Extensible Structural Analysis of Petri Net Product Lines

Elena Gómez-Martínez(✉), Juan de Lara, and Esther Guerra

Modelling and Software Engineering Research Group,
Universidad Autónoma de Madrid, C/ Francisco y Valiente, 11, 28049 Madrid, Spain
{MariaElena.Gomez,Juan.DeLara,Esther.Guerra}@uam.es
http://miso.es

Abstract. Petri nets are a popular formalism to represent concurrent systems. However, their standard form does not offer variability support to model and effectively analyse large sets of variants of a given system. For this purpose, we propose a notion of *product line* of Petri nets to represent a set of similar concurrent systems. The formalization enriches Petri nets with a feature model characterizing the variability of the systems. Moreover, places, transitions and arcs can define presence conditions that determine the subset of system variants they belong to.

To enable an efficient analysis of the set of all net variants, we have lifted several structural analysis methods for Petri nets, to the product line level. Currently, we support the lifted checking of the marked graph, state-machine, and (extended) free-choice properties, which avoids their analysis on each particular net of the product line in isolation.

We demonstrate the feasibility of our proposal using examples in the domain of flexible assembly lines, and introduce an extensible tool infrastructure. The tool is based on Eclipse and FeatureIDE, and permits adding new analysis methods externally. Moreover, we present an evaluation that shows the efficiency gains of our method with respect to an enumerative approach that analyses the properties on every net within the product line separately.

Keywords: Petri nets · Structural analysis · Product lines · Model-driven engineering

1 Introduction

Petri nets are a popular formalism to model concurrent systems [21]. They are widely used due to their rich body of theoretical results enabling analysis, and the plethora of existing supporting tools[1]. However, some scenarios require modelling (possibly a large set of) variants of similar systems. Some examples reported in the literature include the design of the variants of controllers for cyber-physical systems [20], modelling all possible variants of flexible assembly lines [24], or

[1] See for example https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/.

M. Koutny et al. (Eds.): ToPNoC XV, LNCS 12530, pp. 27–49, 2021.
https://doi.org/10.1007/978-3-662-63079-2_2

building families of workflow process models [28]. In these cases, the designer needs to build many variations of a base model. However, if there are many variants, then building, maintaining and analysing this large set of variants becomes challenging.

To facilitate the management of large sets of net variants, we combine Petri nets with software product lines (SPLs) [25,27] to define a notion of Petri net product line (PNPL). This allows modelling the variability space using a feature model, and automatically producing specific Petri nets from given feature configurations [15].

As the main contribution of this paper, we propose lifting some structural analysis techniques of Petri nets to the product line level. This means that we do not need to analyse each Petri net that can be produced from a PNPL separately, but our analysis techniques work on the whole set of Petri nets directly. In this paper, we explain how to lift the analysis of the marked graph, state-machine, and (extended) free-choice [11] properties to PNPLs, but other structural analysis techniques like (extended) asymmetric choice [2] or equal conflict nets [32] can be lifted in a similar way. In the above-mentioned scenarios, these structural analysis techniques can be used to assess soundness of workflow nets [1] by analysing if some/all nets are free choice; to check whether synchronization can interfere with conflicts in a flexible assembly line by analysing if some/all nets are free choice; or checking whether any variant of a controller design can lead to conflicts by checking if all variants are marked graphs.

As a second contribution, we present extensible prototype tool support to model and analyse PNPLs. Our tool is based on Eclipse, and has an extension point to enable contributing further analysis techniques externally. Moreover, we use our tool to evaluate the efficiency of our lifted analysis techniques, which show good improvement compared to enumerating and analysing each Petri net within the product line.

This paper extends our work in [12] by a more comprehensive formalization of the lifting process (Sect. 3.1), the lifted analysis of two additional properties (free-choice and extended free-choice), improved tool support and an expanded evaluation.

In the following, Sect. 2 introduces PNPLs; Sect. 3 proposes lifting the analysis of structural properties to PNPLs and lifts the analysis of the marked graph and (extended) free-choice properties; Sect. 4 presents tool support; Sect. 5 evaluates the efficiency of our lifted analysis; Sect. 6 compares with related research; and Sect. 7 concludes. The Appendix details the lifting of the state-machine property.

2 Petri Net Product Lines

This section defines PNPLs, and how to derive concrete Petri nets via feature configurations. We consider a simple notion of Petri net, as given in Definition 1, but the approach can be easily adapted to other more complex versions.

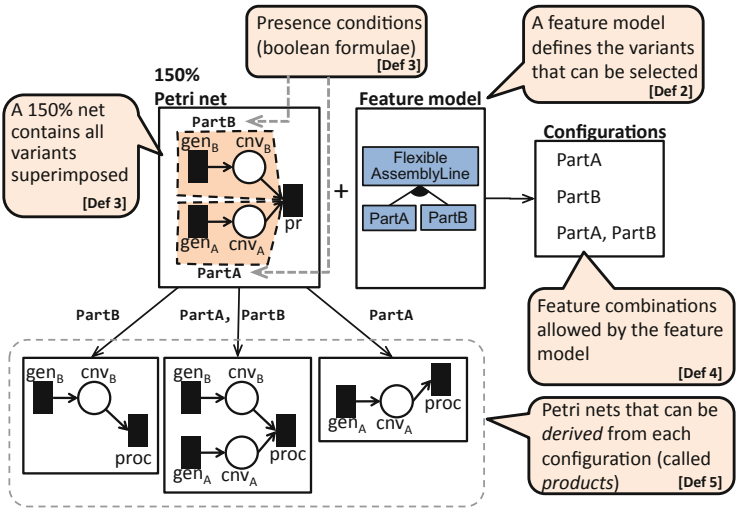

Fig. 1. Ingredients of a PNPL.

Definition 1 (Petri net). *A* Petri net *is a tuple* $PN = (P, T, A)$*, where* P *and* T *are disjoint sets of places and transitions, and* $A \subseteq (P \times T) \cup (T \times P)$ *is the set of arcs connecting either places to transitions or vice versa.*

Given an arc $a \in A$, we use a_0 to refer to its source, and a_1 to refer to its target.

We define a notion of PNPL to support the definition of net variants. Figure 1 shows the concepts that it involves. Firstly, the variability space is represented as a set of features in a feature model. Then, the main idea is to superimpose all net variants within a single net – called 150% Petri net[2] – and annotate its elements with presence conditions (logic formulae over the features in the feature model). Users can retrieve a particular net variant by selecting a subset of the available features. Such a choice is called a configuration. Then, the selected features are substituted by *true* in the presence conditions, and the unselected ones by $false$. This makes each presence condition to evaluate either to true or false. The elements whose presence condition evaluates to false are eliminated from the 150% net, and the remaining elements form the selected net variant. In the following, we define each component of the approach in detail.

PNPLs build on the notion of a feature model that defines the variability space of possible configurations.

Definition 2 (Feature model). *A* feature model $FM = (F, \Psi)$ *consists of a set of propositional variables* $F = \{f_1, ..., f_n\}$ *called* features*, and a propositional formula* Ψ *over the variables in* F*.*

[2] The term 150% model is standard in software product lines. It refers to the fact that a single model contains many variants superimposed.

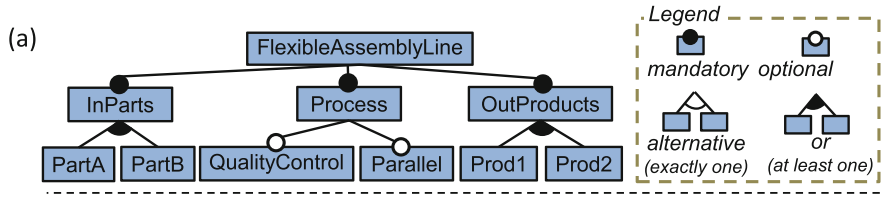

Fig. 2. Feature model for the flexible assembly line using (a) the diagrammatic notation of feature models [15], and (b) Definition 2.

Remark. The propositional formula Ψ in the feature model is used to determine the allowed combinations of feature values (those making the formula true).

Example. As an illustration, we will be using a family of Petri nets describing the behaviour of a flexible assembly line, that is, a production system that can be quickly reconfigured in different set-ups to produce a variety of goods or adapt to customer demands [24]. Here, the problem is to model all such possible configurations in a compact way, and analyse properties of all configurations efficiently. Figure 2(a) shows the feature model using a diagrammatic notation [15], and Fig. 2(b) using Definition 2. Our assembly line can be configured to accept one or two kinds of input parts (PartA, PartB), can optionally have a quality control process (QualityControl) and a parallel conveyor (Parallel), and can produce one or two kinds of products (Prod1, Prod2).

A PNPL is a Petri net whose elements can be annotated with boolean formulae, having as variables the features of the feature model.

Definition 3 (Petri net product line). *A* PNPL $PNL = (FM, PN, \Phi)$ *is made of a feature model* FM*, a Petri net* PN *(called the 150% Petri net), and a mapping* Φ *which consists of pairs* $\langle x, \Phi_x \rangle$ *mapping an element* $x \in P \cup T \cup A$ *to a propositional formula* Φ_x *(called the presence condition (PC) of* x*) over the features in* FM*.*

PNL *is well-formed if* $\forall a \in A : \ (\Phi_a \Rightarrow \Phi_{a_0}) \ \wedge \ (\Phi_a \Rightarrow \Phi_{a_1})$ *is true.*

As noticed, we use an annotative approach to facilitate the analysis. The approach relies on the definition of a 150% Petri net that contains all variants of the PNPL, and the assignment of PCs to its elements. Then, a particular Petri net can be obtained by removing the elements whose PC evaluates to false given a choice (a *configuration*) of feature values. This kind of variability which starts from a maximal description of a set of systems (the 150% Petri net) and deletes elements upon certain conditions is called *negative* [10]. Instead, other approaches to SPLs use *positive* variability, i.e., they start from a minimal description of the systems to which new elements are added depending on the selected features [29]. Our method can also be applied to positive variability approaches as long as they permit deriving a 150% Petri net.

In Definition 3, the well-formedness condition requires the PC of an arc to be stronger than the PC of its source and target elements. This ensures that, if the arc is present in a product Petri net (i.e., a Petri net derived from a configuration by deleting from the 150% net the elements whose PC is false), its source and target elements will be present as well. Definitions 4 and 5 will provide the formal notions of configuration and Petri net derivation.

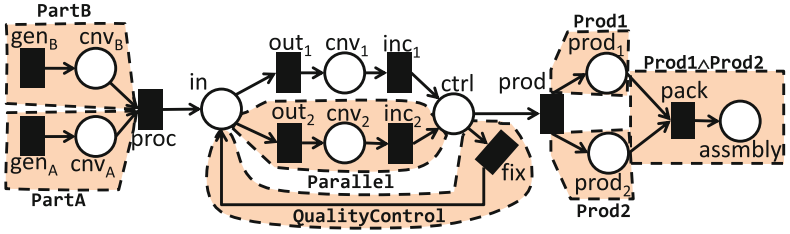

Fig. 3. 150% Petri net with PC annotations, modelling a flexible assembly line.

Example. Figures 2 and 3 show the feature model and the 150% net composing the PNPL of the flexible assembly line. The 150% Petri net in Fig. 3 uses dashed regions as a shortcut to assign the same PC to all the elements in the region. For example, formula PartB in the top-left corner is attached to transition gen_B, to place cnv_B, and to the arcs from/to place cnv_B. If an element does not show an attached PC, then we assume that its PC is true.

The way to obtain a specific *product* Petri net from a PNPL is by selecting a subset of the features in its feature model. This selection is called a *feature configuration*. In the following definition, we use $\Psi[X/true, Y/false]$ to denote the substitution of all variables in X by $true$, and all variables in Y by $false$, in formula Ψ.

Definition 4 (Feature configuration). *A* valid feature configuration $\rho \subseteq F$ *of a PNPL* PNL *with feature model* $FM = (F, \Psi)$ *is a subset of its features satisfying* Ψ*, i.e.,* $\Psi[\rho/true, F \setminus \rho/false]$ *evaluates to true when each* $f \in \rho$ *is substituted by true, and each* $f \in F \setminus \rho$ *is substituted by false. We use* $P(FM) = \{\rho_i\}$ *for the set of all valid feature configurations of* PNL*.*

To improve readability, in the remaining of the paper, feature configurations omit features that are mandatory in any configuration.

Example. Figure 2 admits 36 feature configurations. In all of them, PartA or PartB (inclusive) need to be selected, and similarly, Prod1 or Prod2 need to be selected as well. For instance, some valid configurations are $\rho_0 = \{\mathsf{PartA}, \mathsf{Prod1}\}$, $\rho_1 = \{\mathsf{PartA}, \mathsf{PartB}, \mathsf{Prod1}\}$, and $\rho_2 = \{\mathsf{PartB}, \mathsf{Parallel}, \mathsf{Prod1}\}$. As mentioned above, these configurations would also include features FlexibleAssemblyLine, InParts, Process and OutProducts, but we do not show them as they are mandatory.

Given a feature configuration, we obtain the corresponding product Petri net by removing from the 150% Petri net any element whose PC is false.

Definition 5 (Petri net derivation). *Given a PNPL* $PNL = (FM, PN = (P, T, A), \Phi)$ *and a configuration* $\rho \in P(FM)$*, we derive the net* $PN_\rho = (P_\rho, T_\rho, A_\rho)$ *building each set* $X_\rho \subseteq X$ *(for* $X = \{P, T, A\}$*) as* $\{x \in X \mid \Phi_x[\rho/true, F \setminus \rho/false] = true\}$*. We use* $Prod(PNL) = \{PN_\rho \mid \rho \in P(FM)\}$ *for the set of all derivable nets from* PNL*.*

Example. Figure 4 shows a Petri net derivation example using the feature configuration $\rho_2 = \{\mathsf{PartB}, \mathsf{Parallel}, \mathsf{Prod1}\}$. This way, PN_{ρ_2} contains exactly those elements whose PC evaluates to true.

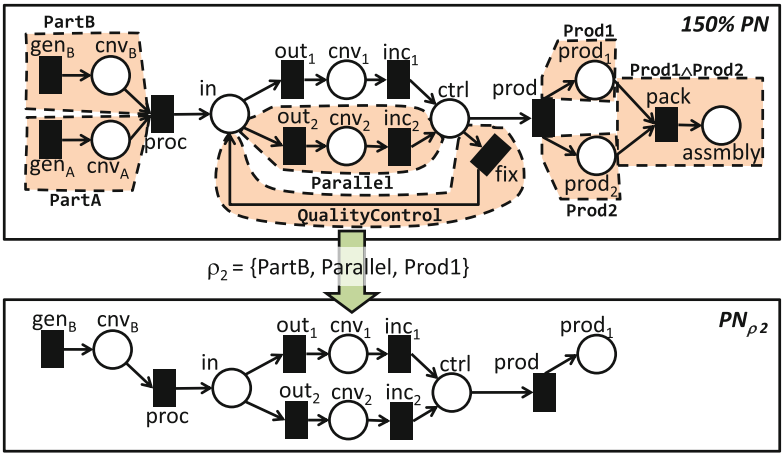

Fig. 4. Petri net derivation example.

To analyse a property in every Petri net that can be derived from a PNPL, a naive method would derive and analyse each product Petri net one by one. However, this can be time-consuming since the number of derivable Petri nets can be exponential on the number of features in the worst case. Hence, the next section proposes a method to lift the analysis of structural properties to the product line level.

3 Structural Analysis of Petri Net Product Lines

This paper is focused on the efficient analysis of structural properties of the set of nets that can be derived from a PNPL. Structural properties depend only on the net topology and are independent of the initial marking [21]. These properties include connectedness, state-machine, marked graph, and (extended) free-choice, among others.

In the following, we first introduce the general scheme and required concepts for the lifted analysis of structural properties (Sect. 3.1). Then, we lift the analysis for the marked graph (Sect. 3.2), free-choice (Sect. 3.3) and extended free-choice properties (Sect. 3.4). The appendix contains the lifting of the state-machine property.

3.1 Lifting the Analysis of Structural Properties

Structural properties look at connectivity patterns of a given Petri net to assert the occurrence of some particular structure. These properties are frequently formulated using first-order logic and auxiliary functions, such as the pre- and post-sets of each place and transition in the net. Figure 5(a) illustrates the formalization of a structural property using a formula P, which is checked on a Petri net PN. We write $PN \models P$ to indicate that property P holds on the Petri net PN.

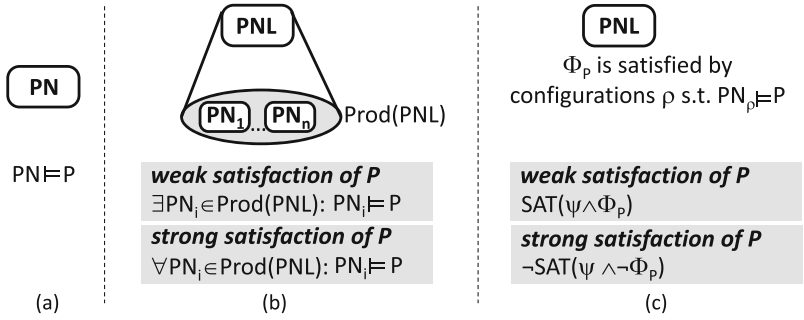

Fig. 5. (a) Checking a structural property P on a Petri net PN. (b) Checking a structural property P on a PNPL PNL using an enumerative approach. (c) Checking a structural property P on a PNPL PNL using a lifted approach.

To check a structural property in the set of nets of a PNPL, we can separately check the property in each derivable net PN_i, as Fig. 5(b) shows. Because we now look at a set of nets, instead of at individual ones, we can distinguish between *weak* and *strong* property satisfaction. Weak satisfaction requires that some product Petri net of the PNPL satisfies the property P (e.g., the marked graph property, cf. Definition 7), while strong satisfaction requires that all product Petri nets satisfy P. The problem with this solution is that checking P on each product net might be too costly as there may be an exponential number of them.

Instead, we propose the solution outlined in Fig. 5(c) to improve the efficiency of the analysis of structural properties for a PNPL. In this solution, we first lift the property P to the product line level. For this purpose, we encode P as a formula Φ_P which takes into account the PCs of the elements in the 150% Petri net, and is satisfied by those configurations ρ such that $PN_\rho \models P$. Then,

we recast the checking of weak/strong property satisfaction as a constraint satisfaction problem. Specifically, if $SAT(\Psi \wedge \Phi_P)$ (with SAT a predicate that holds if the formula is satisfiable, and Ψ the formula of the feature model), then there is some valid configuration which produces a Petri net that satisfies the property P. We can use a constraint solver to obtain a feature configuration that satisfies the formula $\Psi \wedge \Phi_P$. If such a configuration exists, then we have weak property satisfaction.

Conversely, the formula $\neg\Phi_P$ is satisfied by those configurations that produce Petri nets where P does not hold. This way, we have strong satisfiability if $SAT(\Psi \wedge \neg\Phi_P)$ does not hold. This means that no valid configuration (satisfying Ψ) produces a Petri net that does not satisfy P (where $\neg\Phi_P$ holds).

The structural properties that we consider in this paper – state-machine, marked graph, (extended) free-choice – make use of the pre- and post-sets of each place p and each transition t (written ${}^\bullet p$, $p^\bullet$, ${}^\bullet t$ and $t^\bullet$ respectively). Hence, we need to incorporate the PCs within those sets, as Definition 6 shows.

Definition 6 (Lifted pre-/post-sets). *Given a PNPL* $PNL = (FM, PN = (P, T, A), \Phi)$*, for any element* $x \in P \cup T$*, the lifted pre-set of* x *is* ${}^\circ x = \{(y, \Phi_{(y,x)}) \mid (y, x) \in A\}$*, while its lifted post-set is* $x^\circ = \{(y, \Phi_{(x,y)}) \mid (x, y) \in A\}$.

Remark. In the previous definition, we can use the PC of the arc (Φ_a) instead of the PC of its source or target place or transition (Φ_{a_0}, Φ_{a_1}) because, according to Definition 3, in a well-formed PNPL, $\Phi_a \Rightarrow \Phi_{a_0} \wedge \Phi_a \Rightarrow \Phi_{a_1}$, and so, $\Phi_a \wedge \Phi_{a_0} = \Phi_a = \Phi_a \wedge \Phi_{a_1}$.

As an illustration, the following subsections apply this approach to lift the analysis of the marked graph, free-choice and extended free-choice properties. Since the state-machine property is the dual of the marked graph property, we show it in the Appendix. Other structural properties like asymmetric choice can be lifted in a similar way.

3.2 Lifted Analysis of the Marked Graph Property

Firstly, we provide the definition of the marked graph (MG) property. In a MG Petri net, each place has exactly one input transition and one output transition, whereas each transition may have multiple input and output places. Therefore, a MG allows concurrent and synchronization structures with no conflict.

Definition 7 (Marked graph, from [21]). *A Petri net* $PN = (P, T, A)$ *is a* marked graph*, written* $PN \models MG$*, if* $\forall p \in P: \ |{}^\bullet p| = |p^\bullet| = 1$.

We lift this definition of MG to the product line level. Therefore, a PNPL strongly (weakly) satisfies the MG property if all (some of) its derivable nets are MGs.

Definition 8 (Strong and weak MG product line). *A Petri net product line PNL is a strong marked graph iif* $\forall PN_\rho \in Prod(PNL) : PN_\rho \models MG$. *$PNL$ is a weak marked graph iif* $\exists PN_\rho \in Prod(PNL) : PN_\rho \models MG$.

If we can derive from the product line PNL a net that is not a MG, then PNL is not a strong MG product line. In particular, given a feature configuration ρ, a Petri net derivation PN_ρ is not a MG if it has a place p with more than one input transition, more than one output transition, no input transitions, or no output transitions. Therefore, for a PNPL to be a strong MG, we require that the size of the lifted pre-set $^\circ p = \{(t_0, \Phi_{(t_0,p)}), ..., (t_n, \Phi_{(t_n,p)})\}$ and the lifted post-set $p^\circ = \{(t_0, \Phi_{(p,t_0)}), ..., (t_n, \Phi_{(p,t_n)})\}$ of every place p to be one for every possible configuration. For the case of the pre-set, this is the case if the following formula is true:

$$\begin{aligned} \Phi_{^\circ p} \triangleq\ & false \\ & \vee (\Phi_{(t_0,p)} \wedge \neg\Phi_{(t_1,p)} \wedge ... \wedge \neg\Phi_{(t_n,p)}) \\ & \vee (\neg\Phi_{(t_0,p)} \wedge \Phi_{(t_1,p)} \wedge ... \wedge \neg\Phi_{(t_n,p)}) \\ & \vee ... (\neg\Phi_{(t_0,p)} \wedge \neg\Phi_{(t_1,p)} \wedge ... \wedge \Phi_{(t_n,p)}) \end{aligned} \tag{1}$$

The formula is made of a disjunction of conjunctions, where only one term in each conjunction can be true. This ensures that, regardless of the configuration, the pre-set of the place will have size one. The disjunctions start with $false$, so that $\Phi_{^\circ p}$ is false when $^\circ p$ is empty. The terms $\Phi_{(t_i,p)}$ are the PCs in the lifted pre-set of p ($^\circ p$). The formula that ensures that the size of the post-set of a place is one for every possible configuration is defined similarly, but using the terms $\Phi_{(p,t_i)}$ in the lifted post-set of p (p°).

This way, a PNPL includes some Petri net that is a MG if there is a feature configuration ρ such that for every place p in the PNPL:

- p is not in PN_ρ, therefore Φ_p is false; or
- p is in PN_ρ, and therefore $\Phi_{^\circ p}$ and Φ_{p° need to be true.

We can express these conditions as the logical formula in Equation 2.

$$\Phi_{MG} = \wedge_{p \in P}[\neg\Phi_p \vee (\Phi_p \wedge \Phi_{^\circ p} \wedge \Phi_{p^\circ})] \tag{2}$$

If $SAT(\Psi \wedge \Phi_{MG})$ is true, then the PNPL is a weak MG. In such a case, we can use a constraint solver to obtain a feature configuration that satisfies the formula. The Petri net derived using this feature configuration is ensured to be a MG.

Conversely, the feature configurations making the formula Φ_{MG} false yield Petri nets that are not MGs. Hence, a PNPL is a strong MG if $SAT(\Psi \wedge \neg\Phi_{MG})$ is unsatisfiable (i.e., no valid configuration produces a net that is not a MG).

Example. In the PNPL consisting of the feature model in Fig. 2 and the 150% net in Fig. 3, the interesting cases are those for places in and ctrl. In the latter case, any Petri net that contains either both transitions inc_1 and inc_2, or both transitions prod and fix, is not a MG because place ctrl would have either two

incoming or two outgoing arcs. This is the case for the Petri nets derived from configurations that select the features Parallel or QualityControl. Similarly, place in will have two incoming arcs for configurations that select the feature QualityControl, and two outgoing arcs for configurations that select the feature Parallel, resulting in nets that are not MGs. Overall, the example PNPL is not a strong MG product line. However, it is a weak MG product line as, for example, the configuration that only selects features PartA and Prod1 produces a Petri net that is a MG. In practice, if we would like to have no conflicts in the flexible assembly line, we might rule out the problematic variants (i.e., those that are not MGs) by extending the formula Ψ in the feature model. The concrete formula, not reduced, corresponding to the MG property of our example PNPL is the following:

$$
\begin{aligned}
\Phi_{MG} = {} & (\neg PartA \ \vee \ (PartA \ \wedge \ PartA \ \wedge \ PartA)) \\
& \wedge \ (\neg PartB \ \vee \ (PartB \ \wedge \ PartB \ \wedge \ PartB)) \ \wedge \ (QualityControl \ \wedge \ Parallel) \\
& \wedge \ (\neg Parallel \ \vee \ (Parallel \ \wedge \ Parallel \ \wedge \ Parallel)) \\
& \wedge \ (Parallel \ \wedge \ QualityControl) \ \wedge \ (\neg Prod1 \ \vee \ (Prod1 \ \wedge \ Prod1 \\
& \wedge \ (Prod1 \ \wedge \ Prod2))) \ \wedge \ (\neg Prod2 \ \vee \ (Prod2 \ \wedge \ Prod2 \ \wedge \ (Prod1 \ \wedge \ Prod2))) \\
& \wedge \ ((\neg(Prod1 \ \wedge \ Prod2)) \ \vee \ ((Prod1 \ \wedge \ Prod2) \ \wedge \ (Prod1 \ \wedge \ Prod2)))
\end{aligned}
$$

Then, to assess the MG property on the PNPL, we analyse the satisfiability of the conjunction of this formula Φ_{MG} and the formula of the feature model Ψ.

Interestingly, the lifted analysis of the MG property is very similar to the analysis of the state-machine property. A state-machine (SM) is a subclass of Petri net where each transition t has exactly one input place and one output place, while each place may have multiple input and output transitions. This way, analysing whether a PNPL is a weak/strong SM product line is dual to checking the MG property in a PNPL but replacing transitions by places and vice versa (details in the Appendix).

3.3 Lifted Analysis of the Free-Choice Property

Next, we define the free-choice (FC) property. In a FC net, it is not possible to mix choice and synchronization into one routing construct, i.e., either a choice is preceded by a synchronization, or vice versa. FC Petri nets do not have conflicts since every transition has a unique input place.

Definition 9 (Free-choice, from [7]). *A Petri net $PN = (P, T, A)$ is a* free-choice *Petri net, written $PN \models FC$, if for every two transitions t_1 and $t_2 \in T$, $t_1 \neq t_2$: ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing \ \Rightarrow \ |{}^\bullet t_1| = |{}^\bullet t_2| = 1$.*

In other words, a Petri net is FC if every place is either connected to a unique output transition, or all its output transitions have a unique input place. Formally:

$$\forall p \in P : |p^\bullet| = 1 \vee \forall t \in p^\bullet : |{}^\bullet t| = 1 \tag{3}$$

Following the rationale of the previous analysis, we first lift the definition of property FC to the product line level. Hence, a PNPL is a strong (weak) FC if all (some) its derivable nets are FC.

Definition 10 (Strong and weak FC product line). *A Petri net product line PNL is a strong free-choice iif $\forall PN_\rho \in Prod(PNL) : PN_\rho \models FC$. PNL is a weak free-choice iif $\exists PN_\rho \in Prod(PNL) : PN_\rho \models FC$.*

According to Eq. 3, every outgoing arc from a place either is unique, or is the only incoming arc to the target transition of the arc. Therefore, a PNPL includes a FC Petri net if there is a feature configuration ρ such that for every place p in the PNPL:

- p is not in PN_ρ, therefore Φ_p is false; or
- p is in PN_ρ, and therefore either Φ_{p° is true, or for every transition t in the post-set $p^\bullet$:
  - t is not in PN_ρ, therefore Φ_t is false; or
  - t is in PN_ρ, and therefore $\Phi_{\circ t}$ needs to be true.

Equation 4 shows the encoding of these conditions as a logical formula expressing the cases in which a PNPL is a FC product line.

$$\Phi_{FC} = \wedge_{p \in P}[\neg\Phi_p \vee (\Phi_p \wedge (\Phi_{p^\circ} \vee (\wedge_{t \in p^\bullet}[\neg\Phi_t \vee (\Phi_t \wedge \Phi_{\circ t})])))] \tag{4}$$

If $SAT(\Psi \wedge \Phi_{FC})$ is satisfied, then the PNPL is a weak FC product line. On the contrary, configurations leading to nets that are not FC satisfy $\Psi \wedge \neg\Phi_{FC}$. Therefore, a PNPL is a strong FC product line if $SAT(\Psi \wedge \neg\Phi_{FC})$ does not hold.

Example. The sets of conflicting transitions in the PNPL of Fig. 3 (out_1 and out_2; prod and fix) only have one input place, and therefore, the example is a strong FC product line. In practice, this means that our example has a sound design: in no variant of our flexible assembly line, synchronization (i.e., sequencing of part production or movement through the conveyors in the assembly line) interferes with conflicts (i.e., choice of paths for parts in the assembly line).

3.4 Lifted Analysis of the Extended Free-Choice Property

Extended-free choice (EFC) Petri nets satisfy a weaker condition than FC Petri nets, and every FC Petri net is also EFC. Informally, we say that a Petri net is EFC if the result of a choice between two transitions is never influenced by the rest of the system. The following definition formalizes this intuition.

Definition 11 (Extended free-choice, from [7]). *A Petri net $PN = (P, T, A)$ is an* extended free-choice *Petri net, written $PN \models EFC$, if for every two transitions t_1 and $t_2 \in T$, $t_1 \neq t_2 : {}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing \Rightarrow {}^\bullet t_1 = {}^\bullet t_2$.*

In an EFC Petri net, if a transition has two or more input places, then all these places must have the same set of output transitions. Formally:

$$\forall t \in T : \forall p_1, p_2 \in {}^\bullet t \;\Rightarrow\; p_1{}^\bullet = p_2{}^\bullet \tag{5}$$

Next, we lift the definition of the EFC property to the product line level. A PNPL is a strong (weak) EFC if all (some) derivable nets are EFC.

Definition 12 (Strong and weak EFC product line). *A Petri net product line PNL is a strong extended free-choice iif $\forall PN_\rho \in Prod(PNL) : \; PN_\rho \models EFC$. PNL is a weak extended free-choice iif $\exists PN_\rho \in Prod(PNL) : \; PN_\rho \models EFC$.*

According to Eq. 5, we check that each transition t has the following EFC condition:

- t is not in PN_ρ, therefore Φ_t is false; or
- t is in PN_ρ, and therefore Φ_t needs to be true, and moreover, for every two places p_1 and p_2 in the pre-set ${}^\bullet t$ such that $p_1 \neq p_2$:
  1. each transition t' that is not in the post-set of both p_1 and p_2 is not in PN_ρ, and hence $\Phi(p_i, t')$ is false (for $i = 1$ *or* 2); and
  2. each transition t' that is in the post-set of both p_1 and p_2 is in PN_ρ (and hence $\Phi(p_1, t')$ and $\Phi(p_2, t')$ are true), or disappears from both post-sets (and therefore $\Phi(p_1, t')$ and $\Phi(p_2, t')$ are false).

In the previous condition, the first requirement demands configurations where the transitions t' that only belong to one of the post-sets (i.e., to $p_1{}^\bullet \setminus p_2{}^\bullet$ or to $p_2{}^\bullet \setminus p_1{}^\bullet$) disappear from this post-set. The second requirement demands the common transitions in $p_1{}^\bullet \cap p_2{}^\bullet$ to be maintained. Equation 6 captures these conditions as a logical formula:

$$\begin{aligned} \Phi_{EFC(t)} = \neg\Phi_t \vee (\Phi_t \wedge_{p_1,p_2 \in {}^\bullet t | p_1 \neq p_2} [& \wedge_{t' \in p_1{}^\bullet \setminus p_2{}^\bullet} \; \neg\Phi_{(p_1,t')} \\ & \wedge_{t' \in p_2{}^\bullet \setminus p_1{}^\bullet} \; \neg\Phi_{(p_2,t')} \\ & \wedge_{t' \in p_2{}^\bullet \cap p_1{}^\bullet} \; \Phi_{(p_1,t')} \Leftrightarrow \Phi_{(p_2,t')}]) \end{aligned} \tag{6}$$

Therefore, we define $\Phi_{EFC} = \wedge_{t \in T} \; \Phi_{EFC(t)}$. Consequently, if there exists a feature configuration that satisfies $SAT(\Psi \wedge \Phi_{EFC})$, then there is a derivable Petri net that is EFC, and hence the PNPL is a weak EFC product line. A PNPL is strong EFC if $SAT(\Psi \wedge \neg\Phi_{EFC})$ does not hold.

Example. In the PNPL of Fig. 3, there are two transitions that may have two incoming places in some configurations: proc and pack. However, their incoming places only have those transitions as their output, and therefore, the example is a strong EFC. Actually, as we have seen in Sect. 3.3, the example is a strong FC product line, and in consequence, we can conclude that it is a strong EFC product line as well.

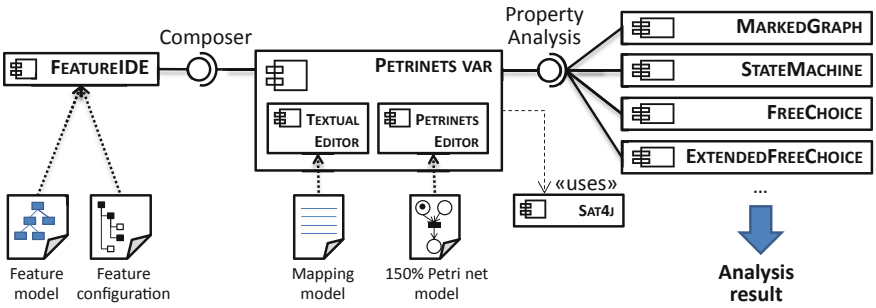

Fig. 6. Architecture of our Petrinets var tool.

4 Tool Support

We have implemented an Eclipse plugin, called Petrinets var, which supports the presented approach. Figure 6 shows its architecture.

Petrinets var provides two editors: one to specify the 150% Petri net, and another to assign PCs to its elements in a so-called mapping model. We use the Eclipse Modeling Framework (EMF) [31] as the underlying modelling technology, and therefore, both the 150% Petri net and the mapping model are EMF-based models that conform to their respective meta-models. The meta-model of the mapping model defines classes to represent the abstract syntax of the boolean formulae making the PCs, together with a cross-reference that points to the Petri net meta-model elements that can be annotated.

We rely on FeatureIDE [19] to specify the feature model and the feature configurations. FeatureIDE provides an extension point Composer that our tool instantiates to automate the derivation of specific Petri nets from the 150% Petri net given a feature configuration. Our tool defines an extension point as well, called Property Analysis, that allows extending the tool with new analysis methods. We currently provide four instances of this extension point to analyse whether some/all Petri nets in a PNPL are state-machines, marked graphs, free-choice or extended free-choice. Since the analysis techniques provided so far rely on the Sat4J solver [5], our plugin provides facilities to transform the conjunction of the analysis formula and the formula of the feature model into conjunctive normal form (CNF), as Sat4J requires. This can simplify the implementation of new analysis techniques by future users.

Figure 7 shows a screenshot of our tool. The Eclipse project explorer (label 1) contains the FeatureIDE project with the definition of the PNPL used as a running example. This project is configured with our composer and declares the 150% Petri net (file *150mm.petrinets*), the feature model (file *model.xml* that is being edited in the window labelled *2*), and the mapping model (file *annotation.vrb* that is being edited in the window labelled *3*). As the figure shows, there are dedicated editors for each kind of file. The textual editor for the PCs has code completion (e.g., offering the available feature names) and validation

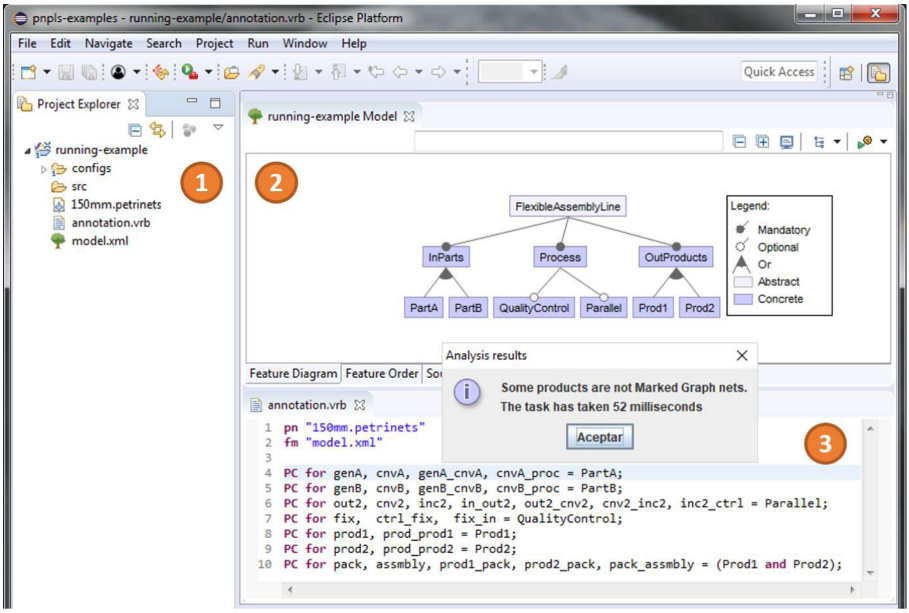

Fig. 7. Screenshot of our Petrinets var tool.

(e.g., it checks that the used features are defined in the feature model). A popup menu on the mapping model allows selecting the lifted analysis to perform. The results of the analyses, including the analysis time, are shown in a dialog box like the one shown in the figure.

Our current implementation uses its own EMF meta-model to represent 150% Petri nets. This meta-model supports a simple notion of net like the one we have used in the paper. However, to improve interoperability with other Petri net tools, we are planning to use the standard Petri Net Markup Language (PNML) [26] instead, for which there is an EMF implementation available.

5 Evaluation

Next, we report on two experiments to assess the efficiency gains of our lifted analyses, compared to generating all derivable nets in a PNPL and analysing each net separately. In the latter case, we perform an explicit analysis of each single net (i.e., without converting the analysis into a constraint satisfaction problem, which may take longer). In our evaluation, we measure the time for analysing the strong satisfaction of the MG, SM, FC and EFC properties (i.e., whether all nets in the PNPL satisfy the property).

We have carried out two experiments based on the running example. In the first experiment, illustrated in Fig. 8, we have analysed the efficiency of our analysis techniques when considering PNPLs with 150% Petri nets of different

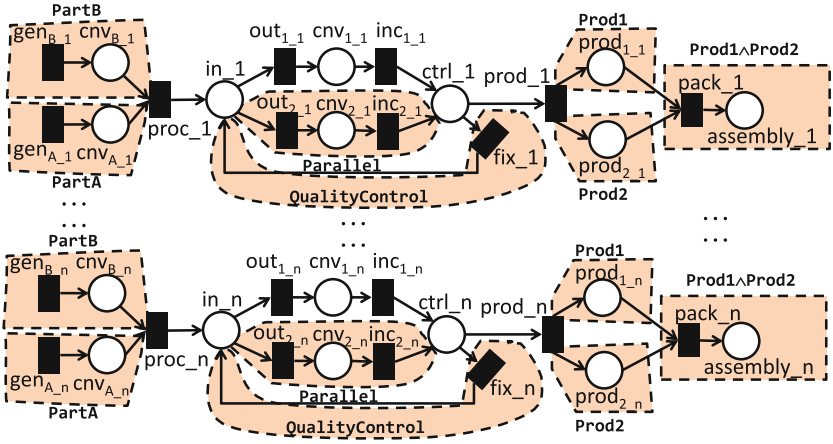

Fig. 8. Experiment 1: PNPL modelling a replicated flexible assembly line.

size but the same number of features. For this purpose, we have created ten PNPLs having the same feature model as in Fig. 2, and whose 150% Petri net contains n replicas of the assembly line in Fig. 3, with n from 1 to 10 (i.e., the first PNPL contains 1 replica, the second one contains two replicas, and so on). All created PNPLs have 36 valid feature configurations. As in the running example, the PNPLs are neither strong MG nor strong SM product lines, but they are strong FC and EFC product lines.

Figure 9 shows the analysis time in milliseconds with logarithmic scale of running 10 times the first execution. We consider just the first execution to discard cache effects. As it can be observed, all lifted analysis techniques were up to three orders of magnitude faster than the time to generate and analyse each net in isolation. In addition, the analysis time did not depend on the size of the 150% Petri net.

The goal of our second experiment is assessing whether an increase in the number of features of the PNPL has an impact in the analysis time. For this purpose, we have created seven PNPLs whose 150% Petri net contains a single assembly line, but they define an increasing number of features to model additional input parts (PartA, PartB, PartC, and so on) and output products (Prod1, Prod2, Prod3, and so on). Figure 10 illustrates the construction of the different feature models. The simplest PNPL contains one input part and one output product, and the most complex one has five input parts and five output products. These PNPLs are constructed by adding replicas of the PartA and Prod1 regions in the PNPL of Fig. 3. The PNPL with one input part and one output product is both a strong MG and a strong SM, but the remaining PNPLs are not. All PNPLs in the experiment are strong FC and EFC product lines.

The graphics in Fig. 11 show the analysis time in milliseconds with logarithmic scale (vertical axis to the left of the graphics), and the number of configurations of each PNPL (vertical axis to the right). Like in the first experiment, we

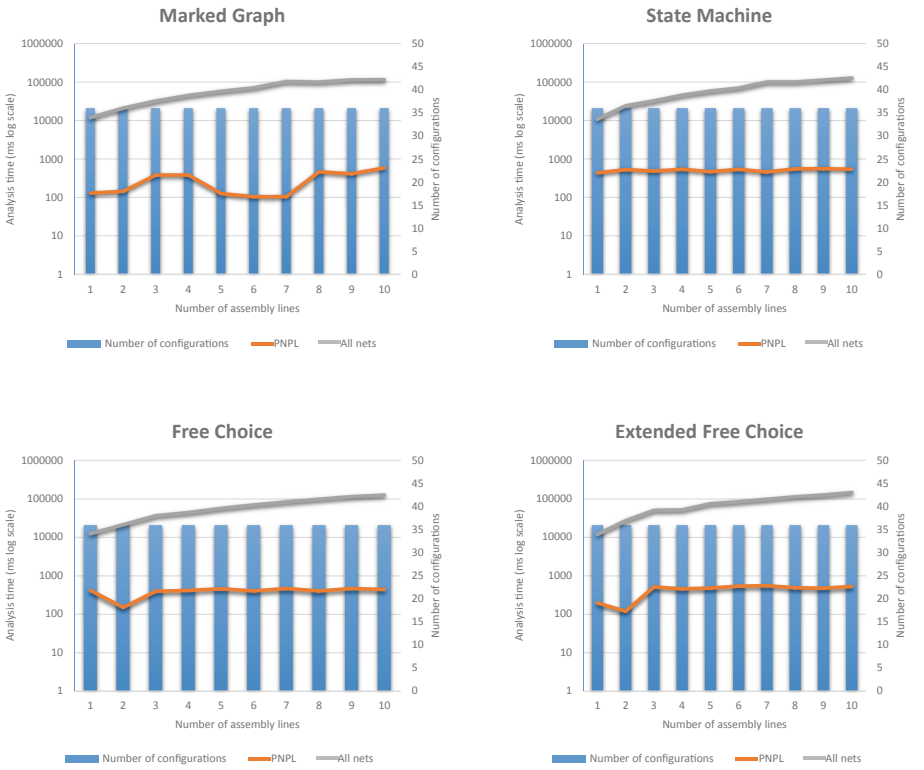

Fig. 9. Analysis time (ms in logarithmic scale) for PNPLs with 150% Petri nets of different size.

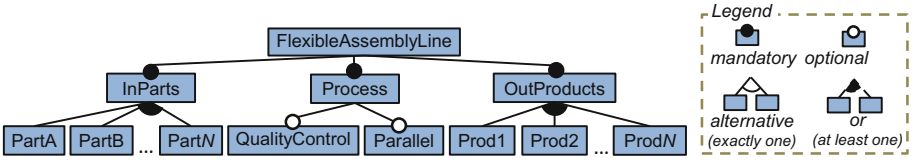

Fig. 10. Experiment 2: PNPL modelling an assembly line with N input parts and output products.

consider just the first execution to discard cache effects. It can be observed that the number of configurations is exponential on the number of features. While the analysis time of all nets in the PNPL one by one is exponential as well, the analysis time of the lifted property is roughly constant. Therefore, the larger the number of features in a PNPL, the bigger the efficiency gains of our lifted analysis compared to an enumerative approach.

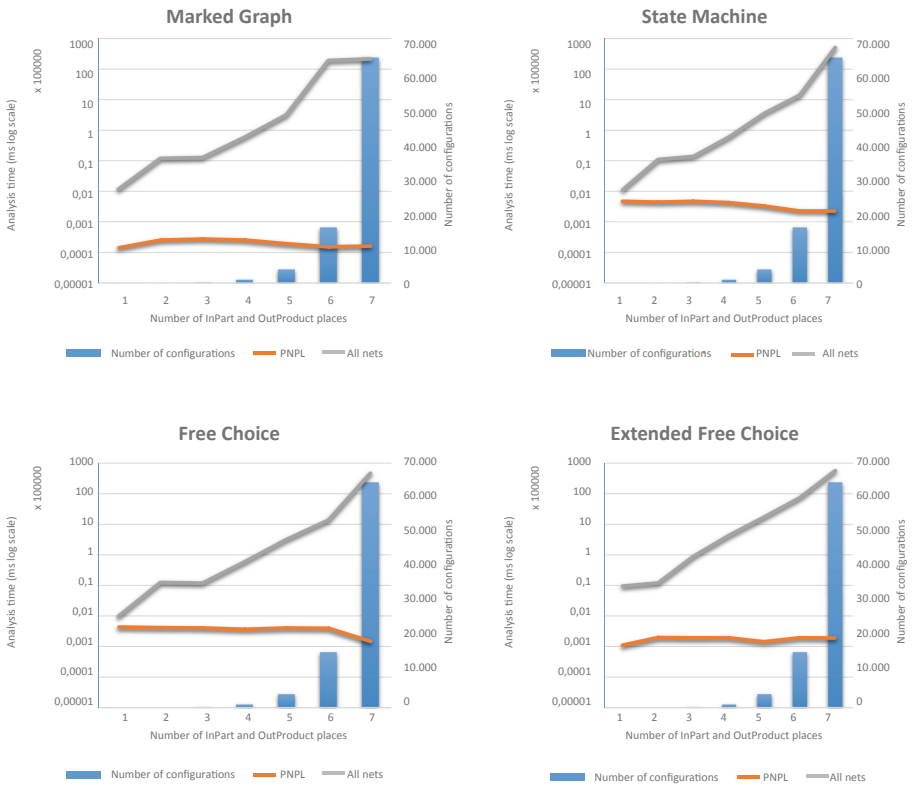

Fig. 11. Analysis time (ms in logarithmic scale) for PNPLs with feature models of different size.

Threats to Validity. While our experiments show gains of at least two orders of magnitude with respect to an enumeration-based approach, the experiments were based on a synthetic net and variations of it generated by replicating either a part of the net or a part of the feature model. Therefore, to further validate these results, we plan to consider models arising in realistic scenarios.

In addition, these results are for checking strong satisfaction of the properties. We plan to extend the experiments to consider weak satisfaction, for which we would require PNPLs with different percentages of product Petri nets satisfying the properties. We expect that our method will provide larger efficiency gains when the percentage of product Petri nets satisfying the property is low, as more nets within the PNPL need to be inspected to find one satisfying the property.

6 Related Work

The main analysis techniques for Petri nets can be classified into three groups [9]: i) enumeration, ii) transformation (mainly reduction), and iii) structural. *Enumeration* methods are based on the construction of a reachability/coverability graph, but they suffer the *state explosion problem*. Transformation methods obtain a slice of a Petri net that is easier to analyse but preserves the properties under study [6]. Structural analysis techniques are based on the net structure and its initial marking, and can be divided into two subgroups: *linear programming techniques* based on the state equation, and *graph-based techniques* based on "ad hoc" reasoning frequently derived from the firing rule. A survey on Petri nets models and their analysis techniques can be found at [30].

There are several mechanisms to model variability for SPLs. Most of them can be classified into annotation-based and composition-based techniques [3]. In annotation-based approaches, parts of a model are annotated with information about their mapping to products of the product line. They are widely used since they are easy to implement but they work under the closed world assumption, i.e., the set of features is fixed. In composition-based modelling, the product line is decomposed into separate modules representing features that can be composed to derive products. They support positive variability, that is, composition units are added on demand. Surveys on SPL modelling techniques can be found in [4, 8,33].

Just like us, some works have added variability to Petri nets using SPL techniques. *Feature Petri nets* (FN) [22] extend Petri nets to allow modelling the behaviour of an entire SPL. A FN transition is activated if its input places are marked and its application condition (a logical constraint over features) is true under the current configuration state. *Dynamic feature Petri nets* (DFPN) [23] extend FN to control feature bindings at runtime, and allow the evaluation of some dynamic properties using model checking. These works lift analysis techniques based on the reachability graph to the product line level, by adding PCs to this graph. They follow an annotative approach to model variability. Similarly, some works have used variability in Petri nets to express variants of higher-level languages – like activity diagrams – and use a variable reachability graph for analysis [14]. This work also uses an annotative approach for SPLs. With respect to these works, our mapping model is more general: [14] only supports variability in edges and [23] only supports variability in arcs and transitions, while our approach permits PCs in arcs, places and transitions. With respect to analysis techniques, the mentioned works focus on the reachability graph, while we lift structural analysis techniques.

In addition to SPL methods, other techniques to handle variability in Petri nets have been proposed. *Conditional Petri nets* [34] associate to each transition a condition defined with the family of $\mathcal{L}$ languages, and transitions are conditioned by the transition sequence previously applied. Likewise, *logical Petri nets* [17] limit transition firing by means of constraints on first-order logic. There are also *reconfigurable nets* [18], which can change the net topology at runtime

by means of rewriting rules. Instead, PNPLs are static: the user needs to provide a configuration to derive a Petri net.

Regarding analysis of model-based product lines, Czarnecki and Pietroszek [10] propose an approach to check whether all possible derivable models satisfy the OCL constraints of their meta-model. We may have encoded the different structural properties in OCL and used that technique. However, our solution permits generating specific constraints for the analysed PNPL (instead of relying on one generic OCL constraint), which therefore can be solved using simpler and potentially more efficient standard SAT-solving techniques. Instead, Czarneck's approach requires extending an existing OCL-based checker to consider PCs, while in practice we just use the Sat4J SAT solver.

Concerning SPL analysis of temporal properties, Legay et al. [16] represent the behaviour of variability-intensive systems by means of an extension of transition systems, called Feature Transition System. These authors also propose model checking algorithms to verify all products of a SPL [8]. Unlike this approach, we only focus on static properties, but we plan to explore behavioural properties in future works.

Altogether, to the best of our knowledge, there are no previous works on lifting the analysis of structural properties of PNPLs. This is relevant to enable an efficient analysis of structural properties for all variants within a PNPL. Structural analysis helps in discovering possible design errors in some product Petri net, e.g., related to the existence of conflicts, or interference of synchronization with conflicts. Our work is a first step in this direction, which we have realized in practice through extensible tooling.

7 Conclusions and Future Work

In this paper, we have proposed the notion of Petri net product line, and showed how to analyse structural properties (the marked graph, state-machine, free choice and extended free choice properties) at the product line level. We have validated the approach in practice by presenting an extensible prototype on top of FeatureIDE, and an experiment that shows the benefits of our approach with respect to an enumerative one.

In the future, we plan to support more types of static analysis techniques, exploit compositionality of Petri nets in these analysis techniques, and perform more thorough experiments. We also plan to consider further types of properties (not only strong and weak), in the line of [13]. Our idea is to develop a domain-specific language to express such analyses, which then can be compiled into standard SAT solving procedures. At the tool level, we will use the PNML meta-model to ease the connection of our approach with Petri net tools like CPN Tools [35]. We are also planning to explore the lifting of dynamic analysis techniques, e.g., based on the incidence matrix and on the reachability graph (similar to [23]). For the former, our idea is to include the PCs on the elements of the matrix, and use constraint solving to find place/transition invariants for some/all Petri net products. For the latter, a first idea – if we restrict to Petri

nets with PCs only in transitions – is to calculate the reachability graph of the 150% Petri net, and then annotate the reachability graph with PCs, in the style of [23]. Finally, we would like to combine product lines with other types of Petri nets (e.g., with inhibitor or read arcs, or with timed transitions), and consider variability of the Petri net language itself.

Acknowledgments. Work funded by the Spanish Ministry of Science (RTI2018-095255-B-I00) and the R&D programme of Madrid (P2018/TCS-4314).

Appendix

This appendix lifts the analysis of the state-machine property. A state-machine (SM) is a subclass of Petri net where each transition t has exactly one input and one output place, while each place may have multiple input and output transitions. SMs allow representing decisions, but not the synchronization of concurrent activities.

Definition 13 (State-machine, from [21]). *A Petri net $PN = (P, T, A)$ is a* state-machine*, written $PN \models SM$, if $\forall t \in T : \ |{}^\bullet t| = |t^\bullet| = 1$.*

Next, we lift the definition of the SM property to the product line level. A PNPL is a strong (weak) SM if all (some) derivable nets are SMs.

Definition 14 (Strong and weak SM product line). *A Petri net product line PNL is a strong state-machine iif $\forall PN_\rho \in Prod(PNL) : \ PN_\rho \models SM$. PNL is a weak state-machine iif $\exists PN_\rho \in Prod(PNL) : \ PN_\rho \models SM$.*

Similar to the case of MGs, to ensure that all derivable nets are SMs, we check that the size of the lifted pre-set ${}^\circ t = \{(p_0, \Phi_{(p_0,t)}), ..., (p_n, \Phi_{(p_n,t)})\}$ and the lifted post-set $t^\circ = \{(p_0, \Phi_{(t,p_0)}), ..., (p_n, \Phi_{(t,p_n)})\}$ of a transition t is one in every configuration. The size of the lifted pre-set ${}^\circ t$ of a transition t is one if the following formula is true. The disjunction starts with $false$ to consider the case when ${}^\circ t$ is empty. The formula Φ_{t° to check that the size of the lifted post-set of a transition t is one is defined similarly.

$$\begin{aligned} \Phi_{{}^\circ t} \triangleq \ & false \\ & \vee \ (\Phi_{(p_0,t)} \wedge \neg\Phi_{(p_1,t)} \wedge ... \wedge \neg\Phi_{(p_n,t)}) \\ & \vee \ (\neg\Phi_{(p_0,t)} \wedge \Phi_{(p_1,t)} \wedge ... \wedge \neg\Phi_{(p_n,t)}) \\ & \vee \ ... \ (\neg\Phi_{(p_0,t)} \wedge \neg\Phi_{(p_1,t)} \wedge ... \wedge \Phi_{(p_n,t)}) \end{aligned} \tag{7}$$

Hence, a PNPL includes some Petri net that is a SM if there is a feature configuration ρ such that for every transition t in the PNPL:

- t is not in PN_ρ, therefore Φ_t is false; or
- t is in PN_ρ, and therefore $\Phi_{{}^\circ t}$ and Φ_{t° need to be true.

Equation 8 shows the formula that captures the two previous conditions.

$$\Phi_{SM} = \wedge_{t \in T}[\neg\Phi_t \vee (\Phi_t \wedge \Phi_{\circ t} \wedge \Phi_{t^\circ})] \tag{8}$$

If there is a feature configuration such that $SAT(\Psi \wedge \Phi_{SM})$ holds, then there is a derivable Petri net that is a SM, and the PNPL is a weak SM. On the contrary, the feature configurations that produce nets which are not SMs are those making the formula $\neg\Phi_{SM}$ true. Hence, the PNPL is a strong SM if $SAT(\Psi \wedge \neg\Phi_{SM})$ does not hold.

References

1. van der Aalst, W.: Structural characterizations of sound workflow nets. Computing Science Reports 9263, Technische Universiteit Eindhoven (1996)
2. van der Aalst, W., Kindler, E., Desel, J.: Beyond asymmetric choice: a note on some extensions. Petri Net Newsl. **55**, 3–13 (1998)
3. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines - Concepts and Implementation. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37521-7
4. Benduhn, F., Thüm, T., Lochau, M., Leich, T., Saake, G.: A survey on modeling techniques for formal behavioral verification of software product lines. In: VaMoS, pp. 80:80–80:87. ACM (2015). https://doi.org/10.1145/2701319.2701332, http://doi.acm.org/10.1145/2701319.2701332
5. Berre, D.L., Parrain, A.: The Sat4j library, release 2.2. JSAT **7**(2–3), 59–64 (2010)
6. Berthelot, G.: Transformations and decompositions of nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) ACPN 1986. LNCS, vol. 254, pp. 359–376. Springer, Heidelberg (1987). https://doi.org/10.1007/978-3-540-47919-2_13
7. Best, E.: Structure theory of petri nets: the free choice hiatus. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) ACPN 1986. LNCS, vol. 254, pp. 168–205. Springer, Heidelberg (1987). https://doi.org/10.1007/978-3-540-47919-2_8
8. Classen, A., Cordy, M., Schobbens, P., Heymans, P., Legay, A., Raskin, J.: Featured transition systems: foundations for verifying variability-intensive systems and their application to LTL model checking. IEEE Trans. Softw. Eng. **39**(8), 1069–1089 (2013)
9. Colom, J., Teruel, E., Silva, M.: Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques. Ed. KRONOS (1998)
10. Czarnecki, K., Pietroszek, K.: Verifying feature-based model templates against well-formedness OCL constraints. In: GPCE, pp. 211–220. ACM (2006)
11. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press, Cambridge (1995)
12. Gómez-Martínez, E., de Lara, J., Guerra, E.: Towards extensible structural analysis of Petri net product lines. In: PNSE, vol. 2424, pp. 37–46. CEUR (2019)
13. Guerra, E., de Lara, J., Chechik, M., Salay, R.: Property satisfiability analysis for product lines of modelling languages. IEEE Trans. Softw. Eng. (2020, in press). https://doi.org/10.1109/TSE.2020.2989506
14. Heuer, A., Stricker, V., Budnik, C.J., Konrad, S., Lauenroth, K., Pohl, K.: Defining variability in activity diagrams and Petri nets. Sci. Comput. Program. **78**(12), 2414–2432 (2013)

15. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (FODA) feasibility study. Technical report. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University (1990)
16. Legay, A., Perrouin, G., Devroey, X., Cordy, M., Schobbens, P.-Y., Heymans, P.: On featured transition systems. In: Steffen, B., Baier, C., van den Brand, M., Eder, J., Hinchey, M., Margaria, T. (eds.) SOFSEM 2017. LNCS, vol. 10139, pp. 453–463. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51963-0_35
17. Liu, W., Wang, P., Du, Y., Zhou, M., Yan, C.: Extended logical Petri nets-based modeling and analysis of business processes. IEEE Access **5**, 16829–16839 (2017)
18. Llorens, M., Oliver, J.: Structural and dynamic changes in concurrent systems: reconfigurable Petri nets. IEEE Trans. Comput. **53**(9), 1147–1158 (2004). https://doi.org/10.1109/TC.2004.66
19. Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T., Saake, G.: Mastering Software Variability with FeatureIDE. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-61443-4
20. Meyers, B., Mierlo, S.V., Maes, D., Vangheluwe, H.: Efficient software controller variant development and validation (ECoVaDeVa) overview of a flemish ICON project. In: STAF Co-Located Events, vol. 2405, pp. 49–54. CEUR (2019)
21. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
22. Muschevici, R., Clarke, D., Proença, J.: Feature petri nets. In: SPLC Workshops, pp. 99–106. Lancaster University (2010)
23. Muschevici, R., Proença, J., Clarke, D.: Feature nets: behavioural modelling of software product lines. Softw. Syst. Model. **15**(4), 1181–1206 (2016). https://doi.org/10.1007/s10270-015-0475-z
24. Nabi, H., Aized, T.: Modeling and analysis of carousel-based mixed-model flexible manufacturing system using colored Petri net. Adv. Mech. Eng. **11**(12), 1–14 (2019)
25. Northrop, L., Clements, P.: Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
26. Petri Net Markup Language. www.pnml.org
27. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering. Foundations Principles and Techniques. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-28901-1
28. Rosa, M.L., van der Aalst, W., Dumas, M., Milani, F.: Business process variability modeling: a survey. ACM Comput. Surv. **50**(1), 2:1–2:45 (2017)
29. Seidl, C., Schaefer, I., Aßmann, U.: DeltaEcore - a model-based delta language generation framework. In: Modellierung. LNI, vol. 225, pp. 81–96. GI (2014)
30. Silva, M.: Half a century after Carl Adam Petri's Ph.D. thesis: a perspective on the field. Ann. Rev. Control **37**(2), 191–219 (2013). https://doi.org/10.1016/j.arcontrol.2013.09.001
31. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0, 2nd edn. Addison-Wesley Professional, Boston (2009)
32. Teruel, E., Silva, M.: Structure theory of equal conflict systems. Theoret. Comput. Sci. **153**(1&2), 271–300 (1996)
33. Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G.: A classification and survey of analysis strategies for software product lines. ACM Comput. Surv. **47**(1), 6:1–6:45 (2014). https://doi.org/10.1145/2580950

34. Tiplea, F., Jucan, T., Masalagiu, C.: Conditional Petri net languages. Elektronische Informationsverarbeitung und Kybernetik **27**(1), 55–66 (1991)
35. Westergaard, M., Kristensen, L.M.: The Access/CPN framework: a tool for interacting with the CPN tools simulator. In: Franceschinis, G., Wolf, K. (eds.) PETRI NETS 2009. LNCS, vol. 5606, pp. 313–322. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02424-5_19