Discrete Time Markov Chain Families: Modeling and Verification of Probabilistic Software Product Lines

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

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

ABSTRACT

Software product line engineering (SPLE) enables systematic reuse in development of a family of related software systems by explicitly defining commonalities and variabilities among the individual products in the family. Nowadays, SPLE is used in a variety of complex domains such as avionics and automotive. As such domains include safety critical systems which exhibit probabilistic behavior, there is a major need for modeling and verification approaches dealing with probabilistic aspects of systems in the presence of variabilities. In this paper, we introduce a mathematical model, Discrete Time Markov Chain Family (DTMCF), which compactly represents the probabilistic behavior of all the products in the product line. We also provide a probabilistic model checking method to verify DTMCFs against Probabilistic Computation Tree Logic (PCTL) properties. This way, instead of verifying each product individually, the whole family is model checked at once, resulting in the set of products satisfying the desired property. This reduces the required cost for model checking by eliminating redundant processing caused by the commonalities among the products.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Model checking*

General Terms

Theory, Verification

Keywords

Software Product Line, Probabilistic Model Checking, Variable Discrete Time Markov Chains

1. INTRODUCTION

Software Product Line (SPL) engineering is one of the promising software development approaches, concerned with exploiting commonalities and variabilities among a related set of software systems within a specific domain (a.k.a. product families). Benefiting from systematic reuse throughout the system life cycle, the use of software product line engineering leads to decrease in time to market and development cost [22]. A common way to capture variabilities among the different products in a product line is to use a *feature model* which contains a hierarchy of the features in the whole product line as well as specifying which features are mandatory and which ones are optional (described in Sect. 2.1 in more detail). Each individual product of the family is then described by a *configuration*, which is the set of features included in that product.

Nowadays, SPLs are adopted increasingly in a variety of domains such as avionics and automotive. As a wide range of these domains contain safety critical systems, the verification of such SPLs is a major concern. Model checking [5] is a formal method used widely for verification of safety critical systems. The basic model checking method deals with absolute correctness of the system but in reality, a large number of systems are subject to probabilistic and nondeterministic behavior and cannot guarantee some properties with complete certainty. Hence, a significant amount of research has been dedicated to development of probabilistic model checking [10, 19] using Markovian structures to describe and analyze the probabilistic behavior of such systems. There is a brief description of some concepts in this area, used in our paper, in Sect. 2.2.

Model checking of software product lines is much more challenging comparing to model checking of a single product [24]. A naïve approach would be to model all valid configurations of a software product line and verify properties against each of them individually. As the number of products in an SPL may grow exponentially in terms of the number of features, This approach can be significantly time consuming.

Recently, some model checking approaches have been proposed in which all the possible behaviors of the products are compactly represented in one model and the properties are verified at once [6, 7, 8, 9]. In these approaches, variability is integrated into the system model by annotating the model elements by features. The behavior of a single product can then be obtained from the whole product line model by selecting the model elements that are annotated by the features present in the corresponding configuration. To the best of our knowledge, none of the existing work in this area concerns models with probabilistic behavior.

In this paper, we introduce a new mathematical model called Discrete Time Markov Chain Family (DTMCF) which compactly represents the probabilistic behavior of all products contained in an SPL. The behavior of each individual product can be easily extracted from this model in terms of a Discrete Time Markov Chain (DTMC). We formalize the syntax and semantics of DTMCF (Sect. 4) and also provide a model checking algorithm, to verify these models against Probabilistic Computation Tree Logic (PCTL) [17] formulas (Sect. 5). This way, we enable reuse in model checking SPLs, by avoiding space/time redundancies caused by commonalities among the products in the family. We use a running example, described in Sect. 3, which is a simplified version of a electromechanical breaking system, to illustrate the concepts and algorithms.

2. PRELIMINARIES

In this section, we briefly overview the basic constructs our work is based on: feature models to capture variability in SPLs, discrete time Markov chains to model probabilistic behavior, and PCTL to express properties to be analyzed.

2.1 Basic SPL Concepts

In the context of software product line engineering, a *feature* is defined as "a prominent or distinctive user-visible aspect, quality, or characteristic of a software system" [18]. The notion of feature has been introduced to model commonalities and variabilities between the products in an SPL. Hence, a software product line contains a set of features and each product is identified by a certain subset of features. There may be different relationships between the features in an SPL such as one may require or exclude some others. A common means to compactly representing a software product line in terms of its features is to use feature models [3].

A feature model is a tree-like structure in which features are hierarchically organized based on their relationships. A feature may have a number of sub-features, some of them may be *optional*, while others are *mandatory*. Two or more sibling sub-features may have *alternative* or *xor* relationship, indicating that exactly one of them must be included in the parent feature. They may have *or* relationship, where at least one of them must be included in the parent. There can be also cross-tree relations between features, namely *requires* (resp. *excludes*), where inclusion of one feature in a product implies inclusion (resp. exclusion) of another one. Figure 1 illustrates the feature model of a (very) simplified Electromechanical Braking System (EBS) in a drive-by-wire car. In this feature model, *Sensor*, *Actuator*, *Communication Media*, and *Control Unit* are mandatory sub-features of the root feature *EBS*. Furthermore, *Redundancy* is an optional sub-feature of *Control Unit* and *Long Range Radar* and *Short Range Radar* are alternatively used as obstacle detection sensors.

Each feature model can be formulated as a propositional logic formula in which each boolean variable corresponds to a feature and its value indicates if the feature is included or excluded [2]. The corresponding formula is the conjunction of implications from (1) every child feature to its parent feature, (2) every parent to its mandatory child features, (3) every parent to or/xor of its children that have an or/xor relationship, (4) every feature f to other features that f requires, (5) and every feature f to the negation of the features that f excludes [23].

Definition 1. (Feature model) A feature model is a pair $\mathcal{F} = (F, \Phi_F)$ where

- F is a finite set of features.
- Φ_F is the propositional formula capturing the relationships between the features, constructed as described above.

The inclusion or exclusion of all or a subset of features can be formulated in terms of configurations as below:

Definition 2. (Configuration) A configuration c over a feature set F is a pair (I_c, E_c) where

- $I_c \subseteq F$ denotes the set of included features,
- $E_c \subseteq F$ denotes the set of excluded features, and
- $I_c \cap E_c = \varnothing$.

Furthermore, we define $\Delta(c) \triangleq (\bigwedge_{f \in I_c} f) \wedge (\bigwedge_{f \in E_c} \neg f)$ to be the propositional formula corresponding to the configuration c.

If $I_c \cup E_c = F$, the configuration c is called *total* otherwise it is called *partial*. A configuration c is *valid* if it does not violate the constraints imposed by the feature model (e.g., it does not include two alternative siblings). Having a propositional logic formula ψ over a set of feature variables, we define the projection of ψ over a configuration c, denoted by $\psi|_c$, as the formula resulting from the substitution of the features in I_c with true and the features in E_c with false. So, a configuration c is valid if $\mathrm{SAT}(\Phi_F|_c)$, where $\mathrm{SAT}(\_)$ denotes the satisfiability of a propositional logic formula (which can be checked using standard SAT-solvers, e.g., [13, 25]). Note that a valid total configuration corresponds to a concrete product, while a valid partial configuration defines a set of products. We use the notation $\mathcal{C}_{\mathcal{F}}$ as the set of all valid configurations with respect to the feature model $\mathcal{F}$. We also denote the set of all propositional formulas over the feature set F by Ψ_F.

A common technique in representing variability in product line models is to *annotate* the parts of the model which belongs only to a certain subset of products. One way to express the annotations is to use *application conditions*, which are propositional formulas over the feature set F. A part of the model annotated by the application condition ψ will be included only in the products whose corresponding configuration satisfies ψ. As will be seen, we use application conditions to annotate transitions in Markov chains. We call two application conditions ψ_1 and ψ_2 *exclusive*, if no valid configuration c can satisfy both:

$$EX(\psi_1, \psi_2) \triangleq \nexists c \in \mathcal{C}_{\mathcal{F}} \cdot SAT(\psi_1|_c \wedge \psi_2|_c)$$

Binary Decision Diagram (BDD) is a well-known data structure for efficient representation and manipulation of propositional formulas [4]. Here we use Ordered Binary Decision Diagrams (OBDDs), an extension of BDDs fitting more to our needs as it will be explained later, to represent application conditions.

2.2 Probabilistic Model Checking

To verify properties on probabilistic system, we first need to model the behavior of the system. Discrete time Markov chain is one of the well-known formalisms used for modeling systems with pure probabilistic behavior (without nondeterminism). A DTMC is formalized as follows [1].

Definition 3. (Discrete Time Markov Chain) A discrete time Markov chain is a tuple $(S, P, \iota_{init}, AP, L)$ where

- S is a finite set of states,
- $P : S \times S \to [0, 1]$ is a probability density function such that
$$\forall s \in S \cdot \quad \sum_{s' \in S} P(s, s') = 1,$$
- $\iota_{init} : S \to [0, 1]$ is the initial distribution function such that
$$\sum_{s \in S} \iota_{init}(s) = 1,$$
- AP is a set of atomic propositions, and
- $L : S \to 2^{AP}$ is the state labeling function.

To express properties on probabilistic systems, various formalisms have been introduced, among which PCTL [17] is a well-known temporal logic. The syntax of PCTL formulas can be represented in terms of two categories: state formulas and path formulas which are defined over a set of atomic propositions AP. State formulas are evaluated on the states of the system and are defined by the following grammar:

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

where $a \in AP$, $\bowtie \in \{\leq, <, >, \geq\}$, $p \in [0, 1]$, and φ is a path formula. Path formulas are evaluated on 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 as in CTL. $\mathbb{P}_{\bowtie p}(\varphi)$ is true in the states which satisfy φ with a probability in the bound specified by $\bowtie p$. The formula $\Phi_1 U^{\leq n} \Phi_2$ means that Φ_2 will be satisfied in less than n steps and till then, Φ_1 stands true.

3. THE RUNNING EXAMPLE

To better illustrate how our proposed method works in modeling and model checking of software product lines with probabilistic behavior, we have chosen a very simplified automated Electromechanical Braking System (EBS) product line, as an example. In such braking systems, sensors, communication media, and actuators replace mechanical devices. The feature model of the EBS product line is illustrated in Figure 1. The root feature represents the main domain concept which is the EBS system. We consider four main features that are mandatory for every configuration of the SPL: Sensors, a Control Unit (CU), Communication Media, and Actuators.

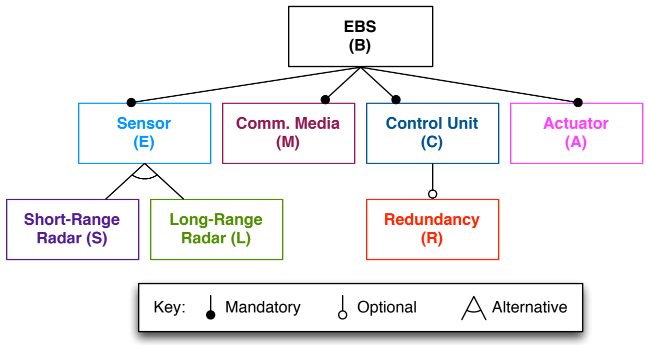

Figure 1: The feature model of EBS product line

A perfect product in this SPL generally functions as follows. The first step is the detection of an obstacle which can be done alternatively by the short range or long range radar sensors. The range of detection obviously differs in this two kind of sensors. Once an obstacle is detected, a signal will be sent to the CU via communication media. Receiving the signal, the CU computes some required parameters aiming to control the speed of the vehicle and braking safely. Then, in the next step, the CU sends the necessary commands to the actuators via communication media which eventually result in the required actuation. As we mentioned before, this is the functionality in the case that every component performs perfectly. But in reality, there could be malfunctions with nonzero probabilities. Failure of the sensors to detect obstacles in an admissible interval, possible message loss, and CU failure are some examples of undesirable but possible characteristics of such systems.

To analyze this system, we first model the system considering all the possible behaviors. To keep the running example simple, we restrict the probabilistic behavior of the system assuming reliable communication between the components and fault-free operation of actuators. Both kind of sensors may fail to detect the obstacles in acceptable time interval with different probabilities. There could be a failure in the CU with some probability which will be masked if there is a redundant version of CU. Again, for the sake of simplicity, we assume all the possible malfunctions lead the system to an unsafe situation (modeled with a single state).

There are different properties EBS products must satisfy, such as “under any circumstances, the probability of ending in the unsafe state once encountering an obstacle is less than 0.5”. One approach to verify this property is to model each individual product by a separate DTMC and check the property against it. However, a more interesting approach would be to model the probabilistic behavior of all products compactly in one model and verify the model at once.

The Discrete Time Markov Chain Family is such a compact model that is introduced in the next section.

4. DISCRETE TIME MARKOV CHAIN FAMILIES

As we mentioned before, a discrete time Markov chain family aims to capture the behavior of all products in a SPL exhibiting probabilistic behavior in a single model. The model is based on discrete time Markov chain with additional variability information encoded into the behavioral model, in the form of application conditions attached to each transition. The application conditions determine the set of products in which the transition is valid. The syntax of a discrete time Markov chain is defined as follows.

Definition 4. (Discrete Time Markov Chain Family) A DTMCF defined over the feature model (F, Φ_F) is a tuple $\mathcal{D} = (S, P, \iota_{init}, AP, L)$ where

- S, ι_{init}, AP, and L are defined as the same as in Def. 3.
- $P : S \times \Psi_F \times S \to [0, 1]$ is the probability density function satisfying conditions (1) and (2) bellow, and

By $s \xrightarrow{\psi/p} s'$, we mean there is a transition from s to s' with nonzero probability p annotated by the application condition ψ (the transition exists only in configurations satisfying ψ). We write $P_\psi(s, s')$ as an alternative to $P(s, \psi, s')$. Figure 2(a) illustrates the behavior of the EBS family as a DTMCF. We define an auxiliary function $\Upsilon : S \times S \to \Psi_F$ which determines the set of application conditions on all the transitions between s and s'. Furthermore, we define $\Upsilon_s = \bigcup_{s' \in S} \Upsilon(s, s')$ as the set of application conditions on all outgoing transitions of s. In the running example, we have $\Upsilon_{s_2} = \Upsilon(s_2, s_3) \cup \Upsilon(s_2, s_5) = \{\mathsf{C} \wedge \neg\mathsf{R}, \mathsf{R}\}$. We also define $Succ_\psi(s)$ as the set of states in S to which s makes a transition annotated by ψ.

There are two main conditions on the outgoing transitions of a state s. Every two application conditions ψ and ψ' on the outgoing transitions of the same state s are either equivalent or exclusive:

$$\forall s \in S \cdot \forall \psi, \psi' \in \Upsilon_s \cdot \psi \neq \psi' \to EX(\psi, \psi') \qquad (1)$$

We represent application conditions using reduced form of OBDDs. The reduced form of OBDDs representing equivalent propositional formulas, have the same structure. Hence, the equivalence checking of application conditions is fulfilled by equivalence checking of reduced OBDDs for which there are efficient algorithms.

Furthermore, the probabilities of the transitions with the same application condition sum up to 1.

$$\forall s \in S \cdot \forall \psi \in \Upsilon_s \cdot \sum_{s' \in Succ_\psi(s)} P_\psi(s, s^{'}) = 1 \qquad (2)$$

These conditions are in line with the need for having fully probabilistic choices in a DTMC which is the probabilistic

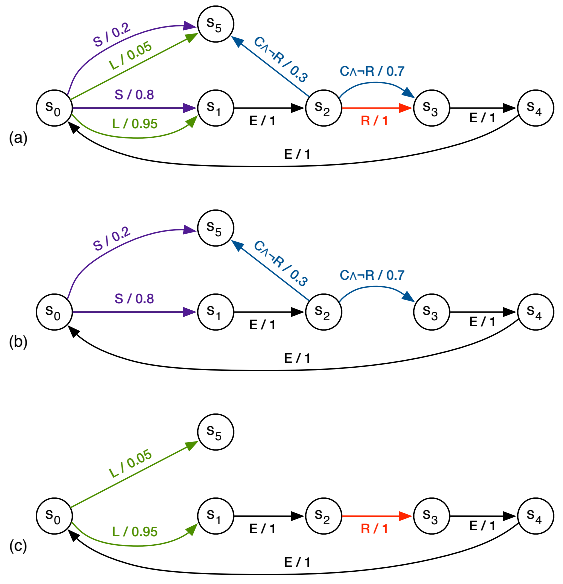

Figure 2: (a) The DTMCF of EBS product line (b),(c) Two different configurations of EBS product line

model of individual products in this paper (i.e., by satisfying these condition, the extracted model of each individual product from the DTMCF is a DTMC).

We define a path, π, in a discrete time Markov chain family as a sequence $\pi = \psi_0 s_0 \psi_1 s_1 \psi_2 s_2 \cdots$ such that $s_{i-1} \xrightarrow{\psi_i/p} s_i$ for $i > 0$, and $\psi_0 = \Phi_F$ is the propositional formula representing the feature model. We define $\pi[i] = (s_i, \bigwedge_{0 \leq k \leq i} \psi_k)$ for $i \geq 0$. The set of infinite paths starting from s is denoted by $Paths(s)$ and the set of all paths starting form the initial states $\{s \in S | \iota_{init}(s) > 0\}$ is denoted by $Paths_I$.

The behavioral model of each individual product in a product line represented by a DTMCF, is a DTMC containing only the transitions that are labeled with application conditions satisfiable by the configuration. As an example, according to the DTMCF of EBS product line in Figure 2(a), the transitions $s_0 \xrightarrow{\mathsf{L}/0.95} s_1$ and $s_0 \xrightarrow{\mathsf{L}/0.05} s_1$ are enabled in configurations including Long-range radar. Hence, these transitions are removed from the DTMC 2(b) which models a product with short-range radar. Extraction of the behavior of a specific product from the whole family model is defined in terms of *projection*. The projection of a DTMCF over a total configuration results in a discrete time Markov chain.

Definition 5. (Projection) A projection of a discrete time Markov chain family $\mathcal{D} = (S^{\mathcal{D}}, P^{\mathcal{D}}, \iota^{\mathcal{D}}_{init}, AP^{\mathcal{D}}, L^{\mathcal{D}})$ over a specific configuration c, denoted by $\mathcal{D}|_c$, is a DTMC $\mathcal{M} = (S^{\mathcal{M}}, P^{\mathcal{M}}, \iota^{\mathcal{M}}_{init}, AP^{\mathcal{M}}, L^{\mathcal{M}})$ where:

- $S^{\mathcal{M}} = S^{\mathcal{D}}, \iota^{\mathcal{M}}_{init} = \iota^{\mathcal{D}}_{init}, AP^{\mathcal{M}} = AP^{\mathcal{D}}, L^{\mathcal{M}} = L^{\mathcal{D}}$.

- $\forall s, s' \in S$ and $\psi \in \Upsilon(s, s')$:

$$P^{\mathcal{M}}_{\psi}(s, s') = \begin{cases} P^{\mathcal{D}}_{\psi}(s, s'), & \text{if } SAT(\psi|_c) \\ \\ 0, & \text{Otherwise} \end{cases}$$

The projection of the DTMCF in Figure 2(a) on two configurations $\mathsf{S} \wedge \neg\mathsf{L} \wedge \mathsf{C} \wedge \neg\mathsf{R} \wedge \mathsf{E}$ and $\neg\mathsf{S} \wedge \mathsf{L} \wedge \mathsf{C} \wedge \mathsf{R} \wedge \mathsf{E}$ are illustrated respectively in Figure 2(b) and (c).

The semantics of a DTMCF can be defined by combining the behavior of all possible products in the family. The behavior of each product is obtained by projection of the DTMCF over the corresponding configuration which results in a DTMC. Hence, we can define the semantics of a DTMCF $\mathcal{D}$ over the feature model $\mathcal{F} = (F, \Phi_F)$, denoted by $[\![\mathcal{D}]\!]$, as the union of all DTMCs resulting from projection of $[\![\mathcal{D}]\!]$ on all valid configurations:

$$[\![\mathcal{D}]\!] = \bigcup_{c \in \mathcal{C}_{\mathcal{F}}} \mathcal{D}|_c$$

5. MODEL CHECKING OF DTMCF

The model checking of PCTL properties over a DTMCF would be more challenging than a DTMC, since each state may be reachable only in a subset of products. Hence, we provide a model checking algorithm which returns a set of states annotated by formulas representing the set of configurations in which the state satisfies the given property. To this end, we first define a satisfaction relation, $\models$, between pairs (s, ψ) (where s is a state and ψ is a propositional formula representing a set of configurations) and a PCTL formula Φ.

In the rest of this section, assume s to be a state of the discrete time Markov chain family $\mathcal{D} = (S, P, \iota_{init}, AP, L)$ which models a product line over the feature model (F, Φ_F). Also 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 semantics of the satisfaction relation for state formulas is defined as follows.

$$\begin{aligned}
&(s, \psi) \models \mathbf{True} \\
&(s, \psi) \models a \;\text{ iff }\; a \in L(s) \\
&(s, \psi) \models \neg\Phi \;\text{ iff }\; \neg(s, \psi) \models \Phi \\
&(s, \psi) \models \Phi_1 \wedge \Phi_2 \;\text{ iff }\; (s, \psi) \models \Phi_1 \;\wedge\; (s, \psi) \models \Phi_2 \\
&(s, \psi) \models \mathbb{P}_J(\varphi) \;\text{ iff }\; Pr((s, \psi) \models \varphi) \in J
\end{aligned}$$

Here, $Pr((s, \psi) \models \varphi)$ is a probability measure which will be defined later in this section. The semantics of the satisfaction relation for path formulas is defined as follows.

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

The model checking of a PCTL formula Φ against a DTMCF is fulfilled by traversing the parse tree of Φ in a bottom-up manner and computing the satisfaction sets for subformulas. In order to compute the satisfaction sets, it is necessary to keep track of the set of products in which a state is reachable. To this end, we define a reachability relation R as follows.

$$R(s) \triangleq \{(s, \psi) | \exists \pi \in Paths_I \wedge \exists i \geq 0 \cdot \pi[i] = (s, \psi)\}$$

Where $Paths_I$ denotes the set of initial paths in $\mathcal{D}$. According to this definition, the initial state is reachable in all valid configurations.

We also define $R(S) = \bigcup_{s \in S} R(s)$ as the annotated state space of the DTMCF. According to the DTMCF of the EBS product line $R(s_1) = \{\mathsf{S} \wedge \Phi_{EBS}, \mathsf{L} \wedge \Phi_{EBS}\}$ where Φ_{EBS} denotes the propositional formula representing the EBS feature model. The reachability relation of a DTMCF can be computed using the algorithm represented in [6] by transforming a DTMCM to an FTS^+ which is easily done by removing the probabilities and merging the outgoing transitions with the same application conditions in each state.

We define the satisfaction set of a PCTL formula Φ, over a DTMCF $\mathcal{D}$, as follows.

$$Sat(\Phi) = \{(s, \psi) | \; (s, \psi) \models \Phi\}$$

According to the above definitions, the satisfaction set of state and path formulas over $\mathcal{D}$ can be defined as follows.

$$\begin{aligned}
&Sat(\mathbf{True}) = R(S) \\
&Sat(a) = \{(s, \psi) \in R(S) | a \in L(s)\} \\
&Sat(\neg\Phi) = R(S) \setminus Sat(\Phi) \\
&Sat(\Phi_1 \wedge \Phi_2) = Sat(\Phi_1) \cap Sat(\Phi_2) \\
&Sat(\mathbb{P}_J(\bigcirc\Phi)) = \{(s, \psi) \in R(S) | Pr((s, \psi) \models \bigcirc\Phi) \in J\} \\
&Sat(\mathbb{P}_J(\Phi_1 U \Phi_2)) = \\
&\qquad \{(s, \psi) \in R(S) | Pr((s, \psi) \models \Phi_1 U \Phi_2) \in J\} \\
&Sat(\mathbb{P}_J(\Phi_1 U^{\leq n} \Phi_2)) = \\
&\qquad \{(s, \psi) \in R(S) | Pr((s, \psi) \models \Phi_1 U^{\leq n} \Phi_2) \in J\}
\end{aligned}$$

As mentioned in Sect. 2, any PCTL path operation ($\bigcirc, U^{\leq n}$, U) should be preceded by the $\mathbb{P}_J$ operator. Here we focus on the model checking of state formulas like $\mathbb{P}_J(\varphi)$, as the model checking of other state formulas is straightforward.

Before explaining the model checking algorithm, we define the following notations:

$$\begin{aligned}
Pre_{\psi_t}(s, \psi) &\triangleq \{(s', \psi') | s' \in Succ_{\psi_t}(s) \wedge SAT(\psi' \wedge \psi_t \wedge \psi)\} \\
Pre(s, \psi) &\triangleq \bigcup_{\psi_t \in \Psi_F} Pre_{\psi_t}(s, \psi) \\
Post_{\psi_t}(s, \psi) &\triangleq \{(s', \psi') | (s, \psi) \in Pre_{\psi_t}(s', \psi')\} \\
Post(s, \psi) &\triangleq \bigcup_{\psi_t \in \Psi_F} Post_{\psi_t}(s, \psi)
\end{aligned}$$

As a notational convenience in the rest of this section, we use the notation $\sum_{a \in A} f(a)[P(a)]$ to calculate the sum of $f(a)$ over those elements $a \in A$ satisfying the formula $P(a)$ (borrowed from [14]).

5.1 Model Checking $\mathbb{P}_J(\bigcirc\Phi)$

The model checking of properties such as $\mathbb{P}_J(\bigcirc\Phi)$ is fulfilled through following steps:

1. Computing the satisfaction set of Φ
2. Computing the satisfaction set of $\mathbb{P}_J(\bigcirc\Phi)$ as described below.

$$Sat(\mathbb{P}_J(\bigcirc\Phi)) = \{(s, \psi \wedge \psi_t) | (s,\psi) \in R(S) \wedge \psi_t \in \Upsilon_s \wedge \sum_{s' \in Succ_{\psi_t}(s)} P_{\psi_t}(s,s')[\exists\psi' \cdot (s',\psi') \in Post_{\psi_t}(s,\psi) \cap Sat(\Phi)] \in J\}$$

The above formula computes the probability of s, reaching the states that satisfy Φ, in one step in the set of configurations specified by $\psi \wedge \psi_t$. As we mentioned before all application conditions on the outgoing transitions of s are either the same or exclusive (i.e., they cannot be enabled in the same set of configurations). Hence, the summation is only over the transitions with the same application conditions which lead to the states satisfying Φ in the same set of configurations. As an example, we assume s_1 is the only state labeled by the atomic proposition *detect*, the result of checking the property:
$\mathbb{P}_{>0.9}(\bigcirc(detect))$ against the DTMCF of the EBS product line would be the set $\{(s_0, \mathsf{L} \wedge \Phi_{EBS})\}$.

5.2 Model Checking $\mathbb{P}_J(\Phi_1 U \Phi_2)$

The model checking of a property such as $\mathbb{P}_J(\Phi_1 U \Phi_2)$ is fulfilled through the following steps. Like the case for the 'neXt' operator, we first compute the satisfaction sets of Φ_1 and Φ_2. Then, we compute three separate subsets of states S^{yes}, S^{no} and $S^?$ defined as:

$$\begin{aligned} S^{yes} &= Sat(\mathbb{P}_{\geq 1}(\Phi_1 U \Phi_2)) \\ S^{no} &= Sat(\mathbb{P}_{\leq 0}(\Phi_1 U \Phi_2)) \\ S^? &= R(S) \setminus S^{yes} \cup S^{no} \end{aligned}$$

The algorithms for computation of these three subsets are basically similar to the original ones in the PCTL model checking which are based on simple graph algorithms [19]. $S^{no} = R(S) - T$ where T is computed as follows. We first let $T := Sat(\Phi_2)$, and then we iteratively update $T := T \cup \{(s,\psi) \in Sat(\Phi_1) \mid \exists(s',\psi') \in T \cdot (s,\psi) \in Pre(s',\psi')\}$ until T is fixed. Likewise, $S^{yes} = R(S) - U$ where U is computed as follows. We first let $U := S^{no}$, then we iteratively update $U := U \cup \{(s,\psi) \in Sat(\Phi_1) \setminus Sat(\Phi_2) \mid \exists(s',\psi') \in U \cdot (s,\psi) \in Pre(s',\psi')\}$ until U is fixed.

As the behavior of the system after reaching these states does not affect the satisfaction of the property, we remove all outgoing transitions from S^{yes} and S^{no}.

Finally, to compute the satisfaction set of $\mathbb{P}_J(\Phi_1 U \Phi_2)$, we need to make an observation about the application conditions on the outgoing transitions from each state as follows. Consider $(s,\psi) \in R(s)$, $\psi_t, \psi'_t \in \Psi_F$,

$$Sat(\mathbb{P}_J(\Phi_1 U \Phi_2)) = \{(s,\psi) \in R(S) | \lim_{n\to\infty} X^n(s,\psi) \in J\}$$

Where X is a probability measure defined recursively as depicted in Figure 3.

Since the infinite computation of X^n is not possible, usually a threshold would be specified for the difference of two successive computed values of X^n.

As an example, we follow the above steps to verify a simple formula containing 'until' operator. Assuming that in the DTMCF of Figure 2(a), s_5 is the only state labeled with the atomic proposition *Crash* and s_4 is the only state labeled with the atomic proposition *Safe*, we want to find the configurations in which the probability of ending in a *safe* state without entering a *crash* state is more than 0.9. The PCTL formula for this property is $\mathbb{P}_{\geq 0.9}(\neg Crash\ U\ Safe)$. At the first step, the two satisfaction sets $Sat(\neg Crash)$ and $Sat(Safe)$ are computed:

$$\begin{aligned} Sat(\neg Crash) &= R(s) \setminus \{(s_5, \Phi_{EBS})\} \\ Sat(Safe) &= \{(s_4, \Phi_{EBS})\} \end{aligned}$$

Then, the sets S^{yes}, S^{no}, and $S^?$ are computed:

$$\begin{aligned} S^{no} &= \{(s_5, \Phi_{EBS})\} \\ S^{yes} &= \{(s_3, \Phi_{EBS}), (s_4, \Phi_{EBS}), (s_1, \Phi_{EBS} \wedge \mathsf{R}), \\ &\quad (s_2, \Phi_{EBS} \wedge \mathsf{R})\} \\ S^? &= R(S) \setminus \{S^{yes} \cup S^{no}\} \end{aligned}$$

After removing the outgoing transitions from the states in S^{yes} or S^{no}, we turn to the recursive computation of probabilities. The value of X for a tuple $(s, \psi \wedge \psi_t)$ in $S^?$ at each step is only dependent on the value of X for the tuples in $Post_{\psi_t}(s,\psi)$. For example:

$$\begin{aligned} X^n(s_2, \Phi_{EBS} \wedge \mathsf{C} \wedge \neg\mathsf{R}) &= P_{\mathsf{C}\wedge\neg\mathsf{R}}(s_2,s_3) X^{n-1}(s_3, \Phi_{EBS}) + \\ &\quad P_{\mathsf{C}\wedge\neg\mathsf{R}}(s_2,s_5) X^{n-1}(s_5, \Phi_{EBS}) \end{aligned}$$

Since $(s_3, \Phi_{EBS}) \in S^{yes}$ and $(s_5, \Phi_{EBS}) \in S^{no}$, we have $X^n(s_3, \Phi_{EBS}) = 1$ and $X^n(s_5, \Phi_{EBS}) = 0$ for every $n \geq 0$. As a result, $X^n(s_2, \Phi_{EBS} \wedge \mathsf{C} \wedge \neg\mathsf{R}) = 0.7 \times 1 + 0.3 \times 0 = 0.7, \forall n \geq 0$. Hence,

$$(s_2, \Phi_{EBS} \wedge \mathsf{C} \wedge \neg\mathsf{R}) \notin Sat(\mathbb{P}_{\geq 0.9}(\neg Crash\ U\ Safe)).$$

5.3 Model Checking $\mathbb{P}_J(\Phi_1 U^{\leq n} \Phi_2)$

The only difference between the computations of bounded until operator with until operator is the specific number of steps n. This way, the model checking of $\mathbb{P}_J(\Phi_1 U^{\leq n} \Phi_2)$ is very similar to the model checking of $\mathbb{P}_J(\Phi_1 U \Phi_2)$ consisting of the following steps. Like other operators, the satisfaction sets of the subformulas $Sat(\Phi_1)$ and $Sat(\Phi_2)$ must be computed first. We define the three subset of states S^{yes}, S^{no} and $S^?$ as:

$$\begin{aligned} S^{yes} &= Sat(\Phi_2) \\ S^{no} &= R(S) \setminus Sat(\Phi_1) \cup Sat(\Phi_2) \\ S^? &= R(S) \setminus S^{yes} \cup S^{no} \end{aligned}$$

Like the 'until' operator, we remove the outgoing transitions from the states in S^{yes} and S^{no}. Finally, the satisfaction set of $\mathbb{P}_J(\Phi_1 U^{\leq n} \Phi_2)$ is computed as follows where the function X is computed based on Figure 3.

$$Sat(\mathbb{P}_J(\Phi_1 U^{\leq n} \Phi_2)) = \{(s,\psi) \in R(S) | X^n(s,\psi) \in J\}$$

Note that all the computed values in terms of summations mentioned above are valid probabilities in the interval [0,1].

$$X^n(s,\psi) = \begin{cases} 0 & \text{if } (s,\psi) \in S^{no} \\ 1 & \text{if } (s,\psi) \in S^{yes} \\ 0 & \text{if } (s,\psi) \in S^{?} \wedge n = 0 \\ \displaystyle\sum_{\substack{(s,\psi_t,s') \\ \in Dom(P)}} P_{\psi_t}(s,s')X^{n-1}(s',\psi)[\exists (s,\psi') \in R(S) \cdot (s,\psi') \in Pre_{\psi_t}(s',\psi) \wedge (\psi_t \wedge \psi' \equiv \psi)] & \text{if } (s,\psi) \in S^{?} \wedge n > 0 \end{cases}$$

Figure 3: The probability measure used in the computation of until and bounded until satisfaction sets

As the application conditions on the outgoing transitions of an state are mutually exclusive, the summations are over the transitions with the same application conditions. Regarding the definition of DTMCF, the probabilities of the transitions with the same application condition sum up to 1. In each step, each of these probabilities is multiplied to a value less than or equal to 1, hence, the summation value on these multiplications are in the range [0,1].

6. RELATED WORK

We divide the existing work related to our method into two parts. The first part includes the existing methods on the quantitative analysis of software product lines, while the second part addresses the existing work on the formal modeling and verification of (non-probabilistic) SPLs.

6.1 Quantitative Analysis of SPLs

In [20] Nunes et al. use model checking of parametric Markov models [15] in order to address reliability analysis of software product line systems. This work deals with the process of generating an arithmetic formula corresponding to the reliability of an SPL, through parametric model checking using the PARAM tool [16]. A feature model is considered as the input to this approach and the output is a reliability formula for the SPL in which variables represent features of the feature model. There are also some techniques proposed to improve handling of the major problems of the parametric approach such as the limitations imposed on both the software architecture and the relations between the features, and the size of the formula which can grow rapidly in terms of the number of features.

There is also another approach proposed in [11] for the verification of quantitative non-functional properties of software product lines. In this work, the UML sequence diagrams are extended by variabilities to capture different behaviors of the software product lines. Two different methods are used to verify some non-functional properties which are specified in terms of PCTL reachability formulas. The first one is trivial one-by-one model checking of the products which is fulfilled by extracting the DTMCs, modeling the behavior of each product, from the mentioned sequence diagrams (using the technique presented in [12]). The other method is the parametric one, which is done via verifying the system behavior described by means of a parametric DTMC or parametric reward DTMC. The first method suffers from the fast growth of the number of the products (usually exponential in the number of features). The latter has some limitations on the system architecture and restrictions on feature relations.

6.2 Formal Verification of SPLs

In [8] Classen et al. introduce a variation of labeled transition systems in which transitions are labeled by features of a specific feature model. The resulting model is called a *feature transition system*. In this model, a priority relation is considered between transitions to express the relations between features as specified by the feature model. There is also an algorithm proposed to verify the feature transition systems against regular and ω-regular properties. This algorithm reduces the time to verify the behavior of all products by verifying the whole system model (expressed by a feature transition system) at once. The feature transition system is extended in [6] by using featured expressions instead of single features as transition labels. Also, a verification algorithm is proposed to verify the model against fLTL (an extension of LTL) formulas.

In [9] Cordy et al. propose a model and verification algorithm based on [8] for behavioral modeling and verification of real-time software product lines. In this work, an extension of timed automata, featured timed automata, is proposed, modeling all the possible behavior of the products compactly and two different techniques are applied to verify the SPL. The first is the one-by-one model checking of products and the second is a variability-aware approach which takes into account the common behavior between products in a SPL.

In [7] a symbolic representation of featured transition systems is presented that can be verified against an extension of CTL formulas, called fCTL. In this work, an extension of language NuSMV [21] is used for high-level description of the behavior of the software product line. It is shown that symbolic representation of featured transition systems leads to a more efficient model checking process.

The overall structure of our method is similar to this category, both in the way we extend DTMCs to incorporate variability information and the way we annotate the states of the DTMCF with application conditions when model-checking the family. However, there are subtleties in addressing the probabilistic nature of the model that arise in both modeling and verification of DTMC families.

7. CONCLUSION AND FUTURE WORK

In this paper we introduced a mathematical model, Discrete Time Markov Chain Family, to model all the possible probabilistic behaviors of the products of a software product line in a single model. We defined the satisfaction relation of PCTL formulas by DTMCFs, and presented a model checking algorithm to verify PCTL properties. Using our method, all products in the family can be verified against a property at once, without the need to derive the product models one-by-one and model check each separately. This would save verification time by avoiding redundant analysis arising from the commonalities among the product.

Although we currently have an implementation of our method, performing a realistic case study is left as a future work, since it needs a high-level language to describe a probabilistic software product line. Using symbolic techniques in model-checking PCTL formulas is another direction in our future work. This will enable model-checking of much larger probabilistic models.

8. REFERENCES

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

[2] D. Batory. Feature models, grammars, and propositional formulas. In *Proc. of the 9th intl. conference on Software Product Lines*, SPLC'05, pages 7–20, Berlin, Heidelberg, 2005. Springer-Verlag.

[3] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. volume 35, pages 615–636, Oxford, UK, UK, Sept. 2010. Elsevier Science Ltd.

[4] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, Aug. 1986.

[5] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[6] 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*, 99(PrePrints):1, 2013.

[7] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay. Symbolic model checking of software product lines. In *Proc. of the 33rd Inl. Conference on Software Engineering*, ICSE '11, pages 321–330, New York, NY, USA, 2011. ACM.

[8] 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*, ICSE '10, pages 335–344, New York, NY, USA, 2010. ACM.

[9] 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*, SPLC '12, pages 66–75, New York, USA, 2012. ACM.

[10] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.

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

[12] 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*, SPLC '11, pages 170–174, Washington, DC, USA, 2011. IEEE Computer Society.

[13] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat-solver. *Discrete Appl. Math.*, 155(12):1549–1561, June 2007.

[14] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

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

[16] 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*, CAV'10, pages 660–664, Berlin, Heidelberg, 2010. Springer-Verlag.

[17] H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *IEEE Real-Time Systems Symposium*, pages 102–111, 1989.

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

[19] M. Kwiatkowska and D. Parker. Advances in probabilistic model checking. In T. Nipkow, O. Grumberg, and B. Hauptmann, editors, *Software Safety and Security - Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 126–151. IOS Press, 2012.

[20] 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*, pages 51–60, 2012.

[21] M. Plath and M. Ryan. Feature integration using a feature construct. *Science of Computer Programming*, 41(1):53–84, 2001.

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

[23] H. Sabouri, M. M. Jaghoori, F. d. Boer, and R. Khosravi. Scheduling and analysis of real-time software families. In *Proc. of the 2012 IEEE 36th Annual Computer Software and Applications Conference*, COMPSAC '12, pages 680–689, Washington, DC, USA, 2012. IEEE Computer Society.

[24] A. von Rhein, S. Apel, C. Kästner, T. Thüm, and I. Schaefer. The pla model: on the combination of product-line analyses. In *Proc. of the Seventh Intl. Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 14:1–14:8, New York, NY, USA, 2013. ACM.

[25] H. Zhang. Sato: An efficient propositional prover. In *Proc. of the 14th Intl. Conference on Automated Deduction*, CADE '14, pages 272–275, London, UK, 1997. Springer-Verlag.