\,({}^\bullet t)$ and $\mathsf{unit\text{-}safe}\,(t^\bullet)$. Then, it suffices to apply item 2 of Prop. 15 twice by taking $M = {}^\bullet t$ and $M = t^\bullet$. □

Proposition 17. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN and let N_α be its abstracted NUPN. If N is unit safe, then for any reachable marking M of N, $\alpha(M)$ is a reachable marking of N_α.*

Proof. We show that, for any sequence of transitions starting from the initial marking of N to reach some marking M, the same sequence of transitions can be safely fired from the initial marking of N_α to reach $\alpha(M)$. Proof by induction on the depth n of the execution sequence. The base case $n = 0$ is obvious, as the initial marking of N_α is defined to be the abstraction $\alpha(M_0)$ of the initial marking M_0 of N. General case: let M_1 be a reachable marking of N at depth n, and let M_2 be any reachable marking of N at depth $(n+1)$ obtained by firing some transition t from M_1, i.e., $M_1 \xrightarrow{t} M_2$. From the induction rule and Prop. 15, it follows that $\alpha(M_1)$ is a reachable marking of N_α such that $\mathsf{enabled}\,(\alpha(M_1), t)$ and $\mathsf{safe\text{-}fire}\,(\alpha(M_1), t)$. Let M' denote the

marking of N_α such that $\alpha(M_1) \xrightarrow{t} M'$. We prove that $M' = \alpha(M_2)$ by double inclusion: (Direct) $M' = (\alpha(M_1) \setminus {}^{\bullet_F}t) \cup t^{\bullet_F} = (\alpha(M_1) \setminus \alpha({}^{\bullet_F}t)) \cup \alpha(t^{\bullet_F}) \subseteq \alpha(M_1 \setminus {}^{\bullet_F}t) \cup \alpha(t^{\bullet_F}) = \alpha((M_1 \setminus {}^{\bullet_F}t) \cup t^{\bullet_F}) = \alpha(M_2)$. (Reverse) One has: $M_1 \setminus {}^{\bullet_F}t \subseteq M_1 \Rightarrow \alpha(M_1 \setminus {}^{\bullet_F}t) \subseteq \alpha(M_1) \Rightarrow \alpha(M_1 \setminus {}^{\bullet_F}t) \setminus \alpha({}^{\bullet_F}t) \subseteq \alpha(M_1) \setminus \alpha({}^{\bullet_F}t)$. Since N is a unit-safe NUPN and M_1 a reachable marking of N, M_1 is unit safe. From $\mathsf{enabled}\,(M_1, t)$ and $\mathsf{unit\text{-}safe}\,(M_1)$, it follows that $(M_1 \setminus {}^{\bullet_F}t)$ does not contain any place belonging to any of the units in which ${}^{\bullet_F}t$ has a place; hence, $\alpha(M_1 \setminus {}^{\bullet_F}t) \cap \alpha({}^{\bullet_F}t) = \varnothing$, i.e., $\alpha(M_1 \setminus {}^{\bullet_F}t) \setminus \alpha({}^{\bullet_F}t) = \alpha(M_1 \setminus {}^{\bullet_F}t)$. The above set inclusion thus becomes: $\alpha(M_1 \setminus {}^{\bullet_F}t) \subseteq \alpha(M_1) \setminus \alpha({}^{\bullet_F}t)$. Then, $\alpha(M_2) = \alpha((M_1 \setminus {}^{\bullet_F}t) \cup t^{\bullet_F}) = \alpha(M_1 \setminus {}^{\bullet_F}t) \cup \alpha(t^{\bullet_F}) \subseteq (\alpha(M_1) \setminus \alpha({}^{\bullet_F}t)) \cup \alpha(t^{\bullet_F}) = (\alpha(M_1) \setminus {}^{\bullet_F}t) \cup t^{\bullet_F} = M'$. □

5.3. Non-Preservation Results

We now give three negative results about properties that are not preserved by the place-fusion abstraction.

Proposition 18. *Let N be a unit-safe NUPN. Its abstracted NUPN N_α may contain reachable markings M' for which N contains no reachable marking M such that $M' = \alpha(M)$.*

Said otherwise, the reverse implication of Prop. 17 is false: the abstraction of the set of reachable markings of a unit-safe NUPN is included in the set of reachable markings of its abstracted NUPN, but not the other way round, as the abstracted NUPN may contain markings and execution sequences that have no counterpart in the original NUPN.

Proof. Consider a NUPN N with two units u_1 and u_2 such that $u_1 \sqsupseteq u_2$, three places p_0, p_1, and p_2 such that $\mathsf{unit}\,(p_0) = \mathsf{unit}\,(p_1) = u_1$ and $\mathsf{unit}\,(p_2) = u_2$, and one transition t from p_1 to p_2; the initial marking of N is $\{p_0\}$ and it is the only reachable marking of N, which is thus unit safe. In the abstracted NUPN N_α, places p_0 and p_1 are merged, so that transition t is no longer disconnected from the initial marking and can fire, leading to marking $M = \{\alpha(p_2)\}$, whereas no reachable marking of N has a token in unit u_2. □

Proposition 19. *If a NUPN N is safe, N_α is not necessarily safe.*

Proof. Consider the following NUPN $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ given by $P = \{p_0, p_1, p_2\}$, $T = \{t\}$, $F = \{(p_0, t), (t, p_1), (t, p_2)\}$ (so that ${}^\bullet t = \{p_0\}$ and $t^\bullet = \{p_1, p_2\}$), $M_0 = \{p_0\}$, $U = \{u_0, u\}$, $u \sqsubset u_0$, $\mathsf{unit}\,(p_0) = u_0$, $\mathsf{unit}\,(p_1) = u_0$, and $\mathsf{unit}\,(p_2) = u$. N is safe (but not unit safe, due to its reachable marking $\{p_1, p_2\}$). In N_α, places p_0 and p_1 are merged together (e.g., into p_0), and $F' = \{(p_0, t), (t, p_0), (t, p_2)\}$, meaning that transition t is turned into a self-loop on p_0 and can accumulate infinitely many tokens in p_2; hence, N_α is not safe. □

Proposition 20. *If a NUPN N is unit safe, N_α is not necessarily unit safe.*

Proof. Consider the NUPN N given in the proof of Prop. 19 but with a different initial marking $M_0 = \{p_1\}$. Since M_0 is the only reachable marking, N is unit safe. For the same reason as in the proof of Prop. 19, N_α is not safe, and thus not unit safe. □

5.4. Sequential and Concurrent Units

The tree hierarchy gives a certain structure to a NUPN, but does not necessarily express all the relations that may exist between the units. For instance, taking the example given above for Prop. 6 and assuming that the NUPN generated from the process-calculus term has the following form $\boxed{\boxed{B_1} \cdot (\boxed{B_2} \,||\, \boxed{B_3}) \cdot \boxed{B_4}}$, the unit tree makes no difference between units B_2 and B_3, which execute in parallel, and units B_1 and B_4, which execute in mutual exclusion. Yet, such information about concurrency between units could be useful, e.g., to make the NUPN simpler by merging non-concurrent units B_1 and B_4 into a single one (or even into their parent unit), still preserving the unit-safeness property, whereas such a merging would not be allowed for the concurrent units B_2 and B_3. Such information is also needed to perform accurate data-flow analysis on interpreted Petri nets equipped with a NUPN structure [20, Sect. 5], as concurrent units may cause read/write conflicts on shared variables.

Definition 10. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. Let u_1 and u_2 be two disjoint units of U. These units are said to be* concurrent *if it exists at least one reachable marking M such that $M \rhd u_1 \neq \varnothing$ and $M \rhd u_2 \neq \varnothing$. These units are said to be* sequential *if no such marking exists.*

Proposition 21. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a unit-safe NUPN and let N_α be its abstracted NUPN. Let u_1 and u_2 be disjoint units of U.*

1. *If u_1 and u_2 are concurrent in N, they are also concurrent in N_α.*
2. *If u_1 and u_2 are sequential in N_α, they are also sequential in N.*

Proof. Item 1: u_1 and u_2 concurrent in N, it exists a reachable M of N such that $M \rhd u_1 \neq \varnothing$ and $M \rhd u_2 \neq \varnothing$. By applying Prop. 17, $\alpha(M)$ is also a reachable marking of N_α. By applying item 3 of Prop. 15, $\alpha(M) \rhd u_1 = \alpha(M \rhd u_1)$ and $\alpha(M) \rhd u_2 = \alpha(M \rhd u_2)$. Given that $M \neq \varnothing \iff \alpha(M) \neq \varnothing$, and because $M \rhd u_1 \neq \varnothing$ and $M \rhd u_2 \neq \varnothing$, it follows that $\alpha(M) \rhd u_1 \neq \varnothing$ and $\alpha(M) \rhd u_2 \neq \varnothing$; thus, u_1 and u_2 are concurrent in N_α. Item 2: by contraposition of item 1. □

Hence, by performing reachability analysis on N_α rather than N, one can efficiently obtain (approximate yet valuable) information about the sequential and concurrent units of N. Experiments conducted with 561 LOTOS specifications [20, Sect. 5] have shown that the analyses on N_α gave the same results as the analyses on N in all cases but one, in which a loss of precision equal to 4% was observed; moreover, all analyses on N_α took less than one second each, whereas the analyses on N were much longer (up to one hour and 41 minutes).

6. Compact Marking Encodings for Unit-Safe NUPNs

6.1. Five Encoding Schemes for NUPN Markings

It is well known that the safeness property of Petri nets allows to optimize the encoding of reachable markings by keeping, for each place, a single bit rather

than an integer number. Therefore, each marking of a safe Petri net with m places is usually represented, in explicit-state verification, by a bit string with m bits, and in symbolic verification, by a BDD (*Binary Decision Diagram*) with m Boolean variables [34] [35]. In the sequel, this linear encoding will be called *scheme (a)* and used as a reference point in future comparisons.

Finding more compact encodings of reachable markings is relevant for, at least, two reasons. From a theoretical point of view, if B bits suffice to encode the markings of a net having m places ($B < m$), then the reachability graph has at most 2^B (rather than 2^m) reachable markings. From a practical point of view, using B bits rather than m allows memory savings in both explicit-state and symbolic verification; moreover, BDD packages support only a limited number of Boolean variables (from 2^9 to 2^{32}, but 2^{16} in most cases [36, Table 3]), so that reducing the number of Boolean variables may enable larger nets to be verified using existing symbolic tools.

The unit-safeness property of NUPNs allows to further optimize marking representation by taking into account all linear inequalities (cf. Prop. 10) that constrain the space of reachable markings. Let $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a unit-safe NUPN. Let $n \stackrel{\text{def}}{=} \mathrm{card}\,(\widetilde{U})$ be the number of units having local places, and let $u_1, ..., u_n$ denote these units of $\widetilde{U}$. For each $u_i \in \widetilde{U}$, let $m_i \stackrel{\text{def}}{=} \mathrm{card}\,(\mathsf{places}\,(u_i))$ be the number of local places in u_i. From Prop. 4, we know that any marking M can be represented by its projections $M \rhd u_1, ..., M \rhd u_n$. The result of Prop. 5 (at most one local place in each unit has a token) can be exploited to optimize the representation of these projections. Indeed, each $M \rhd u_i$ is either empty or reduced to a singleton containing one of the m_i local places of u_i, leading to $(m_i + 1)$ different options. It is thus possible [18, Sect. 8.3.1] to store $M \rhd u_i$ using only $\lceil \log_2(m_i + 1) \rceil$ bits (in explicit-state verification) or Boolean variables (in symbolic BDD-based verification), where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x. This optimized representation will be called *scheme (b)*. It was implemented in the CADP toolbox [17], both in the explicit-state setting, as early as version 1.3 (March 1988) of the CÆSAR tool for LOTOS [18, Sect. 8.3.1] and, later, in the symbolic setting, with CÆSAR.BDD tool for NUPNs (July 2004). Based upon this implementation work, we observed that BDD-based verification clearly outperforms the explicit-state approach on data-less models such as NUPNs.

A slightly different encoding was proposed for BDDs in [37, Sect. 4.1], which suggests to use one bit (actually, a Boolean variable) b_i to express whether u_i has a token or not, and $\lceil \log_2(m_i) \rceil$ more bits to store $M \rhd u_i$ when u_i has a token. This encoding will be called *scheme (c)*. It costs $(\lceil \log_2(m_i) \rceil + 1)$ bits or Boolean variables, and is thus slightly less compact than scheme (b) but was found effective in decreasing BDD size globally. In fact, [37] only introduced the bit b_i when $\neg\mathsf{leaf}\,(u_i)$; however, bit b_i is required for both leaf and non-leaf units, as any unit can lose its token for the reasons given in Sect. 4, unless the unit satisfies stronger assumptions. Such assumptions have been later explored in [38, 39], where units are required to be strongly connected state machines, hereby ensuring that a unit u_i always keeps its token and thus can be encoded

using $\lceil \log_2(m_i) \rceil$ bits only. Although this approach removes the need for the extra bit b_i, it is not general enough, as not every Petri net can be partitioned into strongly connected state machines. Consider, for instance, a simple Petri net containing m places sequentially connected by $(m-1)$ transitions (such a net could, e.g., be generated from a CCS term having the form $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_{m-1} \cdot NIL$, each transition being labelled with an action α_i); this net does not have any subset of places that forms a strongly connected state machine, so that the approach of [38, 39] requires m bits (one per place) to encode the markings, while the NUPN model enables all the m places to be contained in a single unit that is unit safe, thus allowing logarithmic encoding with only $\lceil \log_2(m+1) \rceil$ bits using scheme (b) or $\lceil \log_2(m) \rceil + 1$ using scheme (c).

For nested units, further optimization is possible, based on the result of Prop. 6 (when a unit has a token, none of its ascendent or descendent units have a token). In particular, if a unit u_i has both local places and sub-units (i.e., u_i is neither a void nor a leaf unit), its local places and the local places of its sub-units can never have tokens simultaneously; this suggests to use one bit or Boolean variable b_i to encode whether there is or not a token in $\mathsf{places}\,(u_i)$, and to perform *overlap* by using the same bits or Boolean variables to encode the presence of tokens either in $\mathsf{places}\,(u_i)$ or in $\mathsf{places}^*(u_i) \setminus \mathsf{places}\,(u_i)$. Following this approach, the number $\nu(u_i)$ of bits or Boolean variables needed for a non-leaf unit u_i that has local places is given by the recursive definition $\nu(u_i) \stackrel{\text{def}}{=} 1 + \max\left(\lceil \log_2(m_i) \rceil, \sum_{u \in \mathsf{subunits}\,(u_i)} \nu(u)\right)$, where 1 stands for the bit b_i. For a non-leaf unit u_i with no local places (which may be the case of the root unit), the bit b_i is not needed and the definition gets simpler: $\nu(u_i) \stackrel{\text{def}}{=} \sum_{u \in \mathsf{subunits}\,(u_i)} \nu(u)$. For a leaf unit u_j, one can opt either for $\nu(u_j) \stackrel{\text{def}}{=} \lceil \log_2(m_j + 1) \rceil$ if scheme (b) is chosen, or for $\nu(u_j) \stackrel{\text{def}}{=} (\lceil \log_2(m_j) \rceil + 1)$ if scheme (c) is preferred. The table below summarizes these five possible encoding schemes for the reachable markings of a unit-safe NUPN.

scheme	*overlap*	*number of bits or Boolean variables*
(a)	no	$\sum_{i \in \{1,\ldots,n\}} m_i \quad$ (i.e., m)
(b)	no	$\sum_{i \in \{1,\ldots,n\}} \lceil \log_2(m_i + 1) \rceil$
(c)	no	$\sum_{i \in \{1,\ldots,n\}} (\lceil \log_2(m_i) \rceil + 1)$
(b)	yes	$\nu(u_0)$, with $\mathsf{leaf}\,(u_j) \Rightarrow \nu(u_j) = \lceil \log_2(N_j + 1) \rceil$
(c)	yes	$\nu(u_0)$, with $\mathsf{leaf}\,(u_j) \Rightarrow \nu(u_j) = \lceil \log_2(N_j) \rceil + 1$

6.2. Comparative Compactness of Encoding Schemes

To evaluate the merits of these five schemes, we define for each scheme S and each NUPN N having m places and n units, a ratio $\rho \in [0, 1]$ that is the number of bits B used by scheme S to encode the reachable markings of N divided by the number of bits used by scheme (a), the divider being equal to m since scheme (a) is not optimized. This ratio represents a memory cost: the lower the value of ρ, the more compact the encoding provided by S.

Clearly, the value of ρ does not only depend on the number m of places, but also on the number n of units, on the distribution of local places in units, and, when overlap is used, on the shape of the tree of units, so that, for a given value of m, there can be different values of ρ. The minimal value for ρ is obtained when there is a single unit containing all the places, in which case one can perform logarithmic compression, i.e., $\rho = \log_2(m+1)/m$ using scheme (b) or $\rho = (\log_2(m)+1)/m$ using scheme (c). Conversely, the maximal value of ρ can be obtained, for instance, when the net is trivial (i.e., each place belongs to a one-place unit, thus preventing compression), so that $\rho = m/m = 1$. However, beyond the case of trivial nets, the maximal value for ρ can also be obtained when $n < m$: consider, for instance, a NUPN with $n = \lfloor m/2 \rfloor$ units, each of them containing two places (plus, possibly, one more unit containing a single place if m is even); for this NUPN, scheme (b) uses two bits for the two-place units and one bit for the one-place unit, leading to a ratio $\rho = (2 \times \lfloor m/2 \rfloor + 1 \times (m - 2\lfloor m/2 \rfloor))/m = 1$; this maximal ratio can also be achieved when applying scheme (c) to a NUPN with $n = \lfloor m/3 \rfloor$ units, each of them containing three places.

At this point, despite the intuition that optimizing encoding schemes for NUPNs can achieve significant memory savings in practice, the theoretical bounds for ρ are of little help, as the minimal value tends to zero when n increases, and the maximal value is always equal to one. The only option left is to use experimental data.

To this aim, we evaluated the five schemes on a collection of 8778 non-trivial NUPNs derived from "meaningful" (i.e., written by humans, rather than randomly generated) examples, and generated using various software tools available today to produce NUPNs. It is worth stressing the significance of this collection with respect to the literature on Petri nets: for instance, the experimental results of [38, 39] were based on at most 15 benchmarks; the Petriweb[10] collection [40] had 43 benchmarks; the MCC[11] collection, which grows every year, contains 90 parametric benchmarks totalling 949 instances; a recent article [41] mentions an impressive collection of 1976 Petri nets; finally, compared to the initial article on NUPNs [42], which used 3524 benchmarks, the size of the collection used in the present article has more than doubled.

We first studied whether it exists, in practice, an upper bound on ρ, hopefully lower than the theoretical value of one. For scheme (b), with and without overlap, the following observation was found to hold for all the 8778 benchmarks: $\rho \leq 1.292 \log_2((m/n)+1)/(m/n)$, meaning that, in practice, the number of bits used by scheme (b) satisfies $B \leq 1.292\, n \log_2((m/n)+1)$. Similarly, for scheme (c), with and without overlap, the following observation was made: $\rho \leq 1.547 \log_2((m/n)+1)/(m/n)$.

We then considered whether these formulas could be simplified by finding a relation between the two parameters m and n, but the value of n does not

[10] See http://pnrepository.lip6.fr/pweb/models/all/browser.html
[11] See http://pnrepository.lip6.fr/mcc/models/all/browser.html

appear to be a continuous function of m, and we found no obvious lower/upper bounds that could be fruitfully injected in the above formulas — for instance, m/n varies between 0.714 and 25601, with a standard deviation equal to 449. We also examined the functions that map a number of places m to the largest number of bits observed when encoding, with a given scheme S, all benchmarks having m places in our collection, but could not find any exploitable property for these functions.

Finally, we searched for a mean value of ρ that could summarize, in a single percentage, the compactness of each encoding scheme S. Given a collection of NUPN benchmarks N_i (with i ranging between 1 and $k = 8778$), we note m_i the number of places of N_i and B_i the number of bits required by scheme S to encode the reachable markings of N_i. There are two plausible definitions for a mean value of ρ, which can be either defined as the mean value of the ratios computed for each benchmark taken individually, i.e., $\rho' = (\Sigma_{i\in\{1,\ldots,k\}}\ B_i/m_i)/k$, or as the ratio computed for all benchmarks taken as a whole, i.e., $\rho'' = (\Sigma_{i\in\{1,\ldots,k\}}\ B_i)/(\Sigma_{i\in\{1,\ldots,k\}}\ m_i)$.

Because it was derived from concrete examples written by humans, our collection of benchmarks has particular properties. Although the number of places ranges from 2 to 131,216, the population of benchmarks having the same number of places m is not distributed uniformly, but rather according to some inverse exponential or Pareto law (many benchmarks having few places and few benchmarks having many places). To detect statistical bias, we quotiented our collection (noted C) in two ways: first, by putting into the same equivalence class all benchmarks having both the same number of places and the same number of units (the resulting collection is noted Q_{mn}), then by putting into the same equivalence class all benchmarks having the same number of places (the resulting collection is noted Q_m). The table below gives, for these three collections, statistical information on how the number of places is distributed; clearly, a coarser quotienting reduces the preponderance of small benchmarks and gives more influence to large benchmarks.

collection	*benchmarks — or equivalence classes*	*median*	*mean*	*standard dispersion*
C	8778	19	175	2431
Q_{mn}	1848	103	682	5251
Q_m	619	333	1827	8967

The next table gives the values of ρ' and ρ'' computed on the three collections, for all the schemes but schema (a), for which $\rho' = \rho'' = 1$. The number of bits needed to encode an equivalence class of Q_{mn} or Q_m is defined to be the arithmetic mean of the B_i's for all the benchmarks N_i belonging to this class.

scheme	*overlap*	ρ',C	ρ',Q_{mn}	ρ',Q_n	ρ'',C	ρ'',Q_{mn}	ρ'',Q_n
(b)	no	56.95%	42.13%	31.56%	16.70%	13.17%	10.33%
(c)	no	66.10%	48.46%	36.41%	19.34%	15.16%	11.84%
(b)	yes	55.06%	40.27%	30.19%	16.08%	12.70%	10.00%
(c)	yes	63.71%	46.19%	34.74%	18.58%	14.58%	11.45%

The analysis of the relative values given in this table leads to three conclusions: (i) it establishes that optimized encoding schemes provide noticeable savings, whatever the metrics chosen; (ii) it confirms that scheme (b) is uniformly more compact than scheme (c), both with or without overlap; and (iii) it suggests that overlap brings little additional savings (2.4% at most), but this may be an artefact on our current NUPN collection, which contains less hierarchical benchmarks (37%) than communicating automata.

It is quite difficult to provide one absolute value summarizing the compactness that can be expected from optimized encodings: for the best scheme, percentages vary between 10.00% and 55.06%, depending on the metrics chosen, which give different weights to small and large benchmarks. To address this issue, we chose to remove from collection C all benchmarks having strictly less than 36 places, for the reason that the state spaces of such benchmarks can be exhaustively stored on any standard computer nowadays, by using a bit vector of 2^{35} bits, i.e., 4 Gbytes RAM at most. Keeping only the "challenging" benchmarks gives a collection of 2897 NUPNs, on which scheme (b) with overlap achieves $\rho' = 14.05\%$ and $\rho'' = 39.67\%$. For safety, we retain that latter value, which is also close to the percentage (39.35%) reported in our initial article on NUPNs [42].

6.3. Further Encoding Schemes for NUPN Markings

So far, the proposed encoding schemes have been only based upon the hierarchical structure of NUPNs (i.e., places and units) without analyzing the transitions. However, greater savings could be obtained by taking into account the transition relation. Ideally, the most compact encoding could be obtained by firing all transitions to build the set R of reachable markings and then represent each marking by a number coded on $\lceil \log_2(\mathrm{card}\,(R)) \rceil$ bits, but marking-graph construction is precisely the difficult problem that optimized encodings are trying to make more tractable.

It is nevertheless true that a greater compaction could be achieved, at a reasonable cost, by a structural examination of the transitions. For instance, if a leaf unit u in a unit-safe NUPN has m places, if one of these places belongs to the initial marking, and if each transition having one input place in u also has one output place in u (i.e., is conservative with respect to this unit), then u never loses its token in any reachable marking, and can thus be encoded using $\log_2(m)$ rather than $\log_2(m+1)$ bits. Moreover, overlap is perhaps not the only reduction possible and alternative approaches could be investigated, e.g., by precomputing information about units that can execute concurrently [20, Sect. 5].

Finally, the potential impact of nested units for optimizing the transition relation (and not only marking representation) should also be studied. The experimental results could also be expanded by considering, besides the number of Boolean variables, the number of BDD nodes allocated during symbolic marking-graph construction.

Beyond the case of BDDs, it is likely that unit safeness also permits savings when exploring the state space of NUPNs with other kinds of decision diagrams

than BDDs. Of particular interest would be the investigation of MDDs (*Multi-valued Decision Diagrams*) and MTBDDs (*Multi-Terminal BDDs*), which are often deemed superior to BDDs for reachability analysis of Petri nets [43] [44]. Regarding SDDs (*Hierarchical Symbolic Set Diagrams*) [45, 46], discussions with Alexandre Hamez and Yann Thierry-Mieg show that unit safeness provides useful information that can be easily exploited and, especially, allows to encode each unit using only one SDD variable with satisfactory results (see Sect. 7.2).

7. Implementations of NUPN

The NUPN model is actually used for concrete applications. This section reviews the file formats and software tools that implement this model.

7.1. File Formats for NUPN

Two file formats exist for storing NUPN benchmarks:

- The "`.nupn`" format (see Appendix A for details), which is a concise, human-readable textual format implemented in the CADP toolbox [17].
- The "`.pnml`" format (see Appendix B for details), which is an extension of the PNML standard format [47] for Petri nets. In this format, the NUPN-specific information is stored by using the generic extension mechanism provided by the PNML standard, namely by inserting, in each "`.pnml`" file, a "`toolspecific`" section containing information about units.

7.2. Tools for NUPN

At present, the NUPN model is implemented in thirteen tools developed by seven academic institutions in four countries. We first present five tools that can generate NUPNs:

- PNML2NUPN[12] (Paris, France) translates a "`.pnml`" file containing an ordinary, safe P/T net into a "`.nupn`" file using the translation scheme given for the proof of Prop. 11 (i.e., each place in a separate unit). If the P/T net is not ordinary or not safe, the "`.nupn`" file is still generated, but tagged with the special pragmas mentioned in Sect. 11.
- EXP.OPEN[13] (Grenoble, France) can convert a set of finite-state automata that execute concurrently and synchronize as specified by process-calculi operators and/or synchronization vectors into a "`.nupn`" file using the translation scheme given for the proof of Prop. 12.

[12] See `http://pnml.lip6.fr/pnml2nupn`
[13] See `http://cadp.inria.fr/man/exp.open.html`

- CÆSAR[14] (Grenoble, France) translates a (value-passing) LOTOS specification into a hierarchical interpreted Petri net. When invoked with option "`-nupn`", CÆSAR stores in a "`.nupn`" file the (unit-safe by construction) NUPN corresponding to this interpreted Petri net. CÆSAR invokes the CÆSAR.BDD tool (see below) with options "`-concurrent-units`" and "`-dead-transitions`" to detect units that execute simultaneously (this information is useful for data-flow analysis [20, Sect. 5]) and transitions that are not quasi-live in the NUPN (such transitions are not quasi-live also in the interpreted Petri net, and thus can be removed).

- NUPN_INFO[15] (Grenoble, France) implements various transformations on NUPNs: for instance, it can turn a NUPN into a trivial one by erasing the existing unit tree; it can repair an incorrect NUPN by deleting all non-root void units; it also implements the place-fusion abstraction defined in Sect. 5; etc.

- The RERS benchmark generator [48] (Dortmund, Germany and Twente, The Netherlands) generates, given a property expressed as a temporal-logic formula φ, a unit-safe NUPN N (represented using the "`.pnml`" format described in Appendix B) that is a model for φ, i.e., φ yields true when evaluated on N. The complexity of N can be arbitrarily high, so that it can be used to assess the correctness, performance, and scalability of model checkers for Petri nets.

We then present two tools that compute structural and behavioural properties of NUPNs:

- CÆSAR.BDD[16] (Grenoble, France) reads a "`.nupn`" file, checks that the NUPN is well-formed, and performs various actions depending on the command-line options. Option "`-pnml`" implements the inverse functionality of PNML2NUPN by translating the NUPN into a "`.pnml`" file, which embeds a "`toolspecific`" section (see Appendix B). Option "`-mcc`" computes usual structural and behavioural properties and automatically generates a Petri-net description form in LaTeX according to the conventions of the Model Checking Contest. Options "`-concurrent-units`", "`-dead-transitions`", and "`-exclusive-places`" perform forward reachability analysis to obtain accurate information about places, transitions, and units. CÆSAR.BDD relies on BDDs, as implemented by Fabio Somenzi's CUDD software library[17].

- CÆSAR.SDD (Toulouse, France) is an emulation of CÆSAR.BDD written by Alexandre Hamez. Rather than BDDs, CÆSAR.SDD uses a library[18]

[14] See `http://cadp.inria.fr/man/caesar.html`
[15] See `http://cadp.inria.fr/man/nupn_info.html`
[16] See `http://cadp.inria.fr/man/caesar.bdd.html`
[17] See `http://vlsi.colorado.edu/~fabio/CUDD`
[18] See `http://github.com/ahamez/libsdd`

for Hierarchical Set Decision Diagrams (SDDs) and takes advantage of unit safeness to allocate only one SDD variable per unit (instead of one SDD variable per place with ordinary P/T nets). CÆSAR.SDD may perform reachability analysis faster and process large NUPNs that CÆSAR.BDD fails to handle, but the converse was also observed on other benchmarks.

Finally, we present six model checkers that are able to parse the "`toolspecific`" section of PNML files and exploit the additional information provided by NUPNs and unit safeness to deliver superior performance:

- GreatSPN-Meddly[19] (Torino, Italy) [49] is a symbolic model checker based on Decision Diagrams, as implemented in the Meddly library[20].
- ITS-Tools[21] (Paris, France) [50] is a model checker supporting multiple formalisms (translated to an intermediate pivot language) as well as multiple solution engines.
- LoLA[22] (Rostock, Germany) [51] is a model checker based on explicit-state exploration and structural Petri net methods.
- LTSmin[23] (Twente, The Netherlands) [52] is a language-independent model checker that, among other features, implements multi-core algorithms for symbolic model checking.
- PNMC[24] (Toulouse, France) [53] is a Petri Net model checker that uses the same SDD library as CÆSAR.SDD.
- TINA[25] (Toulouse, France) [54] provides a wide range of tools for editing, simulating, and analyzing various extensions of Petri nets and Time Petri nets. The 2018 version of TINA exploits the NUPN information in its two model checkers "tedd" (enumerative) and "sift" (symbolic).

Interestingly, the model checkers supporting NUPNs exhibit good performance: at the 2017 and 2018 editions of Model Checking Contest, these model checkers won, respectively, 11 and 12 medals out of 15 — three medals (gold, silver, and bronze) being awarded in each of the five competition categories (StateSpace, UpperBounds, Reachability, CTL, and LTL). The PNMC tool did not participate in the 2017 and 2018 editions but, for the StateSpace category, won two silver medals in 2014 and 2015, and a bronze medal in 2016.

[19] See `http://www.di.unito.it/~greatspn`
[20] See `http://sourceforge.net/projects/meddly`
[21] See `http://ddd.lip6.fr`
[22] See `http://www.service-technology.org/lola`
[23] See `http://fmt.cs.utwente.nl/tools/ltsmin`
[24] See `http://github.com/ahamez/pnmc`
[25] See `http://projects.laas.fr/tina`

7.3. NUPN Benchmarks

To obtain NUPN benchmarks, one can use PNML2NUPN, which translates any ordinary, safe P/T net into a NUPN, albeit with one place per unit. To better take advantage of NUPN-specific properties, one can write higher-level specifications in LOTOS (or in any language, such as LNT [55], that automatically translates to LOTOS) and generate a (structured) NUPN using CÆSAR.

Such "genuine" NUPNs generated from higher-level specifications are already available in the collection of benchmarks accumulated during the successive editions of the Model Checking Contest[26] [21, 22]. The 2018 version of this collection contains 248 NUPNs, representing 33% of the total number of P/T-net benchmarks.

The NUPN model has also been adopted for the parallel problems of the RERS (Rigorous Examination of Reactive Systems) challenge[27] [56]. The collection of benchmarks and training benchmarks prepared for the 2017 edition of this challenge gathers 35 NUPNs of increasing complexity.

Another collection specifically dedicated to NUPN benchmarks is being developed. The VLPN (*Very Large Petri Nets*) benchmark suite[28] provides a collection of 350 carefully selected, large-size NUPNs given in both "`.nupn`" and "`.pnml`" formats, so as to provide tool developers with challenging problems originating from realistic case studies.

8. Comparison with Related Work

The concept of units for encapsulating Petri-net places belonging to the same sequential process was briefly mentioned, from a process-calculus point of view, in early publications by the author [18, 19, 20] and fully presented in [42] from a Petri-net perspective. The present article extends and subsumes [42].

One classically distinguishes between three different Petri-net classes ranked by increasing conciseness and expressiveness of the models they can describe: elementary nets (the most fundamental class), P/T nets, and high-level nets. In such a classification, NUPNs are above elementary nets because of the concept of hierarchy brought by units, and below high-level nets, the tokens of which may carry data. NUPNs are incomparable to P/T nets, as the latter allow multiple tokens per place but lack hierarchical structure; however, as mentioned above, one can easily convert P/T nets to NUPNs and vice versa.

The literature on Petri nets is so abundant, and so many extensions of Petri nets have already been proposed, that it would be no surprise if the ideas underlying the NUPN model had already been also published elsewhere. However, to the best of our knowledge, it is not the case. Specifically, the following comparisons can be drawn between NUPNs and the various approaches proposed in the literature:

[26] See `http://mcc.lip6.fr/models.php`
[27] See `http://www.rers-challenge.org`
[28] See `http://cadp.inria.fr/resources/vlpn`

8.1. High-level Petri Nets

According to [57], there have been three generations of high-level extensions to Petri nets, successively introducing data, hierarchy, and object orientation. The generation that brought hierarchical extensions to Petri nets [58] [59, 33] [60] [61, 62] was developed independently from our concept of nested units [18, 19], at the same time or slightly later; actually, the need for hierarchy in Petri nets had been recognized long before, together with early extension proposals, e.g., [63, 64] [65] [66] [67, 68] [69, 70]. All these hierarchical extensions differ from NUPNs in several respects:

- The motivation is not the same. The stated objectives of hierarchical extensions are: (i) to remedy a distinct weakness of traditional "flat" Petri nets, which provide no means to represent the structure of real-world systems and tend to become large, complex, and thus difficult to review and maintain, even for small-size systems; (ii) to equip nets with means for abstraction, encapsulation, and information hiding based on hierarchical structuring; (iii) to support top-down development methodologies ("divide and conquer"), which ease the modelling of involved systems by recursively decomposing them into modules of smaller, more manageable complexity; and (iv) to support bottom-up development methodologies ("reuse"), which enable systems to be designed by assembling components. Such hierarchical extensions are primarily intended to human specifiers who model systems using Petri nets, often with the help of diagram editors. On the contrary, NUPNs are not supposed to be produced or read by humans, but automatically generated and analyzed by computer tools, as NUPN was designed as a "machine-to-machine" formalism for increasing the efficiency of verification algorithms.

- The technical details are different. The common concept to all hierarchical Petri net extensions is the notion of *subnet* (also called *component*, *module*, *page*, *submodel*, or *subsystem*). A subnet usually aggregates *common* (also: *elementary*, *normal*, or *ordinary*) nodes, which are places or transitions, and *macro* (also: *abstract*, *substitution*, or *super*) nodes, which are special places or transitions, each of which represents a subnet. A hierarchical Petri net can be translated to a "flat" Petri net by substituting each macro node with its corresponding subnet, in the same way as macro-expansion is performed by text preprocessors. There is usually some notion of *interface*, often achieved by dedicated places or transitions. The NUPN model does not fit at all into this framework. Units are not subnets, as they only contain places (but neither transitions nor arcs), do not provide abstraction, and have no interfaces. Units are not macro-places either, because the sets of units and places are disjoint, and because no unit can be used where a place can (arcs and transitions are totally unrelated to units); moreover, replacing a unit by a single place does not always preserve the crucial unit-safeness property (cf. Prop. 20 above). Translating a NUPN to a "flat" Petri net does not require any kind of substitution (only the

information about units has to be dropped). Finally, some hierarchical Petri net extensions allow certain places (especially, interface places) to be shared between several subnets, whereas such sharing is forbidden by the tree-like hierarchy of the NUPN model, in which each place (directly) belongs to a single unit.

- The intended behavioural semantics is also different. It is often stated that subnets are the Petri-net equivalent for subroutines (i.e., procedures and functions) and modules of programming languages; this is not the case with units, which focus on the concurrent structure of sequential processes running in parallel. In particular, when NUPNs are generated from process calculi, all of which have a built-in construct to define procedures (i.e., by associating an identifier to a given behavioural term so that it can be called multiple times), unit creation does not arise from the procedure calls themselves but from the occurrences of parallel composition operators; said differently, a call to a procedure that is fully sequential will create no unit of its own, unless it occurs as an operand of some parallel composition operator.

- The unit-safeness property, which plays a crucial role in the analysis of NUPNs, has no counterpart in subnets. Even if certain behavioural properties are sometimes defined for subnets (e.g., uniformness, conservativeness, and state-machine property in [33, Sect. 4.1]), such properties are optional and seemingly of secondary importance.

8.2. Nets Within Nets

Nested Petri Nets [71, 72] and *Object Petri Nets* [73, 74, 75] describe Petri nets whose tokens are also Petri nets, thus inducing a multi-level hierarchy of "nets within nets"; in comparison, NUPNs are much simpler, as they only have data-less tokens.

8.3. Nets Decomposed into State Machines

As shown by Prop. 12, NUPNs subsume communicating automata. There have been many prior attempts, since the early 70s, at expressing Petri nets as the parallel composition of state machines. To this aim, various definitions of state-machine components have been proposed, leading to many net classes, namely: *synchronized state-machine nets*, *state-machine decomposable nets* (which contain *live & bounded free-choice nets*), *state-machine allocatable nets*, *proper nets*, *superposed automata nets*, *basic modular nets*, and *medium composable nets* (including *deterministic systems of sequential processes*), which are all[29] summarized and discussed in [23, Sect. 5–8]. Compared to these net classes, NUPNs exhibit the following differences:

[29] We exclude from this list *regular nets* and *constructable Petri nets*, which are not primarily based upon state-machine decomposition; such net classes will be addressed below in Sect. 8.5.

- NUPN units can be recursively nested at an arbitrary depth, whereas all these net classes support a single-level decomposition of Petri nets into state machines.

- NUPN units may lose their tokens, whereas all these net classes are built upon (possibly strongly connected) finite-state machines, each of which always has a token in each reachable marking, meaning that all transitions must be conservative with respect to all state machines, i.e., have the same number (zero or one) of input and output places in each state machine. NUPN transitions are not subject to such requirement.

- NUPN units have disjoint local places, whereas some of these net classes (synchronized state-machine nets, proper nets, and medium composable nets) allow the existence of shared places between state machines; in other words, synchronization in NUPNs is achieved using shared transitions rather than shared places.

8.4. Nets Generated from Process Calculi

The concept of nested units is a distinctive trait of the CÆSAR compiler for LOTOS [18, 19]. The same idea was implicitly present in a later LOTOS compiler, IBM's LOEWE software [76, 77] that translated LOTOS to Extended Finite State Machines, a formalism that inherently represents the concurrent structure that Petri nets without hierarchical extensions cannot express. Noticeably, for other process calculi than LOTOS, nested units have not been used by the translation approaches generating Petri nets from CCS [78] [79, 80, 81] [82] [25, 83, 84] [85, 86] [87, 88], CSP [89], CCS+CSP [90, 91] [92, 93], ACP [1], and OCCAM [94, 95]. We believe, however, that many of these approaches could be easily adapted to produce NUPN-like structured models rather than "flat" nets.

8.5. Nets Described Using Extensions of Regular Expressions

Conversely, there have been studies to represent elementary nets and various classes of Petri nets under the form of algebraic terms, especially usual regular expressions (i.e., sequence, choice, iteration) extended with additional operators expressing the concepts of concurrency (parallel composition, interleaving, synchronization, etc.) [6, Th. 9 and Prop. 10–11] [67, 96, 68, 97, 98] [99].

Perhaps, the closest approach to the NUPN idea is the representation of (a subclass of) one-safe Petri nets as sequential-parallel posets or series-rational languages [100, 101, 102], which can be seen as sets of terms in an algebra having sequential and parallel composition, or as structured programs built using "**while**" loops and "**cobegin**"–"**coend**" parallel constructs.

8.6. Process Calculi with Boxes

Petri Box Calculus [103, 104, 105], *Box Algebra* [106, 107], *Petri Net Algebra* [108, 109], and *Asynchronous Box Calculus* [110, 111] are process calculi

specifically designed so that all process terms of these calculi can be compositionally translated to equivalent Petri-net fragments called *boxes*. At first sight, these boxes may bear some similarity to NUPN units, but the radical differences between both models clearly show that units are not boxes:

- Units enclose places only, whereas boxes are nets and thus contain places as well as transitions.
- Units are just based upon elementary nets, whereas boxes are based upon labelled Petri nets, meaning that additional information must be attached to box places (namely, a three-value attribute: entry, exit, or internal) and to box transitions (namely, actions or multisets of actions belonging to some communication alphabet).
- Regarding structural properties, units only require a proper partitioning of places, whereas boxes lay totally different kinds of constraints, such as: each transition must have at least one input and one output place; a box must have at least one entry and one exit place; entry places have no incoming arcs and exit places have no outgoing arcs; etc.
- Regarding behavioural properties, both units and boxes usually assume that each place has at most one token (with the notable exception of the Asynchronous Box Calculus [110, 111], which extends the box approach to nets that are not one-safe, thus going beyond the capabilities of NUPNs). But units also require the aforementioned, stronger unit-safeness property (which is not mandatory for boxes), whereas boxes require a *cleanness* property, which expresses that tokens should progress from entry to exit places without staying in any of the nonexit places (this property is irrelevant for units, the places of which are not labelled and which can lose their tokens). Also, unit safeness leads to inequality relations (see Prop. 10), whereas box properties are naturally expressed in terms of equality relations (S-invariants) [103, 112, 113].
- Any Petri net generated by the translation of a process term containing parallel composition has several units but only one box. Indeed, when translating a parallel composition $p_1 || p_2$, the two units corresponding to p_1 and p_2 are kept side by side and enclosed into a third unit, whereas the two boxes corresponding to p_1 and p_2 are merged into one single box. Said differently, units remain *after* the translation is complete, whereas boxes only exist *during* the translation.

8.7. Process Calculi with Localities

The various approaches (cited in Sect. 1 and surveyed in [16]) for enhancing process calculi with localities are posterior to NUPNs, with which they share some similar goals, but from which they differ in several respects:

- Units express localities by encapsulating (Petri-net) places, whereas these approaches express localities by attaching location information to (process-calculus) transitions.

- Units are a structural concept, which does not modify the behavioural semantics of Petri nets (i.e., the "token game" rules), whereas locality extensions modify the SOS rules defining the operational semantics of the underlying process calculus (namely, CCS).

- Units keep transition labels unchanged, so that the graph of reachable markings remains compatible and analyzable with the usual bisimulation relations, whereas locality extensions require the definition of ad hoc bisimulations taking into account location information.

9. Conclusion

The NUPN (*Nested-Unit Petri Net*) model is an extension of Petri nets with additional information about locality, concurrency, and hierarchy, based upon the decomposition of Petri nets into recursively nested sequential processes. For thirty years, this model has remained hidden in the internals of the CÆSAR compiler for LOTOS [18, 19, 20]. With the advent of the Model Checking Contest, it became manifest that NUPN could be of interest to a broader community, as this model combines three major advantages:

- *It is easy to generate when Petri nets are produced from higher-level, structured descriptions.* This can be seen, e.g., on three main types of such descriptions. First, in the case of communicating automata, each automaton directly corresponds to a NUPN unit. Second, in the case of process calculi, the parallel composition operators determine NUPN units that are unit safe by construction; straightforward optimizations help to reduce the depth of unit nesting according to the associativity property of parallel composition; such an approach is implemented in the CÆSAR compiler. Third, in the case of high-level Petri nets, we believe that existing unfolding algorithms could be easily modified to retain in NUPN units all hierarchy-related information that is usually lost when generating "flat" unfolded Petri nets.

- *It allows significant improvements in state-space exploration and verification of behavioural properties.* As explained in Sect. 6, the unit-safeness property permits logarithmic savings in the encoding of markings, both in explicit-state and symbolic settings. Our longstanding observations with the CÆSAR compiler, bolstered by recent experimental results, confirm the real benefits of this approach in terms of performance and scalability.

- *It is not a disruptive extension that requires major overhaul in software tools.* Adding support for NUPN in an existing Petri-net tool only requires limited changes, namely: (i) being able to read NUPN information, which is easy if the tool already embeds a PNML parser, and (ii) taking advantage of the NUPN information to optimize the encoding of markings. The implementation of transition firings can remain unchanged, unless one uses NUPN information to perform extra (e.g., partial-order) reductions.

As regards future research directions, we believe that the NUPN model raises a number of interesting problems:

1. *Is there an algorithm to determine if certain NUPNs are unit safe without building their marking reachability graph?* Prop. 8 gives some necessary conditions for unit safeness concerning, e.g., the initial marking or the input/output places and quasi-liveness of particular transitions, but having a more general, efficient decision procedure or, at least, sufficient conditions would be desirable.
2. *What is the best algorithmic approach to compute behavioural properties of a NUPN, such as deadlock freeness, quasi-liveness, etc.?* Do NUPNs allow algorithms with a lower complexity than the ones known for safe elementary nets [114] [115], such as those verification algorithms dedicated to hierarchical state machines [116, 30, 31, 117, 118, 119, 120, 121] or hierarchical Petri nets [122, 123] [124, 125, 126, 127]? Also, what would be the most efficient technique for handling NUPNs? At present, there are only fragmentary answers to this latter question. Our experiments with two state-space exploration algorithms we implemented for NUPNs indicate that a BDD-based symbolic approach outperforms an explicit-state approach. Other results reported by Alexandre Hamez indicate that SDDs often scale better than BDDs when analyzing NUPNs. The application of other types of decision diagrams (ADDs, DDDs, MDDs, MTBDDs, ZDDs, etc.) to NUPNs remains to be investigated. It is also likely that information about the concurrent structure of NUPNs can be profitably exploited to perform state-space reductions based on partial orders and stubborn sets.
3. *How to optimally translate a given ordinary, safe P/T net to a NUPN?* Prop. 11 shows that such a P/T net can be easily converted to a trivial, unit-safe NUPN by putting each place in a distinct unit, but no algorithmic improvement can be expected from such a simple approach that makes no attempt at discovering the concurrent structure of the net. A better translation should target at reducing the number of units while maximizing the number of places per unit. There have been many publications on how to decompose a Petri net into concurrent state machines; however, the NUPN hierarchy of nested units is likely to raise new challenges compared to prior approaches that merely target a flat composition of state machines.
4. *How does the concept of nested units extend to high-level nets?* The NUPN model is based on elementary nets; yet, nested units were originally introduced not for such "data-less" low-level nets, but for the interpreted Petri nets generated by the CÆSAR compiler as an intermediate model for the translation of LOTOS. It would therefore be interesting to study whether nested units can also be applied to other forms of high-level Petri nets, such as colored nets and predicate/transition nets.
5. *Can nested units support the unbounded creation/destruction of concurrent processes?* The NUPN model and the unit-safeness property have been

designed as an extension of elementary nets, to represent algebraic terms in which processes are launched and terminated dynamically, yet in a finite way, as, e.g., in the term $B_1 \cdot (B_2 \parallel B_3) \cdot B_4$ or in the recursive process $P = B_1 \cdot (B_2 \parallel B_3) \cdot B_4 \cdot P$. However, for algebraic terms not having such a finite-control property, e.g., $P = B_1 \cdot (B_2 \parallel P)$, elementary nets are not sufficient and must be replaced with more expressive formalisms, such as (bounded or unbounded) P/T nets. In such case, the concurrent and hierarchical structure of such multiple-token nets can still be expressed using NUPNs, but the safeness and unit-safeness properties no longer hold and, ideally, should be replaced with other, more general flow relations.

Acknowledgements

The present article was written under the aegis of the Alexander-von-Humboldt foundation. Damien Bergamini implemented the first version of CÆSAR.BDD in 2004. Frédéric Lang, Lian Apostol, and the anonymous reviewers provided valuable comments about this article. Alexander Graf-Brill simplified the counterexample given for Prop. 20. The author is also grateful to Alexandre Hamez, Lom Messan Hillah, and Fabrice Kordon for their support in making the NUPN model broadly available, and to Rob van Glabbeek, Holger Hermanns, Hernan Ponce de Leon, Yann Thierry Mieg, and Jaco van de Pol for their insightful comments and interest in the NUPN model. This article is dedicated to the memory of Oded Maler.

Appendix A. The ".nupn" File Format

The CADP toolbox [17] provides a textual format[30] for storing NUPNs in files that are assumed to have the "`.nupn`" extension. This format was designed to be concise, easy to produce and to parse by programs, and also readable by humans. Here is a small commented example:

`!creator caesar`	*The NUPN was created by the CÆSAR tool*
`!unit_safe`	*The creator tool warrants that unit safeness holds*
`places #5 0...4`	*There are 5 places numbered from 0 to 4*
`initial place 0`	*The initial marking contains only place 0*
`units #3 0...2`	*There are 3 units numbered from 0 to 2*
`root unit 0`	*The root unit is unit 0*
`U0 #1 0...0 #2 1 2`	*Unit 0 contains 1 place (0) and 2 sub-units (1, 2)*
`U1 #2 1...2 #0`	*Unit 1 contains 2 places (1, 2) and no sub-unit*
`U2 #2 3...4 #0`	*Unit 2 contains 2 places (3, 4) and no sub-unit*
`transitions #3 0...2`	*There are 3 transitions numbered from 0 to 2*
`T0 #1 0 #2 1 3`	*Trans. 0 has 1 input place (0) and 2 output places (1, 3)*
`T1 #1 1 #1 2`	*Trans. 1 has 1 input place (1) and 1 output place (2)*
`T2 #1 3 #1 4`	*Trans. 2 has 1 input place (3) and 1 output place (4)*

[30] The definition is available from `http://cadp.inria.fr/man/caesar.bdd.html`

Non-ordinary and/or non-safe P/T nets can be represented in this format by erasing information about arc multiplicity and token counts in the initial marking. To this aim, the ".nupn" format provides pragmas (namely, `!multiple_arcs` and `!multiple_initial_tokens`) to retain part of the erased information, so as to preserve a few behavioural properties — in addition to the structural ones.

Appendix B. The ".pnml" File Format

The NUPN model is not supported by the PNML standard [47] but there is a simple way to enrich a PNML file with NUPN-related information. This can be done without leaving the PNML framework, by inserting into a "`.pnml`" file, which describes an ordinary, safe P/T net (P, T, F, M_0), a "`toolspecific`" section that adds the description of $(U, u_0, \sqsubseteq, \mathsf{unit})$. This is the approach followed for the Model Checking Contest, which has specified the format of such "`toolspecific`" section in natural language, XSD (XML Schema Definition), DTD (Document Type Definition), RNC (RELAX NG Compact Syntax), and RMG (RELAX NG XML Syntax)[31]. Here is the "`toolspecific`" section corresponding to the NUPN example of Appendix A:

```
<toolspecific tool="nupn" version="1.1">
   <size places="5" transitions="3" arcs="7"/>
   <structure units="3" root="u0" safe="true">
      <unit id="u0">
         <places>p0</places>
         <subunits>u1 u2</subunits>
      </unit>
      <unit id="u1">
         <places>p1 p2</places>
         <subunits/>
      </unit>
      <unit id="u2">
         <places>p3 p4</places>
         <subunits/>
      </unit>
   </structure>
</toolspecific>
```

References

[1] R. J. van Glabbeek, F. W. Vaandrager, Petri Net Models for Algebraic Theories of Concurrency, in: J. W. de Bakker, A. J. Nijman, P. C. Treleaven (Eds.), Proceedings of the 1st International Conference on Parallel Architectures and Languages Europe (PARLE'87), Volume II, Eind-

[31] These definitions are available from `http://mcc.lip6.fr/nupn.php`

hoven, The Netherlands, Vol. 259 of Lecture Notes in Computer Science, Springer, 1987, pp. 224–242.

[2] R. Milner, A Calculus of Communicating Systems, Vol. 92 of Lecture Notes in Computer Science, Springer, 1980.

[3] J. A. Bergstra, J. W. Klop, Process Algebra for Synchronous Communication, Information and Computation 60 (1984) 109–137.

[4] S. D. Brookes, C. A. R. Hoare, A. W. Roscoe, A Theory of Communicating Sequential Processes, J. ACM 31 (3) (1984) 560–599.

[5] ISO/IEC, LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, International Standard 8807, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Geneva (Sep. 1989).

[6] A. W. Mazurkiewicz, Trace Theory, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties, Bad Honnef, Germany, Vol. 255 of Lecture Notes in Computer Science, Springer, 1986, pp. 279–324.

[7] G. Winskel, Event Structures, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties, Part II, Bad Honnef, Germany, Vol. 255 of Lecture Notes in Computer Science, Springer, 1986, pp. 325–392.

[8] G. Boudol, I. Castellani, M. Hennessy, A. Kiehn, A Theory of Processes with Localities, Formal Aspects of Computing 6 (2) (1994) 165–200.

[9] G. Boudol, I. Castellani, M. Hennessy, A. Kiehn, Observing Localities, in: A. Tarlecki (Ed.), Proceedings of the 16th International Symposium on Mathematical Foundations of Computer Science (MFCS'91), Kazimierz Dolny, Poland, Vol. 520 of Lecture Notes in Computer Science, Springer, 1991, pp. 93–102.

[10] G. Boudol, I. Castellani, M. Hennessy, A. Kiehn, A Theory of Process with Localities, in: R. Cleaveland (Ed.), Proceedings of the 3rd International Conference on Concurrency Theory (CONCUR'92), Stony Brook, NY, USA, Vol. 630 of Lecture Notes in Computer Science, Springer, 1992, pp. 108–122.

[11] I. Castellani, Observing Distribution in Processes, in: A. M. Borzyszkowski, S. Sokolowski (Eds.), Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science (MFCS'93), Gdansk, Poland, Vol. 711 of Lecture Notes in Computer Science, Springer, 1993, pp. 321–331.

[12] G. Boudol, I. Castellani, M. Hennessy, A. Kiehn, Observing Localities, Theoretical Computer Science 114 (1) (1993) 31–61.

[13] I. Castellani, Observing Distribution in Processes: Static and Dynamic Localities, International Journal on Foundations of Computer Science 6 (4) (1995) 353–393.

[14] M. Mukund, M. Nielsen, CCS, Location and Asynchronous Transition Systems, in: R. K. Shyamasundar (Ed.), Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science, New Delhi, India, Vol. 652 of Lecture Notes in Computer Science, Springer, 1992, pp. 328–341.

[15] L. Aceto, A Static View of Localities, Formal Aspects of Computing 6 (2) (1994) 201–222.

[16] I. Castellani, Process Algebras with Localities, in: J. A. Bergstra, A. Ponse, S. A. Smolka (Eds.), Handbook of Process Algebra, North-Holland, 2001, Ch. 15, pp. 945–1045.

[17] H. Garavel, F. Lang, R. Mateescu, W. Serwe, CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes, Springer International Journal on Software Tools for Technology Transfer (STTT) 15 (2) (2013) 89–107.

[18] H. Garavel, Compilation et vérification de programmes LOTOS, Thèse de Doctorat, Université Joseph Fourier (Grenoble) (Nov. 1989).

[19] H. Garavel, J. Sifakis, Compilation and Verification of LOTOS Specifications, in: L. Logrippo, R. L. Probert, H. Ural (Eds.), Proceedings of the 10th IFIP International Symposium on Protocol Specification, Testing and Verification (PSTV'90), Ottawa, Canada, North-Holland, 1990, pp. 379–394.

[20] H. Garavel, W. Serwe, State Space Reduction for Process Algebra Specifications, Theoretical Computer Science 351 (2) (2006) 131–145.

[21] F. Kordon, H. Garavel, L. M. Hillah, E. Paviot-Adet, L. Jezequel, C. Rodríguez, F. Hulin-Hubard, MCC'2015 – The Fifth Model Checking Contest, Transactions on Petri Nets and Other Models of Concurrency XI (2016) 262–273.

[22] F. Kordon, H. Garavel, L. Hillah, E. Paviot-Adet, L. Jezequel, F. Hulin-Hubard, E. Amparore, M. Beccuti, B. Berthomieu, H. Evrard, P. G. Jensen, D. Le Botlan, T. Liebke, J. Meijer, J. Srba, Y. Thierry-Mieg, J. van de Pol, K. Wolf, MCC'2017 – The Seventh Model Checking Contest, Transactions on Petri Nets and Other Models of Concurrency XIII (2018) 181–209.

[23] L. Bernardinello, F. de Cindio, A Survey of Basic Net Models and Modular Net Classes, in: G. Rozenberg (Ed.), Advances in Petri Nets – The DEMON Project, Vol. 609 of Lecture Notes in Computer Science, Springer, 1992, pp. 304–351.

[24] G. Rozenberg, J. Engelfriet, Elementary Net Systems, in: W. Reisig, G. Rozenberg (Eds.), Lectures on Petri Nets I: Basic Models, Vol. 1491 of Lecture Notes in Computer Science, Springer, 1996, pp. 12–121.

[25] P. Degano, R. De Nicola, U. Montanari, A Distributed Operational Semantics for CCS Based on Condition/Event Systems, Acta Informatica 26 (1/2) (1988) 59–91.

[26] G. Memmi, G. Roucairol, Linear Algebra in Net Theory, in: W. Brauer (Ed.), Net Theory and Applications – Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, Germany, Vol. 84 of Lecture Notes in Computer Science, Springer, 1979, pp. 213–223.

[27] K. Lautenbach, Linear Algebraic Techniques for Place/Transition Nets, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties (APN'86, Part I), Bad Honnef, Germany, Vol. 254 of Lecture Notes in Computer Science, Springer, 1986, pp. 142–167.

[28] T. Murata, Petri Nets: Analysis and Applications, Proceedings of the IEEE 77 (4) (1989) 541–580.

[29] D. Harel, Statecharts: A visual formalism for complex systems, Sci. Comput. Programming 8 (3) (1987) 231–274.

[30] R. Alur, M. Yannakakis, Model Checking of Hierarchical State Machines, in: Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering, Lake Buena Vista, Florida, USA, ACM, 1998, pp. 175–188.

[31] R. Alur, S. Kannan, M. Yannakakis, Communicating Hierarchical State Machines, in: J. Wiedermann, P. van Emde Boas, M. Nielsen (Eds.), Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP'99), Prague, Czech Republic, Vol. 1644 of Lecture Notes in Computer Science, Springer, 1999, pp. 169–178.

[32] H. Garavel, M. Sighireanu, On the Introduction of Exceptions in LOTOS, in: R. Gotzhein, J. Bredereke (Eds.), Proceedings of the IFIP Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (FORTE/PSTV'96), Kaiserslautern, Germany, Chapman & Hall, 1996, pp. 469–484.

[33] K. Jensen, Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use – Volume 1, EATCS Monographs on Theoretical Computer Science, Springer, 1992.

[34] K. Hamaguchi, H. Hiraishi, S. Yajima, Design Verification of Asynchronous Sequential Circuits Using Symbolic Model Checking, in: Proceedings of the International Symposium on Logic Synthesis and Microprocessor Architecture, 1992, pp. 84–90.

[35] E. Pastor, O. Roig, J. Cortadella, R. M. Badia, Petri Net Analysis Using Boolean Manipulation, in: R. Valette (Ed.), Proceedings of the 15th International Conference on Application and Theory of Petri Nets (ICATPN'94), Zaragoza, Spain, Vol. 815 of Lecture Notes in Computer Science, Springer, 1994, pp. 416–435.

[36] G. Janssen, A Consumer Report on BDD Packages, in: Proceedings of the 16th Annual Symposium on Integrated Circuits and Systems Design (SBCCI'03), Sao Paulo, Brazil, IEEE Computer Society, 2003, pp. 217–222.

[37] A. Kerbrat, Méthodes symboliques pour la vérification de processus communicants: Etude et mise en œuvre, Thèse de Doctorat, Université Joseph Fourier (Grenoble) (Nov. 1994).

[38] E. Pastor, J. Cortadella, Efficient Encoding Schemes for Symbolic Analysis of Petri Nets, in: P. Dewilde, F. J. Rammig, G. Musgrave (Eds.), Proceedings on the International Conference on Design, Automation and Test in Europe (DATE'98), Paris, France, IEEE Computer Society, 1998, pp. 790–795.

[39] E. Pastor, J. Cortadella, M. A. Peña, Structural Methods to Improve the Symbolic Analysis of Petri Nets, in: S. Donatelli, H. C. M. Kleijn (Eds.), Proceedings of the 20th International Conference Application and Theory of Petri Nets (ICATPN'99), Williamsburg, VA, USA, Vol. 1639 of Lecture Notes in Computer Science, Springer, 1999, pp. 26–45.

[40] R. Goud, K. M. van Hee, R. D. J. Post, J. M. E. M. van der Werf, Petriweb: A repository for petri nets, in: S. Donatelli, P. S. Thiagarajan (Eds.), Proceedings of the 27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN'06), Turku, Finland, Vol. 4024 of Lecture Notes in Computer Science, Springer, 2006, pp. 411–420.

[41] J. Esparza, P. J. Meyer, An SMT-based Approach to Fair Termination Analysis, in: R. Kaivola, T. Wahl (Eds.), Proceedings of the 15th Conference on Formal Methods in Computer-Aided Design (FMCAD'15), Austin, Texas, USA, IEEE, 2015, pp. 49–56.

[42] H. Garavel, Nested-Unit Petri Nets: A Structural Means to Increase Efficiency and Scalability of Verification on Elementary Nets, in: R. R. Devillers, A. Valmari (Eds.), Proceedings of the 36th International Conference on Application and Theory of Petri Nets and Concurrency (PETRI

NETS'15), Brussels, Belgium, Vol. 9115 of Lecture Notes in Computer Science, Springer, 2015, pp. 179–199.

[43] N. Arora, Comparison of Encoding Schemes for Symbolic Model Checking of Bounded Petri Nets, Master thesis (computer science), Iowa State University, Ames, IA, USA, paper 11511 (2010).

[44] G. Ciardo, Y. Zhao, X. Jin, Ten Years of Saturation: A Petri Net Perspective, Transactions on Petri Nets and Other Models of Concurrency 6900 (2012) 51–95.

[45] A. Hamez, Y. Thierry-Mieg, F. Kordon, Building Efficient Model Checkers using Hierarchical Set Decision Diagrams and Automatic Saturation, Fundamenta Informaticae 94 (3–4) (2009) 413–437.

[46] Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, F. Kordon, Hierarchical Set Decision Diagrams and Regular Models, in: S. Kowalewski, A. Philippou (Eds.), Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09), York, UK, Vol. 5505 of Lecture Notes in Computer Science, Springer, 2009, pp. 1–15.

[47] ISO/IEC, High-level Petri Nets – Part 2: Transfer Format, International Standard 15909-2:2011, International Organization for Standardization – Information Technology – Systems and Software Engineering, Geneva (2011).

[48] B. Steffen, M. Jasper, J. Meijer, J. van de Pol, Property-Preserving Generation of Tailored Benchmark Petri Nets, in: Proceedings of the 17th International Conference on Application of Concurrency to System Design (ACSD'17), Zaragoza, Spain, IEEE Computer Society, 2017, pp. 1–8.

[49] E. G. Amparore, M. Beccuti, S. Donatelli, (Stochastic) Model Checking in GreatSPN, in: G. Ciardo, E. Kindler (Eds.), Proceedings of the 35th International Conference on the Application and Theory of Petri Nets and Concurrency (PETRI NETS'14), Tunis, Tunisia, Vol. 8489 of Lecture Notes in Computer Science, Springer, 2014, pp. 354–363.

[50] Y. Thierry-Mieg, Symbolic Model-Checking Using ITS-Tools, in: C. Baier, C. Tinelli (Eds.), Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15), London, UK, Vol. 9035 of Lecture Notes in Computer Science, Springer, 2015, pp. 231–237.

[51] K. Wolf, Running LoLA 2.0 in a Model Checking Competition, Transactions on Petri Nets and Other Models of Concurrency 11 (2016) 274–285.

[52] G. Kant, A. Laarman, J. Meijer, J. van de Pol, S. Blom, T. van Dijk, LTSmin: High-Performance Language-Independent Model Checking, in:

C. Baier, C. Tinelli (Eds.), Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'15), London, UK, Vol. 9035 of Lecture Notes in Computer Science, Springer, 2015, pp. 692–707.

[53] A. Hamez, A Symbolic Model Checker for Petri Nets: pnmc, Transactions on Petri Nets and Other Models of Concurrency 11 (2016) 297–306.

[54] B. Berthomieu, P.-O. Ribet, F. Vernadat, The Tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets, International Journal of Production Research 42 (14) (2004) 2741–2756.

[55] H. Garavel, F. Lang, W. Serwe, From LOTOS to LNT, in: J.-P. Katoen, R. Langerak, A. Rensink (Eds.), ModelEd, TestEd, TrustEd – Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday, Vol. 10500 of Lecture Notes in Computer Science, Springer, 2017, pp. 3–26.

[56] M. Jasper, M. Fecke, B. Steffen, M. Schordan, J. Meijer, J. van de Pol, F. Howar, S. F. Siegel, The RERS 2017 Challenge and Workshop, in: H. Erdogmus, K. Havelund (Eds.), Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software (SPIN'17), Santa Barbara, CA, USA, ACM, 2017, pp. 11–20.

[57] X. He, T. Murata, High-Level Petri Nets – Extensions, Analysis, and Applications, in: W.-K. Chen (Ed.), Electrical Engineering Handbook, Elsevier Academic Press, 2005, pp. 459–476.

[58] G. Dittrich, Specification with Nets – Report on Activities in Connection with "Requirements Capture with Nets", in: F. Pichler, R. Moreno-Díaz (Eds.), Proceedings of the International Workshop on Computer-Aided Systems Theory (EUROCAST'89), Las Palmas, Spain, Vol. 410 of Lecture Notes in Computer Science, Springer, 1989, pp. 111–124.

[59] P. Huber, K. Jensen, R. M. Shapiro, Hierarchies in Coloured Petri Nets, in: G. Rozenberg (Ed.), Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (APN'91), Bonn, Germany, Vol. 483 of Lecture Notes in Computer Science, Springer, 1989, pp. 313–341.

[60] R. Fehling, A Concept of Hierarchical Petri Nets with Building Blocks, in: G. Rozenberg (Ed.), Proceedings of the 12th International Conference on Applications and Theory of Petri Nets (APN'91), Gjern, Denmark, Vol. 674 of Lecture Notes in Computer Science, Springer, 1991, pp. 148–168.

[61] X. He, J. Lee, A Methodology for Constructing Predicate Transition Net Specifications, Software, Practice & Experience 21 (8) (1991) 845–875.

[62] X. He, A Formal Definition of Hierarchical Predicate Transition Nets, in: J. Billington, W. Reisig (Eds.), Proceedings of the 17th International Conference on Application and Theory of Petri Nets (APN'96), Osaka,

Japan, Vol. 1091 of Lecture Notes in Computer Science, Springer, 1996, pp. 212–229.

[63] J. D. Noe, G. J. Nutt, Macro E-Nets for Representation of Parallel Systems, IEEE Transactions on Computers C-22 (8) (1973) 718–727.

[64] J. D. Noe, Nets in Modeling and Simulation, in: W. Brauer (Ed.), Net Theory and Applications – Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, Germany, Vol. 84 of Lecture Notes in Computer Science, Springer, 1980, pp. 347–368.

[65] J. L. Peterson, Petri Nets, ACM Computing Surveys 9 (3) (1977) 223–252.

[66] R. Valette, Analysis of Petri Nets by Stepwise Refinements, Journal of Computer and System Sciences 18 (1) (1979) 35–46.

[67] V. E. Kotov, An Algebra for Parallelism Based on Petri Nets, in: J. Winkowski (Ed.), Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science (MFCS'78), Zakopane, Poland, Vol. 64 of Lecture Notes in Computer Science, Springer, 1978, pp. 39–55.

[68] V. E. Kotov, L. Cherkasova, On Structural Properties of Generalized Processes, in: G. Rozenberg, H. J. Genrich, G. Roucairol (Eds.), Proceedings of the 1983 and 1984 European Workshop on Applications and Theory in Petri Nets (APN'84), Toulouse, France and Aarhus, Denmark, Vol. 188 of Lecture Notes in Computer Science, Springer, 1984, pp. 288–306.

[69] I. Suzuki, T. Murata, Stepwise Refinements of Transitions and Places, in: C. Girault, W. Reisig (Eds.), Proceedings of the 1st and 2nd European Workshops on Application and Theory of Petri Nets (APN'81), Strasbourg, France and Bad Honnef, Germany, Vol. 52 of Informatik-Fachberichte, Springer, 1981, pp. 136–141.

[70] I. Suzuki, T. Murata, A Method for Stepwise Refinement and Abstraction of Petri Nets, Journal of Computer and System Sciences 27 (1) (1983) 51–76.

[71] I. A. Lomazova, Nested Petri Nets – a Formalism for Specification and Verification of Multi-Agent Distributed Systems, Fundamenta Informaticae 43 (1–4) (2000) 195–214.

[72] I. A. Lomazova, Nested Petri Nets: Multi-level and Recursive Systems, Fundamenta Informaticae 47 (3–4) (2001) 283–293.

[73] R. Valk, Petri Nets as Token Objects: An Introduction to Elementary Object Nets, in: J. Desel, M. Silva (Eds.), Proceedings of the 19th International Conference on Application and Theory of Petri Nets (ICATPN'98), Lisbon, Portugal, Vol. 1420 of Lecture Notes in Computer Science, Springer, 1998, pp. 1–25.

[74] R. Valk, Object Petri Nets: Using the Nets-within-Nets Paradigm, in: J. Desel, W. Reisig, G. Rozenberg (Eds.), Lectures on Concurrency and Petri Nets, from the 4th Advanced Course on Petri Nets (ACPN'03), Eichstätt, Germany, Vol. 3098 of Lecture Notes in Computer Science, Springer, 2003, pp. 819–848.

[75] O. Kummer, F. Wienberg, M. Duvigneau, J. Schumacher, M. Köhler, D. Moldt, H. Rölke, R. Valk, An Extensible Editor and Simulation Engine for Petri Nets: Renew, in: J. Cortadella, W. Reisig (Eds.), Proceedings of the 25th International Conference on Applications and Theory of Petri Nets 2004 (ICATPN'04), Bologna, Italy, Vol. 3099 of Lecture Notes in Computer Science, Springer, 2004, pp. 484–493.

[76] G. Karjoth, Implementing LOTOS Specifications by Communicating State Machines, in: R. Cleaveland (Ed.), Proceedings of the 3rd International Conference on Concurrency Theory (CONCUR'92), Stony Brook, NY, USA, Vol. 630 of Lecture Notes in Computer Science, Springer, 1992, pp. 386–400.

[77] G. Karjoth, C. Binding, J. Gustafsson, LOEWE: A LOTOS Engineering Workbench, Computer Networks and ISDN Systems 25 (7) (1993) 853–874.

[78] F. de Cindio, G. de Michelis, L. Pomello, C. Simone, Milner's Communicating Systems and Petri Nets, in: A. Pagnoni, G. Rozenberg (Eds.), Proceedings of the 3rd European Workshop on Applications and Theory of Petri Nets (APN'82), Varenna, Italy, Vol. 66 of Informatik-Fachberichte, Springer, 1982, pp. 40–59.

[79] U. Goltz, A. Mycroft, On the Relationship of CCS and Petri Nets, in: J. Paredaens (Ed.), Proceedings of the 11th Colloquium on Automata, Languages and Programming (ICALP'84), Antwerp, Belgium, Vol. 172 of Lecture Notes in Computer Science, Springer, 1984, pp. 196–208.

[80] U. Goltz, On Representing CCS Programs by Finite Petri Nets, in: M. Chytil, L. Janiga, V. Koubek (Eds.), Proceedings of the 13th Symposium on Mathematical Foundations of Computer Science (MFCS'88), Carlsbad, Czechoslovakia, Vol. 324 of Lecture Notes in Computer Science, Springer, 1988, pp. 339–350.

[81] U. Goltz, CCS and Petri Nets, in: I. Guessarian (Ed.), Semantics of Systems of Concurrent Processes – Proceedings of the LITP Spring School on Theoretical Computer Science, La Roche Posay, France, Vol. 469 of Lecture Notes in Computer Science, Springer, 1990, pp. 334–357.

[82] M. Nielsen, CCS and its Relationship to Net Theory, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties (APN'86, Part II),

Bad Honnef, Germany, Vol. 255 of Lecture Notes in Computer Science, Springer, 1986, pp. 393–415.

[83] N. D. Francesco, U. Montanari, D. Yankelevich, Axiomatizing CCS, Nets and Processes, Science of Computer Programming 21 (3) (1993) 225–261.

[84] U. Montanari, D. Yankelevich, Combining CCS and Petri Nets Via Structural Axioms, Fundamenta Informaticae 20 (1/2/3) (1994) 193–229.

[85] G. Boudol, I. Castellani, Three Equivalent Semantics for CCS, in: I. Guessarian (Ed.), Semantics of Systems of Concurrent Processes – Proceedings of the LITP Spring School on Theoretical Computer Science, La Roche Posay, France, Vol. 469 of Lecture Notes in Computer Science, Springer, 1990, pp. 96–141.

[86] G. Boudol, I. Castellani, Flow Models of Distributed Computations: Three Equivalent Semantics for CCS, Information and Computation 114 (2) (1994) 247–314.

[87] R. Gorrieri, U. Montanari, Distributed Implementation of CCS, in: G. Rozenberg (Ed.), Proceedings of the 12th International Conference on Applications and Theory of Petri Nets (APN'91), Gjern, Denmark, Vol. 674 of Lecture Notes in Computer Science, Springer, 1991, pp. 244–266.

[88] R. Gorrieri, U. Montanari, On the Implementation of Concurrent Calculi in Net Calculi: Two Case Studies, Theoretical Computer Science 141 (1&2) (1995) 195–252.

[89] U. Goltz, W. Reisig, CSP-Programs with Individual Tokens, in: G. Rozenberg, H. J. Genrich, G. Roucairol (Eds.), Proceedings of the 1983 and 1984 European Workshop on Applications and Theory in Petri Nets (APN'84), Toulouse, France and Aarhus, Denmark, Vol. 188 of Lecture Notes in Computer Science, Springer, 1984, pp. 169–196.

[90] E.-R. Olderog, Operational Petri Net Semantics for CCSP, in: G. Rozenberg (Ed.), Proceedings of the 7th European Workshop on Applications and Theory of Petri Nets (APN'87), Oxford, UK, Vol. 266 of Lecture Notes in Computer Science, Springer, 1986, pp. 196–223.

[91] E.-R. Olderog, Nets, Terms, and Formulas: Three Views of Concurrent Processes and Their Relationship, Cambridge University Press, 1991.

[92] D. Taubner, Finite Representations of CCS and TCSP Programs by Automata and Petri Nets, Vol. 369 of Lecture Notes in Computer Science, Springer, 1989.

[93] D. Taubner, Representing CCS Programs by Finite Predicate/Transition Nets, Acta Informatica 27 (6) (1990) 533–565.

[94] J. G. Hall, R. P. Hopkins, O. Botti, F. de Cindio, A Petri Net Semantics of OCCAM2, Technical report 329, University of Newcastle upon Tyne, Computing Laboratory, available from http://www.cs.ncl.ac.uk/publications/trs/papers/329.pdf (Oct. 1991).

[95] R. P. Hopkins, J. G. Hall, O. Botti, A Basic-net Algebra for Program Semantics and its Application to OCCAM, in: G. Rozenberg (Ed.), Advances in Petri Nets – The DEMON Project, Vol. 609 of Lecture Notes in Computer Science, Springer, 1992, pp. 179–214.

[96] L. Cherkasova, V. E. Kotov, Structured nets, in: J. Gruska, M. Chytil (Eds.), Proceedings of the 10th Symposium on Mathematical Foundations of Computer Science (MFCS'81), Strbske Pleso, Czechoslovakia,, Vol. 118 of Lecture Notes in Computer Science, Springer, 1981, pp. 242–251.

[97] L. Cherkasova, V. E. Kotov, Descriptive and Analytical Process Algebras, in: G. Rozenberg (Ed.), Proceedings of the 9th European Workshop on Applications and Theory in Petri Nets, Venice, Italy, Vol. 424 of Lecture Notes in Computer Science, Springer, 1988, pp. 77–104.

[98] L. Cherkasova, V. E. Kotov, An Algebra of Concurrent Non-Deterministic Processes, Theoretical Computer Science 90 (1) (1991) 151–170.

[99] V. K. Garg, M. T. Ragunath, Concurrent Regular Expressions and Their Relationship to Petri Nets, Theoretical Computer Science 96 (2) (1992) 285–304.

[100] K. Lodaya, P. Weil, Series-Parallel Posets: Algebra, Automata and Languages, in: M. Morvan, C. Meinel, D. Krob (Eds.), Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS'98), Paris, France, Vol. 1373 of Lecture Notes in Computer Science, Springer, 1998, pp. 555–565.

[101] K. Lodaya, P. Weil, Series-Parallel Languages and the Bounded-Width Property, Theoretical Computer Science 237 (1-2) (2000) 347–380.

[102] K. Lodaya, D. Ranganayakulu, K. Rangarajan, Hierarchical Structure of 1-Safe Petri Nets, in: V. A. Saraswat (Ed.), Proceedings of the 8th Asian Computing Science Conference on Programming Languages and Distributed Computation (ASIAN'03), Mumbai, India, Vol. 2896 of Lecture Notes in Computer Science, Springer, 2003, pp. 173–187.

[103] E. Best, R. R. Devillers, J. G. Hall, The Box Calculus: A New Causal Algebra with Multi-Label Communication, in: G. Rozenberg (Ed.), Advances in Petri Nets – The DEMON Project, Vol. 609 of Lecture Notes in Computer Science, Springer, 1992, pp. 21–69.

[104] E. Best, R. R. Devillers, J. Esparza, General Refinement and Recursion Operators for the Petri Box Calculus, in: P. Enjalbert, A. Finkel, K. W.

Wagner (Eds.), Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science (STACS'93), Würzburg, Germany, Vol. 665 of Lecture Notes in Computer Science, Springer, 1993, pp. 130–140.

[105] M. Koutny, J. Esparza, E. Best, Operational Semantics for the Petri Box Calculus, in: B. Jonsson, J. Parrow (Eds.), Proceedings of the 5th International Conference on Concurrency Theory (CONCUR'94), Uppsala, Sweden, Vol. 836 of Lecture Notes in Computer Science, Springer, 1994, pp. 210–225.

[106] E. Best, R. R. Devillers, M. Koutny, The Box Algebra – A Model of Nets and Process Expressions, in: S. Donatelli, H. Kleijn (Eds.), Proceedings of the 20th International Conference on Application and Theory of Petri Nets (ICATPN'99), Williamsburg, VA, USA, Vol. 1639 of Lecture Notes in Computer Science, Springer, 1999, pp. 344–363.

[107] E. Best, R. R. Devillers, M. Koutny, The Box Algebra = Petri Nets + Process Expressions, Information and Computation 178 (1) (2002) 44–100.

[108] E. Best, R. R. Devillers, M. Koutny, A Unified Model for Nets and Process Algebras, in: J. Bergstra, A. Ponse, S. Smolka (Eds.), Handbook of Process Algebra, Elsevier, 2001, Ch. 14, pp. 873–944.

[109] E. Best, R. R. Devillers, M. Koutny, Petri Net Algebra, EATCS Monographs in Theoretical Computer Science, Springer, 2001.

[110] R. R. Devillers, H. Klaudel, M. Koutny, F. Pommereau, An Algebra of Non-safe Petri Boxes, in: H. Kirchner, C. Ringeissen (Eds.), Proceedings of the 9th International Conference on Algebraic Methodology and Software Technology (AMAST'02), Saint-Gilles-les-Bains, Reunion Island, France, Vol. 2422 of Lecture Notes in Computer Science, Springer, 2002, pp. 192–207.

[111] R. R. Devillers, H. Klaudel, M. Koutny, F. Pommereau, Asynchronous Box Calculus, Fundamenta Informaticae 54 (4) (2003) 295–344.

[112] R. R. Devillers, Construction of S-invariants and S-components for Refined Petri Boxes, in: M. A. Marsan (Ed.), Proceedings of the 14th International Conference on Application and Theory of Petri Nets (APN'93), Chicago, IL, USA, Vol. 691 of Lecture Notes in Computer Science, Springer, 1993, pp. 242–261.

[113] R. R. Devillers, S-Invariant Analysis of General Recursive Petri Boxes, Acta Informatica 32 (4) (1995) 313–345.

[114] A. Cheng, J. Esparza, J. Palsberg, Complexity Results for 1-Safe Nets, Theoretical Computer Science 147 (1–2) (1995) 117–136.

[115] M. Praveen, K. Lodaya, Parameterized Complexity Results for 1-safe Petri Nets, in: J.-P. Katoen, B. König (Eds.), Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR'11), Aachen, Germany, Vol. 6901 of Lecture Notes in Computer Science, Springer, 2011, pp. 358–372.

[116] R. Alur, Efficient Formal Verification of Hierarchical Descriptions, in: V. Arvind, R. Ramanujam (Eds.), Proceedings of the 18th Conference on the Foundations of Software Technology and Theoretical Computer Science, Chennai, India, Vol. 1530 of Lecture Notes in Computer Science, Springer, 1998, p. 269.

[117] R. Alur, Exploiting Hierarchical Structure for Efficient Formal Verification, in: C. Palamidessi (Ed.), Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00), University Park, PA, USA, Vol. 1877 of Lecture Notes in Computer Science, Springer, 2000, pp. 66–68.

[118] M. Benedikt, P. Godefroid, T. W. Reps, Model checking of unrestricted hierarchical state machines, in: F. Orejas, P. G. Spirakis, J. van Leeuwen (Eds.), Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01), Crete, Greece,, Vol. 2076 of Lecture Notes in Computer Science, Springer, 2001, pp. 652–666.

[119] R. Alur, M. McDougall, Z. Yang, Exploiting Behavioral Hierarchy for Efficient Model Checking, in: E. Brinksma, K. G. Larsen (Eds.), Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02), Copenhagen, Denmark, Vol. 2404 of Lecture Notes in Computer Science, Springer, 2002, pp. 338–342.

[120] R. Alur, Formal Analysis of Hierarchical State Machines, in: N. Dershowitz (Ed.), Verification: Theory and Practice – Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday, Vol. 2772 of Lecture Notes in Computer Science, Springer, 2003, pp. 42–66.

[121] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, M. Yannakakis, Analysis of Recursive State Machines, ACM Transactions on Programming Languages and Systems 27 (4) (2005) 786–818.

[122] M. Notomi, T. Murata, Hierarchically Organized Petri Net State Space for Reachability and Deadlock Analysis, in: V. K. Prasanna, L. H. Canter (Eds.), Proceedings of the 6th International Parallel Processing Symposium, Beverly Hills, CA, USA, IEEE Computer Society, 1992, pp. 616–623.

[123] M. Notomi, T. Murata, Hierarchical Reachability Graph of Bounded Petri Nets for Concurrent-Software Analysis, IEEE Transactions on Software Engineering 20 (5) (1994) 325–336.

[124] P. Buchholz, A Hierarchical View of GCSPNs and Its Impact on Qualitative and Quantitative Analysis, Journal of Parallel and Distributed Computing 15 (3) (1992) 207–224.

[125] P. Buchholz, Hierarchies in Colored GSPNs, in: M. A. Marsan (Ed.), Proceedings of 14th International Conference on Application and Theory of Petri Nets (ICATPN'93), Chicago, IL, USA, Vol. 691 of Lecture Notes in Computer Science, Springer, 1993, pp. 106–125.

[126] P. Buchholz, P. Kemper, Modular State Level Analysis of Distributed Systems Techniques and Tool Support, in: R. Cleaveland (Ed.), Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99), Amsterdam, The Netherlands, Vol. 1579 of Lecture Notes in Computer Science, Springer, 1999, pp. 420–434.

[127] P. Buchholz, P. Kemper, Hierarchical Reachability Graph Generation for Petri Nets, Formal Methods in System Design 21 (3) (2002) 281–315.