Nested-Unit Petri Nets: A Structural Means to Increase Efficiency and Scalability of Verification on Elementary Nets

Hubert Garavel[1,2,3,4]

[1] Inria
[2] Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
[3] CNRS, LIG, F-38000 Grenoble, France
[4] Saarland University, Saarbrücken, Germany
hubert.garavel@inria.fr
http://convecs.inria.fr

Abstract. Petri nets can express concurrency and nondeterminism but not 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 the 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. Six tools already implement the NUPN model, which is also part of the next edition of the Model Checking Contest.

1 Introduction

Process calculi and Petri nets are two major branches of concurrency theory and have been extensively compared from many different viewpoints. Regarding the ways in which the hierarchical structure of complex systems can be formally described, process calculi have features that low-level Petri nets are lacking. This is precisely the issue adressed in the present article, which proposes to extend Petri nets with hierarchical structuring features inspired from process calculi.

Our proposal is rooted in a longstanding effort to develop the comprehensive CADP toolbox [22] for the design and verification of concurrent systems. The toolbox includes an efficient compiler [21,24,23] for LOTOS, a value-passing process calculus standardized by ISO [40]. This compiler translates LOTOS to labelled transition systems using, as an intermediate step, interpreted Petri nets that possess a hierarchical structure reflecting the concurrent structure of the source LOTOS specifications. Actually, the suggestion that the Petri nets generated by the compiler 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 twenty-five years, this concept has been in use, but internally to the CADP toolbox only. Specifically, the LOTOS compiler uses two different types of hierarchically-structured nets: an interpreted Petri net (which comprises variables, expressions, assignments, guards, etc.) 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, initially called BPN (*Basic Petri Net*) until we realized that this acronym was heavily overloaded[5]; for this reason, the acronym was changed to NUPN[6] (*Nested-Unit Petri Net*) in 2013.

Recently, this model found a new application field in the framework of the Model Checking Contest. For the 2014 edition of the contest, the software tools built around the NUPN model helped to correct and complete the descriptions (structural and behavioural properties) of the P/T nets proposed as challenges to the competitors; additionally, six new challenges were automatically derived from realistic process-calculus specifications using the NUPN tools. For the 2015 edition of the contest, NUPN will move from the back- to the front-office and become visible to competitors, as certain challenges will be provided as P/T nets enriched with NUPN information.

The present article is organised as follows. Sec. 2 defines the NUPN model and states its main properties. Sec. 3 does an extensive review of the state of the art to position the NUPN model with respect to related work. Sec. 4 indicates how the representation of markings can be optimized for NUPNs in both explicit-state and symbolic verification settings. Sec. 5 provides an overview of implementation efforts to equip the NUPN model with file formats, software tools, and collections of benchmarks. Finally, Sec. 6 gives concluding remarks and draws open perspectives for future work.

2 Nested-Unit Petri Nets

2.1 Structure

This subsection defines the "structural" aspects of the NUPN model; these correspond to the description of syntax and static semantics for a computer language.

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.

[5] 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.

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

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 $(\forall u_1, u_2 \in U)\ u_1 \sqsupseteq u_2 \overset{\text{def}}{=} u_2 \sqsubseteq u_1$; thus, $\sqsubseteq$ is reflexive, antisymmetric, transitive, and u_0 is the greatest element of U for this relation; intuitively, $u_1 \sqsubseteq u_2$ espresses 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.*

We have chosen to base our definitions on elementary nets rather than P/T nets, the main difference being that elementary nets are ordinary (i.e., all arc weights are equal to one) and usually expected to be safe (i.e., each place can contain at most one token). We however use the terms "places" and "transitions" rather than their counterparts "conditions" and "events" in elementary nets. 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 $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN, and let u, u_1, and u_2 be any three units of U:*

- *$u_1 \sqsubset u_2 \overset{\text{def}}{=} (u_1 \sqsubseteq u_2) \wedge (u_1 \neq u_2)$ is the strict nesting partial order.*
- *$\mathsf{disjoint}\,(u_1, u_2) \overset{\text{def}}{=} (u_1 \not\sqsubseteq u_2) \wedge (u_2 \not\sqsubseteq u_1)$ characterizes pairs of units neither equal nor nested one in the other.*
- *$\mathsf{subunits}^*(u) \overset{\text{def}}{=} \{u' \in U \mid (u' \sqsubset u)\}$ gives all units transitively nested in u.*
- *$\mathsf{subunits}\,(u) \overset{\text{def}}{=} \{u' \in U \mid (u' \sqsubset u) \wedge (\nexists u'' \in U)\ (u' \sqsubset u'') \wedge (u'' \sqsubset u)\}$ gives all units directly nested in u.*
- *$\mathsf{leaf}\,(u) \overset{\text{def}}{=} (\mathsf{subunits}\,(u) = \varnothing)$ characterises the minimal elements of $(U, \sqsubseteq)$.*
- *$\mathsf{places}\,(u) \overset{\text{def}}{=} \{p \in P \mid \mathsf{unit}\,(p) = u\}$ gives all places directly contained in u; these are called the* local places *(or* proper places*) of u.*
- *$\mathsf{places}^*(u) \overset{\text{def}}{=} \{p \in P \mid (\exists u' \in U)\ (u' \sqsubseteq u) \wedge (\mathsf{unit}\,(p) = u')\}$ gives all places transitively contained in u or its sub-units.*
- *$\widetilde{U} \overset{\text{def}}{=} \{u \in U \mid \mathsf{places}\,(u) \neq \varnothing\}$ is the set of all units but u_0 if the root unit has no local place.*

Proposition 1. *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.

2.2 Execution

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.

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 ${}^\bullet t \overset{\text{def}}{=} \{p \in P \mid (p, t) \in F\}$.*
- *The* post-set *of t is the set of places defined as $t^\bullet \overset{\text{def}}{=} \{p \in P \mid (t, p) \in F\}$.*
- *A* marking *M is defined as a set of places ($M \subseteq P$).*
- *A transition t is* enabled *in some marking M iff it satisfies the predicate $\mathsf{enabled}\,(M, t) \overset{\text{def}}{=} ({}^\bullet t \subseteq M)$.*
- *A transition t can* safely fire *from some marking M iff it satisfies the predicate $\mathsf{safe\text{-}fire}\,(M, t) \overset{\text{def}}{=} \mathsf{enabled}\,(M, t) \wedge ((M \setminus {}^\bullet t) \cap t^\bullet = \varnothing)$*
- *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 markings $M_1, M_2, ..., M_n$ such that $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \; ... \xrightarrow{t_n} M_n$.*
- *The NUPN is* safe *(or* one-safe*) 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 and strict-firing rules coincide.*

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 $M \triangleright u \overset{\text{def}}{=} M \cap \mathsf{places}\,(u)$.*

Proposition 2. *Let $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. Any marking M can be expressed as $M = (M \triangleright u_1) \uplus ... \uplus (M \triangleright u_n)$, 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. 1, given that the family $\mathsf{places}\,(u_1)$, ..., $\mathsf{places}\,(u_n)$ is a partition of P.

2.3 Unit Safeness

This subsection introduces the so-called *unit-safeness* property, which does not exist in "classical" Petri nets and 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: $\mathsf{unit\text{-}safe}\,(M) \overset{\text{def}}{=} (\forall p_1, p_2 \in M)\ (p_1 \neq p_2) \Rightarrow \mathsf{disjoint}\,(\mathsf{unit}\,(p_1), \mathsf{unit}\,(p_2))$; that is, all places of a unit-safe marking are contained in disjoint units.*

Proposition 3. *Let $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. For each marking M and unit u, $\mathsf{unit\text{-}safe}\,(M) \Rightarrow card(M \triangleright u) \leq 1$; that is, a unit-safe marking cannot contain two different local places of the same unit.*

Proof. By contradiction. If $\text{card}\,(M \rhd u) > 1$, 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 4. *Let* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. For each marking* M *and units* (u, u')*, one has:* $\mathsf{unit\text{-}safe}\,(M) \wedge (M \rhd u \neq \varnothing) \wedge (u' \sqsubset u \vee u \sqsubset u') \Rightarrow (M \rhd u' = \varnothing)$*; 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 $p \in \mathsf{unit}\,(u)$ and at least one place $p' \in \mathsf{unit}\,(u')$. 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. 4 can be given an intuitive explanation in a process calculus setting. Consider a process term of the form $\boxed{B_1 \;;\; (\;\boxed{B_2}\;||\;\boxed{B_3}\;)\;;\;B_4}$ 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 converse 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 as the condition that all reachable markings are unit safe, which would imply safeness as a particular case of not having more than one token in the same unit.

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 Sec. 6. The following conditions give preliminary, yet useful checks that can be easily performed.

Proposition 5. *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 $\neg\textsf{unit-safe}\,({}^\bullet t)$ or $\neg\textsf{unit-safe}\,(t^\bullet)$ then either $\neg\textsf{unit-safe}\,(M_1)$ or $\neg\textsf{unit-safe}\,(M_2)$; thus, N is not unit safe.

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, which are linear equations.

Proposition 6. *Let $(P, T, F, M_0, U, u_0, \sqsubseteq, \textsf{unit})$ be a safe NUPN. N is unit safe iff any reachable marking M satisfies the following system of inequalities:*

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

where each variable x_p is equal to 1 if place p belongs to M, or 0 otherwise.

Proof. Direct implication: If N is unit safe, then $\textsf{unit-safe}\,(M)$ is true. Prop. 3 ensures all inequalities $(I_{u,u'})$ with $u = u'$, since $\sum_{p \in \textsf{places}\,(u)} x_p = \text{card}\,(M \rhd u)$. Prop. 4 ensures all inequalities $(I_{u,u'})$ with $u \sqsubset u'$, taking into account that $u \neq u' \Rightarrow \sum_{p \in \textsf{places}\,(u) \cup \textsf{places}\,(u')} x_p = \text{card}\,(M \rhd u) + \text{card}\,(M \rhd u')$ and that, from Prop. 4, $(M \rhd u \neq \varnothing) \Rightarrow (M \rhd u' = \varnothing)$ and $(M \rhd u' \neq \varnothing) \Rightarrow (M \rhd u = \varnothing)$, i.e., $(M \rhd u = \varnothing) \vee (M \rhd u' = \varnothing)$, which leads to $\text{card}\,(M \rhd u) + \text{card}\,(M \rhd u') \leq 1$ after applying Prop. 3 twice. Reverse implication: If N is not unit safe, but safe, there exists some reachable marking M such that $\neg\textsf{unit-safe}\,(M)$. Thus, there exist two distinct places p_1 and p_2, and two units $u_1 = \textsf{unit}\,(p_1)$ and $u_2 = \textsf{unit}\,(p_2)$ such that $\neg\textsf{disjoint}\,(u_1, u_2)$, i.e., $u_1 \sqsubseteq u_2$ or $u_2 \sqsubseteq u_1$. In both cases, $\sum_{p \in \textsf{places}\,(u_1) \cup \textsf{places}\,(u_2)} x_p \geq x_{p_1} + x_{p_2} = 2$, 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 $u_2 \sqsubseteq u_1$.

We now study the preservation of NUPN properties under the abstraction (somehow related to the concept of "place fusion" in Coloured Petri Nets [42]) given in [23] to determine which pairs of units can execute concurrently.

Definition 7. *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \textsf{unit})$ be a NUPN. Let N_α denote the 8-tuple $(P', T, F', M'_0, U, u_0, \sqsubseteq, \textsf{unit}')$ derived from N by merging, in each unit u, all the local places of u into a single place local to u. Formally:*

- *Let $P' \subseteq P$ denote the places of N_α after merging: $\textit{card}\,(P') = \textit{card}\,(\widetilde{U})$.*
- *Let α be the abstraction function $P \to P'$ that maps each place of N to its corresponding place in N_α.*
- *Let $F' \subseteq (P' \times T) \cup (T \times P')$ be the finest arc relation that satisfies $(\forall p \in P)\ (\forall t \in T)\ (F(p,t) \Rightarrow F'(\alpha(p), t)) \wedge (F(t,p) \Rightarrow F'(t, \alpha(p)))$.*
- *Let $M'_0 \subseteq P'$ be equal to $\{\alpha(p) \mid p \in M_0\}$.*
- *Let $\textsf{unit}'$ be the function $P' \to U$ defined by $(\forall p \in P)\ \textsf{unit}'(\alpha(p)) = \textsf{unit}\,(p)$.*

Proposition 7. *Let N be a NUPN and let N_α be defined as in Def. 7. Then:*

1. *N_α is also a NUPN.*
2. *If N is safe, N_α is not necessarily safe.*
3. *If N is unit safe, N_α is not necessarily unit safe.*

Proof. For item 1, it easily follows that, because N satisfies all the conditions of Def. 1, N_α also satisfies these conditions. For item 2, consider the following NUPN $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \textsf{unit})$ given by $P = \{p_0, p_1, p_2\}$, $T = \{t\}$, $F = \{(p_0, t), (t, p_1), (t, p_2)\}$, $M_0 = \{p_0\}$, $U = \{u_0, u\}$, $u \sqsubset u_0$, $\textsf{unit}(p_0) = u_0$, $\textsf{unit}(p_1) = u_0$, and $\textsf{unit}(p_2) = u$; N is safe (but not unit-safe). 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. For item 3, consider the same NUPN N as for item 2 but with a different initial marking $M_0 = \{p_1\}$; M_0 is the only reachable marking, so that N is unit safe; for the same reason as with item 2, N_α is not safe, and thus not unit safe (notice that t is not quasi-live, which suggests that preservation could hold under stronger assumptions).

2.4 Expressiveness

This subsection 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 converse implication.

Proposition 8. *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 $(U, u_0, \sqsubseteq, \textsf{unit})$ such that $(P, T, F, M_0, U, u_0, \sqsubseteq, \textsf{unit})$ is a unit-safe NUPN.*

Proof. Let $p_1, ..., p_n$ be the places of P, where $n = \text{card}(P) \geq 1$. 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 $\textsf{unit}$ be the function $P \rightarrow U$ such that $(\forall i \in \{1, ..., n\})\ \textsf{unit}(p_i) = u_i$, meaning that only the root unit u_0 has no local place. Therefore, the NUPN $(P, T, F, M_0, U, u_0, \sqsubseteq, \textsf{unit})$ satisfies all the structural conditions of Def. 1. This NUPN is safe because (P, T, F, M_0) is safe. This NUPN is also unit safe, as any marking $M \subseteq P$ (reachable or not) satisfies $\textsf{unit-safe}(M)$ because, for any two distinct places (p_i, p_j) in M, $\textsf{disjoint}(\textsf{unit}(p_i), \textsf{unit}(p_j)) = \textsf{disjoint}(u_i, u_j) = \textit{false}$ since $i > 0$, $j > 0$, and $i \neq j$.

Notice that this simple encoding (each place in a distinct unit) is not necessarily the only one: there may exist better encodings with fewer units having more local places each; this issue will be discussed in Sec. 6. Also, this encoding justifies why Def. 1 allows the root unit to have no local place, whereas all other units must have at least one — this latter condition preventing the existence of useless "empty" units in a NUPN.

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., [52, p. 557], and [3] for a survey).

Proposition 9. *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. The existence of a state machine decomposition implies that: (i) there exists a collection of place sets $P_1, ..., P_n$ that is a partition[7] of P, and (ii) for each $i \in \{1, ..., n\}$, the subnet $N_i = (P_i, T_i, F_i, M_0 \cap P_i)$ restricted to P_i is a state machine, i.e., $\text{card}\,(M_0 \cap P_i) = 1$ and $(\forall t \in T_i)\ \text{card}\,({}^\bullet t) = \text{card}\,(t^\bullet) = 1$. 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 $P \to U$ 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 state-machine decomposable, thus safe. Due to the state-machine decomposition, each reachable marking M has the form $\{p_1, ..., p_n\}$ where each p_i belongs to P_i; thus, for any two distinct places (p_i, p_j) in M, $\mathsf{disjoint}\,(\mathsf{unit}\,(p_i), \mathsf{unit}\,(p_j)) = \mathsf{disjoint}\,(u_i, u_j) =$ *false* since $i > 0$, $j > 0$, and $i \neq j$; the NUPN is therefore unit safe.

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 [34] and hierarchical communicating state machines [1].
2. For each state machine P_i, these exists a 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 a S-invariant but a boundedness inequality of the form $\sum_{p \in \mathsf{places}\,(u_i)} x_p \leq 1$ (cf. Prop. 6). The possibility of having no token in a unit has proven useful when encoding safe nets as NUPNs (cf. proof of Prop. 8); in practice, it also provides greater modelling flexibility:
   - 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 (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 [40] or the raise of an exception [25]).
   - 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 case is useful to model process terms ending with deadlock, such as "a; **stop**", for which the transition implementing action a needs no output place (the smaller the net, the more efficient the verification).

[7] Notice that some authors do not require $P_1, ..., P_n$ to be pairwise disjoint.

3 Comparison with Related Work

Although 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 prior publications by the author [21,24,23], the present article is the first to specifically cover this topic and provide a broad synthesis from a Petri-net perspective.

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:

1. *High-level Petri Nets*: According to [37], 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 [18] [39,42] [19] [36,35] was developed independently from our concept of nested units [21,24], 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., [55,54] [58] [63] [46,47] [59,60]. All these hierarchical extensions differ from nested units 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. 7 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.
  - Moreover, NUPN units have to satisfy the unit-safeness property, which has no counterpart in subnets. Even if certain properties are sometimes defined for subnets (e.g., uniformness, conservativeness, and state-machine property in [42, Sec. 4.1]), such properties are merely optional.
2. *Nested Petri Nets* [49,50] and *Object Petri Nets* [64,65,48]: Such models 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.
3. *Translation from process calculi to nets*: The concept of nested units is a distinctive trait of the CÆSAR compiler for LOTOS [21,24]. The same idea was implicitly present in a later LOTOS compiler, IBM's LOEWE software

[43,44] 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 [12] [28,26,27] [53] [13,20,51] [9,10] [30,31], CSP [29], CCS+CSP [56,57] [61,62], ACP [66], and OCCAM [32,38]. We believe, however, that many of these approaches could be easily adapted to produce NUPN-like structured models rather than "flat" nets.

4. *Petri Box Calculus* [4], *Box Algebra* [5,8], *Petri Net Algebra* [6,7], and *Asynchronous Box Calculus* [16,17]: These 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 there are enough radical differences between both models to sustain the claim 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 [16,17], 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. 6), whereas box properties are naturally expressed in terms of equality relations (S-invariants) [4,14,15].
   – 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.

4 Efficient Marking Encodings for Unit-Safe NUPNs

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 N places is usually represented, in explicit-state verification, by a bit string with N bits, and in symbolic verification, by a BDD (*Binary Decision Diagram*) with N Boolean variables. In the sequel, this linear encoding will be called *scheme (a)* and used as a reference point in future comparisons.

The unit-safeness property of NUPNs allows to further optimize marking representation by taking into account all linear inequalities (cf. Prop. 6) 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 $n \overset{\text{def}}{=} \text{card}\,(\widetilde{U})$ 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 $N_i \overset{\text{def}}{=} \text{card}\,(\mathsf{places}\,(u_i))$ be the number of local places in u_i.

From Prop. 2, we know that any marking M can be represented by its projections $M \rhd u_1, ..., M \rhd u_n$. The result of Prop. 3 (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 N_i local places of u_i, leading to $(N_i + 1)$ different options. It is thus possible [21, Sec. 8.3.1] to store $M \rhd u_i$ using only $\lceil \log_2(N_i + 1) \rceil$ 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)*.

A slightly different encoding is proposed in [45, Sec. 4.1], which suggests to use one bit or Boolean variable to express whether u_i has a token or not[8], and $\lceil \log_2(N_i) \rceil$ more bits to store $M \rhd u_i$ when u_i has a token. This encoding will be called *scheme (c)*. It is less compact than scheme (b), as it costs $(\lceil \log_2(N_i) \rceil + 1)$ bits or Boolean variables, but is claimed to favour global reduction of BDD size.

For nested units, further optimization is possible, based on the result of Prop. 4 (when a unit has a token, none of its ascendent or descendent units has a token). In particular, if a unit u has sub-units (i.e., $\mathsf{subunits}\,(u) \neq \varnothing$), its local places and the local places of its sub-units can never have tokens simultaneously [45]; this suggests to use one bit or Boolean variable to encode whether there is or not a token in $\mathsf{places}\,(u)$, and to perform *overlapping* by using the same bits or Boolean variables to encode the presence of tokens either in $\mathsf{places}\,(u)$ or in $\mathsf{places}^*(u) \setminus \mathsf{places}\,(u)$. Following this approach, the number $\nu(u_i)$ of bits or Boolean variables needed for a non-leaf unit u_i is given by the recursive definition $\nu(u_i) \overset{\text{def}}{=} 1 + \max\Big(\lceil \log_2(N_i) \rceil, \sum_{u \in \mathsf{subunits}\,(u_i)} \nu(u)\Big)$. For a leaf unit u_j, one can opt either for $\nu(u_j) \overset{\text{def}}{=} \lceil \log_2(N_j + 1) \rceil$ if scheme (b) is chosen, or for $\nu(u_j) \overset{\text{def}}{=} (\lceil \log_2(N_j) \rceil + 1)$ if scheme (c) is preferred.

[8] [45] only introduces this bit when $\neg\mathsf{leaf}\,(u_i)$; however, this bit is required for both leaf and non-leaf units, as any unit can lose its token for the reasons given in Sec. 2.4, unless the unit satisfies stronger assumptions (i.e., is a state machine).

Scheme (b) without overlapping is the approach implemented in the CADP toolbox [22], in both explicit-state setting (CÆSAR tool for LOTOS, when invoked with option "`-e7old`") and symbolic setting (CÆSAR.BDD tool for NUPNs). We observed that BDD-based verification clearly outperforms the explicit-state approach on data-less models such as NUPNs. We assessed these five encoding/overlapping combinations on a collection of 3524 "non-trivial" NUPNs (i.e., such that card $(U) <$ card (P)), with the following results:

scheme	*overlapping*	*number of bits or Boolean variables*	*average size*
(a)	no	$\sum_{i \in \{1,\ldots,n\}} N_i$ (i.e., N)	100.00%
(b)	no	$\sum_{i \in \{1,\ldots,n\}} \lceil \log_2(N_i + 1) \rceil$	40.52%
(c)	no	$\sum_{i \in \{1,\ldots,n\}} (\lceil \log_2(N_i) \rceil + 1)$	46.44%
(b)	yes	$\nu(u_0)$ with $\mathsf{leaf}\,(u_j) \Rightarrow \nu(u_j) = \lceil \log_2(N_j + 1) \rceil$	39.35%
(c)	yes	$\nu(u_0)$ with $\mathsf{leaf}\,(u_j) \Rightarrow \nu(u_j) = \lceil \log_2(N_j) \rceil + 1$	44.94%

It appears that schemes (b) or (c) alone provide a marking-size reduction greater than 50%. Overlapping seems to have a much lower impact (less than 2%) but this may be an artefact on our current NUPN collection, in which communicating automata largely predominate over hierarchical models.

These experimental results could be expanded in at least three directions: (i) besides the number of Boolean variables, the number of BDD nodes allocated could be considered; (ii) overlapping is perhaps not the only reduction possible and better approaches could be investigated, e.g., by precomputing information about units that can execute concurrently [23]; and (iii) the potential impact of nested units for optimizing the transition relation (and not only marking representation) should also be studied.

Beyond the case of BDDs, it is likely that unit safeness could also permit 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 [2] [11]. Regarding SDDs (*Hierarchical Symbolic Set Diagrams*) [33], discussions with Alexandre Hamez led to the finding that unit safeness permits to keep only one SDD variable per unit, with satisfactory results (see Sec. 5.3).

5 Implementation of NUPN

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

5.1 The ".nupn" File Format

The CADP toolbox [22] provides a textual format[9] for storing NUPNs in files that are assumed to have the "`.nupn`" extension. This format was designed to

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

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).*

Non-ordinary and/or non-safe P/T nets can be encoded 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.

5.2 The ".pnml" File Format

The NUPN model is not supported by the PNML standard [41] 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)[10]. Here is the "`toolspecific`" section corresponding to the NUPN example of Sec. 5.1:

```
<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>
```

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

5.3 Tools for NUPN

At present, the NUPN model is implemented in six tools developed at three different academic institutions:

1. CÆSAR[11] 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 model corresponding to this interpreted Petri net. CÆSAR relies on options "`-concurrent-units`" and "`-dead-transitions`" of the CÆSAR.BDD tool (see below) to detect units that execute simultaneously (this information is useful to data-flow analysis [23]) and transitions that are not quasi-live in the NUPN (such transitions are neither quasi-live in the interpreted Petri net, and thus can be removed).
2. PNML2NUPN[12] is a tool developed by Lom-Messan Hillah. It translates a "`.pnml`" file containing an ordinary, safe P/T net into a "`.nupn`" file using the encoding scheme given for the proof of Prop. 8 (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 Sec. 8.
3. EXP.OPEN[13] is a tool developed by Frédéric Lang. Its latest version 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 encoding scheme given for the proof of Prop. 9.
4. CÆSAR.BDD[14] is a tool developed by Damien Bergamini in 2004 and progressively extended since then. It 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 Sec. 5.2). 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; in 2014, the combined use of PNML2NUPN and CÆSAR.BDD enabled the author to detect and correct fourty erroneous properties in the contest's database of models. 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[15].
5. CÆSAR.SDD is an emulation of CÆSAR.BDD written by Alexandre Hamez. Rather than BDDs, CÆSAR.SDD uses A. Hamez's library[16] for Hierarchical Set Decision Diagrams (SDDs) and takes advantage of unit safeness

[11] See `http://cadp.inria.fr/man/caesar.html`
[12] See `http://pnml.lip6.fr/pnml2nupn`
[13] See `http://cadp.inria.fr/man/exp.open.html`
[14] See `http://cadp.inria.fr/man/caesar.bdd.html`
[15] See `http://vlsi.colorado.edu/~fabio/CUDD`
[16] See `https://github.com/ahamez/libsdd`

to allocate only one SDD variable per unit (instead of one SDD variable per place with ordinary P/T nets). Preliminary experiments indicate that CÆSAR.SDD performs reachability analysis faster and can process large NUPNs that CÆSAR.BDD fails to handle.

6. PNMC[17] is a Petri Net model checker developed by Alexandre Hamez. PNMC is also built on the aforementioned SDD library, and is able to parse the "`toolspecific`" section of PNML files to exploit unit safeness. Although PNMC is a very recent tool, it ranked second in the "State Space" category at the 2014 edition of the Model Checking Contest.

5.4 NUPN Benchmarks

To obtain NUPN models, 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, that automatically translates to LOTOS) and generate a (structured) NUPN model using CÆSAR.

Such higher-level generated models are already available from the data base of models for the Model Checking Contest[18]. A present, six NUPNs are in the data base, and more will be added for the 2015 edition of the contest.

We will publish in 2015 the VLPN (*Very Large Petri Nets*) benchmark suite[19], a collection of 350 large-size NUPN models, which will be given in both "`.nupn`" and "`.pnml`" formats, and will provide tool developers with realistic examples and challenging problems.

6 Conclusion

The NUPN (*Nested-Unit Petri Net*) model is an extension of Petri nets with additional information about concurrent structure, i.e., decomposition into hierarchically nested sequential processes. For twenty-five years, this model has remained hidden in the internals of the CÆSAR compiler for LOTOS [21,24,23]. 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;

[17] See `https://github.com/ahamez/pnmc`
[18] See `http://mcc.lip6.fr/models.php`
[19] See `http://cadp.inria.fr/resources/vlpn`

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 above, 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, conforted by recent experimental results obtained on certain benchmarks of the Model Checking Contest, confirm the real benefits of this approach in terms of performance and scalability.
– *It is not a disruptive extension that would require 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 embedds a PNML parser, and (ii) take advantage of the NUPN information to optimize the representation of markings. The implementation of transition firings can remain unchanged, unless one wishes to use 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 issues:

1. *Is there an algorithm to determine if certain NUPNs are unit-safe without building their marking reachability graph?* Prop. 5 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 would be desirable.
2. *What is the best algorithmic approach to compute behavioural properties of a NUPN, such as deadlock freeness, quasi-liveness, etc.?* At present, there are merely fragmentary answers to this question. For instance, we implemented and compared two state-space exploration approaches for NUPN, an explicit-state one and a symbolic one based on BDDs, both with the scheme (b) reduction made possible by the presence of units; clearly, the BDD-based implementation outperforms the explicit-state one. Also, recent results reported by Alexandre Hamez indicate that SDDs often scale better than BDDs when analyzing NUPN models. The application of other types of decision diagrams (ADDs, DDDs, MDDs, MTBDDs, etc.) to NUPN models remains to be investigated. It is also likely that information about the concurrent structure of NUPN models 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?* As mentioned above, such a P/T net can be easily converted to a 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 defined in the present article 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 to represent algebraic terms in which processes are launched and terminated dynamically, yet in a finite way, as in, e.g., "$B_1 ; (B_2 || B_3) ; B_4$" or "**process** $P = B_1 ; (B_2 || B_3) ; B_4 ; P$". However, for algebraic terms not having such a finite-control property, e.g., "**process** $P = B_1 ; (B_2 || P)$", the corresponding Petri nets can still be expressed as NUPNs, but the safeness and unit-safeness properties no longer hold and, ideally, should be replaced with other, more general flow relations.

Acknowledgements

This article was written under the aegis of the Alexander-von-Humboldt foundation. Frédéric Lang provided valuable comments about this article. Alexander Graf-Brill simplified the counterexample given for item 3 of Prop. 7. The author is grateful to Alexandre Hamez, Lom-Messan Hillah, and Fabrice Kordon for their support in making the NUPN model broadly available.

References

1. Alur, R., Yannakakis, M.: Model Checking of Hierarchical State Machines. In: Proc. ACM SIGSOFT Int. Symp. on Foundations of Software Engineering, pp. 175–188. ACM (1998)
2. Arora, N.: Comparison of Encoding Schemes for Symbolic Model Checking of Bounded Petri Nets. Master thesis, paper 11511, Iowa State University, USA (2010)
3. Bernardinello, L., de Cindio, F.: A Survey of Basic Net Models and Modular Net Classes. In: Rozenberg, G. (ed.) Advances in Petri Nets – The DEMON Project. LNCS, vol. 609, pp. 304–351. Springer (1992)
4. Best, E., Devillers, R.R., Hall, J.G.: The Box Calculus: A New Causal Algebra with Multi-label Communication. In: Rozenberg, G. (ed.) Advances in Petri Nets, LNCS, vol. 609, pp. 21–69. Springer (1992)
5. Best, E., Devillers, R.R., Koutny, M.: The Box Algebra – A Model of Nets and Process Expressions. In: Donatelli, S., Kleijn, H. (eds.) ICATPN'99, LNCS, vol. 1639, pp. 344–363. Springer (1999)
6. Best, E., Devillers, R.R., Koutny, M.: A Unified Model for Nets and Process Algebras. In: Handbook of Process Algebra, chap. 14. Elsevier (2001)

7. Best, E., Devillers, R.R., Koutny, M.: Petri Net Algebra. EATCS Monographs in Theoretical Computer Science, Springer (2001)
8. Best, E., Devillers, R.R., Koutny, M.: The Box Algebra = Petri Nets + Process Expressions. Information and Computation 178(1) (2002)
9. Boudol, G., Castellani, I.: Three Equivalent Semantics for CCS. In: Guessarian, I. (ed.) Semantics of Systems of Concurrent Processes, LNCS, vol. 469, pp. 96–141. Springer (1990)
10. Boudol, G., Castellani, I.: Flow Models of Distributed Computations: Three Equivalent Semantics for CCS. Information and Computation 114(2) (1994)
11. Ciardo, G., Zhao, Y., Jin, X.: Ten Years of Saturation: A Petri Net Perspective. Transactions on Petri Nets and Other Models of Concurrency 6900 (2012)
12. de Cindio, F., de Michelis, G., Pomello, L., Simone, C.: Milner's Communicating Systems and Petri Nets. In APN'82, Informatik-Fachberichte, vol. 66. Springer (1982)
13. Degano, P., De Nicola, R., Montanari, U.: A Distributed Operational Semantics for CCS Based on Condition/Event Systems. Acta Inf. 26(1/2) (1988)
14. Devillers, R.R.: Construction of S-invariants and S-components for Refined Petri Boxes. In: Marsan, M.A. (ed.) APN'93, LNCS, vol. 691, pp. 242–261. Springer (1993)
15. Devillers, R.R.: S-Invariant Analysis of General Recursive Petri Boxes. Acta Informatica 32(4) (1995)
16. Devillers, R.R., Klaudel, H., Koutny, M., Pommereau, F.: An Algebra of Non-safe Petri Boxes. In: Kirchner, H., Ringeissen, C. (eds.) AMAST'02, LNCS, vol. 2422, pp. 192–207. Springer (2002)
17. Devillers, R.R., Klaudel, H., Koutny, M., Pommereau, F.: Asynchronous Box Calculus. Fundamenta Informaticae 54(4) (2003)
18. Dittrich, G.: Specification with Nets – Report on Activities in Connection with "Requirements Capture with Nets". In: Pichler, F., Moreno-Díaz, R. (eds.) EUROCAST'89, LNCS, vol. 410. Springer (1989)
19. Fehling, R.: A Concept of Hierarchical Petri Nets with Building Blocks. In: Rozenberg, G. (ed.) APN'91, LNCS, vol. 674, pp. 148–168. Springer (1991)
20. Francesco, N.D., Montanari, U., Yankelevich, D.: Axiomatizing CCS, Nets and Processes. Science of Computer Programming 21(3) (1993)
21. Garavel, H.: Compilation et Vérification de Programmes LOTOS. Doctorate thesis, Université Joseph Fourier (Grenoble) (Nov 1989)
22. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. Springer International Journal on Software Tools for Technology Transfer (STTT) 15(2) (Apr 2013)
23. Garavel, H., Serwe, W.: State Space Reduction for Process Algebra Specifications. Th. Comp. Sci. 351(2) (Feb 2006)
24. Garavel, H., Sifakis, J.: Compilation and Verification of LOTOS Specifications. In: Proc. 10th Int. Symp. on Protocol Specification, Testing and Verification (PSTV'90), North-Holland (Jun 1990)
25. Garavel, H., Sighireanu, M.: On the Introduction of Exceptions in LOTOS. In: Proc. Int. Joint Conf. on Formal Description Techniques and Protocol Specification, Testing, and Verification FORTE/PSTV'96, IFIP, Chapman & Hall (1996)
26. Goltz, U.: On Representing CCS Programs by Finite Petri Nets. In: Chytil, M., Janiga, L., Koubek, V. (eds.) MFCS'88, LNCS, vol. 324, pp. 339–350. Springer (1988)
27. Goltz, U.: CCS and Petri Nets. In: Guessarian, I. (ed.) Semantics of Systems of Concurrent Processes, LNCS, vol. 469, pp. 334–357. Springer (1990)

28. Goltz, U., Mycroft, A.: On the Relationship of CCS and Petri Nets. In: Paredaens, J. (ed.) ICALP'84, LNCS, vol. 172, pp. 196–208. Springer (1984)
29. Goltz, U., Reisig, W.: CSP-Programs with Individual Tokens. In: Rozenberg, G., Genrich, H.J., Roucairol, G. (eds.) APN'84, LNCS, vol. 188, pp. 169–196. Springer (1984)
30. Gorrieri, R., Montanari, U.: Distributed Implementation of CCS. In: Rozenberg, G. (ed.) APN'91, LNCS, vol. 674, pp. 244–266. Springer (1991)
31. Gorrieri, R., Montanari, U.: On the Implementation of Concurrent Calculi in Net Calculi: Two Case Studies. Theoretical Computer Science 141(1&2) (1995)
32. Hall, J.G., Hopkins, R.P., Botti, O., de Cindio, F.: A Petri Net Semantics of OCCAM2. Tech. report 329, Univ. of Newcastle upon Tyne, Computing Lab. (1991)
33. Hamez, A., Thierry-Mieg, Y., Kordon, F.: Building Efficient Model Checkers using Hierarchical Set Decision Diagrams and Automatic Saturation. Fundamenta Informaticae 94(3-4) (2009)
34. Harel, D: Statecharts: A Visual Formalism for Complex Systems. Sci. Comput. Program. 8(3), pp. 231–274, 1987.
35. He, X.: A Formal Definition of Hierarchical Predicate Transition Nets. In: Billington, J., Reisig, W. (eds.) APN'96, LNCS, vol. 1091, pp. 212–229. Springer (1996)
36. He, X., Lee, J.: A Methodology for Constructing Predicate Transition Net Specifications. Software, Practice & Experience 21(8) (1991)
37. He, X., Murata, T.: High-Level Petri Nets – Extensions, Analysis, and Applications. In: Electrical Engineering Handbook. Elsevier Academic Press (2005)
38. Hopkins, R.P., Hall, J.G., Botti, O.: A Basic-net Algebra for Program Semantics and its Application to OCCAM. In: Rozenberg, G. (ed.) Advances in Petri Nets, LNCS, vol. 609, pp. 179–214. Springer (1992)
39. Huber, P., Jensen, K., Shapiro, R.M.: Hierarchies in Coloured Petri Nets. In: Rozenberg, G. (ed.) APN'91, LNCS, vol. 483, pp. 313–341. Springer (1989)
40. ISO/IEC: LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. International Standard ISO/IEC 8807, (1989)
41. ISO/IEC: High-level Petri Nets – Part 2: Transfer Format. International Standard ISO/IEC 15909-2 (2011)
42. Jensen, K.: Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use – Vol. 1. EATCS Monographs on Th. Computer Science, Springer (1992)
43. Karjoth, G.: Implementing LOTOS Specifications by Communicating State Machines. In: Cleaveland, R. (ed.) CONCUR'92, LNCS, vol. 630, pp. 386–400. Springer (Aug 1992)
44. Karjoth, G., Binding, C., Gustafsson, J.: LOEWE: A LOTOS Engineering Workbench. Computer Networks and ISDN Systems 25(7) (1993)
45. Kerbrat, A.: Méthodes Symboliques pour la Vérification de Processus Communicants: Etude et Mise en Œuvre. Doct. thesis, Univ. J. Fourier (Grenoble) (1994)
46. Kotov, V.E.: An Algebra for Parallelism Based on Petri Nets. In: Winkowski, J. (ed.) MFCS'78, LNCS, vol. 64, pp. 39–55. Springer (1978)
47. Kotov, V.E., Cherkasova, L.: On Structural Properties of Generalized Processes. In: Rozenberg, G., Genrich, H.J., Roucairol, G. (eds.) APN'84, LNCS, vol. 188, pp. 288–306. Springer (1984)
48. Kummer, O., Wienberg, F., Duvigneau, M., Schumacher, J., Köhler, M., Moldt, D., Rölke, H., Valk, R.: An Extensible Editor and Simulation Engine for Petri Nets: Renew. In: Cortadella, J., Reisig, W. (eds.) ICATPN'04, LNCS, vol. 3099, pp. 484–493. Springer (2004)
49. Lomazova, I.A.: Nested Petri Nets – a Formalism for Specification and Verification of Multi-Agent Distributed Systems. Fundamenta Informaticae 43(1–4) (2000)

50. Lomazova, I.A.: Nested Petri Nets: Multi-level and Recursive Systems. Fundamenta Informaticae 47(3–4) (2001)
51. Montanari, U., Yankelevich, D.: Combining CCS and Petri Nets Via Structural Axioms. Fundamenta Informaticae 20(1/2/3) (1994)
52. Murata, T.: Petri nets: Analysis and Applications. Proc. of the IEEE 77(4) (1989)
53. Nielsen, M.: CCS and its Relationship to Net Theory. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN'86 (Part I), LNCS, vol. 255, pp. 393–415. Springer (1986)
54. Noe, J.D.: Nets in Modeling and Simulation. In: Brauer, W. (ed.) LNCS, vol. 84, pp. 347–368. Springer (1980)
55. Noe, J.D., Nutt, G.J.: Macro E-Nets for Representation of Parallel Systems. IEEE Transactions on Computers C-22(8) (Aug 1973)
56. Olderog, E.R.: Operational Petri Net Semantics for CCSP. In: Rozenberg, G. (ed.) APN'87, LNCS, vol. 266, pp. 196–223. Springer (1986)
57. Olderog, E.R.: Nets, Terms, and Formulas: Three Views of Concurrent Processes and Their Relationship. Cambridge University Press (1991)
58. Peterson, J.L.: Petri Nets. ACM Computing Surveys 9(3) (1977)
59. Suzuki, I., Murata, T.: Stepwise Refinements of Transitions and Places. In APN'81, Informatik-Fachberichte, vol. 52. Springer (1981)
60. Suzuki, I., Murata, T.: A Method for Stepwise Refinement and Abstraction of Petri Nets. Journal of Computer and System Sciences 27(1) (1983)
61. Taubner, D.: Finite Representations of CCS and TCSP Programs by Automata and Petri Nets, LNCS, vol. 369. Springer (1989)
62. Taubner, D.: Representing CCS Programs by Finite Predicate/Transition Nets. Acta Informatica 27(6) (1989)
63. Valette, R.: Analysis of Petri Nets by Stepwise Refinements. Journal of Computer and System Sciences 18(1) (1979)
64. Valk, R.: Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In: Desel, J., Silva, M. (eds.) ICATPN'98, LNCS, vol. 1420, pp. 1–25. Springer (1998)
65. Valk, R.: Object Petri Nets: Using the Nets-within-Nets Paradigm. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) ACPN'03, LNCS, vol. 3098, pp. 819–848. Springer (2003)
66. van Glabbeek, R.J., Vaandrager, F.W.: Petri Net Models for Algebraic Theories of Concurrency. In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) PARLE'87, LNCS, vol. 259, pp. 224–242. Springer (1987)