Journal of Logical and Algebraic Methods in Programming 130 (2023) 100824

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Lifted structural invariant analysis of Petri net product lines

Elena Gómez-Martínez [*], Esther Guerra, Juan de Lara, Antonio Garmendia

Departamento de Ingeniería Informática, Universidad Autónoma de Madrid, Spain

ARTICLE INFO

Article history:
Received 21 February 2022
Received in revised form 23 September 2022
Accepted 24 September 2022
Available online 28 September 2022

Dataset link: https://github.com/antoniogarmendia/titan

Keywords:
Petri nets
Software product lines
Structural invariant analysis
Constraint solving

ABSTRACT

Petri nets are commonly used to represent concurrent systems. However, they lack support for modelling and analysing system families, like variants of controllers, different variations of a process model, or the possible configurations of a flexible assembly line.

To facilitate modelling potentially large collections of similar systems, in this paper, we enrich Petri nets with variability mechanisms based on product line engineering. Moreover, we present methods for the efficient analysis of the place and transition invariants in all defined versions of a Petri net. Efficiency is achieved by analysing the system family as a whole, instead of analysing each possible net variant separately. For this purpose, we lift the notion of incidence matrix to the product line level, and rely on constraint solving techniques. We present tool support and evaluate the benefits of our techniques on synthetic and realistic examples, achieving in some cases speed-ups of two orders of magnitude with respect to analysing each net variant separately.

1. Introduction

Petri nets are a popular formalism to describe and analyse concurrent systems [1]. They are widely used due to their rich body of theoretical results enabling analysis, and the maturity of the supporting tools. Some domains where Petri nets have been successfully applied include manufacturing systems [2], cluster tools [3], distributed protocols [4], process modelling [5], web modelling [6], cloud systems [7], reversible computation [8], and Internet of Things (IoT) technologies [9], to name a few areas.

Some scenarios require modelling a family of systems that share common elements but differ in some of their parts. The literature reports different examples such as the design of variants of controllers for cyber-physical systems [10], the enumeration of possible configurations of flexible assembly lines [11], the analysis of parameterised architectures [12], or modelling the variability of service robots [13]. In such scenarios, modelling and analysing each system variant separately is suboptimal: the common parts have to be repeatedly defined in each variant, maintenance and evolution become complex since modifications may need to be replicated in several variants, and verification is costly as each variant needs to be analysed on its own.

In this respect, software product lines [14,15] have proven valuable for reducing the complexity when dealing with software systems variants' in general, and recently, they have been applied to define Petri nets with variability [16–18]. Petri Net Product Lines (PNPLs) permit the compact definition of a system family by means of a Petri net where all variants are superimposed, and from which specific variants can be obtained by slicing. This simplifies the management of the

[*] Corresponding author.
E-mail addresses: mariaelena.gomez@uam.es (E. Gómez-Martínez), esther.guerra@uam.es (E. Guerra), juan.delara@uam.es (J. de Lara), antonio.garmendia@uam.es (A. Garmendia).

https://doi.org/10.1016/j.jlamp.2022.100824

$$\begin{tikzcd}
PNL \arrow[r, dotted, "\text{extract net with configuration } \rho"] \arrow[d, dotted, "\text{calculate generalised incidence matrix}"'] \arrow[dr, phantom, "="] & \mathcal{N}_\rho \arrow[d, dotted, "\text{calculate incidence matrix}"] \\
\mathbb{L}_{PNL} \arrow[r, dotted, "\text{project over configuration } \rho"] & \mathbb{C}_{\mathcal{N}_\rho}
\end{tikzcd}$$

Fig. 1. Main theoretical result of this paper.

system family because just one artefact (the PNPL) suffices to define all variants of the family in a unified way. In addition, since analysing each system variant on its own may be costly when the number of variants is high, some authors have devised ways to *lift* standard Petri net analyses to the PNPL level, avoiding the need to analyse each net of the family in isolation. We can find lifted analysis for structural properties in [17], and based on the reachability graph in [18].

In this paper, we focus on specific analyses based on the incidence matrix of a Petri net. This matrix describes the effect of firing each transition on each place of the net. Its analysis enables proving some structural properties of Petri nets without constructing the reachability graph, as this latter may suffer from the state space explosion problem [1]. In particular, it permits the identification of transition and place invariants (T- and P-invariants in short). On the one hand, T-invariants identify loops in the net, i.e., sequences of transitions whose overall effect is null and cause the net to go back to its initial state. This analysis is useful to reason about reversibility of states and to investigate transition liveness (i.e., the fact that each transition can be enabled again and again) [19]. On the other hand, P-invariants indicate that the number of tokens in any reachable marking satisfies some linear invariant (e.g., that the number of tokens in a given set of places remains constant). This can be useful to prove resource preservation, mutual exclusion properties or as a pre-processing step for reachability analysis [20]. Section 2 motivates both analyses by means of a running example of a family of vending machines, where the selling process of every specific machine should end up in the initial state (a T-invariant), and no vending machine can sell two products concurrently (a P-invariant).

This paper is a continuation of our previous work [16,17], where we lifted the analysis of structural properties (marked graph, state-machine and extended free-choice) to the product line level. In this work, we lift P- and T-invariant analysis, which requires radically different techniques. In particular, we generalise the standard incidence matrix of Petri nets to the product line level, encode the net variability in the generalised matrix, and analyse the matrix using constraint solving [21]. We demonstrate soundness and completeness of our method, for which we rely on the main result illustrated by Fig. 1. Specifically, given a product line *PNL* and a configuration ρ of *PNL*, we show that extracting the Petri net $\mathcal{N}_\rho$ corresponding to ρ, and then calculating its incidence matrix $\mathbb{C}_{\mathcal{N}_\rho}$, yields the same result as calculating the generalised incidence matrix $\mathbb{L}_{PNL}$ of PNL, and then projecting over the configuration ρ.

Overall, the novel contributions of this paper are the following:

(i) We propose techniques for the lifted analysis of P- and T-invariants in PNPLs, based both on a generalisation of the notion of incidence matrix, constraint solving and an elaborate notion of PNPL that considers parallel arcs;
(ii) We present tool support for our new analysis and improved modelling support in the Titan tool, an Eclipse plugin that is freely available at https://github.com/antoniogarmendia/titan;
(iii) We report on experiments that demonstrate the improved efficiency of our lifted invariant analysis in comparison to analysing each variant separately.

The rest of this paper is organised as follows. Section 2 provides motivation and a running example. Section 3 overviews the basics of Petri nets and their matrix-based analysis. Section 4 presents our notion of PNPL. Section 5 lifts the matrix-based analysis to the product line level. Section 6 describes our tool support. Section 7 evaluates the performance of our lifted analysis. Finally, Section 8 compares with related works, and Section 9 ends with the conclusions and future work.

2. Motivation and running example

As we argued in the introduction, some scenarios require defining and analysing several variants of a system, which are modelled as Petri nets in our case. This may be challenging if there are many variants. As an example (adapted from [22]) assume we aim to model and analyse several variants of a vending machine that differ in the products that they sell. Specifically, each vending machine variant permits selling one or more among three kinds of items (tea, coffee, and solid food), and may offer or not two kinds of supplement (milk and sugar). Moreover, no vending machine offers milk unless it provides coffee, or sugar unless it provides tea or coffee. In total, the vending machine family comprises 21 variants. Fig. 2 shows three of them, selling coffee and tea with no supplements (a); coffee and tea with sugar (b); and tea with sugar (c).

Defining each variant separately is time-consuming and error-prone. As the figure shows, the nets have common parts, so that changes in one net may need to be replicated in several others. Moreover, extending the family to account for new items (e.g., decaf) or supplements (e.g., saccharine) results in an exponential growth in the number of net versions to be

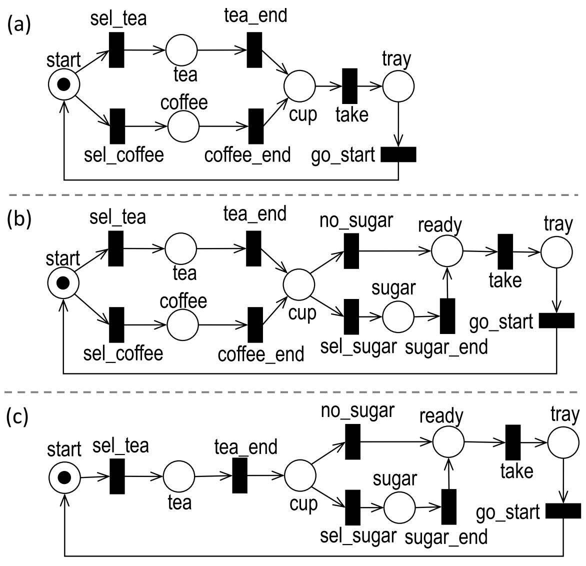


Fig. 2. Variants of a vending machine that sells: (a) coffee and tea with no supplement; (b) coffee and tea with sugar; (c) tea with sugar.

considered. In this paper, we are interested in the analysis of Petri net invariants. For instance, we would like to assess that no matter the vending machine variant or the bought product, any sale ends with the vending machine in the initial state, ready for starting a new sale. This can be expressed with a T-invariant, which can be analysed for each particular Petri net. Similarly, we may like to check that the net of each vending machine variant preserves the number of tokens, hence ensuring that all variants avoid serving coffee, tea or solids at the same time. This property can be expressed as a P-invariant. However, analysing each net version one by one to find T- and P-invariants can be time-consuming if the number of variants is large.

To alleviate this problem, we resort to product lines to enable the compact representation of all Petri nets of a family in a single net [17]. A product line of Petri nets merges all nets of a family into one net, and provides mechanisms to identify which parts of the merged net belong to each variant, and to easily retrieve each variant as needed using a configuration. Regarding analysis, enumerating and analysing each net variant one by one is time-consuming. Likewise, a naive, direct analysis of the merged net is not sufficient either, as the result of analysing the merged net may not correctly capture the result of analysing each net variant. For example, the merged net may have invariants that do not belong to any valid net variant it contains. Hence, we propose a method to lift the analysis of Petri net P- and T-invariants to the product line level, in order to reduce the analysis time with respect to analysing each net in isolation. This method permits analysing all nets of a family at the same time, instead of one by one, and ensures correct results.

3. Petri nets and matrix-based analysis

Next, we provide the background on Petri nets (PNs) needed to understand the rest of the paper: basics of PNs (Section 3.1), calculation of the incidence matrix and the state equations (Section 3.2), and place/transition invariant analysis (Section 3.3).

3.1. Basics of Petri nets

Petri nets are a graphical and mathematical modelling tool for describing concurrent systems. A PN model of a system consists of: (i) a *net structure* that captures the static part of the system as a weighted bipartite directed graph (Definition 1); and (ii) a *marking* representing the system state, distributed across the net structure (Definition 2).

Definition 1 *(Petri net structure).* A PN structure is a tuple $\mathcal{N} = (P, T, A_{PT}, A_{TP}, src_{PT}, tar_{PT}, src_{TP}, tar_{TP})$ made of:

- Finite, pairwise disjoint sets P of places; T of transitions; A_{PT} of place-to-transition arcs; and A_{TP} of transition-to-place arcs.
- The functions $src_{PT}: A_{PT} \to P$, $tar_{PT}: A_{PT} \to T$ identifying the source place and target transition of place-to-transition arcs.
- The functions $src_{TP}: A_{TP} \to T$, $tar_{TP}: A_{TP} \to P$ identifying the source transition and target place of transition-to-place arcs.

We use $A = A_{PT} \cup A_{TP}$ for the set of all arcs, and define two derived functions $w_{PT}(p,t)$ and $w_{TP}(t,p)$ – called arc *weights* – returning the number of arcs from a place to a transition, and vice versa. Formally, given a place $p \in P$ and a transition $t \in T$, $w_{PT}(p,t) = |\{a \in A_{PT} \mid src_{PT}(a) = p \wedge tar_{PT}(a) = t\}|$ and $w_{TP}(t,p) = |\{a \in A_{TP} \mid src_{TP}(a) = t \wedge tar_{TP}(a) = p\}|$.

Remark. Typically, most formalisations of PNs [1] represent the set A of arcs as a subset of the relation $(P \times T) \cup (T \times P)$, and parallel arcs as a weight function $w : A \to \mathbb{N}$. Instead, we represent arcs explicitly to enable their annotation with the variants they belong to (this annotation is called *presence condition*, PC). This way, the weight of arcs is a derived notion in our formalisation.

In the following, we use $\circ t$ to denote the pre-set of incoming arcs of t, $\{a \in A_{PT} \mid tar_{PT}(a) = t\}$, $t\circ$ for the post-set of outgoing arcs from t, $\{a \in A_{TP} \mid src_{TP}(a) = t\}$, and similarly for places ($\circ p$ and $p\circ$). In addition, we assume an implicit ordering for places and transitions, using p_i for the i-th place in P, and t_j for the j-th transition in T.

The state of a PN is distributed, defined by the number of tokens in each place (the *local state* of that place).

Definition 2 *(Marked Petri net).* A marked PN, $\mathcal{S} = (\mathcal{N}, \mathbf{M_0})$, is made of a PN structure $\mathcal{N}$ (as in Definition 1) and an *initial marking* $\mathbf{M_0} = [m_1, ..., m_n]^T$. The latter is a (column) vector with size $|P|$ of natural numbers (including 0), where each entry m_i is the number of tokens of place p_i.

Marked PNs can be simulated by the token game. A transition t is *enabled* at a marking $\mathbf{M}$ if and only if all its input places p_i have at least as many tokens as the arc weight $w_{PT}(p_i, t)$. Enabled transitions may *fire*, producing a change in the system state. When a transition t fires, its input places p_i are removed $w_{PT}(p_i, t)$ tokens, and its output places p'_j are added $w_{TP}(t, p'_j)$ tokens.

Graphically, places are depicted as white circles, transitions are depicted as black bars, directed arcs (arrows) connect places to transitions and transitions to places, and tokens are represented by black dots inside places (cf. Fig. 2).

3.2. The incidence matrix and the state equation

This subsection follows closely the definitions in [1]. Different kinds of properties can be analysed on PNs to obtain valuable information of the modelled systems. Properties can be generally classified into *behavioural* and *structural*, based on whether they depend on the initial marking or not. Behavioural properties (e.g., reachability, safeness, liveness [1]) can be proved by constructing the reachability graph, which may become exponential on the number of places [23]. Instead, analyses of structural properties overcome the state explosion problem by relying just on the net structure. Specifically, many structural analyses construct an incidence matrix encoding the connections between places and transitions, and then typically derive transition or place invariants that cover parts or all the net. This is the kind of properties this paper is interested in. The properties linked to this matrix-based approach include structural boundedness, conservativeness, repetitiveness and consistency [1].

Definition 3 formalises the concept of incidence matrix.

Definition 3 *(Incidence matrix [1]).* Given a PN structure $\mathcal{N}$, its incidence matrix $\mathbb{C}_{\mathcal{N}} = [c_{ij}]$ is a $|T| \times |P|$ matrix of integers, with each entry given by $c_{ij} = w_{TP}(t_i, p_j) - w_{PT}(p_j, t_i)$.

In the previous definition, $w_{PT}(p_j, t_i)$, $w_{TP}(t_i, p_j)$ and c_{ij} represent respectively the number of tokens removed, added and overall changed in place p_j when transition t_i fires. Hence, $\mathbb{C}_{\mathcal{N}}$ can also be expressed as a substraction of matrices of the form $\mathbb{C}_{\mathcal{N}} = \mathbb{C}^+_{\mathcal{N}} - \mathbb{C}^-_{\mathcal{N}}$, with $\mathbb{C}^+_{\mathcal{N}} = [c^+_{ij} = w_{TP}(t_i, p_j)]$ and $\mathbb{C}^-_{\mathcal{N}} = [c^-_{ij} = w_{PT}(p_j, t_i)]$.

Example 1. The incidence matrix of the Petri net in Fig. 2(a) is:

$$\mathbb{C}_{\mathcal{N}_1} = \left[\begin{array}{r|rrrrr} & \textit{start} & \textit{tea} & \textit{coffee} & \textit{cup} & \textit{tray} \\ \hline \textit{sel\_tea} & -1 & 1 & 0 & 0 & 0 \\ \textit{sel\_coffee} & -1 & 0 & 1 & 0 & 0 \\ \textit{tea\_end} & 0 & -1 & 0 & 1 & 0 \\ \textit{coffee\_end} & 0 & 0 & -1 & 1 & 0 \\ \textit{take} & 0 & 0 & 0 & -1 & 1 \\ \textit{go\_start} & 1 & 0 & 0 & 0 & -1 \end{array}\right] = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix $\mathbb{C}_{\mathcal{N}_1}$ has as many rows as transitions, and as many columns as places. Each cell $c_{i,j}$ reflects the net effect of firing transition i on place j. For example, $c_{tea\_end,tea} = -1$ since firing tea_end removes a token from place tea, while $c_{tea\_end,cup} = 1$ since firing tea_end adds a token to place cup.

A transition t_i is enabled at a marking $\mathbf{M}$ if $\mathbf{M}(j) \geq \mathbb{C}^-_{\mathcal{N}}(i, j)$ for $1 \leq j \leq |P|$, where $\mathbf{M}(j)$ is the j-th position of the vector $\mathbf{M}$, representing the marking of place p_j, and $\mathbb{C}^-_{\mathcal{N}}(i, j)$ is the cell (i, j) of matrix $\mathbb{C}^-_{\mathcal{N}}$, representing the tokens to be removed from place p_j when transition t_i fires.

Example 2. In Fig. 2(a), transitions sel_tea (with index 1) and sel_coffee (with index 2) are enabled in marking $\mathbf{M} = [1\ 0\ 0\ 0\ 0]^T$, since $\mathbf{M}(j) \geq \mathbb{C}^-_{\mathcal{N}_1}(1, j) = [1\ 0\ 0\ 0\ 0](j)$, and $\mathbf{M}(j) \geq \mathbb{C}^-_{\mathcal{N}_1}(2, j) = [1\ 0\ 0\ 0\ 0](j)$, for $1 \leq j \leq 5$.

The matrix $\mathbb{C}_{\mathcal{N}}$ allows computing the next net state upon firing an enabled transition. For this purpose, we define a firing vector $\mathbf{u}_k = [0...1...0]^T$ (a column vector with size $|T|$) to represent the k-th firing in some firing sequence, where the only 1 in the i-th position indicates the firing of the transition t_i. We say that the firing vector is *applicable* at marking $\mathbf{M}$ if transition t_i is enabled at $\mathbf{M}$. Since the i-th row of the incidence matrix $\mathbb{C}_{\mathcal{N}}$ denotes the marking changes as a result of firing t_i, the state equation of a PN with a marking $\mathbf{M_{k-1}}$ is:

$$\mathbf{M_k} = \mathbf{M_{k-1}} + \mathbb{C}^T_{\mathcal{N}} \cdot \mathbf{u}_k,\ k = 1, 2, ...$$

where $\mathbb{C}^T_{\mathcal{N}}$ is the transpose of matrix $\mathbb{C}_{\mathcal{N}}$.

Example 3. In Fig. 2(a), the next state after firing transition sel_tea can be calculated as:

$$\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}-1 & -1 & 0 & 0 & 0 & 1\\ 1 & 0 & -1 & 0 & 0 & 0\\ 0 & 1 & 0 & -1 & 0 & 0\\ 0 & 0 & 1 & 1 & -1 & 0\\ 0 & 0 & 0 & 0 & 1 & -1\end{bmatrix} \cdot \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}-1\\1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}$$

Firing transition sel_tea removes one token from place start and adds a token in place tea. Therefore, the resulting marking has one token in place tea and zero in the rest of the places.

If a marking $\mathbf{M_d}$ is reachable from the initial marking $\mathbf{M_0}$, then a sequence $\langle \mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_d \rangle$ of firing vectors must exist [24]. By adding those vectors, we can express several firing steps with one equation as follows:

$$\mathbf{M_d} = \mathbf{M_0} + \mathbb{C}^T_{\mathcal{N}} \cdot \sum_{k=1}^{d} \mathbf{u}_k$$

3.3. Place and transition invariants

Invariants [25] are assertions guaranteed to be true in all reachable states of a given net. Here, we focus on structural invariants, a kind of invariants that do not require computing the reachability graph of the net, which generally is very expensive to calculate. Structural invariants are useful for deriving certain properties, such as conservativeness, liveness, home states and consistency, among others [26]. We focus on two kinds of invariants: place invariants (P-invariants) and transition invariants (T-invariants) [1].

A T-invariant identifies a set of transition firings that can return the net to the same marking, therefore indicating a possible loop. In the running example, identifying the T-invariants of the nets in Fig. 2 can be used to assess that any possible sequence of transition firings will end up leading to the same initial marking. In other words: after completing a sale, the vending machine always returns to the starting point to allow new selections.

Definition 4 *(T-invariant [1]).* Given a PN $\mathcal{N}$, a *T-invariant* is a non-trivial integer column vector $\mathbf{x}$ of size $|T|$ satisfying:

$$\mathbb{C}^T_{\mathcal{N}} \cdot \mathbf{x} = 0\text{, where each element } \mathbf{x}(i) \text{ of the vector } \mathbf{x} \text{ is} \geq 0$$

We use $inv_T(\mathcal{N})$ for the set of all T-invariants of $\mathcal{N}$.

T-invariants denote possible cycles in the reachability graph, but they may not be realisable (i.e., there may not be a reachable marking where the sequence of transition firings is applicable). A vector $\mathbf{x}$ of size $|T|$ is a realisable T-invariant iff there exists a reachable marking $\mathbf{M}$ and a sequence $\langle \mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_d \rangle$ of applicable firing vectors changing the marking $\mathbf{M}$ back to $\mathbf{M}$, where the sum of the firing vectors equals $\mathbf{x}$ [1].

Example 4. The net of Fig. 2(a) has two loops caused by the firing sequences sel_tea; tea_end; take; go_start and sel_coffee; coffee_end; take; go_start. In the case of the first loop, this is so because:

$$\left[\begin{array}{c|cccccc} & sel\_tea & sel\_coffee & tea\_end & coffee\_end & take & go\_start\\ \hline start & -1 & -1 & 0 & 0 & 0 & 1\\ tea & 1 & 0 & -1 & 0 & 0 & 0\\ coffee & 0 & 1 & 0 & -1 & 0 & 0\\ cup & 0 & 0 & 1 & 1 & -1 & 0\\ tray & 0 & 0 & 0 & 0 & 1 & -1\end{array}\right] \cdot \begin{bmatrix}1\\0\\1\\0\\1\\1\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix}$$

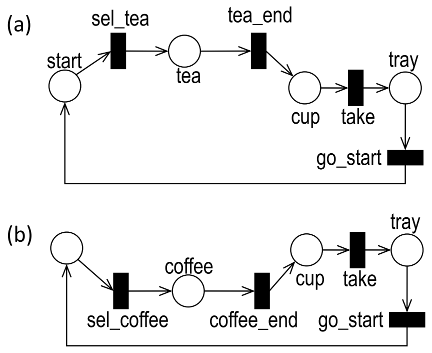


Fig. 3. Two T-invariants of the Petri net in Fig. 2(a).

A similar equality holds for the second loop. The vector $[1\ 0\ 1\ 0\ 1\ 1]^T$ is realisable. It can be decomposed into four firing vectors: $[1\ 0\ 0\ 0\ 0\ 0]^T$, $[0\ 0\ 1\ 0\ 0\ 0]^T$, $[0\ 0\ 0\ 0\ 1\ 0]^T$, $[0\ 0\ 0\ 0\ 0\ 1]^T$. The first vector is applicable to $\mathbf{M_0} = [1\ 0\ 0\ 0\ 0]^T$, the subsequent vectors are applicable in sequence, and the final marking is again $\mathbf{M_0}$.

The set of transitions corresponding to the non-zero entries of the T-invariant vector **x** is called the support of **x**, written *supp*(**x**). An invariant is *minimal* if its support does not contain the support of any other invariant, and the greatest common divisor of all non-zero entries of **x** is 1. This way, while T-invariants can be understood as multi-sets, minimal invariants are frequently sets. The set of minimal invariants forms a generating set of base vectors for all possible invariants that can be formed as linear combinations of the minimal invariants [27]. A minimal T-invariant defines a connected subnet within a PN, consisting of a set of transitions (those in the support), their pre- and post-places, and all arcs between them.

Example 5. Fig. 3 graphically shows the two T-invariants described in Example 4 for the net in Fig. 2(a). Every element in the original net is covered by one of the T-invariants, meaning that the net always returns to the initial state regardless of the chosen beverage. Generally, liveness of bounded nets covered by T-invariants can be checked efficiently [19].

Similarly, a P-invariant denotes a set of places on which the weighted sum of their tokens remains constant in any possible reachable marking. P-invariants can be used to prove mutual exclusion and can be seen as a token-preserving net component. In our example, we can use P-invariants to prove that none of the machines serves both tea and coffee at the same time.

Definition 5 *(P-invariant [1]).* Given a PN $\mathcal{N}$, a *P-invariant* is a non-trivial integer column vector **y** of size $|P|$ satisfying:

$$\mathbb{C}_{\mathcal{N}} \cdot \mathbf{y} = 0\text{, where each element } \mathbf{y}(i) \text{ of the vector } \mathbf{y} \text{ is} \geq 0$$

We use $inv_P(\mathcal{N})$ for the set of all P-invariants of $\mathcal{N}$.

The P-invariant vector **y** identifies a set of places where the weighted sum of tokens remains constant in all reachable markings regardless the initial marking. This means that given any two reachable markings $\mathbf{M_i}$ and $\mathbf{M_j}$, $\mathbf{M_i} \cdot \mathbf{y} = \mathbf{M_j} \cdot \mathbf{y}$. When all entries in **y** are 0 or 1, the sum of tokens in the places corresponding to the non-zero entries remains unchanged in all reachable markings.

Just like for T-invariants, there is also the notion of minimal P-invariant. This defines a connected subnet within a PN, consisting of a set of places, their pre- and post-transitions, and all arcs in between. A net is covered by P-invariants if every place belongs to some P-invariant.

Example 6. The three nets in Fig. 2 have one minimal P-invariant that indicates the preservation of the number of tokens (1) in all possible markings. This means that all places in the nets are mutually exclusive, and demonstrates that a vending machine cannot serve tea and coffee concurrently. As an illustration, $[1\ 1\ 1\ 1\ 1]^T$ is a P-invariant for the net in Fig. 2(a) because:

$$\mathbb{C}_{\mathcal{N}_1} = \left[\begin{array}{r|rrrrr} & start & tea & coffee & cup & tray \\ \hline sel\_tea & -1 & 1 & 0 & 0 & 0 \\ sel\_coffee & -1 & 0 & 1 & 0 & 0 \\ tea\_end & 0 & -1 & 0 & 1 & 0 \\ coffee\_end & 0 & 0 & -1 & 1 & 0 \\ take & 0 & 0 & 0 & -1 & 1 \\ go\_start & 1 & 0 & 0 & 0 & -1 \end{array}\right] \cdot \begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix}$$

Table 1
Conditions for some structural properties [1].

Property	Condition
Structurally bounded	$\exists \mathbf{y}.\ \mathbb{C}_{\mathcal{N}} \cdot \mathbf{y} \leq 0$ with $\mathbf{y}(i) > 0$
Conservative	$\exists \mathbf{y}.\ \mathbb{C}_{\mathcal{N}} \cdot \mathbf{y} = 0$ with $\mathbf{y}(i) > 0$
Partially conservative	$\exists \mathbf{y} \neq \mathbf{0}.\ \mathbb{C}_{\mathcal{N}} \cdot \mathbf{y} = 0$ with $\mathbf{y}(i) \geq 0$
Consistent	$\exists \mathbf{x}.\ \mathbb{C}_{\mathcal{N}}^{T} \cdot \mathbf{x} = 0$ with $\mathbf{x}(i) > 0$
Partially consistent	$\exists \mathbf{x} \neq \mathbf{0}.\ \mathbb{C}_{\mathcal{N}}^{T} \cdot \mathbf{x} = 0$ with $\mathbf{x}(i) \geq 0$
Repetitive	$\exists \mathbf{x}.\ \mathbb{C}_{\mathcal{N}}^{T} \cdot \mathbf{x} \geq 0$ with $\mathbf{x}(i) > 0$
Partially repetitive	$\exists \mathbf{x} \neq \mathbf{0}.\ \mathbb{C}_{\mathcal{N}}^{T} \cdot \mathbf{x} \geq 0$ with $\mathbf{x}(i) \geq 0$

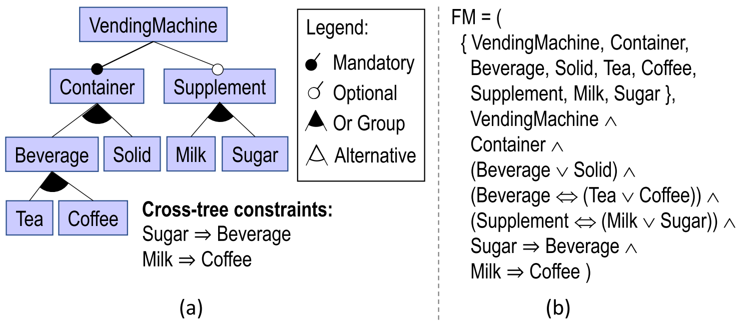


Fig. 4. Feature model of the running example. (a) Using the feature diagram notation. (b) Using Definition 6.

The set of T- and P-invariants can be calculated using matrix-based computations, e.g., with the Farkas algorithm [28,29]. Instead, as we will see in Section 5, our approach applies constraint solving [21] to solve the equations given by Definitions 4 and 5, where each entry of the vectors $\mathbf{x}$ and $\mathbf{y}$ is treated as a variable in the domain of the natural numbers including 0.

As an optimisation, the places and transitions that are unconnected or only have self-loop connections can be removed from the incidence matrix $\mathbb{C}_{\mathcal{N}}$. This is so because they have no effect on the P- and T-invariants. Such elements appear as zero-rows (in case of transitions) or zero-columns (for places). Hence, given an incidence matrix $\mathbb{C}_{\mathcal{N}}$, we write ${}^{P_0}_{T_0}\mathbb{C}_{\mathcal{N}}$ for the simplified matrix of size $|T \setminus T_0| \times |P \setminus P_0|$ where zero-rows (as given by the transitions in T_0) and zero-columns (those corresponding to the places in P_0) have been deleted.

In addition to P- and T-invariants, other structural properties can be expressed as variations of the equations in Definitions 4 and 5 [1]. These include:

- *Structural boundedness.* A PN is structurally bounded if the tokens in each place are bounded in each possible initial marking.
- *Conservativeness.* A PN is (partially) conservative if it has a P-invariant $\mathbf{y}$ with all (some) component $\mathbf{y}(i) > 0$.
- *Consistency.* A PN is (partially) consistent if it has a T-invariant $\mathbf{x}$ with all (some) component $\mathbf{x}(i) > 0$.
- *Repetitiveness.* A PN is (partially) repetitive if there is a marking $\mathbf{M_0}$ and a firing sequence from $\mathbf{M_0}$ such that every (some) transition occurs infinitely often in the sequence.

Table 1 summarises the necessary and sufficient conditions for each one of the previous properties.

4. Petri net product lines

This section extends PNs with variability. We start by defining the concept of feature model, which is a standard way to represent the allowed variability within a system [30].

Definition 6 *(Feature model [17]).* 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.

Example 7. Fig. 4(a) shows the feature model for the running example using the standard feature diagram notation [30], while Fig. 4(b) uses Definition 6. The feature model requires selecting VendingMachine and Container mandatorily, as well as choosing at least one Container item (Tea, Coffee or Solid) and optionally one or more Supplements (Milk, Sugar). The *cross-tree constraints* at the bottom of Fig. 4(a) specify that Sugar is only allowed if some Beverage is chosen; and Milk is only available for Coffee.

Selecting one system variant is done by providing a configuration of selected and unselected features.

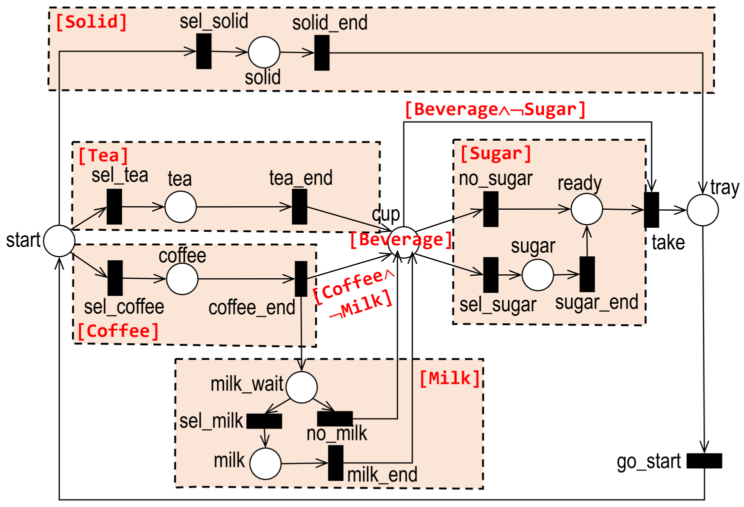


Fig. 5. 150% PN annotated with presence conditions (PCs) for the running example. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

Definition 7 *(Feature configuration [17]).* A *valid feature configuration* $\rho \subseteq F$ of a feature model $FM = (F, \Psi)$ is a subset of its features satisfying Ψ, i.e., $\Psi[true/\rho, false/(F \setminus \rho)]$ evaluates to true when each $f \in \rho$ is substituted by true, and each $f \in F \setminus \rho$ is substituted by false.

We write $Conf(FM) = \{\rho_i\}$ for the set of all valid feature configurations of *FM*.

Example 8. The feature model of the running example admits 21 configurations. These include {Solid}, {Tea, Coffee} and {Tea, Coffee, Milk, Sugar} (where we only list the selected features that have no sub-features). Instead, configurations like {Tea, Milk} and {Solid, Sugar} are invalid as they make the formula Ψ false.

Next, we define product lines of PNs (PNPLs). These comprise a feature model and a PN – called 150% PN[1] – where all variants of the family are superimposed [16]. In addition, each element of the 150% PN is annotated with a formula – called its presence condition (PC) – specifying the variants the element belongs to. This notion of product line only considers the structure of the PN but not the marking, since the variability only affects the PN structure.

Definition 8 *(Petri net product line; adapted from [17]).* A *PNPL* $PNL = (FM, \mathcal{N}, \Phi)$ is made of a feature model *FM*, a PN structure $\mathcal{N}$ (called the 150% PN), and a mapping Φ consisting of pairs $\langle x, \Phi_x \rangle$ that assign to each element $x \in P \cup T \cup A$ a propositional formula Φ_x (the presence condition of x) over the features in FM.

PNL is well-formed if:

- $\forall a \in A_{PT} : \; (\Phi_a \Rightarrow \Phi_{src_{PT}(a)}) \; \wedge \; (\Phi_a \Rightarrow \Phi_{tar_{PT}(a)})$
- $\forall a \in A_{TP} : \; (\Phi_a \Rightarrow \Phi_{src_{TP}(a)}) \; \wedge \; (\Phi_a \Rightarrow \Phi_{tar_{TP}(a)})$

The previous definition has been adapted from [17] to consider our notion of PN, which is more expressive than in [17] as it formalises arcs as objects instead of functions and permits parallel arcs. The well-formedness conditions demand that the PC (Φ_a) of each arc a be stronger (i.e., there is an implication) than the PC of the source and target nodes of a. This ensures that in any configuration where the arc is present, so are its source and target place/transition, hence avoiding dangling arcs.

Example 9. Fig. 5 shows the 150% PN annotated with PCs for the running example. The 150% PN superimposes the underlying structure of all PN variants in the PNPL. The PCs of elements are shown in red between square brackets and, in this example, use features from the feature model in Fig. 4. For readability reasons, we hide the PCs when their expression is true, meaning that the element affected by the PC is present in every possible variant (see, e.g., place start). Moreover, when several elements have the same PC, they are shown in a dashed shaded region together with their PC. As an example, the place solid, the transitions sel_solid and solid_end, and their incoming and outgoing arcs, have the PC Solid. This entails that these elements are present just in the configurations that select the feature Solid. Likewise, the other PCs in the 150% PN control the variability in the choice of tea, coffee, sugar and milk.

[1] The term 150% is standard in product line engineering to denote the superimposition of all variants of a given artefact: a software system [31,32], a model [33], a meta-model [34], or a PN as in our case.

Remark. Our formalisation could have considered setting PCs on the arcs only, and then assume that the PC of each place and transition is the disjunction of the PC of all their adjacent arcs. This way, a configuration ρ making the PC of all adjacent arcs to a transition t false, would make the PC of t false, too. However, this approach would be unable to express that transition t needs to be kept in such a configuration ρ. Alternatively, our formalisation could have set PCs on the arcs only, and then assume the PC of transitions and places to be always true. While this would keep all isolated places and transitions in all configurations, it would not enable distinguishing PN variants where those isolated elements are not present.

Next, we describe how to obtain a concrete PN variant out of a PNPL and a configuration. This process is called *derivation*. Intuitively, given a configuration, the derivation removes from the 150% PN the elements whose PC evaluates to false. This approach is called *negative* variability in the literature [35]. Similar to Definition 8, the following Definition 9 adapts the one in [17] to consider our particular notion of PN.

Definition 9 *(Derivation; adapted from [17]).* Given a PNPL $PNL = (FM, \mathcal{N} = (P, T, A_{PT}, A_{TP}, src_{PT}, tar_{PT}, src_{TP}, tar_{TP}), \Phi)$ and a configuration $\rho \in Conf\ (FM)$, we derive the PN structure $\mathcal{N}_\rho = (P_\rho, T_\rho, A_{PT\rho}, A_{TP\rho}, src_{PT\rho}, tar_{PT\rho}, src_{TP\rho}, tar_{TP\rho})$ by building each set $X_\rho \subseteq X$ (for $X \in \{P, T, A_{PT}, A_{TP}\}$) as $\{x \in X \mid \Phi_x[true/\rho, false/(F \setminus \rho)] = true\}$, and restricting the functions $src_{X\rho} = src_X|_{A_{X\rho}}$ and $tar_{X\rho} = tar_X|_{A_{X\rho}}$ (for $X \in \{PT, TP\}$). We use $Prod(PNL) = \{\mathcal{N}_\rho \mid \rho \in Conf(FM)\}$ for the set of all derivable nets from PNL.

Example 10. Given the PNPL made of the 150% PN in Fig. 5 and the feature model in Fig. 4, and given the configuration {Tea, Coffee}, we derive the PN of Fig. 2(a). The derivation deletes from the 150% PN all the elements whose PC evaluates to false after substituting the features Tea, Coffee and their parent features by true, and the remaining features by false. For instance, the regions with PC Milk, Sugar, and Solid are deleted. Similarly, the PN in Fig. 2(b) is derived by configuration {Coffee, Tea, Sugar}, and the net in Fig. 2(c) by configuration {Tea, Sugar}.

This notion of derivation assumes that a *total* configuration ρ is selected. Instead, a more flexible approach could consider *partial* configurations, where some features may remain undefined (neither selected nor unselected) [34]. Hence, a notion of derivation using partial configurations would produce a more concrete PNPL, where the PCs would become (partially) evaluated using the features of the partial configuration that are not undefined. This *concretisation* process would be equivalent to the *derivation* process in Definition 9 when none of the features in the partial configuration are undefined. Partial configurations may be useful to analyse a subset of the products of the PNPL, instead of all of them. For simplicity, we refrain from using partial configurations, and instead we can preset some of the features in the feature model to true or false before constraint solving in the analysis process (as we will explain in Section 5).

To analyse the P- and T-invariants of a product line PNL, a naive approach would derive all possible nets in $Prod(PNL)$ by using all valid configurations, and then analyse individually each of them by using the incidence matrix of each derived net. However, this may be time consuming as the number of nets may be exponential in the number of configurations. Instead, our proposal lifts the analysis to the product line level – creating a single, generalised incidence matrix for the PNPL – to avoid analysing each derived net in isolation, as the next section details.

5. Invariant analysis of Petri net product lines

In our previous works [16,17], we developed lifted analysis techniques for the structural properties marked graph, state machine, free-choice, and extended free-choice. These techniques build a logical formula capturing the cases in which a PNPL has one of the aforementioned properties, and use Boolean satisfiability (SAT) solving to check whether the formula is (or is not) satisfiable.

In this paper, we propose a novel lifted technique for the analysis of the P- and T-invariants of a PNPL. Our goal is reducing the time needed to analyse the invariants of all derivable nets of a PNPL. We do so by enabling the analysis of all derivable nets in one go, instead of having to perform a net-by-net analysis. For this purpose, our technique lifts the incidence matrix to the product line level, and applies constraint programming [21] to solve the resulting equations. This way, instead of performing the invariant analysis once per derivable net, our lifted analysis only needs to solve a constraint solving problem. In Section 7, we will show that this strategy generally speeds up the analysis. Next, we describe our proposed technique in detail.

5.1. 150% incidence matrix

The main idea is to capture the net structure of the 150% PN as a matrix of algebraic expressions that use the features $f_i \in F$ as variables with domain $\{0, 1\}$. Such variables take the value 0 when they are false, and 1 when they are true. To create the matrix, our technique transforms the PC of the arcs to algebraic expressions. For this purpose, it uses the equivalences between boolean formulae and algebraic integer equations shown in Table 2, that is, it uses sum for disjunction, product for conjunction, and $1 - f'$ for the negation of f. Moreover, we assume that in PCs, negation only occurs for features.

Table 2
From propositional logic to integer programming.

Logic	Algebraic expression (Exp)
$\bigvee f_i$	$\sum f_i'$
$\bigwedge f_i$	$\prod f_i'$
$\neg f$	$1 - f'$
true	1
false	0

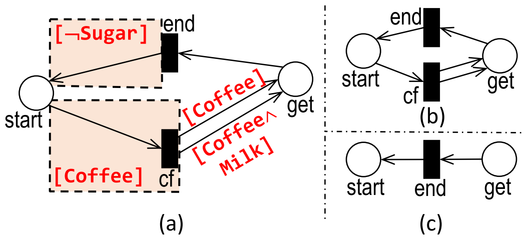


Fig. 6. (a) 150% PN of example PNPL. (b) $\mathcal{N}_{\{Coffee,Milk\}}$. (c) $\mathcal{N}_{\{Solid\}}$.

Remark. Conjunction could also be expressed as $\sum_{i=1}^{n} f_i' - (n-1)$ to obtain a linear equation, but for simplicity, we use the product form. In any case, in practice, one can resort to optimisations of the underlying solver to obtain linear systems when possible [36].

Example 11. Given the boolean expression (Milk $\vee$ Sugar) $\wedge$ Coffee, its equivalent integer equation, written Exp((Milk $\vee$ Sugar) $\wedge$ Coffee), is (Milk'+Sugar')·Coffee', where the variables Milk', Sugar' and Coffee' are integers in the domain $\{0, 1\}$. The boolean expression is true iff Coffee is true and either Milk or Sugar are true. The integer equation is satisfied (i.e., the equation is equal to 1) iff Coffee' is 1 and either Milk' or Sugar' are 1.

Definition 10 *(150% incidence matrix).* Given a PNPL $PNL = (FM, \mathcal{N}, \Phi)$, its 150% incidence matrix $\mathbb{L}_{PNL} = [l_{ij}]$ is a $|T| \times |P|$ matrix of integer expressions, with each entry l_{ij} given by:

$$l_{ij} = \sum_{a_{ij} \in (t_i \circ) \cap (\circ p_j)} Exp(\Phi_{a_{ij}}) - \sum_{a_{ji} \in (p_j \circ) \cap (\circ t_i)} Exp(\Phi_{a_{ji}})$$

Each entry l_{ij} of the 150% incidence matrix sums the algebraic expressions built from the PC of each arc $a_{ij} \in (t_i \circ) \cap (\circ p_j)$ going from transition t_i to place p_j, and subtracts the algebraic expressions built from the PC of each arc $a_{ji} \in (p_j \circ) \cap (\circ t_i)$ from place p_j to transition t_i. Recall that our notion of PN represents an arc with weight n as n parallel arcs, each having a PC. Hence, if each parallel arc from t_i to p_j has a PC that evaluates to true, then $\sum_{a_{ij} \in (t_i \circ) \cap (\circ p_j)} Exp(\Phi_{a_{ij}}) = w_{TP}(t_i, p_j)$ (and similar for incoming places to t_i).

Example 12. Fig. 6(a) shows an example PNPL *PNL* based on the running example, where we assume the feature model of Fig. 4. Note that there are two arcs from transition cf to place get, with different PC each. The 150% incidence matrix of this PNPL is:

$$\mathbb{L}_{PNL} = \left[\begin{array}{c|cc} & start & get \\ \hline end & 1 - Sugar' & -1 \\ cf & -Coffee' & Coffee' + Coffee' \cdot Milk' \end{array}\right]$$

where $l_{end\ get} = -1$ since the PC of the arc from place get to transition end is true. For the sake of illustration, Figs. 6(b) and 6(c) show two examples of nets derived from the PNPL using configurations $\rho_1 = \{$ Coffee, Milk$\}$ and $\rho_2 = \{$Solid$\}$, respectively. The net in Fig. 6(b) keeps the two arcs from cf to get since the PC of both arcs evaluates to true in ρ_1. The net in Fig. 6(c) excludes transition cf and its arcs because their PCs become false when substituting all features but Solid by false.[2]

The following definition describes how to obtain a regular incidence matrix out of a 150% incidence matrix and a configuration ρ. Then, Theorem 1 will prove that the obtained matrix $\mathbb{C}_{\mathcal{N}_\rho}$ – modulo simplification of zero rows and columns – is the incidence matrix of $\mathcal{N}_\rho$.

[2] More precisely, when Solid, VendingMachine and Container are substituted by true, and the other features are substituted by false.

Definition 11 *(Incidence matrix derivation).* Given a PNPL $PNL = (FM, \mathcal{N}, \Phi)$ and a configuration $\rho \in Conf(FM)$, the evaluation of $\mathbb{L}_{PNL}$ for ρ, written $\mathbb{L}_{PNL}[\rho]$, yields a $|T| \times |P|$ matrix of integers, where its elements $l'_{ij} = l_{ij}[1/\rho, 0/\ (F \setminus \rho)]$ are the result of evaluating l_{ij} upon the substitution of the variables in ρ by 1, and the variables in $F \setminus \rho$ by 0.

Example 13. Given the 150% incidence matrix in Example 12, corresponding to a PNPL whose 150% PN shows Fig. 6(a), its evaluation for configuration $\rho_1 = \{\text{Coffee, Milk}\}$ yields:

$$\mathbb{L}_{PNL}[\rho_1] = \left[\begin{array}{c|cc} & start & get \\ \hline end & 1 & -1 \\ cf & -1 & 2 \end{array}\right]$$

which actually is the incidence matrix of $\mathcal{N}_{\rho_1}$ (cf. Fig. 6(b)). Each entry in the derived incidence matrix $\mathcal{N}_{\rho_1}$ is obtained by replacing, in the 150% incidence matrix of Example 12, Coffee' and Milk' by 1 (since both features are selected by ρ_1) and the rest of variables by 0. For example, $l'_{cf\ start} = -1$ since the algebraic expression -Coffee' becomes -1 after the replacement.

If the PC Φ_{t_i} of transition t_i is false in a configuration ρ, then so is the PC of each arc in $\circ t_i$ and $t_i \circ$ by the well-formedness condition of Definition 8. In such a case, the row i in the derived incidence matrix is a zero-row. Similarly, a place p_j whose PC is false yields a zero-column j in the derived incidence matrix. If we remove these zero-rows and zero-columns from $\mathbb{L}_{PNL}[\rho]$, we obtain the incidence matrix of $\mathcal{N}_\rho$.

Theorem 1 *(Incidence matrix derivation yields incidence matrix). Given a PNPL $PNL = (FM, \mathcal{N}, \Phi)$ and a configuration $\rho \in Conf(FM)$, ${}^{P \setminus P_\rho}_{T \setminus T_\rho}\mathbb{L}_{PNL}[\rho] = \mathbb{C}_{\mathcal{N}_\rho}$.*

Proof. Each element l_{ij} of $\mathbb{L}_{PNL}[\rho]$ is defined by replacing in the equation of Definition 10 the variables in ρ by 1, and those in $F \setminus \rho$ by 0. Taking the first term of the equation and the equivalences in Table 2, we have $\sum_{a_{ij} \in (t_i \circ) \cap (\circ p_j)} Exp(\Phi_{a_{ij}})[\rho] = |\{a_{ij} \in A_{TP} \mid src_{TP}(a_{ij}) = t_i \wedge tar_{TP}(a_{ij}) = p_j \wedge \Phi_{a_{ij}} = true\}|$. But this is exactly $w_{TP}(t_i, p_j)$ for those arcs appearing in configuration ρ. A similar reasoning holds for the second term of the equation and $w_{PT}(p_j, t_i)$. Therefore, elements $l_{ij}[\rho]$ are defined as $w_{TP}(t_i, p_j) - w_{PT}(p_j, t_i)$ using the weights of the net $\mathcal{N}_\rho$, just like in Definition 3.

However, a transition t_i or a place p_j of the 150% PN is absent from $\mathcal{N}_\rho$ if $\Phi_{t_i}[\rho] = false$ or $\Phi_{p_j}[\rho] = false$. In such a case, row i or column j will only contain zeros. The set of places (resp. transitions) removed by configuration ρ is $P \setminus P_\rho$ (resp. $T \setminus T_\rho$). Using the matrix simplification operator to remove these zero-rows and zero-columns yields ${}^{P \setminus P_\rho}_{T \setminus T_\rho}\mathbb{L}_{PNL}[\rho] = \mathbb{C}_{\mathcal{N}_\rho}$. ■

Remark. Theorem 1 states that, given a PNPL *PNL*, there are two ways to compute the incidence matrix for the net $\mathcal{N}_\rho$ corresponding to a configuration ρ:

1. Calculate the 150% incidence matrix $\mathbb{L}_{PNL}$ (cf. Definition 10), evaluate this matrix for configuration ρ (cf. Definition 11), and remove from the resulting matrix the rows and columns corresponding to places and transitions not selected by ρ to yield ${}^{P \setminus P_\rho}_{T \setminus T_\rho}\mathbb{L}_{PNL}[\rho]$.
2. Derive the net $\mathcal{N}_\rho$ from *PNL* (cf. Definition 9), and calculate the incidence matrix (cf. Definition 3) to yield $\mathbb{C}_{\mathcal{N}_\rho}$.

The theorem shows that both ways of calculation yield the same result, as the commutative diagram of Fig. 1 depicts graphically.

Example 14. The evaluation of the 150% incidence matrix of the net of Fig. 6(a) for $\rho_2 = \{\text{Solid}\}$ (cf. Fig. 6(c)) yields:

$$\mathbb{L}_{PNL}[\rho_2] = \left[\begin{array}{c|cc} & start & get \\ \hline end & 1 & -1 \\ cf & 0 & 0 \end{array}\right]$$

The row corresponding to transition *cf* is zero because the transition is not present in $\mathcal{N}_{\rho_2}$, so it can be removed to obtain:

$$^{\emptyset}_{\{cf\}}\mathbb{L}_{PNL}[\rho_2] = \left[\begin{array}{c|cc} & start & get \\ \hline end & 1 & -1 \end{array}\right] = \mathbb{C}_{\mathcal{N}_{\rho_2}}$$

5.2. Petri net product line invariants

Next, we lift the definition of P- and T-invariants to the PNPL level. The invariants of a PNPL are those in any net of the family.

Definition 12 *(Petri net product line invariant).* Given a PNPL $PNL = (FM, \mathcal{N}, \Phi)$, a vector $\mathbf{y}$ is a P-invariant of PNL if:

$$\exists \mathcal{N}_\rho \in Prod(PNL).\, \mathbf{y} \in inv_P(\mathcal{N}_\rho)$$

(and similarly for T-invariants).

The following definition uses the 150% incidence matrix to establish a set of equations whose solutions (of the form $\langle \mathbf{y}, \overline{\rho} \rangle$) represent an invariant $\mathbf{y}$ and a configuration $\overline{\rho}$ which yields a PN where the invariant holds. More precisely, $\overline{\rho}$ is a binary vector of size $|F|$ (with F the feature set of the feature model) where $\overline{\rho}(i) = 1$ iff the configuration ρ contains feature f_i ($f_i \in \rho$) and 0 otherwise. To restrict to valid configurations, the set of equations include $Exp(\Psi) > 0$ (i.e., the algebraic equation equivalent to the formula Ψ of the feature model FM).

Definition 13 *(150% incidence matrix equation).* Given a PNPL $PNL = ((F, \Psi), \mathcal{N}, \Phi)$, the 150% incidence matrix equation is defined by:

$$\begin{aligned} &\mathbb{L}_{PNL} \cdot \mathbf{y} = 0\,, \text{with}\, \mathbf{y}(i) \in \{0, 1\}\, (\text{for } 1 \le i \le |P|) \\ &Exp(\Psi) > 0\,, \text{with}\, \overline{\rho}(j) \in \{0, 1\}\, (\text{for } 1 \le j \le |F|) \end{aligned} \tag{1}$$

The next lemma states that the solutions $\langle \mathbf{y}, \overline{\rho} \rangle$ of the equations of Definition 13 are sound: $\mathbf{y}$ is an invariant for $\mathcal{N}_\rho$.

Lemma 1 *(Soundness of PNPL invariants). Given a PNPL $PNL = ((F, \Psi), \mathcal{N}, \Phi)$, if $\langle \mathbf{y}, \overline{\rho} \rangle$ is a solution of Equation (1), then ${}_{P\setminus P_\rho}\mathbf{y} \in inv_P(\mathcal{N}_\rho)$, where ${}_{P\setminus P_\rho}\mathbf{y}$ is the vector $\mathbf{y}$ after removing the elements corresponding to places deleted from $\mathcal{N}$ by ρ (and similarly for T-invariants).*

Proof. Assume that $\langle \mathbf{y}, \overline{\rho} \rangle$ is a solution of Equation (1). Then, ρ is a valid configuration, and $\overline{\rho}$ is a solution of $Exp(\Psi) > 0$. Hence, we have $\mathbb{L}_{PNL}[\rho] \cdot \mathbf{y} = 0$. By Theorem 1, ${}^{P\setminus P_\rho}_{T\setminus T_\rho}\mathbb{L}_{PNL}[\rho] = \mathbb{C}_{\mathcal{N}_\rho}$, and so, $\mathbb{C}_{\mathcal{N}_\rho} \cdot {}_{P\setminus P_\rho}\mathbf{y} = 0$ as required. ■

Example 15. $\langle [1\ \ 1]^T, \rho_2 = \{\text{Solid}\} \rangle$ is a solution of Equation (1) for the PNPL in Fig. 6(a), and $[1\ \ 1]^T$ is a P-invariant for $\mathcal{N}_{\{Solid\}}$ since $\mathbb{C}_{\mathcal{N}_{\rho_2}} \cdot [1\ \ 1]^T = [0]$.

The next lemma states that the equations in Definition 13 permit obtaining the complete set of invariants for any net $\mathcal{N}_\rho$ that can be derived under any valid configuration ρ.

Lemma 2 *(Completeness of PNPL invariants). Let $PNL = ((F, \Psi), \mathcal{N}, \Phi)$ be a PNPL. Then, $\forall \mathcal{N}_\rho \in Prod(PNL).\ \mathbf{y} \in inv_P(\mathcal{N}_\rho)$ implies that $\langle \mathbf{y}_{P\setminus P_\rho}, \overline{\rho} \rangle$ is a solution of Equation (1), where $\mathbf{y}_{P\setminus P_\rho}$ is the vector $\mathbf{y}$ after adding 0 in the positions corresponding to the places of $\mathcal{N}$ not present in $\mathcal{N}_\rho$ (and similarly for T-invariants).*

Proof. Assume that $\mathbf{y}$ is a solution of $\mathbb{C}_{\mathcal{N}_\rho} \cdot \mathbf{y} = 0$. Since ${}^{P\setminus P_\rho}_{T\setminus T_\rho}\mathbb{L}_{PNL}[\rho] = \mathbb{C}_{\mathcal{N}_\rho}$, then $\mathbf{y}$ is a solution of ${}^{P\setminus P_\rho}_{T\setminus T_\rho}\mathbb{L}_{PNL}[\rho] \cdot \mathbf{y} = 0$, and so, $\mathbb{L}_{PNL}[\rho] \cdot \mathbf{y}_{P\setminus P_\rho} = 0$. Thus, $\langle \mathbf{y}_{P\setminus P_\rho}, \overline{\rho} \rangle$ is a solution for Equation (1) as required. ■

Finally, Theorem 2 states that using the 150% incidence matrix is a sound and complete method for obtaining the invariants of any derivable net from a PNPL. It is a direct consequence of Lemmas 1 and 2.

Theorem 2 *(Soundness and completeness of PNPL invariants). Given a PNPL $PNL = ((F, \Psi), \mathcal{N}, \Phi)$, then:*

- Soundness*: if $\langle \mathbf{y}, \overline{\rho} \rangle$ is a solution of Equation (1), then ${}_{P\setminus P_\rho}\mathbf{y} \in inv_P(\mathcal{N}_\rho)$ (and similarly for T-invariants).*
- Completeness*: $\forall \mathcal{N}_\rho \in Prod(PNL).\ \mathbf{y} \in inv_P(\mathcal{N}_\rho)$ implies that $\langle \mathbf{y}_{P\setminus P_\rho}, \overline{\rho} \rangle$ is a solution of Equation (1) (and similarly for T-invariants).*

Proof. Direct consequence of Lemmas 1 and 2. ■

Lifting the analysis of the structural properties in Table 1 (boundedness, conservativeness, consistency, repetitiveness) can be done similarly as in Definition 13, by considering $\mathbb{L}_{PNL} \cdot \mathbf{y} \le 0$ or $\mathbb{L}^T_{PNL} \cdot \mathbf{x} \ge 0$ (instead of $\mathbb{L}_{PNL} \cdot \mathbf{y} = 0$) and restricting the components of $\mathbf{y}$ or $\mathbf{x}$ to be strictly positive, as required by each property.

In order to analyse a subset of PNs of the PNPL instead of all of them, it is enough to preset the value of some of the features in the feature model to true or false (i.e., preset the values of some $\overline{\rho}(i)$ to 1 or 0) in Equation (1) before solving.

Given that Equation (1) provides a sound and complete method to obtain the P- and T- invariants of a PNPL, the following section describes a tool that uses constraint solving to obtain such solutions, and offers a graphical editor to create PNPLs. Then, Section 7 will show that using constraint solving on Equation (1) is more efficient than a case-by-case analysis.

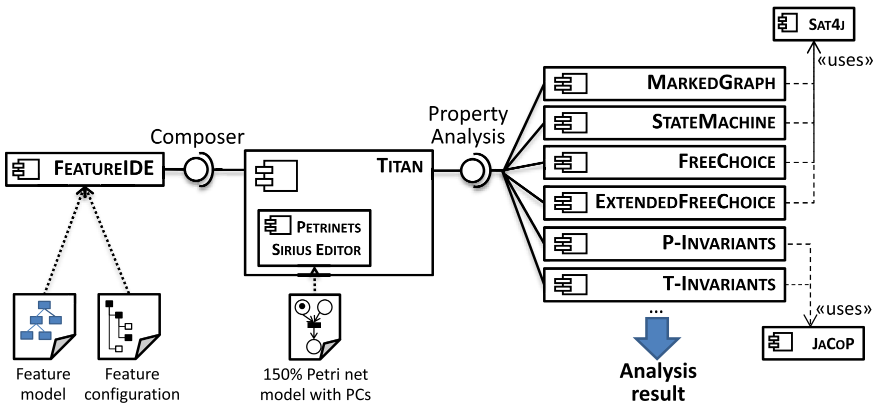


Fig. 7. Architecture of Titan.

6. Tool support

We have developed a tool called Titan (Tool for Petri net product line analysis) supporting our approach. Titan is an Eclipse plug-in, freely available at https://github.com/antoniogarmendia/titan (but not open-source for the moment). Next, we explain its architecture (Section 6.1) and the front-end (Section 6.2).

6.1. Architecture

Fig. 7 shows the architecture of Titan. Its core component is the *Petri nets Sirius Editor*, which allows the graphical definition of PNPLs, annotating their element with PCs. This editor relies on Sirius [37], a modern framework for the construction of graphical modelling environments. The editor is integrated with FeatureIDE [38], a popular plug-in for product line engineering with support for the creation and analysis of feature diagrams, and the derivation of products out of configurations. The latter requires implementing the Composer interface provided by FeatureIDE to specify how to derive a PN product out of a feature configuration and a 150% PN.

FeatureIDE uses the feature diagram notation. Hence, to calculate the PNPL invariants, first, Titan translates the feature diagram into a boolean formula (as required by Definition 6) following the rules described in [39]. Then, Titan transforms the PNPL into Equation (1), expressing it as a constraint satisfaction problem (CSP). Formally, a CSP consists of a set of variables, a set of finite domains, and a set of constraints restricting the values of the variables [40]. CSPs are solved by means of constraint programming (CP) techniques. Specifically, Titan relies on the JaCoP Java library as CP solver [41].

Titan's analysis architecture is extensible via an Eclipse extension point (called Property Analysis) which allows contributing analysis techniques in an external way (i.e., without requiring Titan's source code). The lifted P- and T-invariant analyses presented in this paper have been implemented by profiting from the extension point. Titan also supports analysing structural properties of the PNPL (state machine, marked graph and (extended) free-choice), as described in [17], for which the Sat4J SAT solver [42] is used.

6.2. The Titan tool

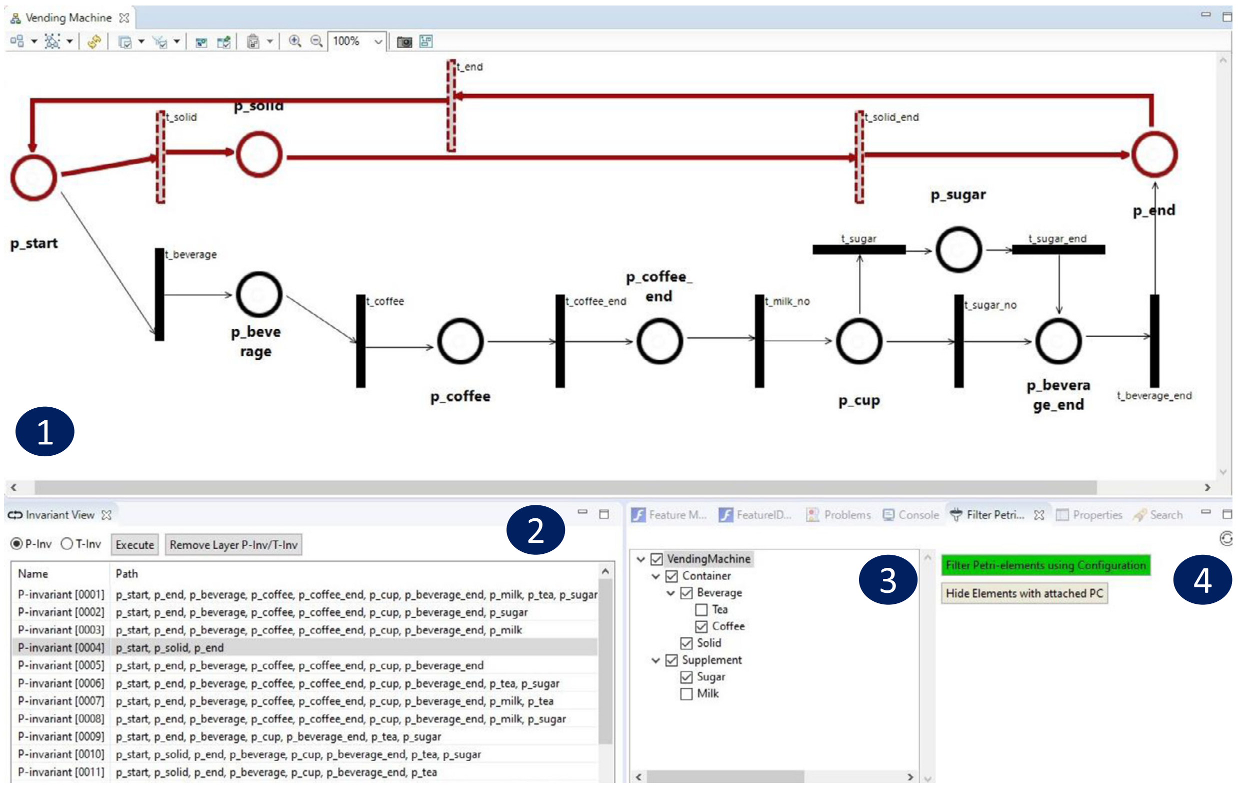
Fig. 8 shows Titan being used with the running example. This environment comprises a graphical editor (label 1 in the figure) and two views: the *Invariant View* (label 2) and the *Filter View* (labels 3 and 4). The *Invariant View* displays all the minimal P- and T-invariants of the PNPL identified by the constraint solver. In the figure, it shows the P-invariants for the running example. When selecting an invariant on this view, the corresponding connected subnet is highlighted on the canvas.

The *Filter View* (label 3) helps in the modelling phase of the PNPL and eases its comprehension. Upon selecting a (partial) configuration in the view, only the elements with non-false PC under the configuration are displayed. Another filter in this view (label 4) permits hiding the elements with attached PCs, so that only the common parts to all net variants remain visible. These filters help the designer of the PNPL to understand the effects of different feature selections on the resulting nets.

7. Evaluation

In this section, we evaluate Titan with regards to the following research questions (RQs):

- RQ1: *Does the lifted invariant analysis perform better than a case-by-case analysis?*
- RQ2: *What factors influence the most in the efficiency of the lifted invariant analysis?*



● P-Inv ○ T-Inv [Execute] [Remove Layer P-Inv/T-Inv]

Name	Path
P-invariant [0001]	p_start, p_end, p_beverage, p_coffee, p_coffee_end, p_cup, p_beverage_end, p_milk, p_tea, p_sugar
P-invariant [0002]	p_start, p_end, p_beverage, p_coffee, p_coffee_end, p_cup, p_beverage_end, p_sugar
P-invariant [0003]	p_start, p_end, p_beverage, p_coffee, p_coffee_end, p_cup, p_beverage_end, p_milk
P-invariant [0004]	p_start, p_solid, p_end
P-invariant [0005]	p_start, p_end, p_beverage, p_coffee, p_coffee_end, p_cup, p_beverage_end
P-invariant [0006]	p_start, p_end, p_beverage, p_coffee, p_coffee_end, p_cup, p_beverage_end, p_tea, p_sugar
P-invariant [0007]	p_start, p_end, p_beverage, p_coffee, p_coffee_end, p_cup, p_beverage_end, p_milk, p_tea
P-invariant [0008]	p_start, p_end, p_beverage, p_coffee, p_coffee_end, p_cup, p_beverage_end, p_milk, p_sugar
P-invariant [0009]	p_start, p_end, p_beverage, p_cup, p_beverage_end, p_tea, p_sugar
P-invariant [0010]	p_start, p_solid, p_end, p_beverage, p_cup, p_beverage_end, p_tea, p_sugar
P-invariant [0011]	p_start, p_solid, p_end, p_beverage, p_cup, p_beverage_end, p_tea

- ✓ VendingMachine
  - ✓ Container
    - ✓ Beverage
      - ☐ Tea
      - ✓ Coffee
    - ✓ Solid
  - ✓ Supplement
    - ✓ Sugar
    - ☐ Milk

[Filter Petri-elements using Configuration]
[Hide Elements with attached PC]

Fig. 8. Screenshot of the Titan environment.

To answer these RQs, we conducted two experiments. The first one considers variants of synthetic PNPLs, and the second one uses examples from the literature. In both experiments, the goal was to compare the time required to generate the P- and T- minimal invariants of all nets in the PNPL, on the one hand using standard invariant analysis algorithms on each net derived from the PNPL, and on the other hand applying our proposed lifted algorithm (i.e., considering all derivable nets at a time) which is based on Equation (1) and CSP. We call the first approach *case-by-case*, and the second one *lifted*.

More in detail, in the *case-by-case* approach, we measured the time required to generate and then analyse every derivable net of a PNPL one by one. For the analysis, we used two different algorithms for a better confidence on the comparison with our approach. As representative algorithms from Petri net tools, we adapted the APT library [43] – an open-source analysis tool for PNs and transition systems – to analyse the invariants of each net by means of the Farkas algorithm based on the incidence matrix [28,29], and also used a method proposed in the PIPE tool [44].

In the *lifted* approach, we measured the time of applying our lifted invariant analysis to the PNPL. In this case, we distinguished the time of the first execution of the analysis which discards cache effects (*cold start*) and subsequent executions of the same analysis (*warm cache*). For comparison, we took the worst time of our approach (which was always the cold start) against the best time of any of the two case-by-case analysis algorithms.

We performed the experiments on a Windows 10 machine with an Intel Core i7-9700 processor and 16 Gb of RAM memory. The data of the experiments, including the used models, are available at https://github.com/antoniogarmendia/titan.

7.1. Synthetic PNPLs

We have prepared two sets of experiments. The first set is directed to understand the benefits of our method when the size of the net or the number of configurations grow. The second set of experiments targets the exploration of corner cases, when the nets have many transitions and few places, many places and few transitions, or the products of the PNPL do not share elements.

7.1.1. Increasing 150% net size and number of configurations

The first experiment considers variants of a synthetic PNPL, which is inspired in a flexible manufacturing system described in [45,46]. This consists of three workstations, two part-receiving stations, one completed-parts station, and five automated guided vehicles. From the PN point of view, it is composed of 64 places, 53 transitions, and 255 arcs. The FM has 5 features which generates 36 valid configurations. From the base system description, we built several PNPLs considering two scenarios: (i) varying the number of parallel cells by replicating subnets of the PN model, while maintaining the

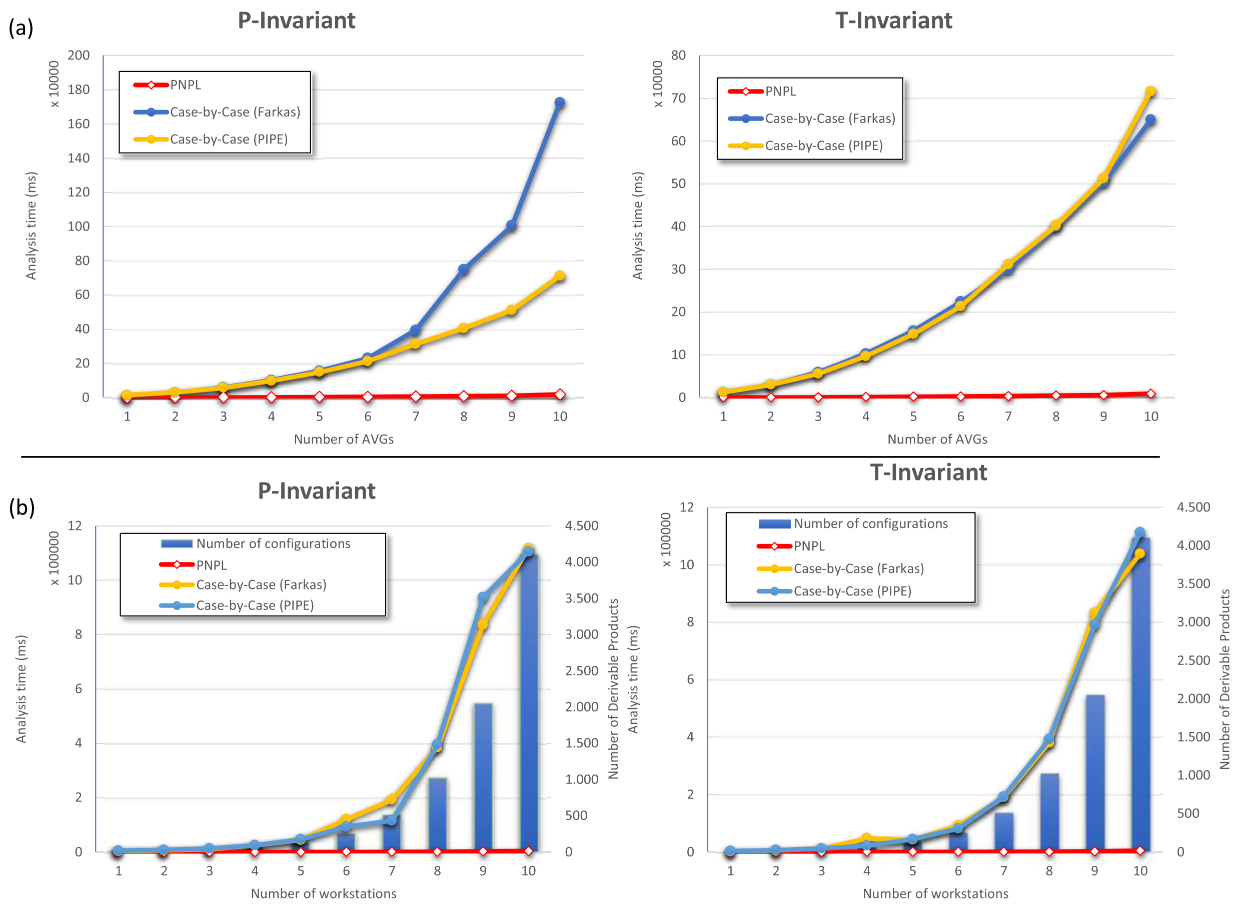


Fig. 9. Analysis time in ms: (a) Replicating the manufacturing cells; (b) Increasing the number of features (workstations).

number of features; and (ii) varying the number of workstations by adding them as new features and therefore increasing the number of feature configurations. That is, the first scenario considers PNPLs with 150% nets of increasing size but the same number of derivable PNs, while the second scenario considers PNPLs with increasing number of derivable nets.

Fig. 9(a) shows the analysis time in milliseconds (ms) for the first scenario. In all cases, the PNPLs have 36 configurations, and the lifted analysis (for both P- and T-invariants) is faster than analysing each derivable net case-by-case. Actually, the time improvement increases as the nets get bigger. Among the case-by-case analyses, Farkas exhibits the worst scalability for the case of P-invariants, while for T-invariants, both Farkas and PIPE require similar times.

Fig. 9(b) shows the analysis times for the second scenario, which considers PNPLs ranging from 12 to 4100 configurations. In all cases, the lifted invariant analysis was faster than analysing each net in isolation. All algorithms used in a case-by-case analysis suffer an exponential time increase, proportional to the number of derivable PNs (shown as blue bars in the graphics).

7.1.2. *Experimenting with corner cases*

To better understand the limits of our approach, we have created synthetic PNPLs with corner cases (many transitions and few places, many places and few transitions, and PNPLs with unrelated products and disjoint nets). Fig. 10 shows a scheme of the 150% PN of the prepared PNPLs. The net in Fig. 10(a) is composed of only three places but 240 transitions, to achieve a high transition to place ratio. Conversely, the net in Fig. 10(b) has 3 transitions and 240 places. Finally, the net in Fig. 10(c) comprises 80 disjoint subnets with 3 places and 3 transitions each.

Tables 3 and 4 summarise their analysis times for P- and T-invariants, respectively. For each corner case, the tables show the time of the first execution of the lifted analysis (*Lifted cold-start*), the time of a subsequent execution (*Lifted warm-cache*), the time for the case-by-case analysis using the two considered algorithms, and the speed-up of the cold-start lifted analysis with respect to the best time of the two case-by-case analyses. In the tables, we have highlighted in bold the worse time of the lifted analysis (always the cold start) and the best time of the case-by-case analyses.

The P-invariant analysis time is higher in nets with many places and few transitions, and conversely, the time to analyse T-invariants is higher when there are many transitions and few places. The case-by-case approach with Farkas is the one with the worst performance in all cases but in the analysis of T-invariants for nets with many places. Our lifted approach is

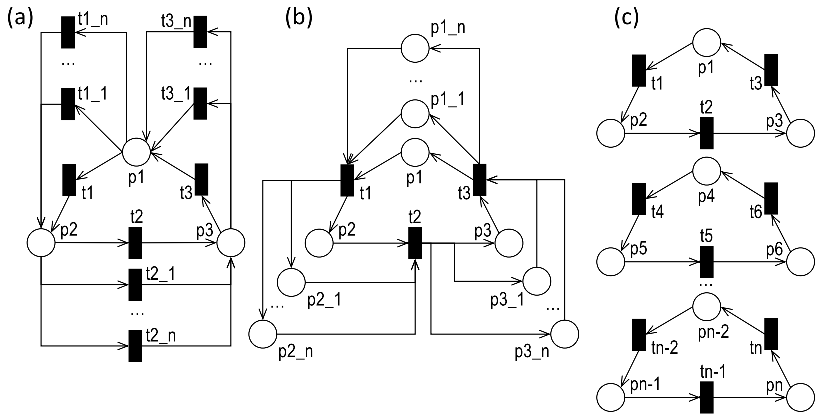


Fig. 10. Synthetic PNPLs modelling corner cases: (a) Net with high transition/place ratio; (b) Net with high place/transition ratio; (c) Disjoint nets.

Table 3
Execution time (in ms) for the P-invariant analysis of corner cases.

Model	Size $\lvert P\rvert \times \lvert T\rvert$	Number of features	Number of config.	Lifted cold-start	Lifted warm-cache	Case-by-case (Farkas)	Case-by-case (PIPE)	Speed-up
Many transitions	3 × 240	2	4	**182**	25	994	**887**	4.4
Many places	240 × 3	2	4	**576**	64	43310190	**607643**	1054.9
Disjoint	240 × 240	2	4	**439**	104	1511	**798**	1.8
Disjoint	240 × 240	5	31	**2105**	1632	110543	**109305**	51.9

Table 4
Execution time (in ms) for the T-invariant analysis of corner cases.

Model	Size $\lvert P\rvert \times \lvert T\rvert$	Number of features	Number of config.	Lifted cold-start	Lifted warm-cache	Case-by-case (Farkas)	Case-by-case (PIPE)	Speed-up
Many transitions	3 × 240	2	4	**3413**	3172	40968262	**182196**	53.4
Many places	240 × 3	2	4	**337**	12	**320**	371	0.9
Disjoint	240 × 240	2	4	**591**	104	715	**588**	1.0
Disjoint	240 × 240	5	31	**2166**	1624	109359	**109342**	50.5

the most efficient in the analysis of P-invariants, with speed-ups ranging from 1.8x (disjoint nets and few configurations) to 1054.9x (few transitions and many places).

Regarding T-invariant analysis, the lifted approach is the best performant in two of the cases, while in two others a case-by-case analysis is slightly better. In particular, in the net with many places and few transitions, Farkas is faster by 17 milliseconds (320 ms Farkas, 337 ms lifted cold-start), and in the case of disjoint nets, PIPE is faster by 3 milliseconds. However, we also observe that, when the PNPL grows in configurations (disjoint net with 31 configurations), our lifted approach becomes better again, with a speedup of 50.5x. For P-invariants, we also observe this efficiency gain as the number of configurations grows (from 1.8x to 51.9x).

7.2. PN models from the literature

The second set of experiments uses PN models with variability taken from the literature. We specified each of them as a 150% PN and a feature model using Titan. Thus, we can characterise the complexity of the PNPLs by two independent variables: the size of the 150% PN, and the number of feature configurations (i.e., the number of derivable nets). In our experiment, the size of the PNPLs ranges from 9 to 64 places and from 8 to 53 transitions, and their variability ranges from 3 to 324 configurations. To calculate the speed-up, we have used the minimum time of the case-by-case analyses.

As in the previous experiment, we analysed the invariants of the PNPLs using our lifted analysis, and also generating and analysing every derivable net using Farkas and PIPE. Tables 5 and 6 summarise the results for P- and T- invariant analysis, respectively. The lifted invariant analysis was faster than generating and analysing each net individually, by at least one order of magnitude. Speed-ups range from 1.5x to 266.6x, tending to increase as the number of configurations grows. There are some exceptions, though, most notably the P- and T-invariant analysis of the *flexible manufacturing cell* [45] which achieves a high speed-up (≥ 55) while having a medium number of configurations. We argue that this may be due to the large size of its 150% net. Finally, the speed-up value is more dependent on the number of configurations that on the 150%

Table 5
Execution time (in ms) for the P-invariant analysis of PN models from the literature.

Model	Size $\|P\| \times \|T\|$	Number of config.	Lifted cold start	Lifted warm cache	Case-by-case (Farkas)	Case-by-case (PIPE)	Speed-up
Phone system [47]	14 × 18	3	**92**	18	412	**335**	3.6
Philosophers' dinner [12]	12 × 8	4	**172**	11	**581**	700	3.4
Washing machine [48]	13 × 8	6	**76**	11	**582**	818	7.7
Vending machine [22]	11 × 15	21	**427**	33	**1882**	2287	4.4
Flexible manufacturing cell [45]	64 × 53	36	**197**	71	**10956**	13296	55.6
Film production [49]	15 × 17	48	**163**	37	**4238**	4609	26.0
Assembly line [17]	9 × 10	196	**115**	17	**1683**	2888	14.6
Extended vending machine [50]	18 × 26	324	**219**	69	58731	**58392**	266.6

Table 6
Execution time (in ms) for the T-invariant analysis of PN models from the literature.

Model	Size $\|P\| \times \|T\|$	Number of config.	Lifted cold start	Lifted warm cache	Case-by-case (Farkas)	Case-by-case (PIPE)	Speed-up
Phone system [47]	14 × 18	3	**107**	19	**158**	215	1.5
Philosophers' dinner [12]	12 × 8	4	**106**	15	**453**	502	4.3
Washing machine [48]	13 × 8	6	**83**	12	**500**	586	6.0
Vending machine [22]	11 × 15	21	**347**	38	1665	**1559**	4.5
Flexible manufacturing cell [45]	64 × 53	36	**204**	79	**12079**	12254	58.9
Film production [49]	15 × 17	48	**159**	36	**3944**	4061	24.8
Assembly line [17]	9 × 10	196	**114**	14	**1489**	1951	13.1
Extended vending machine [50]	18 × 26	324	**250**	88	146473	**56656**	226.6

net size. This is so as, for example, the two PNPLs with the smallest 150% nets have 96 (*philosophers' dinner* [12]) and 90 (*assembly line* [17]) elements in their incidence matrices, but the lifted analysis achieves very different speed-ups in each case (around 3.4 and 14.6 respectively), due to the different number of configurations (4 and 196). Interestingly, while in our previous experiment with synthetic corner cases (cf. Section 7.1.2), Farkas generally performed worse than PIPE, in this experiment using realistic nets, Farkas was faster than PIPE in more cases.

7.3. Discussion of results

Altogether, we conclude that the lifted invariant analysis is faster than a case-by-case one (RQ1). In the experiments, we observed speed-ups of at least one order of magnitude, and up to three orders in one corner case. The lifted analysis is especially suited for systems with large variability, that is, with a high amount of variants (RQ2). As Fig. 9(b) and Tables 5 and 6 show, speed-ups tend to increase with the number of configurations, while there is no correlation between the achieved speed-up and the number of places and transitions in the PNPL. For some corner cases (Tables 3 and 4) a case-by-case analysis yields slightly better results when there are few configurations, but the lifted analysis yields better results when the number of configurations grow. These results confirm those obtained in previous works for the lifted analysis of other structural properties [17].

7.4. Threats to validity

Regarding internal validity, we considered the time of the first lifted analysis execution to discard cache effects. For the case-by-case analysis, we used two algorithms (from the APT library) developed by a third party, to avoid any bias due to using our own implementation of the analysis.

Regarding external validity, we aimed to validate the results obtained with synthetic models by using models created by third parties. For this, we investigated existing literature. While we used examples of PNs with variability [17,45,47], others were adapted from related formalisms, like state machines [12,48] or process models [22,49]. We did our best in such translations, but we may have committed omissions. In any case, such a variety of systems' formalisms shows that the technique is also valid when PNs are used as a semantic domain for analysing other higher-level languages.

8. Related work

Next, we analyse related approaches in the area of Petri nets (Section 8.1), formal notations with variability (Section 8.2), and analysis of systems with variability (Section 8.3).

8.1. Petri nets

Some researchers have added static variability to PNs. Muschevici et al. [18,51] propose *Feature nets*, which are PNs (with no weighted arcs) where either arcs or transitions (but not both at the same time) have PCs. The authors propose different alternatives for their analysis, including their mapping into standard Petri nets, and into Featured Transition Systems (FTSs) [52]. In practice, they show how to use the mCRL2 tool set [53] to model check properties of feature nets. Heuer et al. [47] extend PNs (with no weighted arcs) with variability on arcs, and show how to map variable activity diagrams to them. For analysis, they use techniques based on the variable reachability graph. Compared to these, our variability notion is richer: we use feature models to express the configurations, every element of a PN can have variability (places, transitions and arcs), and our PN model is richer as well since it supports weighted arcs. Moreover, we focus on structural properties, avoiding the state explosion that behavioural properties may produce. As a trade-off, approaches based on model checking enable the analysis of a richer class of properties than the ones we support.

PNs have also been used to represent and analyse feature models [54]. Instead, we represent feature models as logic formulae, and use PNs as the notation over which variability is added.

PNPLs enable the static reconfiguration of nets. Other works also propose mechanisms for changing the structure of PNs. For example, Muschevici et al. [18] extend feature nets with dynamicity, so that transition firings can trigger the assignment of features, leading to a net reconfiguration. The analysis of dynamic feature nets is based on the variable reachability graph. Mobile PNs [55] were proposed to express distributed processes with changing structure at runtime. Places in mobile PNs represent channels, tokens are names for places, and the firing of transitions can add new transitions to the net, resulting in dynamic nets. Adaptive PNs [56] can enable or disable a subnet based on the amount of tokens in a set of places. This kind of PNs can then be flattened to regular PNs with inhibitor arcs. Reconfigurable PNs [57] combine PNs with rewriting systems to change the net structure at runtime. Our PNPLs enable the derivation of one net in the family via a configuration, but contrary to the previous approaches, configurations are static and cannot change at runtime.

Many Petri net tools exist nowadays (cf. [58] for a survey and [59] for a database of tools), and structural analysis is a popular analysis technique among them. According to the database [59] 23 out of 96 tools can calculate P- and T-invariants. The technique used for their calculation is typically based on the incidence matrix [44,43]. In our case, we lift this analysis to the product line level, and use constraint solving for finding the invariants.

8.2. Formal notations with variability

Several works have added variability to formal notations beyond Petri nets. For example FLan is a family of languages based on process algebra, which are enriched with variability [60]. The family includes probabilistic extensions (PFLan), and quantitative constraint modelling options (QFLan) [61]. This family of languages offers analysis tools for the product lines, including statistical model checkers, and Satisfiability Modulo Theory (SMT) solvers. Transition Systems have also been extended with variability, to yield Featured Transition Systems (FTSs) [52]. These propose analysis algorithms based one model checking, for which they introduce the featured linear temporal logic. Hence, these approaches allow a compact modelling of a large set of system variants, and lift analysis methods to the product line level for their analysis. Our approach follows the same philosophy, but we do not rely on model checking – which, even if supporting a richer class of analysis properties, they may suffer from the state-space explosion – but on structural techniques based on the incidence matrix.

We use annotations (PCs) over the net elements to represent points of variation. This approach, called negative variability [35], is a common technique used over other notations. Alternatively, we could have relied on composition-based mechanisms, combining parts of the net on-demand depending on the chosen configuration [62]. For this purpose, we could have used different notions of PN modules [63,64]. However, our lifted invariant analysis benefits from having an explicit representation of the overlay of all possible net variants in the 150% net.

8.3. Analysis mechanisms for systems with variability

Generally, several methods to analyse variable systems have been proposed [65]. A lifted approach (also called family-based, or variability-aware) like the one we propose modifies an existing analysis (e.g., based on the incidence matrix) to make it aware of the product line variability. We can find many lifted approaches for different notations, like automata [12,66], UML/OCL models [67] or meta-models [34]. These works lift techniques like model checking [66], model finding [67,34], or invariant synthesis [12]. In the latter case, the authors derive invariants for parameterised architectures (finite automata enabling replication of components) by an embedding into monadic second order logic. Our contribution to this type of analyses consists in lifting invariant analyses for PNs based on the incidence matrix. To our knowledge, this is a novel technique.

Other methods to analyse variable systems [65] include using sampling (analysing samples of the set of products) [68], or generating and analysing all products. In our case, the latter approach leads to worse times that greatly increase with the number of variants, as we have seen in Section 7. A sample-based approach would not be adequate either, since we could miss invariants for nets not belonging to the analysed sample.

9. Conclusions and future work

In this paper, we have presented a product line-based approach for the compact definition of variants of a given Petri net. In our previous works [16,17], we introduced the concept of PNPL and carried out analysis using SAT solving. In this paper, we have lifted analysis techniques based on the incidence matrix to the product line level, and realised them using constraint programming. Our experiments show the efficiency of our lifted analysis, with experiments using synthetic examples and examples developed by third parties, which may achieve speed-ups of two orders of magnitude with respect to analysing each net variant separately.

As future work, we would like to exploit the system of equations as a basis for other analysis, like finding the *best* invariant (for some notion of *best* using some goal function that can be used within the constraint solver); products with (without) P/T-invariants; invariants belonging to all products; the smallest set of features that can produce an invariant; or the *best* set of features with (or without) P/T-invariants. For some of these analyses we may resort to partial configurations (e.g., in the style of [34]). In addition, we would like to lift other types of analyses, for example based on reduction techniques [1]. Moreover, we plan to support other Petri net extensions in product lines, like time, and lift the corresponding analysis techniques accordingly. We are currently working on dynamic reconfiguration of the Petri net [18], for which we have developed an initial mapping of the feature model and the 150% net into coloured Petri nets [69]. Finally, the availability of common benchmarks for variability-enhanced formalisms is important to enable fair comparison and experimentation of different approaches and tools. Hence, we plan to build a repository of cases of systems with variability, but this is certainly a challenge to be tackled by the community as a whole.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

We would like to thank the anonymous reviewers for their useful comments, which helped to improve a previous version of this manuscript. This work has been funded by the Spanish Ministry of Science (PID2021-122270OB-I00) and the R&D programme of Madrid (P2018/TCS-4314).

References

[1] T. Murata, Petri nets: properties, analysis and applications, Proc. IEEE 77 (4) (1989) 541–580.
[2] Y. Feng, K. Xing, M. Zhou, X. Wang, H. Liu, Robust deadlock prevention for automated manufacturing systems with unreliable resources by using general Petri nets, IEEE Trans. Syst. Man Cybern. Syst. 50 (10) (2020) 3515–3527.
[3] N. Wu, F. Chu, C. Chu, M. Zhou, Petri net-based scheduling of single-arm cluster tools with reentrant atomic layer deposition processes, IEEE Trans. Autom. Sci. Eng. 8 (1) (2011) 42–55, https://doi.org/10.1109/TASE.2010.2046736.
[4] R. Wang, L.M. Kristensen, H. Meling, V. Stolz, Automated test case generation for the paxos single-decree protocol using a coloured Petri net model, J. Log. Algebraic Methods Program. 104 (2019) 254–273, https://doi.org/10.1016/j.jlamp.2019.02.004.
[5] W. van der Aalst, C. Stahl, Modeling Business Processes: A Petri Net-Oriented Approach, The MIT Press, 2011.
[6] T. Brant-Ribeiro, R. Araujo, I. Mendonça, M. Soares, R. Cattelan, Interactive web interfaces modeling, simulation and analysis using colored Petri nets, Softw. Syst. Model. 18 (1) (2019) 721–737.
[7] A. Brogi, A. Canciani, J. Soldani, P. Wang, A Petri net-based approach to model and analyze the management of cloud applications, Trans. Petri Nets Other Model. Concurr. 11 (2016) 28–48.
[8] A. Philippou, K. Psara, Reversible computation in nets with bonds, J. Log. Algebraic Methods Program. 124 (2022) 100718, https://doi.org/10.1016/j.jlamp.2021.100718.
[9] D. Kozma, P. Varga, F. Larrinaga, Dynamic multilevel workflow management concept for industrial IoT systems, IEEE Trans. Autom. Sci. Eng. 18 (3) (2021) 1354–1366, https://doi.org/10.1109/TASE.2020.3004313.
[10] B. Meyers, S.V. Mierlo, D. Maes, H. Vangheluwe, Efficient software controller variant development and validation (ECoVaDeVa) overview of a Flemish ICON project, in: STAF Co-Located Events, in: CEUR, vol. 2405, 2019, pp. 49–54.
[11] H. Nabi, T. Aized, Modeling and analysis of carousel-based mixed-model flexible manufacturing system using colored Petri net, Adv. Mech. Eng. 11 (12) (2019) 1–14.
[12] M. Bozga, J. Esparza, R. Iosif, J. Sifakis, C. Welzel, Structural invariants for the verification of systems with parameterized architectures, in: Proc. TACAS Part I, in: LNCS, vol. 12078, Springer, 2020, pp. 228–246.
[13] S. García, D. Strüber, D. Brugali, A.D. Fava, P. Schillinger, P. Pelliccione, T. Berger, Variability modeling of service robots: experiences and challenges, in: Proc. Workshop on Variability Modelling of Software-Intensive Systems, VAMOS, ACM, 2019, pp. 8:1–8:6.
[14] L. Northrop, P. Clements, Software Product Lines: Practices and Patterns, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[15] K. Pohl, G. Böckle, F.J. van der Linden, Software Product Line Engineering. Foundations, Principles and Techniques, Springer-Verlag, Berlin, Heidelberg, 2005.
[16] E. Gómez-Martínez, J. de Lara, E. Guerra, Towards extensible structural analysis of Petri net product lines, in: Proc. PNSE, in: CEUR Workshop Proceedings, vol. 2424, CEUR-WS.org, 2019, pp. 37–46.

[17] E. Gómez-Martínez, J. de Lara, E. Guerra, Extensible structural analysis of Petri net product lines, Trans. Petri Nets Other Model. Concurr. XV (12530) (2021) 1–23.
[18] R. Muschevici, J. Proença, D. Clarke, Feature nets: behavioural modelling of software product lines, Softw. Syst. Model. 15 (4) (2016) 1181–1206.
[19] K. Lautenbach, H. Ridder, Liveness in bounded Petri nets which are covered by t-invariants, in: R. Valette (Ed.), Application and Theory of Petri Nets 1994, 15th International Conference, in: Lecture Notes in Computer Science, vol. 815, Springer, 1994, pp. 358–375.
[20] K. Schmidt, Using Petri net invariants in state space construction, in: Proc. of Tools and Algorithms for the Construction and Analysis of Systems TACAS, in: LNCS, vol. 2619, Springer, 2003, pp. 473–488.
[21] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, Elsevier, 2006.
[22] M. Çağri Kaya, S. Suloglu, G. Tokdemir, B. Tekinerdogan, A.H. Dogru, Variability incorporated simultaneous decomposition of models under structural and procedural views, in: Software Engineering for Variability Intensive Systems, Auerbach Publications/Taylor & Francis, 2019, pp. 95–116.
[23] A. Finkel, The minimal coverability graph for Petri nets, in: Applications and Theory of Petri Nets, in: LNCS, vol. 674, Springer, 1991, pp. 210–243.
[24] T. Murata, State equation, controllability, and maximal matchings of Petri nets, IEEE Trans. Autom. Control 22 (3) (1977) 412–416.
[25] P. Cousot, N. Halbwachs, Automatic discovery of linear restraints among variables of a program, in: Proc. POPL, ACM Press, 1978, pp. 84–96.
[26] M.P. Cabasino, A. Giua, C. Seatzu, Structural analysis of Petri nets, in: C. Seatzu, M. Silva, J.H. van Schuppen (Eds.), Control of Discrete-Event Systems, in: Lecture Notes in Control and Information Sciences, vol. 433, Springer, 2013, pp. 213–233.
[27] K.S.M. Heiner, Structural analysis to determine the core of hypoxia response network, Supplementary material, PLoS ONE 5 (2010).
[28] J. von Farkas, Theorie der einfachen ungleichungen, J. Reine Angew. Math. (1902) 1–27.
[29] J.M. Colom, M.S. Suárez, Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal p-semiflows, in: 10th International Conference on Applications and Theory of Petri Nets, in: Lecture Notes in Computer Science, vol. 483, Springer, 1989, pp. 79–112.
[30] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-Oriented Domain Analysis (FODA) feasibility study, Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.
[31] A. Schlie, S. Schulze, I. Schaefer, Recovering variability information from source code of clone-and-own software systems, in: Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems, VAMOS '20, Association for Computing Machinery, New York, NY, USA, 2020.
[32] D. Beuche, M. Schulze, M. Duvigneau, When 150 centric viewpoints in an industrial product line, in: Proceedings of the 20th International Systems and Software Product Line Conference, SPLC'16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 262–269.
[33] D. Reuling, C. Pietsch, U. Kelter, T. Kehrer, Towards projectional editing for model-based SPLS, in: Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems, VAMOS '20, Association for Computing Machinery, New York, NY, USA, 2020.
[34] E. Guerra, J. de Lara, M. Chechik, R. Salay, Property satisfiability analysis for product lines of modelling languages, IEEE Trans. Softw. Eng. 48 (2) (2022) 397–416.
[35] K. Czarnecki, M. Antkiewicz, Mapping features to models: a template approach based on superimposed variants, in: Proc. GPCE, in: LNCS, vol. 3676, Springer, 2005, pp. 422–437.
[36] M. Fränzle, C. Herde, Efficient SAT engines for concise logics: accelerating proof search for zero-one linear constraint systems, in: Proc. LPAR, in: LNCS, vol. 2850, Springer, 2003, pp. 302–316.
[37] Sirius, https://www.eclipse.org/sirius/.
[38] J. Meinicke, T. Thüm, R. Schröter, F. Benduhn, T. Leich, G. Saake, Mastering Software Variability with FeatureIDE, Springer, 2017.
[39] D. Benavides, P.T. Martín-Arroyo, A.R. Cortés, Automated reasoning on feature models, in: Proc. CAiSE, in: LNCS, vol. 3520, Springer, 2005, pp. 491–503.
[40] K.N. Brown, I. Miguel, Chapter 21 - Uncertainty and change, in: Handbook of Constraint Programming, in: Foundations of Artificial Intelligence, vol. 2, Elsevier, 2006, pp. 731–760.
[41] K. Kuchcinski, R. Szymanek, JaCoP - Java constraint programming solver, in: CP Solvers: Modeling, Applications, Integration, and Standardization, 2013.
[42] D.L. Berre, A. Parrain, The Sat4j library, release 2.2, J. Satisf. Boolean Model. Comput. 7 (2–3) (2010) 59–64.
[43] E. Best, U. Schlachter, Analysis of Petri nets and transition systems, Electron. Proc. Theor. Comput. Sci. 189 (2015) 53–67, https://doi.org/10.4204/eptcs.189.6.
[44] N.J. Dingle, W.J. Knottenbelt, T. Suto, PIPE2: a tool for the performance evaluation of generalised stochastic Petri nets, ACM SIGMETRICS Perform. Eval. Rev. 36 (4) (2009) 34–39.
[45] Y. Li, W.M. Wonham, Control of vector discrete-event systems. II. Controller synthesis, IEEE Trans. Autom. Control 39 (3) (1994) 512–531, https://doi.org/10.1109/9.280750.
[46] K. Yamalidou, J.O. Moody, M.D. Lemmon, P.J. Antsaklis, Feedback control of Petri nets based on place invariants, Automatica 32 (1) (1996) 15–28.
[47] A. Heuer, V. Stricker, C.J. Budnik, S. Konrad, K. Lauenroth, K. Pohl, Defining variability in activity diagrams and Petri nets, Sci. Comput. Program. 78 (12) (2013) 2414–2432.
[48] R. Salay, M. Famelis, J. Rubin, A.D. Sandro, M. Chechik, Lifting model transformations to product lines, in: Proc. ICSE, ACM, 2014, pp. 117–128.
[49] M.L. Rosa, W.M.P. van der Aalst, M. Dumas, F. Milani, Business process variability modeling: a survey, ACM Comput. Surv. 50 (1) (2017) 2:1–2:45.
[50] A. Sree-Kumar, E. Planas, R. Clarisó, Analysis of feature models using alloy: a survey, in: Proc. FMSPLE, in: EPTCS, vol. 206, 2016, pp. 46–60.
[51] R. Muschevici, J. Proença, D. Clarke, Modular modelling of software product lines with feature nets, in: Proc. SEFM, in: LNCS, vol. 7041, Springer, 2011, pp. 318–333.
[52] A. Classen, M. Cordy, P. Schobbens, P. Heymans, A. Legay, J. Raskin, Featured transition systems: foundations for verifying variability-intensive systems and their application to LTL model checking, IEEE Trans. Softw. Eng. 39 (8) (2013) 1069–1089.
[53] S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, W. Wesselink, T.A.C. Willemse, An overview of the mcrl2 toolset and its recent advances, in: N. Piterman, S.A. Smolka (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS, in: Lecture Notes in Computer Science, vol. 7795, Springer, 2013, pp. 199–213.
[54] C. Martínez, N. Díaz, S. Gonnet, H. Leone, A Petri net variability model for software product lines, Electron. J. SADIO 13 (2014) 35–53.
[55] A. Asperti, N. Busi, Mobile Petri nets, Math. Struct. Comput. Sci. 19 (6) (2009) 1265–1278, https://doi.org/10.1017/S0960129509990193.
[56] C. Mai, R. Schöne, J. Mey, T. Kühn, U. Assmann, Adaptive Petri nets: a Petri net extension for reconfigurable structures, in: Proc. ADAPTIVE, Springer, 2018.
[57] M. Llorens, J. Oliver, Structural and dynamic changes in concurrent systems: reconfigurable Petri nets, IEEE Trans. Comput. 53 (9) (2004) 1147–1158.
[58] W.J. Thong, M.A. Ameedeen, A survey of Petri net tools, in: H.A. Sulaiman, M.A. Othman, M.F.I. Othman, Y.A. Rahim, N.C. Pee (Eds.), Advanced Computer and Communication Engineering Technology, Springer International Publishing, Cham, 2015, pp. 537–551.
[59] Petri Nets World, Data base of Petri net tools, https://www.informatik.uni-hamburg.de/TGI/PetriNets/index.php.
[60] M.H. ter Beek, A. Legay, A. Lluch-Lafuente, A. Vandin, Statistical model checking for product lines, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, Proceedings, Part I, in: Lecture Notes in Computer Science, vol. 9952, 2016, pp. 114–133.
[61] M.H. ter Beek, A. Legay, A. Lluch-Lafuente, A. Vandin, A framework for quantitative modeling and analysis of highly (re)configurable systems, IEEE Trans. Softw. Eng. 46 (3) (2020) 321–345.
[62] S. Apel, D.S. Batory, C. Kästner, G. Saake, Feature-Oriented Software Product Lines - Concepts and Implementation, Springer, 2013.

[63] J. Padberg, H. Ehrig, Petri net modules in the transformation-based component framework, J. Log. Algebraic Methods Program. 67 (1–2) (2006) 198–225.
[64] E. Kindler, L. Petrucci, Towards a standard for modular Petri nets: a formalisation, in: Applications and Theory of Petri Nets, Springer, Berlin, Heidelberg, 2009, pp. 43–62.
[65] T. Thüm, S. Apel, C. Kästner, I. Schaefer, G. Saake, A classification and survey of analysis strategies for software product lines, ACM Comput. Surv. 47 (1) (2014) 6:1–6:45.
[66] A.S. Dimovski, A. Wasowski, Variability-specific abstraction refinement for family-based model checking, in: Proc. FASE, in: LNCS, vol. 10202, Springer, 2017, pp. 406–423.
[67] K. Czarnecki, K. Pietroszek, Verifying feature-based model templates against well-formedness OCL constraints, in: Proc. GPCE, ACM, 2006, pp. 211–220.
[68] S. Apel, A. von Rhein, P. Wendler, A. Größlinger, D. Beyer, Strategies for product-line verification: case studies and experiments, in: Proc. ICSE, IEEE Computer Society, 2013, pp. 482–491.
[69] E. Gómez-Martínez, E. Guerra, J. de Lara, Analysing product lines of concurrent systems with coloured Petri nets, in: 34th International Conference on Software Engineering & Knowledge Engineering, SEKE, Springer, 2022, pp. 118–123.