2014 21st Asia-Pacific Software Engineering Conference

Model Checking of Software Product Lines in Presence of Nondeterminism and Probabilities

Mahsa Varshosaz
School of Electrical and Computer Engineering,
College of Engineering, University of Tehran, Tehran, Iran
Email: m.varshosaz@ut.ac.ir

Ramtin Khosravi
School of Electrical and Computer Engineering,
College of Engineering, University of Tehran, Tehran, Iran
Email: r.khosravi@ut.ac.ir

***Abstract*—Nowadays, Software Product Lines (SPLs) are being used in a variety of domains including safety-critical systems for which verification of the systems is a matter of concern. Formal modeling and verification of SPLs has been majorly investigated recently. Due to the potential large number of the products in a SPL, individual verification of all products could be costly or even impractical. Hence, there is a need for verification methods that can verify the whole family's behavior at once. In this paper, we focus on the probabilistic model checking of software product lines in which the behavior of individual products can be described in terms of Markov decision processes. We introduce a mathematical model, Markov Decision Process Family (MDPF), to compactly represent the behavior of the whole family. We also provide a model checking algorithm in order to verify MDPFs against properties expressed in probabilistic computational tree logic.**

Keywords*— *Software Product Line, Markov Decision Process, Probabilistic Model Checking, Markov Decision Process Family.

I. Introduction

Software Product Line (SPL) engineering deals with developing a set of related software systems (families of products) within an specific domain. Software product line engineering enables systematic reuse by explicitly defining commonalities and variabilities between the products in a family. Thus, using SPLs results in higher development quality and reduce of cost and time to market [1]. In an SPL, different aspects of the products are expressed by means of *features*. Features may be of different types such as *mandatory* or *optional* and can be related to each other via different types of relationships (described in Sect. II-A in more details). As SPL engineering is applied in a variety of domains including the ones containing safety critical systems such as avionics and automotive, verification of SPLs is an important subject to be considered. Many of the products in the mentioned domains exhibit probabilistic and nondeterministic behavior which must be addressed by the verification method. In this work, we study the verification of SPLs exhibiting such behavior. As a result, we propose a method which combines *SPL verification* techniques and *probabilistic model checking* methods.

There have been different techniques introduced to verify software product lines. The simplest way to verify a SPL is to verify all the products in the family individually but this solution may sometimes be very costly or even impractical due to the potentially large number of the products. So, to overcome this challenge there have been investigations for methods to verify the whole family altogether [2], [3], [4], [5]. In these methods the behavior of the whole family is described by a single model and the properties are checked against this model at once.

Probabilistic model checking is a promising method used to verify different quantitative and qualitative properties of the systems. In this technique, the behavior of the system is usually described in terms of a Markovian structure (Sect. II-B). These mathematical structures are capable of modeling the probabilities and nondeterminism in the system behavior. The quantitative properties of the systems can be described using one of the existing formalisms such as Probabilistic Computation Tree Logic (PCTL) [6] and then verified against the model.

We have previously introduced a technique to verify SPLs in which each product can be represented by a Discrete Time Markov Chain (DTMC) [7]. However, the fact that DTMC cannot capture nondeterminism in the system's behavior limits the expressiveness of the proposed model. In this paper, we take nondeterminism into account and address SPLs in which the behavior of each product can be described by a Markov Decision Process (MDP) [8]. With the aim of exploiting the commonalities among the different products in the system, we present a method to verify the whole software family against a PCTL formula at once, instead of building the model for each product and verifying them one by one. To this end, we introduce a new mathematical model, Markov Decision Process Family (MDPF), which represents the whole family behavior compactly (Sect. IV), by annotating each transition with a condition specifying the subset of the products in which taking that transition is possible. We also define the parallel composition of MDPFs, which makes it possible to construct the model of large-scale systems by composing the models of their comprised components. Then, we provide a probabilistic model checking algorithm to verify MDPFs against PCTL properties (Sect. V). Our algorithm follows the general framework for model checking MDPs [8], but when exploring various executions of the system, it keeps track of various subsets of the products leading to different evaluations of the properties. Finally, the algorithm reports the subset of the products satisfying the given property.

This paper differs from our previous work in [7], not only in handling nondeterminism and introducing parallel composition, but also in presenting the model-checking method based on a matrix-based notation using custom-defined summation

1530-1362/14 $31.00 © 2014 IEEE
DOI 10.1109/APSEC.2014.18

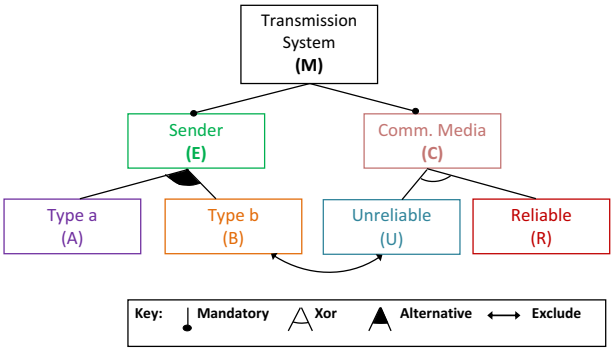

Fig. 1. Feature model of a transmission system product line.

and multiplication operators. The new notation is more general, more structured, and easier to reason about.

II. Preliminaries

This section includes a brief overview of some basic constructs which our work is based on. In Sect. II-A we explain some basic concepts related to SPLs and in Sect. II-B we cover the required primitives about probabilistic model checking.

A. Basic SPL Concepts

One of the basic concepts in the context of software product lines is *feature*. A feature can be described as a visible and distinctive aspect or quality of a system [9]. A software product line is comprised of a set of products that have commonalities and variabilities regarding to a set of features. Hence, a certain product can be distinguished by a subset of features. There are different types of relationships between the features of a product line which are commonly visualized by means of *feature models* [10]. A feature model is a hierarchical structure consisting of features connected to each other via different relationships. A feature may have sub-features which can be *optional* or *mandatory*. A feature can also have *alternative* or *xor* relationship with its sibling features. One or more of the alternative features and at most one of the features involved in a xor relation can be included in each product. A feature f can also have *requires* and *excludes* relationships with other features which respectively imply inclusion and exclusion of other features from the products containing f. Fig. 1 represents a feature model of a simple transmission system family. In this feature model the root feature M has two mandatory sub-features E and C. Feature C has two sub-features with xor relationship and feature E has two sub-features with or relationship. There is also a cross tree relationship indicating that U and B exclude each other.

Each feature model can be formulated as a propositional logic formula. In this formula each boolean variable corresponds to a feature and its value indicates if the feature is included or excluded in a specific product [11]. In this paper, we represent the boolean variables corresponding to a feature with the same symbol for the feature (e.g. C denotes the boolean variable corresponding to feature C). The formula includes conjunctions between a set of implications. These implications are from every parent to its mandatory sub-features (e.g. $\mathsf{M} \rightarrow \mathsf{C}$), every sub-feature to its parent feature, every parent to or/xor of its sub-features that have or/xor relationship, every feature to other features that has *require* relationship with and every feature to the negation of the features that has *exclude* relationship with.

Regarding the above explanations, a feature model can be fully represented by a pair (F, Φ_F) where F is the set of features of the feature model and Φ_F is the corresponding propositional formula.

Configuration is another important concept that captures the inclusion or exclusion of features in a specific product. A configuration c is a pair like (I_c, E_c) where I_c and E_c are disjoint subsets of F respectively indicating the included and excluded features. The configuration c can also be represented by the propositional formula $(\bigwedge_{f \in I_c} f) \wedge (\bigwedge_{f \in E_c} \neg f)$. A configuration is called valid with regards to a feature model $\mathcal{F} = (F, \Phi_F)$ if it satisfies all the constraints imposed by $\mathcal{F}$ (e.g. it does not exclude mandatory features). A valid configuration is called complete if $I_c \cup E_c = F$ and partial otherwise. Each complete configuration represents a certain product. In this paper we use $C_{\mathcal{F}}$ to denote the set of all valid configurations regarding to the feature model $\mathcal{F}$.

Let Ψ_F be the set of all possible propositional formulas over the set of feature variables F. The *projection* of a propositional formula $\psi \in \Psi_F$ over a configuration c denoted by $\psi|_c$, is defined as the substitution of each feature variable f in I_c (resp. E_c) with *true* (resp. *false*) in the propositional formula. A configuration c satisfies a propositional formula ψ if SAT($\psi|_c$), where SAT(_) denotes the satisfiability of a propositional logic formula. We use Ψ_F^c to denote the set of propositional formulas in Ψ_F satisfiable by c.

Annotating the model is one of the common techniques in representing variabilities. One way to express the annotations is to use propositional formulas over the feature set F, called *application conditions*. A part of the model annotated by the application condition ψ will be included only in the products whose corresponding configuration satisfies ψ.

Assuming two application conditions ψ and ψ' regarding to a product line, we define a set of auxiliary relations. We say ψ is weaker than ψ' if the set of products satisfying ψ is a superset of the same set for ψ'. This relation is represented using the operator $\preceq$ such that:

$$\psi' \preceq \psi \quad \textit{iff} \quad \{c \in C_{\mathcal{F}} | SAT(\psi'|_c)\} \subseteq \{c \in C_{\mathcal{F}} | SAT(\psi|_c)\}$$

For example, $\mathsf{A} \wedge \mathsf{B} \preceq \mathsf{A}$. We also define the *equivalent* and *exclusive* [7] relations respectively as:

$$EQ(\psi, \psi') \triangleq (\psi \preceq \psi') \wedge (\psi' \preceq \psi)$$

$$EX(\psi, \psi') \triangleq \nexists c \in C_{\mathcal{F}} \cdot \; SAT(\psi|_c \wedge \psi'|_c)$$

B. Probabilistic Model Checking

As mentioned before in this paper we consider product lines in which the behavior of the products can be described in terms of Markov decision processes. MDPs can be seen as variants of Markov chains that make it possible to simultaneously model the probabilistic and non-deterministic behavior. In each state of an MDP there can be several enabled actions

which can be chosen non-deterministically and each action leads to successor states with probabilities according to some specified distribution. In other words each action represents a distribution determining the probability of transitions to the successor states. The formal definition of an MDP taken from [8] is as follows.

Definition 1: (Markov Decision Process) A Markov decision process $\mathcal{M}$ is a tuple $(S, s_{init}, P, Act, AP, L)$ where:

- S is the set of states and s_{init} is the set of initial states.
- $P : S \times Act \times S \to [0, 1]$ is the transition probability function such that for each state $s \in S$ and action $\alpha \in Act$:
$$\sum_{s' \in S} P(s, \alpha, s') \in \{0, 1\}$$
- Act is a set of actions, AP is a set of atomic propositions and $L : S \to [0, 1]$ is a labeling function.

An infinite path in $\mathcal{M}$ is a sequence like $s_0\alpha_1 s_1\alpha_2 \cdots$ such that $\forall i \geq 0$ we have $P(s_i, \alpha_{i+1}, s_{i+1}) > 0$. The finite paths can be defined accordingly. We use $Act(s)$ to denote the set of actions enabled in state s. In order to reason about the probabilities of different sets of paths in MDPs, it is necessary to resolve the non-determinism in all the states. *Schedulers* are common means to resolve the non-determinism in MDPs. In other terms, each scheduler determines a subset of the paths of the MDP.

There are various formalisms to specify quantitative and qualitative properties of probabilistic systems. PCTL [6] is a well-known temporal logic used to express such properties. The syntax of PCTL formulas is categorized to state formulas and path formulas. These formulas are defined over a set of atomic propositions *AP*. State formulas represent properties regarding to the states of the system, based on the following grammar:

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \mathbb{P}_{\sim J}(\varphi)$$

where $a \in AP$, $\sim \in \{\leq, <, >, \geq\}$, $J \in [0, 1]$, and φ is a path formula. The $\mathbb{P}$ operator is used to specify probability bounds for the satisfaction of path formulas. Path formulas concern the properties over the paths of the system and are defined by the following grammar:

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 U \Phi_2 \mid \Phi_1 U^{\leq n} \Phi_2$$

where Φ, Φ_1, and Φ_2 are state formulas.

The operators $\bigcirc$ and U denote *next* and *until* operators and has the same meaning as in CTL. The *bounded until* operator $U^{\leq n}$ is just the same as until operator with only a bound on the number of the computation steps. For example $\Phi_1 U^{\leq n} \Phi_2$ means that Φ_2 will be satisfied in less than n steps and till then, Φ_1 stands true.

III. Running Example

In this section we describe a simple transmission system product line which is used as our running example in the rest of this paper. Generally a system in this product line includes one or two senders to send messages via a communication media (for the sake of simplicity we only model the sender side of the system). We consider two different variabilities in the products in this product line: (1) the type of sender, (2) the reliability of the communication media. Fig. 1 represents the feature model regarding to this product line.

Here, we assume there are two type of senders, *a* and *b*, that can be included in the products. In case that there is a message to be sent, both types of senders first decide to either send the message or wait. In sender *a* the decision of sending the message is nondeterministic while sender *b* sends the message with probability 0.8 and it may wait with probability 0.2. A product can contain one or both of the senders. If a product contains both senders, a message is sent by one of the senders nondeterministically.

As mentioned above, the reliability of the communication media is assumed as another variation point. The communication media may be reliable which means that a message will be sent successfully with probability one or it can be unreliable. Here, a message passed through an unreliable communication media will be sent successfully with probability 0.7 and may be lost with probability 0.3. We also consider an exclude relation in our product line such that an unreliable communication media excludes sender type *b* which means that it can be used in products that only include sender type *a*.

IV. Markov Decision Process Families

In this section we propose the formal definition of a Markov decision process family. The general idea of using MDPFs is to compactly represent the behavior of the whole family. As we mentioned before in this paper we cover the modeling of SPLs, in which the behavior of each product can be described in terms of a MDP. We extend the structure of MDPs by annotating transitions with application conditions (Sect. II-A). The structure of a MDPF is defined as follows.

Definition 2: (Markov decision process family) A Markov decision process family $\mathcal{M}$ defined over the feature model $\mathcal{F} = (F, \Phi_F)$, is a tuple $(S, s_{init}, P_F, Act, AP, L)$ where:

- S, s_{init}, Act, AP and L are defined as the same as in Def 1.
- P_F: $S \times Act \times \Psi_{\mathcal{F}} \times S \to$[0,1] is the (partial) transition probability function satisfying conditions (1) and (2) explained in the following.

Figure 2, represents the MDPF of the transmission system product line in our running example (we use dotted arcs to demonstrate transitions with the same action labels). The first condition concerns the sum of the probabilities of the transitions with same action labels which can be enabled in a same product. For each state $s \in S$, action $\alpha \in Act$ and complete configuration $c \in C_{\mathcal{F}}$:

$$\sum_{\substack{s' \in S \\ \psi \in \Psi_{\mathcal{F}}^c}} P_F(s, \alpha, \psi, s') \in \{0, 1\} \quad (1)$$

We assume that for each state s and action $\alpha \in Act(s)$, Υ_s^α denotes the set of application conditions belonging to the set of outgoing transitions from s with action label α. For example in MDPF in Fig. 2, $\Upsilon_{s_1}^{try} = \{\mathsf{A}, \mathsf{B}\}$. The second condition indicates that for each state s, all the application conditions of

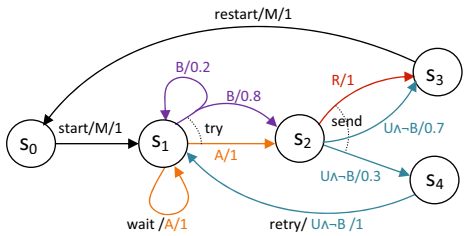

Fig. 2. MDPF describing the transmission system product line. The dotted arcs connect transitions with the same action.

the outgoing transitions with the same action are either equal or exclusive pairwise. That is $\forall s \in S,\ \forall \alpha \in Act(s)$:

$$\forall \psi, \psi' \in \Upsilon_s^{\alpha} \cdot \quad EQ(\psi, \psi') \vee EX(\psi, \psi') \tag{2}$$

These two conditions are imposed to preserve the behavior of each product as a MDP. An infinite path π in $\mathcal{M}$ is a sequence $s_0(\alpha_1, \psi_1)s_1(\alpha_2, \psi_2)\cdots$ such that for all $i \geq 1$, $P_F(s_{i-1}, \alpha_i, \psi_i, s_i) > 0$ and SAT($\bigwedge_{i>0} \psi_i$). This condition over the paths is defined due to the fact that there are sequences as above which include exclusive application conditions and hence do not describe a behavior in any of the products. In the MDPF in Fig. 2, $s_0(start, \mathsf{M})(s_1(try, \mathsf{B}))^{\omega}$ is an infinite path. By $Paths_{fin}(\mathcal{M})$ (resp. $Paths(\mathcal{M})$) we mean the set of finite (resp. infinite) paths in $\mathcal{M}$. A path $\pi = s_0(\alpha_1, \psi_1)s_1(\alpha_2, \psi_2)\cdots$ in MDPF $\mathcal{M}$ is called a valid path for a configuration c if $SAT(\bigwedge_{i\geq 0} \psi_i|_c)$ for $i \geq 1$. For all $j \geq 1$, we define $\pi[j] = s_j$.

Now we define a set of notations regarding to $\mathcal{M}$, which are used in the rest of this paper. We use $s \xrightarrow{\alpha/\psi/p} s'$ as an alternative to $((s, \alpha, \psi, s'), p) \in P_F$ which denotes a transition from state s to state s' with probability p when action α is chosen from $Act(s)$ and this transition only exists in the products satisfying ψ. We define $\gamma_{\alpha}(s, s')$ with respect to the transitions between s and s' labeled with action α which is formally expressed as $\forall s, s' \in S\ \forall \alpha \in Act(s)$:

$$\gamma_{\alpha}(s, s') = \{(\psi, p) | \exists t \in P_F \cdot t : s \xrightarrow{\alpha/\psi/p} s'\}$$

For example in the MDPF in Fig. 2, $\gamma_{try}(s_1, s_2) = \{(\mathsf{A}, 1), (\mathsf{B}, 0.8)\}$. We assume that a transition annotated with *true* is equal to just the same transition with no application condition. Hence, the behavior of each markov decision process is equal to an MDPF obtained by annotating all of its transitions with *true*. In order to extract the behavior of an individual or a subset of the products from an MDPF, we define the projection of an MDPF on a configuration.

Definition 3: (Projection) Let $\mathcal{M}$=(S, s_{init}, $P_{\mathcal{F}}$, Act, AP, L) be an MDPF defined over the feature model $\mathcal{F}$=(F,Φ_F) and let c be a configuration in $C_{\mathcal{F}}$. The projection of $\mathcal{M}$ over c, denoted by $\mathcal{M}|_c$ is an MDPF $\mathcal{M}'$=(S, s_{init}, $P^c_{\mathcal{F}}$, Act, AP, L) where:

$$P^c_{\mathcal{F}} = \{s \xrightarrow{\alpha/\psi'/p} s' | \exists s \xrightarrow{\alpha/\psi/p} s' \in P_{\mathcal{F}} \wedge EQ(\psi', \psi|_c)\}$$

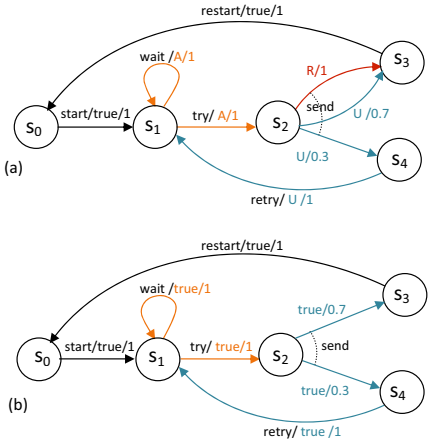

Fig. 3. The projection of the transmission system product line MDPF over (a) The partial configuration M ∧ ¬B and (b) The complete configuration M ∧ U ∧ ¬B.

The projection of an MDPF over a complete configuration results in an MDP and the projection over a partial configuration results in another MDPF which describes the behavior of a subset of the products. The projection of the MDPF of the transmission system product line over the partial configuration M ∧ ¬B is illustrated in Fig. 3 (a) and its projection over the complete configuration M ∧ U ∧ ¬B is illustrated in 3 (b).

Recall that schedulers are used to resolve nondeterminism in MDPs. We use schedulers for the same purpose over MDPFs. Assuming scheduler σ over the MDPF defined in Def. 2, for a finite path $\pi = s_0(\alpha_1, \psi_1)s_1(\alpha_2, \psi_2)s_2 \cdots (\alpha_n, \psi_n)s_n \in Paths_{fin}(\mathcal{M})$ we have $\sigma(\pi) = \alpha$ such that $\alpha \in Act(s_n)$ and also $\exists s \in S,\ \exists (\psi, p) \in \gamma(s_n, s) \cdot SAT(\psi \wedge \bigwedge_{i=1}^{n} \psi_n)$. The latter condition is imposed based on the definition of paths in an MDPF.

Parallel composition is an important mean to design large scale systems comprising a number of sub-systems working together. This way, each sub-system can be modeled separately and then be composed in parallel with other models. This can effectively reduce the complexity of the modeling. Here we define the parallel composition of the MDPFs based on the interleaved execution of their actions and handshaking over a subset of their common actions.

Definition 4: Assume two MDPFs $\mathcal{M}_i$=(S_i, s^i_{init}, $P^i_{\mathcal{F}}$, Act_i, L_i) for $i \in \{1, 2\}$ are defined over the feature model $\mathcal{F}$. We define the parallel composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ handshaking over the set of actions $H \subseteq Act_1 \cap Act_2$, denoted by $\mathcal{M}_1 \|_H \mathcal{M}_2$, as an MDPF $\mathcal{M}$=($S_1 \times S_2$, $s^1_{init} \times s^2_{init}$, $P_{\mathcal{F}}$, $Act_1 \cup Act_1$, L) where:

- $L(s_1, s_2) = L(s_1) \cup L(s_2)$
- $P_{\mathcal{F}}$ can be defined as the smallest relationship satisfying:
  - For $\alpha \in H$:

$$\frac{s_1 \xrightarrow{\alpha/\psi_1/p_1} s'_1 \wedge s_2 \xrightarrow{\alpha/\psi_2/p_2} s'_2}{(s_1, s_2) \xrightarrow{\alpha/\psi_1 \wedge \psi_2/p_1 \times p_2} (s'_1, s'_2)}$$

- For $\alpha \notin H$:

$$\frac{s_1 \xrightarrow{\alpha/\psi_1/p_1} s_1'}{(s_1,s_2) \xrightarrow{\alpha/\psi_1/p_1} (s_1',s_2)} \qquad \frac{s_2 \xrightarrow{\alpha/\psi_2/p_2} s_2'}{(s_1,s_2) \xrightarrow{\alpha/\psi_2/p_2} (s_1,s_2')}$$

Note that we assumed the two MDPFs are defined over the same feature model. In case this assumption is not valid, e.g., one of them is reused from another context, we can first merge the feature models based on existing techniques, e.g., [12], and then use the above definition for parallel composition.

V. Model checking of MDPFs

As the presence of transitions in an MDPF is conditional regarding to the application conditions, the probabilistic model checking can not be accomplished by applying the same algorithm used for MDPs. In this section we propose an algorithm to verify MDPFs against PCTL formulas. This algorithm is similar to the existing one in the sense that it checks the satisfiability of a PCTL formula by computing the satisfaction sets for the sub-formulas in a bottom-up manner. But the computed satisfaction sets in addition to the set of states, also include the set of products in which a state satisfies the property (this is necessary according to the fact that a state may satisfy a property only in a subset of the products).

In order to formulate when a PCTL formula is satisfied by a state in a specific subset of the products, we define the semantics of PCTL formulas with respect to the MDPF $\mathcal{M} = (S, s_{init}, P_{\mathcal{F}}, Act, AP, L)$, defined with regards to the feature model $\mathcal{F} = (F, \Phi_F)$.

Let $s \in S$ be a state and $\psi \in \Psi_F$ be a propositional formula, the semantics of PCTL formula Φ is defined by a satisfaction relation, denoted by $\models$, between pairs (s, ψ) and Φ. The interpretation of $(s, \psi) \models \Phi$ is that state s satisfies Φ in the set of products represented by ψ.

Let Φ, Φ_1, and Φ_2 be PCTL state formulas, φ be a PCTL path formula, $\psi \in \Psi_F$, and $a \in AP$. The satisfaction relation for PCTL state formulas over $\mathcal{M}$ can be defined almost the same way as for DTMCFs in [7] as follows:

$$\begin{aligned}
&(s,\psi) \models \textbf{True}\\
&(s,\psi) \models a \;\text{ iff }\; a \in L(s)\\
&(s,\psi) \models \neg\Phi \;\text{ iff }\; (s,\psi) \not\models \Phi\\
&(s,\psi) \models \Phi_1 \wedge \Phi_2 \;\text{ iff }\; (s,\psi) \models \Phi_1 \;\wedge\; (s,\psi) \models \Phi_2
\end{aligned}$$

The satisfaction relation for a PCTL path formula $\pi \in Paths(\mathcal{M})$ is defined as:

$$\begin{aligned}
&\pi \models \bigcirc\Phi \;\text{ iff }\; \exists\psi \in \Psi_F \cdot\; (\pi[1],\psi) \models \Phi\\
&\pi \models (\Phi_1 U \Phi_2) \text{ iff } \exists k \cdot \forall j < k \cdot \exists \psi,\psi' \in \Psi_F \cdot\\
&\qquad\qquad (\pi[j],\psi) \models \Phi_1 \wedge (\pi[k],\psi') \models \Phi_2\\
&\pi \models (\Phi_1 U^{\leq n} \Phi_2) \text{ iff } \exists k \leq n \cdot\; \forall j < k \cdot\; \exists \psi,\psi' \in \Psi_F \cdot\\
&\qquad\qquad (\pi[j],\psi) \models \Phi_1 \;\wedge\; (\pi[k],\psi') \models \Phi_2
\end{aligned}$$

The semantics of PCTL formulas such as $\mathbb{P}_{\sim p}[\varphi]$ is defined considering all the possible paths in the model. Assuming that $\mathfrak{S}$ denotes the set of all schedulers over the MDPF $\mathcal{M}$, for $\sim \in \{<, \leq\}$ we define:

$$(s,\psi) \models \mathbb{P}_{\sim J}[\varphi] \longleftrightarrow sup_{\sigma\in\mathfrak{S}} \quad P^{\sigma}((s,\psi),\varphi) \sim J$$

Similarly, for $\sim \in \{>, \geq\}$ we define:

$$(s,\psi) \models \mathbb{P}_{\sim J}[\varphi] \longleftrightarrow inf_{\sigma\in\mathfrak{S}} \quad P^{\sigma}((s,\psi),\varphi) \sim J$$

where *sup* and *inf* represent the supremum and infimum values of a set and $P^{\sigma}((s,\psi),\varphi)$ denotes the probability measure of all the paths induced by scheduler σ starting from s, which belong to the set of the products satisfying ψ and also satisfy the path formula φ. The satisfaction set of a PCTL formula Φ over the MDPF $\mathcal{M}$ denoted by, $Sat(\Phi)$ is defined as the smallest relation of the form:

$$\{(s, \textstyle\bigvee_{i=1}^{n} \psi_i) |\; \forall\; 0 \leq i \leq n \;\cdot\; (s,\psi_i) \models \Phi\}$$

According to the above definition the satisfaction set for PCTL formulas over MDPF $\mathcal{M}$ can be defined as follows.

$$\begin{aligned}
&Sat(\textbf{True}) = \{(s, true) | s \in S\}\\
&Sat(a) = \{(s, true) | s \in S \wedge a \in L(s)\}\\
&Sat(\neg\Phi) = \{(s,\neg\psi) | (s,\psi) \in Sat(\Phi)\}\\
&\qquad\cup \{(s, true) | s \notin Dom(Sat(\Phi))\}\\
&Sat(\Phi_1 \wedge \Phi_2) = \{(s,\psi) \in Sat(\Phi_1) | \exists (s,\psi') \in Sat(\Phi_2) \cdot \psi \preceq \psi'\}\\
&\qquad\cup \{(s,\psi) \in Sat(\Phi_2) | \exists (s,\psi') \in Sat(\Phi_1) \cdot \psi \preceq \psi'\}\\
&Sat(\mathbb{P}_{\sim J}(\textstyle\bigcirc\Phi)) = \{(s, \bigvee_{i=0}^{n} \psi_i) \mid \forall 0 \leq i \leq n \;\cdot\; \forall\sigma \in \mathfrak{S} \cdot\\
&\qquad P^{\sigma}((s,\psi_i) \textstyle\bigcirc \Phi_1) \sim J\}\\
&Sat(\mathbb{P}_{\sim J}(\Phi_1 U \Phi_2)) = \{(s, \textstyle\bigvee_{i=0}^{n} \psi_i) \mid \forall 0 \leq i \leq n \;\cdot\; \forall\sigma \in \mathfrak{S} \cdot\\
&\qquad P^{\sigma}((s,\psi_i), \Phi_1 U \Phi_2) \sim J\}\\
&Sat(\mathbb{P}_{\sim J}(\Phi_1 U^{\leq n} \Phi_2)) = \{(s, \textstyle\bigvee_{i=0}^{n} \psi_i) \mid \forall 0 \leq i \leq n \;\cdot\; \forall\sigma \in \mathfrak{S} \cdot\\
&\qquad P^{\sigma}((s,\psi_i), \Phi_1 U^{\leq n} \Phi_2) \sim J\}
\end{aligned}$$

Where Dom(_) denotes the domain of a function. In the rest of this section we focus on the computation of three last satisfaction sets listed above, as the computation of the other ones is straightforward. The computation of the probability measure used in the above formulas is implied in our model checking algorithm.

A. Model checking $\mathbb{P}_{\sim J}[\bigcirc\Phi]$

In this section we explain the details of the model checking algorithm for the PCTL formula $\mathbb{P}_{\sim J}[\bigcirc\Phi]$. To specify the intuitions making the rational behind our algorithm, we start by reviewing the main steps of the model checking of an MDP $M = (S, s_{init}, P, Act, AP, L)$ against such PCTL formula [8] and then we explain the model checking algorithm of MDPFs.

The model checking starts by the computation of $Sat(\Phi)$. Then for each state s, the probability of reaching $Sat(\Phi)$ in one step via each of the actions enabled in s is computed. These probabilities are computed for all states at once by multiplication of the matrix representation of P and a vector λ of size $|S|$ with probability value 1 in the rows corresponding to the states in $Sat(\Phi)$ and 0 in other rows. The matrix representation of P is a matrix in which there are $|Act(s)|$ rows corresponding to each state s where each row represents the probability of the outgoing transitions from s to other states for one of the actions in $Act(s)$. The result of the multiplication is a vector which has $|Act(s)|$ rows corresponding to each state, where the element in each row represents the probability of reaching $Sat(\Phi)$ in one step via one of the actions enabled in that state. If in the above PCTL formula $\sim \in \{\geq, >\}$, for each

$$[P_F] = \begin{pmatrix} t/0 & \mathsf{M}/1 & t/0 & t/0 & t/0 \\ t/0 & \mathsf{B}/0.2 & \mathsf{A}/1;\mathsf{B}/0.8 & t/0 & t/0 \\ t/0 & \mathsf{A}/1 & t/0 & t/0 & t/0 \\ t/0 & t/0 & t/0 & \mathsf{R}/1;\mathsf{U}\wedge\neg\mathsf{B}/0.7 & \mathsf{U}\wedge\neg\mathsf{B}/0.3 \\ \mathsf{M}/1 & t/0 & t/0 & t/0 & t/0 \\ t/0 & \mathsf{U}\wedge\neg\mathsf{B}/1 & t/0 & t/0 & t/0 \end{pmatrix} \lambda = \begin{pmatrix} t/0 \\ t/0 \\ t/1 \\ t/0 \\ t/0 \end{pmatrix}$$

Fig. 4. The matrix representations of the transmission system product line

state s the minimum probability among probabilities computed according to different actions is compared with the bound J and if it preserves the bound then s is in the final satisfaction set (for $\sim\in\{\leq,<\}$ the maximum value is compared with the bound). The model checking of the MDPF $\mathcal{M}$ against $\mathbb{P}_{\sim J}[\bigcirc\Phi]$ is accomplished by following the general steps of the above framework. But here the computed probabilities in different steps are annotated by application conditions (just like the probabilities of the transitions in the model) to distinguish the results for different subsets of the products. We explain our algorithm for properties in which $\sim\in\{\geq,>\}$, as the computations for the other set can be obtained by some simple changes. For more clarification, we illustrate the model checking of the MDPF in Fig. 2 against the PCTL formula $\mathbb{P}_{\geq 0.8}[\bigcirc \mathit{Sending}]$ while describing our algorithm. (We assume that in this MDPF the state s_2 is the only state labeled with the application condition $\mathit{Sending}$)

The algorithm starts by computing $\mathit{Sat}(\Phi)$. This satisfaction set, based on the definition of the satisfaction relation in Sect. V, is a set of states each paired with an application condition. In our example $\mathit{Sat}(\mathit{Sending}) = \{(s_2, \mathit{true})\}$. In the next step, the vector λ and the matrix representation of P_F, denoted by $[P_F]$, are constructed and then multiplied. The matrix and the vector corresponding to our example are represented in Fig. 4.

In both $[P_F]$ and λ the elements are sets of probabilities paired with application conditions. Two colored rows in $[P_F]$ represent the application condition/probability of the outgoing transitions from s_1 respectively with the actions try and wait. The element in the first row under column 3, $\mathsf{A}/1;\mathsf{B}/0.8$, depicts the probability of two transitions from s_1 to s_2 and their corresponding application conditions A and B. In λ, the element with value $t/1$ denotes that state s_1 satisfies $\mathit{Sending}$ in all the products (we use t as a short form for true representing all the products) and on the other hand $t/0$ denotes that the probability of satisfying $\mathit{Sending}$ in any of the products is 0.

The elements of the above matrices can be defined as functions from Ψ_F to $[0,1]$. We define R as the set of all possible such functions. We assume that ψ/p and $\psi_1/p_1;\cdots;\psi_n/p_n$ are respectively alternative representations of the functions (ψ, p) and $\{(\psi_1, p_1),\cdots,(\psi_n, p_n)\}$. To fulfill the multiplication of $[P_F]$ and λ, first we define the operators $\otimes, \oplus$ as the multiplication and the summation operators for the elements of the matrices. The summation of two functions representing outgoing transitions from a state is a function mapping each application condition on the transitions to the sum of the probabilities corresponding to that application condition. Assuming functions $r, r_1, \cdots, r_n \in R$, application conditions $\psi \in \Psi_F \backslash t$ and $\psi_1, \cdots, \psi_n \in \Psi_F$, and probabilities $p, p_1, \cdots, p_n \in [0,1]$ we define:

$$\psi/p \oplus r = \begin{cases} (r - \psi/r(\psi)) \cup \psi/(p + r(\psi)) & \textit{if } \psi \in \mathit{Dom}(r) \\ r \cup \psi/p & \textit{otherwise} \end{cases}$$

$$t/0 \oplus r = r$$
$$\psi_1/p_1, \cdots, \psi_n/p_n \oplus r = \psi_2/p_2; \cdots; \psi_n/p_n \oplus (\psi_1/p_1 \oplus r)$$

It is easy to see that the summation operation is commutative and associative. So, we can define:

$$\sum_{i=1}^{n} r_i = r_1 \oplus r_2 \oplus \cdots \oplus r_2$$

The operator $\otimes$ combines successive transitions in the way that taking a transition ψ_1/p_1 followed by a transition ψ_2/p_2, is possible only in the products satisfying $\psi_1 \wedge \psi_1$, and leads to the destination with probability $p_1 \times p_2$. The following definition defines the operator in general:

$$r_1 \otimes r_2 = \sum_{\substack{\psi_1/p_1 \in r_1 \\ \psi_2/p_2 \in r_2}} (\psi_1 \wedge \psi_2)/(p_1 \times p_2)$$

When multiplying $[P_F]$ and λ, we use $\oplus$, $\otimes$ instead of two mathematical operators $+$, $\times$ in ordinary matrix multiplication. For example considering the first colored row in $[P_F]$, the computations is done as follows (parentheses are used for more clarity):

$$\big(\sum_{i=0}^{3}(t/0 \otimes t/0)\big) \oplus (\mathsf{B}/0.2 \otimes t/0) \oplus ((\mathsf{A}/1;\mathsf{B}/0.8) \otimes t/1) =$$
$$\big(\sum_{i=0}^{3}(t \wedge t/0 \times 0)\big) \oplus (\mathsf{B} \wedge t/0.2 \times 0) \oplus (\mathsf{A} \wedge t/1 \times 1; \mathsf{B} \wedge t/0.8 \times 1) =$$
$$\mathsf{A}/1;\mathsf{B}/0.8$$

The vector in the right hand side of Eq. 3 is the result of the multiplication of $[P_F]$ and λ. This vector includes $|\mathit{Act}(s)|$ rows corresponding to each state s. The colored rows represent the results computed for two actions enabled in s_1. These rows contain two sets $\mathsf{A}/1;\mathsf{B}/0.8$ for action *try* and $\mathsf{A}/0$ for action *wait* which means that for products containing feature A the probability of $\bigcirc \mathit{Sending}$ being satisfied in s_1 by taking action wait is 0 and by taking action try is 1. For the products containing feature B this probability is 0.8 by taking the only possible action wait. So, for products containing feature B but not A, the minimum probability of $\bigcirc \mathit{Sending}$ is 0.8. However,

if a product contains feature A (regardless of containing feature B), this probability is zero (by taking wait transition).

$$[P_F].\lambda = \begin{pmatrix} t/0 \\ \hline \mathsf{A}/1; \mathsf{B}/0.8 \\ \hline A/0 \\ \hline t/0 \\ \hline t/0 \\ \hline t/0 \end{pmatrix} \xrightarrow{min} \begin{pmatrix} t/0 \\ \hline \mathsf{A}/0; \mathsf{B}/0.8 \\ \hline t/0 \\ \hline t/0 \\ \hline t/0 \end{pmatrix} \quad (3)$$

Following the general algorithm explained for MDPs, we must take the minimum probabilities among different actions for each state. Here, we must define the minimum operator to keep the pairs ψ/p with minimum p for each ψ. We can take one step further and if having two pairs ψ/p and ψ'/p' with $\psi \preceq \psi'$ and $p' \leq p$ we only keep ψ'/p'. For example, taking minimum between $\mathsf{A}/0.2; \mathsf{C}/0.8$ and $\mathsf{A} \wedge \mathsf{B}/0.9; \mathsf{C}/0.7$ results in $\mathsf{A}/0.2; \mathsf{C}/0.7$. Formally, we define the operator min over the set of functions $r_i \in R$ for $0 \leq i \leq n$, as follows.

$$\min_{0 \leq i \leq n}(r_i) = \bigcup_{i=1}^{n} r_i \setminus \{\psi/p \in \bigcup_{i=1}^{n} r_i \mid \exists \psi'/p' \in \bigcup_{i=1}^{n} r_i \cdot \ \psi \preceq \psi' \wedge p' \leq p\}$$

To formally describe the computations in the algorithm, we define $\lambda : S \to R$ such that:

$$\lambda(s) = \begin{cases} t/0 & \text{if } \forall \psi \in \Psi_F \cdot \ \nexists (s,\psi) \in Sat(\Phi) \\ \{(\psi, 1) | (s,\psi) \in Sat(\Phi)\} & \text{otherwise} \end{cases}$$

Considering $[P_F]$, the element in the row representing the outgoing transitions from s to s' with action α can be specified by $\gamma_\alpha(s,s')$ as defined in Sect. IV. So, we can formulate the computations as:

$$X(s) = \min_{\alpha \in Act(s)} (\sum_{s' \in S} \gamma_\alpha(s,s') \otimes \lambda(s))$$

Finally, we formulate the satisfaction set for the next operator as bellow:

$$X_{\sim J}(s) = \{\psi \mid (\psi, p) \in X(s) \wedge p \sim J\}$$

$$Sat(\mathbb{P}_{\sim p}[\bigcirc \Phi]) = \{(s, \bigvee_{\psi \in X_{\sim J}(s)} \psi) | s \in S \wedge X_{\sim J}(s) \neq \emptyset\}$$

The satisfaction set computed in our example is :

$$Sat(\mathbb{P}_{\geq 0.8}[\bigcirc S_2]) = \{(s_2, \mathsf{B}/0.8)\}$$

The steps of the algorithm for properties such as $\mathbb{P}_{\sim p}[\bigcirc \Phi]$ where $\sim \in \{\leq, <\}$, are obtained by replacing $\{\geq, >\}$ with $\{\leq, <\}$ and the min operator with the max operator. The definition of the max operator is just similar to the definition of the min operator with $\leq$ replaced by $\geq$. This way the operator computes the maximum probabilities.

B. Model checking $\mathbb{P}_{\sim J}[\Phi_1 U^{\leq k} \Phi_2]$ and $\mathbb{P}_{\sim J}[\Phi_1 U \Phi_2]$

The steps of the model checking algorithm for above properties can be induced from the computations in Sect. V-A, as here our algorithm (for each state) computes the probability of reaching the states that satisfy Φ_2 through a sequence of states in which Φ_1 is satisfied in the next step. The computation of the satisfaction sets for both above properties is similar except that for bounded until the number of the computation steps is bounded by k. The first step is to find the satisfaction sets for Φ_1 and Φ_2. Then we define three disjoint subsets of states as follows:

- $S^{yes} = \{s \in S | \exists \psi \in \Psi_F \cdot \ (s,\psi) \in Sat(\Phi_2)\}$
- $S^{no} = \{s \in S | \forall \psi \in \Psi_F \cdot \ (s,\psi) \notin Sat(\Phi_2)\}$
- $S^{?} = S \setminus S^{yes} \cup S^{no}$

For a PCTL formula such as $\mathbb{P}_{\sim J}[\Phi_1 U^{\leq k} \Phi_2]$, we compute $X^k(s)$ using the following definition.

$$\begin{cases} t/1 & \text{if } s \in S^{yes} \\ t/0 & \text{if } s \in S^{no} \\ t/0 & \text{if } s \in S^{?} \wedge k = 0 \\ \min\limits_{\alpha \in Act(s)} (\sum\limits_{s' \in S} \gamma_\alpha(s,s') \otimes X^{k-1}(s')) & \text{if } s \in S^{?} \wedge k > 0 \end{cases}$$

The satisfaction set is computed as:

$$X_{\sim J}(s) = \{\psi \mid (\psi, p) \in X^k(s) \wedge p \sim J\}$$

$$Sat(\mathbb{P}_{\sim J}[\Phi_1 U^{\leq k} \Phi_2]) = \{(s, \bigvee_{\psi \in X_{\sim J}(s)} \psi) | s \in S \wedge X_{\sim J}(s) \neq \emptyset\}$$

All the above computations are the same for Until operator except that for each state s, we compute $\lim\limits_{k \to \infty} X^k(s)$. As unbounded computations is not possible, usually a large number will be assumed as a bound for the computations.

VI. Related Work

To the best of our knowledge, apart from our previous work on DTMC-based method [18] (mentioned in Sect. 1), there are no existing work on modeling and verification of probabilistic software product lines. However,there are a few works done on quantitative verification of SPLs till now. In [13], Nunes et al use model checking of parametric Markov models [14] to analyse the reliability of software product lines. In this work using the PARAM tool [15], by considering a feature model as the input, the reliability is computed as a formula in which the variables represent the features in the feature model. In [16], an approach is proposed to verify some *non-functional* properties of SPLs. In this work the UML sequence diagrams are extended by features to represent the variabilities in the SPL. Then, the properties are verified using two different methods. In the first one the properties are verified against DTMCs, modeling the behavior of each product, which are extracted from sequence diagrams using a technique represented in [17]. In the second method the properties are checked against a parametric reward DTMC. This method has some limitations regarding to the software structure and restrictions on feature relations while the first one is costly regarding to the fast growth of the number of the products. In [18], a compositional modeling framework is proposed in order to reason about quantitative properties of dynamic product lines where the features of the product line are not fixed and can change during runtime.

There are also a family of related works done on verification of non-probabilistic SPLs. In [4] Classen et al. introduce an extension of transition systems, Featured Transition System (FTS), by annotating the transitions with single features and

propose a model checking algorithm to verify FTSs against ω-regular properties. Regarding to the results reported, using FTS can reduce the cost of model checking SPLs. In [2] this work is extended by annotating the transitions using feature expressions. An algorithm is also provided in order to check an extension of LTL formulas called featured LTL (fLTL). In [5] an extension of timed automata by features is introduced to verify the behavior of real-time SPLs. In this work, first the properties are verified using the trivial on-by-one verification technique and then the properties are verified using a variability aware technique. In [3] a symbolic representation of feature transition systems is presented that is verified against an extension of CTL formulas called fCTL, which results to fulfilling the model checking more efficiently. In this work an extension of NuSMV [19] is proposed as a high level modeling language to describe the behavior of SPLs.

In this paper we have also used an annotative technique to model the behavior of the whole family but there are more complications, which are imposed by the underlying probabilistic model, demonstrated in both the modeling and the model checking algorithm.

VII. Conclusion and Future Work

As software product line engineering is applied in a variety of domains containing safety critical systems, the verification of the software product lines is vital. In this paper we considered the model checking of software product lines including products with nondeterministic and probabilistic behavior. As modeling and verification of all products individually is costly and sometimes not possible, we provided a model, Markov Decision Process Family, which describes the behavior of the whole products. We also provided the model checking algorithm to check MDPFs against PCTL formulas. This way, it possible to model and verify the behavior of all products at once which reduces the cost of model checking by reducing the total number of the states to be verified.

In order to evaluate our approach, building an efficient implementation of the model checking algorithm is in progress. But we have already developed a simple implementation in C++ and used it to model check a number of small samples. We have successfully used this implementation to check a larger sample where the number of the states in the MDPF is about 2000 states and the total number of the states in the product line was 36000. As our future work, we continue developing the efficient implementation of our algorithm to compare the efficiency of our method compared to the results of model checking products individually with existing tools. As another future work, we are going to provide a high level modeling language to model the behavior of product lines with probabilistic and nondeterministic behavior and automate the generation of MDPFs from the high level description.

References

[1] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[2] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin, "Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, p. 1, 2013.

[3] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay, "Symbolic model checking of software product lines," in *Proc. of the 33rd Intl. Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 321–330. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985838

[4] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin, "Model checking lots of systems: efficient verification of temporal properties in software product lines," in *Proc. of the 32nd ACM/IEEE Intl. Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 335–344. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806850

[5] M. Cordy, P.-Y. Schobbens, P. Heymans, and A. Legay, "Behavioural modelling and verification of real-time software product lines," in *Proc. of the 16th Intl. Software Product Line Conference - Volume 1*, ser. SPLC '12. New York, USA: ACM, 2012, pp. 66–75. [Online]. Available: http://doi.acm.org/10.1145/2362536.2362549

[6] H. Hansson and B. Jonsson, "A framework for reasoning about time and reliability," in *IEEE Real-Time Systems Symposium*, 1989, pp. 102–111.

[7] M. Varshosaz and R. Khosravi, "Discrete time markov chain families: Modeling and verification of probabilistic software product lines," in *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*, ser. SPLC '13 Workshops. New York, NY, USA: ACM, 2013, pp. 34–41. [Online]. Available: http://doi.acm.org/10.1145/2499777.2500725

[8] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.

[9] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-90-TR-21, 1990.

[10] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," vol. 35, no. 6. Oxford, UK, UK: Elsevier Science Ltd., Sep. 2010, pp. 615–636. [Online]. Available: http://dx.doi.org/10.1016/j.is.2010.01.001

[11] D. Batory, "Feature models, grammars, and propositional formulas," in *Proc. of the 9th intl. conference on Software Product Lines*, ser. SPLC'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 7–20. [Online]. Available: http://dx.doi.org/10.1007/11554844_3

[12] M. Acher, P. Collet, P. Lahire, and R. France, "Composing feature models," in *Proceedings of the Second International Conference on Software Language Engineering*, ser. SLE'09. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 62–81. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12107-4_6

[13] V. Nunes, P. Fernandes, V. Alves, and G. N. Rodrigues, "Variability management of reliability models in software product lines: An expressiveness and scalability analysis," in *SBCARS*, 2012, pp. 51–60.

[14] E. M. Hahn, "Parametric markov model analysis," Master's thesis, Saarland University, Germany, 2008.

[15] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "Param: a model checker for parametric markov models," in *Proc. of the 22nd intl. conference on Computer Aided Verification*, ser. CAV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 660–664. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14295-6_56

[16] C. Ghezzi and A. Molzam Sharifloo, "Quantitative Verification of Non-Functional Requirements with Uncertainty," in *Sixth Intl. Conference on Dependability and Computer Systems*. Springer, 2011, pp. 47–62.

[17] C. Ghezzi and A. M. Sharifloo, "Verifying non-functional properties of software product lines: Towards an efficient approach using parametric model checking," in *Proc. of the 15th Intl. Software Product Line Conference*, ser. SPLC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 170–174. [Online]. Available: http://dx.doi.org/10.1109/SPLC.2011.33

[18] C. Dubslaff, S. Klüppelholz, and C. Baier, "Probabilistic model checking for energy analysis in software product lines," *CoRR*, vol. abs/1312.7758, 2013.

[19] M. Plath and M. Ryan, "Feature integration using a feature construct," *Science of Computer Programming*, vol. 41, no. 1, pp. 53–84, 2001.