Science of Computer Programming 152 (2018) 116–160

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

All roads lead to Rome: Commuting strategies for product-line reliability analysis

Thiago Castro [a,b,*], André Lanna [a], Vander Alves [a,d], Leopoldo Teixeira [c], Sven Apel [d], Pierre-Yves Schobbens [e]

[a] Computer Science Department, University of Brasília, Campus Universitário Darcy Ribeiro – Edifício CIC/EST, 70910-900, Asa Norte, Brasília – DF, Brazil
[b] Systems Development Center, Brazilian Army, QG do Exército – Bloco G – 2ª Andar, 70630-901, Setor Militar Urbano, Brasília – DF, Brazil
[c] Informatics Center, Federal University of Pernambuco, Av. Jornalista Aníbal Fernandes, s/n – Cidade Universitária (Campus Recife), 50740-560, Recife – PE, Brazil
[d] Department of Informatics and Mathematics, University of Passau, Innstr. 33, 94032 Passau, Germany
[e] Faculty of Computer Science, University of Namur, rue Grandgagnage 21, 5000 Namur, Belgium

ARTICLE INFO

Article history:
Received 22 November 2016
Received in revised form 20 October 2017
Accepted 24 October 2017
Available online 31 October 2017

Keywords:
Software product lines
Product-line analysis
Reliability analysis
Model checking
Verification

ABSTRACT

Software product line engineering is a means to systematically manage variability and commonality in software systems, enabling the automated synthesis of related programs (*products*) from a set of reusable assets. However, the number of products in a software product line may grow exponentially with the number of features, so it is practically infeasible to quality-check each of these products in isolation. There is a number of *variability-aware* approaches to product-line analysis that adapt single-product analysis techniques to cope with variability in an efficient way. Such approaches can be classified along three analysis dimensions (product-based, family-based, and feature-based), but, particularly in the context of reliability analysis, there is no theory comprising both (a) a formal specification of the three dimensions and resulting analysis strategies and (b) proof that such analyses are equivalent to one another. The lack of such a theory hinders formal reasoning on the relationship between the analysis dimensions and derived analysis techniques. We formalize seven approaches to reliability analysis of product lines, including the first instance of a feature-family-product-based analysis in the literature. We prove the formalized analysis strategies to be sound with respect to the probabilistic approach to reliability analysis of a single product. Furthermore, we present a commuting diagram of intermediate analysis steps, which relates different strategies and enables the reuse of soundness proofs between them.

1. Introduction

Software product line engineering is a means to systematically manage variability and commonality in software systems, enabling the automated synthesis of related programs (known as *variants* or simply *products*) from a set of reusable

* Corresponding author at: Computer Science Department, University of Brasília, Campus Universitário Darcy Ribeiro – Edifício CIC/EST, 70910-900, Asa Norte, Brasília – DF, Brazil.
E-mail addresses: thiago.mael@aluno.unb.br (T. Castro), andrelanna@unb.br (A. Lanna), valves@unb.br (V. Alves), lmt@cin.ufpe.br (L. Teixeira), apel@uni-passau.de (S. Apel), pierre-yves.schobbens@unamur.be (P.-Y. Schobbens).

https://doi.org/10.1016/j.scico.2017.10.013
0167-6423/

assets (known as *domain artifacts*) [15,42,1]. In a product line, variability is modeled in terms of features, which are distinguishable characteristics that are relevant to stakeholders of the system [16]. This methodology improves productivity and time-to-market, and it eases mass customization of software [42].

In recent years, product lines have been widely applied in both industry [50,36] and academia [1,15,28,42], in particular to safety- and mission-critical systems [50,20,19,34,46]. Model checking is of particular interest to quality assurance of such systems. It is a verification technique that explores all possible system states in a systematic manner, effectively checking that a given system model satisfies a certain property [5].

The number of products in a product line may grow exponentially with the number of features, giving rise to an *exponential blowup* of the configuration space [1,6,14,13]. So, it is often infeasible to quality-check each of these products in isolation. Nonetheless, software verification techniques for the single-product case are widely used by the industry, and it is beneficial to exploit their maturity to increase quality while reducing cost and risk [5].

There is a number of approaches to product-line analysis that adapt established analysis techniques to cope with variability [47]. In particular, several model checking techniques have been successfully lifted to operate on product lines [47, 11–13,21,9,22,32,46,40]. Among these techniques, we focus on reliability analysis, which is the verification of a probabilistic existence property [23] and can be seen as the probability that a system does not fail.

Product-line analyses can be classified along three dimensions: product-based (the analysis is performed on generated products or models thereof), family-based (only domain artifacts and valid combinations thereof are checked), and feature-based (domain artifacts implementing a given feature are analyzed in isolation, regardless of their valid combinations) [47]. More than one dimension can be exploited in a given technique, giving rise to feature-family-based and family-product-based analyses, for instance. However, existing approaches to the problem of lifting standard analysis techniques to product lines often focus on the family-based dimension [39,45,21,9,7] and relate it only to the product-based dimension to ensure soundness. In the context of reliability analysis, particularly, there is no theory comprising both (a) a formal specification of the three dimensions and resulting analysis strategies and (b) proof that such analyses are equivalent to one another (i.e., they compute the same reliability).

The lack of such a theory hinders formal reasoning on the relationship between the dimensions and derived analyses. Indeed, proving that an analysis method yields a correct result is a fundamental issue, especially for critical systems. Furthermore, a practitioner needs to be able to choose among existing analysis strategies according to the problem at hand, based on their trade-offs in terms of space and time [47]. As long as there is no evidence that different strategies are mutually equivalent, empirical studies comparing them will have limited results.

Based on the product-line analysis taxonomy proposed by Thüm et al. [47], we formalize seven approaches to reliability analysis of product lines: two product-based, a family-based, a family-product-based, a feature-family-based, a feature-product-based, and a feature-family-product-based. In particular, the latter of these is a novel approach, according to a recent survey [47].

We prove the formalized analysis strategies to be sound with respect to the probabilistic approach to reliability analysis of individual products. Furthermore, we present a commuting diagram of intermediate analysis steps, which relates different strategies and enables the reuse of soundness proofs between them. In this sense, all strategy choices are guaranteed to yield the same result.

The main contributions of this work are the following:

- The formalization of seven strategies for reliability analysis of software product lines, conforming to the classification by Thüm et al. [47] (Section 4).
- A novel feature-family-product-based strategy for model checking of product lines (Section 4.5). To the best of our knowledge, and according to the survey by Thüm et al. [47], this is the first strategy in its category.
- Proofs of commutativity between different strategies (Section 4). This improves the current understanding on how analysis strategies for product lines relate to one another and establishes their soundness.
- A commuting diagram of intermediate analysis steps (Fig. 10), which relates different strategies and enables the reuse of soundness proofs between them.
- A general principle for lifting analyses to product lines using algebraic decision diagrams (Section 4.2.2, Theorem 2).

This work first provides fundamental concepts necessary for the discussion of product-line analyses (Section 2), followed by an explanation of stochastic models of product-line behavior and corresponding model checking techniques (Section 3). We proceed with formal definitions of strategies and proofs of their commutativity (Section 4), ending with a discussion of related work (Section 5) and ideas for further research regarding commonality of product line analysis strategies (Section 6).

2. Background

In this section, we lay the foundations for the upcoming discussion. The reader is expected to have some familiarity with discrete-time Markov chains (DTMC), in particular from a state-based perspective (as presented, for instance, in the book by Baier and Katoen [5]).

2.1. Software product lines

A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs of a particular market or mission, and that are developed from a common set of core assets in a prescribed way [15]. A feature model documents the features of a product line and their relationships [1]. For a feature model *FM*, we denote its set of features by F. Each feature in this set has a name; feature names are used as atomic propositions to express feature relationships as propositional logic formulae.

A given product is specified by a *configuration*. A configuration is a selection of features and, as such, is represented by a set of atoms: a negated atom denotes feature absence, while a positive one denotes presence. We denote the set of configurations over a feature set F as C. This set contains all $2^{|F|}$ combinations of feature atoms, each of which must appear in either positive or negative form, but never both. Valid configurations, that is, configurations that satisfy the constraints expressed by the feature model *FM*, are denoted by $[\![FM]\!] \subseteq C$. Each $c \in [\![FM]\!]$ specifies the features of a product of the product line.

Product derivation is the process by which reusable assets are combined to form a product, according to a specified configuration. Actual behavior is included or excluded from a generated product by means of *presence conditions*, which are propositional formulae over features [17].

To operationalize satisfaction of presence conditions, we need to define Boolean functions over feature selections. Therefore, we define an arbitrary (but fixed) total order of features by turning the set F of features into a list. This way, we can unambiguously denote a configuration $c \in [\![FM]\!]$ as a Boolean tuple in $\mathbb{B}^{|F|}$, where $\mathbb{B} = \{0, 1\}$ is the set of Boolean values (where 0 and 1 denote the Boolean values `FALSE` and `TRUE`, respectively). Such Boolean tuples have a fixed position for each feature, with the i-th position denoting presence or absence of the i-th feature[1] by the values 1 and 0, respectively. In the upcoming discussion, whenever we refer to k-ary Boolean functions, we assume that Boolean k-tuples can be used as arguments.

2.2. Reliability analysis

Reliability analysis can be defined as a probabilistic existence property [23]. This means the reliability of a system is the probability that, starting from an initial state, the system reaches a set of *target* (also *success*) states. This value is called *reachability probability*. To analyze this property, we first model the system's behavior as a DTMC—a tuple $(S, s_0, \mathbf{P}, T)$, where S is a set of states, $s_0 \in S$ is the initial state, $\mathbf{P}$ is the transition probability matrix $\mathbf{P} : S \times S \to [0, 1]$, and $T \subseteq S$ is the set of target states.[2] Moreover, each row of the transition probability matrix sums to 1, that is, $\forall_{s \in S} \cdot \mathbf{P}(s, S) = 1$, where $\mathbf{P}(s, S) = \sum_{s' \in S} \mathbf{P}(s, s')$.

For every state $s \in S$, we say that a state s' is a *successor* of s iff $\mathbf{P}(s, s') > 0$. Accordingly, the set of successor states of s, $Succ(s)$, is defined as $Succ(s) = \{s' \in S \mid \mathbf{P}(s, s') > 0\}$. A DTMC induces an underlying digraph where states act as vertices and edges link states to their successors. This way, we say that a state s' of a DTMC is *reachable* from a state s, denoted by $s \rightsquigarrow s'$, iff s' is reachable from s in the DTMC's underlying digraph. Likewise, we write $s \not\rightsquigarrow s'$ to denote that s' is unreachable from s. This notation is also used with respect to a set T of states: $s \rightsquigarrow T$ iff there is at least one state $s' \in T$ such that $s \rightsquigarrow s'$, and $s \not\rightsquigarrow T$ otherwise.

The reachability probability for a DTMC can be computed using probabilistic model checking algorithms, implemented by off-the-shelf tools [5,33]. An intuitive and correct view of reachability probability, although not well-suited for efficient implementation, is that a target state is reached either directly or by first transitioning to a state that is able to recursively reach it. We present a formalization of this property, adapted from Baier and Katoen [5], that suits the purpose of this work.

Property 1 *(Reachability probability for DTMCs).* Given a DTMC $\mathcal{D} = (S, s_0, \mathbf{P}, T)$, a state $s \in S$, and a set $T \subseteq S$ of target states, the probability of reaching a state $t \in T$ from s satisfies the following property:

$$Pr^{\mathcal{D}}(s, T) = \begin{cases} 1 & \text{if } s \in T \\ 0 & \text{if } s \not\rightsquigarrow T \\ \sum_{s' \in S \setminus T} \mathbf{P}(s, s') \cdot Pr^{\mathcal{D}}(s', T) + \sum_{t \in T} \mathbf{P}(s, t) & \text{if } s \notin T \wedge s \rightsquigarrow T \end{cases}$$

Whenever T is a singleton $\{t\}$, we write $Pr^{\mathcal{D}}(s, t)$ to denote $Pr^{\mathcal{D}}(s, T)$.

In a product line, different products give rise to distinct behavioral models. To handle the behavioral variability that is inherent to product lines, we resort to *Parametric Markov Chains* [18].

2.2.1. Parametric Markov Chains

Parametric Markov Chains (PMC) extend DTMCs with the ability to represent *variable* transition probabilities. Whereas probabilistic choices are fixed at modeling time and represent possible behavior that is unknown until run time, variable

[1] The actual order of features does not affect our results, since its only purpose is to consistently refer to values in Boolean tuples.

[2] This definition departs from the one by Baier and Katoen [5] in two ways: (a) we abstract the possibility of multiple initial states and the computation of other temporal properties (to focus on reliability analysis) and (b) we incorporate target states in the model (to abbreviate model checking notation).

transitions represent behavior that is unknown already at modeling time. These variable transition probabilities can be leveraged to represent product-line variability [46,22,9].

Definition 1 *(Parametric Markov Chain).* A Parametric Markov Chain is defined by Hahn et al. [26] as a tuple $\mathcal{P} = (S, s_0, X, \mathbf{P}, T)$, where S is a set of states, s_0 is the initial state, $X = \{x_1, \ldots, x_n\}$ is a finite set of parameters, $\mathbf{P}$ is the transition probability matrix $\mathbf{P} : S \times S \to \mathcal{F}_X$, and $T \subseteq S$ is the set of *target* (or *success*) states. The set $\mathcal{F}_X$ comprises the *rational expressions* over $\mathbb{R}$ with variables in X, that is, fractions of polynomials with Real coefficients. This way, the semantics of a rational expression ε is a *rational function* $f_\varepsilon(x_1, \ldots, x_n) = \frac{p_1(x_1,\ldots,x_n)}{p_2(x_1,\ldots,x_n)}$ from $\mathbb{R}^n$ to $\mathbb{R}$, where p_1 and p_2 are Real polynomials. For brevity, we hereafter refer to rational expressions simply as *expressions*.

By attributing values to the variables, it is possible to obtain an ordinary (non-parametric) DTMC. Parameters are given values by means of an *evaluation*, which is a total function[3] $u : X \to \mathbb{R}$ for a set X of variables. For an expression $\varepsilon \in \mathcal{F}_X$ and an evaluation $u : X' \to \mathbb{R}$ (where X' is a set of variables), we define $\varepsilon[X/u]$ to denote the expression obtained by replacing every occurrence of $x \in X \cap X'$ in ε by $u(x)$, also denoted by $\varepsilon[x_1/u(x_1), \ldots, x_n/u(x_n)]$.

For instance, suppose we have sets of variables $X = \{x, y\}$ and $X' = \{x, y, z\}$, and an evaluation $u = \{x \mapsto 2, y \mapsto 5, z \mapsto 3\}$. If $\varepsilon \in \mathcal{F}_X$ is the rational expression $x - 2y$, then $\varepsilon[X/u] = \varepsilon[x/2, y/5] = 2 - 2 \cdot 5 = -8$. Note that, if u's domain, X', is different from the set X of variables in ε, then $\varepsilon[X/u] = \varepsilon[(X \cap X')/u]$.

This definition can be extended to substitutions by other expressions. Given two variable sets X and X', their respective induced sets of expressions $\mathcal{F}_X$ and $\mathcal{F}_{X'}$, and an expression $\varepsilon \in \mathcal{F}_X$, a generalized evaluation function $u : X \to \mathcal{F}_{X'}$ substitutes each variable in X for an expression in $\mathcal{F}_{X'}$. The generalized evaluation $\varepsilon[X/u]$ then yields an expression $\varepsilon' \in \mathcal{F}_{X'}$. Moreover, successive expression evaluations can be thought of as rational function compositions: for $u : X \to \mathcal{F}_{X'}$ and $u' : X' \to \mathbb{R}$,

$$\varepsilon[X/u][X'/u'] = \varepsilon[x_1/u(x_1)[X'/u'], \ldots, x_k/u(x_k)[X'/u']] \tag{1}$$

for $x_1, \ldots, x_k \in X$ (since u is a total function, we do not need to consider non-evaluated variables).

The PMC induced by an evaluation u is denoted by $\mathcal{P}_u = (S, s_0, \emptyset, \mathbf{P}_u, T)$ (alternatively, $\mathcal{P}[X/u]$), where $\mathbf{P}_u(s, s') = \mathbf{P}(s, s')[X/u]$ for all $s, s' \in S$. To ensure the resulting chain after evaluation is indeed a valid DTMC, one must use a *well-defined* evaluation.

Definition 2 *(Well-defined evaluation).* An evaluation $u : X \to \mathbb{R}$ is *well-defined* for a PMC $\mathcal{P} = (S, s_0, X, \mathbf{P}, T)$ iff, for all $s, s' \in S$, it holds that

- $\mathbf{P}_u(s, s') \in [0, 1]$ (all transitions evaluate to valid probabilities)
- $\mathbf{P}_u(s, S) = 1$ (stochastic property—the probability of disjoint events must add up to 1)

Hereafter, we drop explicit mentions to well-definedness whenever we consider an evaluation or a DTMC induced by one, because we are only interested in this class of evaluations. Nonetheless, we still need to prove that specific evaluations are indeed well-defined.

2.2.2. Parametric probabilistic reachability

To compute the reachability probability in a model with variable transitions, we use a parametric probabilistic reachability algorithm. A parametric model checking algorithm for probabilistic reachability takes a PMC $\mathcal{P}$ as input and outputs a corresponding expression ε representing the probability of reaching its set T of target states. Hahn et al. [26] present such an algorithm and prove that evaluating ε with an evaluation u yields the reachability probability for the DTMC induced in $\mathcal{P}$ by the same evaluation u.

Fig. 1 [26] illustrates a single step of this parametric probabilistic reachability algorithm. The main idea is that, for a given state s, the probability of one of its predecessors (s_1) reaching one of its successors (s_2) is given by the sum of the probability of transitioning through s and the probability of bypassing it. In this example, other states and respective transitions are omitted. Note that, since there is a self-loop with probability p_c, there are infinite possible paths going through s, each corresponding to a number of times the loop transition is taken before transitioning to s_2. Hence, the sum of probabilities for these paths correspond to the infinite sum $\sum_{i=0}^{\infty} p_a (p_c)^i p_b = p_a (\sum_{i=0}^{\infty} p_c^i) p_b = p_a \frac{1}{1-p_c} p_b$.[4]

Definition 3 *(State elimination step).* Given a PMC $\mathcal{P} = (S, s_0, X, \mathbf{P}, T)$ and an arbitrary state $s \in S$, a state elimination step of the algorithm by Hahn et al. [26] updates the transition matrix $\mathbf{P}$ to $\mathbf{P}'$, such that, for all states $s_1, s_2 \in S \setminus \{s\}$,

$$\mathbf{P}'(s_1, s_2) = \mathbf{P}(s_1, s_2) + \mathbf{P}(s_1, s) \cdot \frac{1}{1 - \mathbf{P}(s, s)} \cdot \mathbf{P}(s, s_2)$$

[3] Hahn et al. [26] actually define it in a more general way as a partial function. However, for our purpose, it suffices to consider total functions.

[4] Whenever $0 < x < 1$, we have the following convergent sum: $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$.

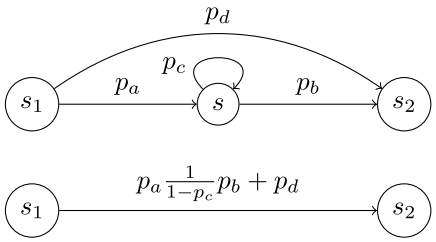


Fig. 1. Elimination of state s in the algorithm by Hahn et al. [26].

The soundness of the parametric probabilistic reachability algorithm by Hahn et al. [26] is expressed by the following lemma and summarized by the commuting diagram in Fig. 2.

Lemma 1 *(Parametric probabilistic reachability soundness). Let $\mathcal{P} = (S, s_0, X, \mathbf{P}, T)$ be a PMC, u be a well-defined evaluation for $\mathcal{P}$, and ε be the output of the parametric probabilistic reachability algorithm by Hahn et al. [26] for $\mathcal{P}$ and T. Then, $Pr^{\mathcal{P}_u}(s_0, T) = \varepsilon[X/u]$.*

Proof. The algorithm by Hahn et al. [26] is based on eliminating states until only the initial and the target ones remain. Its proof consists of showing that each elimination step preserves the reachability probability. We refer the reader to the work by Hahn et al. [26] for more details on the algorithm itself and the proof mechanics. □

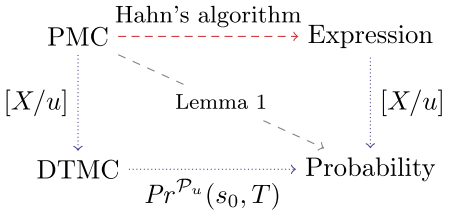


Fig. 2. Statement of Lemma 1.

2.3. Algebraic Decision Diagrams

An Algebraic Decision Diagram (ADD) [4] is a data structure that encodes k-ary Boolean functions $\mathbb{B}^k \to \mathbb{R}$. As an example, Fig. 3 depicts an ADD representing a binary function f. Each internal node in the ADD (one of the circular nodes) marks a decision over a single parameter. Function application is achieved by walking the ADD along a path that denotes this decision over the values of actual parameters: if the parameter represented by the node at hand is 1 (*true*), we take the solid edge; otherwise, if the actual parameter is 0 (*false*), we take the dashed edge. The evaluation ends when we reach a terminal node (one of the square nodes at the bottom).

In the example, to evaluate $f(1, 0)$, we start in the x node, take the solid edge to node y (since the actual parameter x is 1), then take the dashed edge to the terminal 0.8. Thus, $f(1, 0) = 0.8$. Henceforth, we will use a function application notation for ADDs, meaning that, if A is an ADD that encodes function f, then $A(b_1, \ldots, b_k)$ denotes $f(b_1, \ldots, b_k)$. For brevity, we also denote indexed parameters $b_1, \ldots, b_k$ as $\bar{b}$, and the application $A(\bar{b})$ by $[\![A]\!]_{\bar{b}}$.

ADDs have several applications, two of which are of direct interest to this work. The first one is the efficient application of arithmetics over Boolean functions. We employ Boolean functions to represent mappings from product-line configurations (Boolean tuples) to their respective reliabilities. An important aspect that motivated the use of ADDs for this variability-aware arithmetics is that the enumeration of all configurations to perform Real arithmetics on the corresponding reliabilities is usually subject to exponential blowup. ADD arithmetic operations are linear in the input size, which, in turn, can also be exponential in the number of Boolean parameters (i.e., ADD variables), in the worst case. However, given a suitable variable ordering, ADD sizes are often polynomial, or even linear [4]. Thus, for most practical cases, ADD operations are more efficient than enumeration.

$$f(x, y) = \begin{cases} 0.9 & \text{if } x \wedge y \\ 0.8 & \text{if } x \wedge \neg y \\ 0 & \text{otherwise} \end{cases}$$

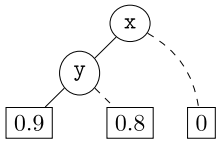


Fig. 3. ADD A_f representing the Boolean function f on the left.

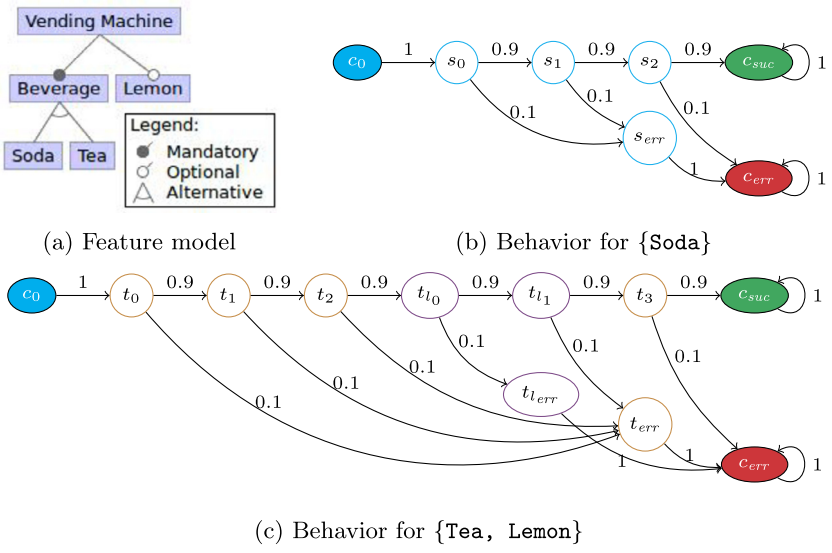


Fig. 4. Vending machine product line example.

An arithmetic operation over ADDs is equivalent to performing the same operation on corresponding terminals of the operands. Thus, we denote ADD arithmetics by corresponding real arithmetics operators. Formally, given a valuation for Boolean parameters $\bar{b} = b_1, \ldots, b_k \in \mathbb{B}^k$, it holds that:

1. $\forall_{\odot \in \{+,-,\times,\div\}} \cdot (A_1 \odot A_2)(\bar{b}) = A_1(\bar{b}) \odot A_2(\bar{b})$
2. $\forall_{i \in \mathbb{N}} \cdot A_1^i(\bar{b}) = A_1(\bar{b})^i$

The second application of interest is the algorithmic encoding of the result of an *if-then-else* operation over ADDs again as another ADD. For the ADDs A_{cond}, A_{true}, and A_{false}, we define the ternary operator ITE (*if-then-else*) as

$$\texttt{ITE}(A_{cond}, A_{true}, A_{false})(c) = \begin{cases} A_{true}(c) & \text{if } A_{cond}(c) \neq 0 \\ A_{false}(c) & \text{if } A_{cond}(c) = 0 \end{cases}$$

More details on the algorithms for ADD operations are outside the scope of this work and can be found elsewhere [4].

3. Markov-chain models of product lines

Reliability analysis, in our setting, is the application of probabilistic model checking to a probabilistic model of a software system. However, for a product line, it may not be feasible to manually model each product (i.e., its probabilistic model) and then analyze it, due to exponential blowup. Hence, we model the product line as a whole in terms of its common and variable behavior, to enable the automatic derivation of probabilistic models corresponding to the behavior of each product of the product line. Such variable behavioral models have properties that allow them to be used with different analysis strategies, as we will show in Section 4. Although we show and use precise definitions of the resulting models, it is outside the scope of this work to present modeling techniques to create them. Models can be produced, for example, by using behavioral UML diagrams annotated with component reliabilities [22,41] or feature-oriented formalisms [9].

Since single-product analysis relies on DTMCs to model software behavior, we use PMCs to represent DTMC variability in product-line analysis. To illustrate our approaches to variability representation and product-line analysis, yet without loss of generality, we rely on an example product line of beverage vending machines (Fig. 4), slightly modified from the examples in the work by Ghezzi and Molzam Sharifloo [22] and Classen et al. [14]. This product line consists of models of vending machines that are able to deliver tea or soda (but never both) and, for each case, there is a beverage-specific optional behavior of adding a certain quantity of lemon juice.

The feature model for this product line is depicted in Fig. 4a, where `Soda` and `Tea` are alternative features (i.e., they cannot be simultaneously present in a feature selection) representing the behaviors of serving soda and tea, respectively. Since adding lemon to a beverage is an optional behavior, it is modeled by the optional feature `Lemon`. If a product is generated with the feature selection {`Soda`} (i.e., `Lemon` is not selected), a possible model of its probabilistic behavior is depicted in Fig. 4b. If the feature selection is {`Tea, Lemon`}, the derived product has a probabilistic behavioral model as in Fig. 4c.

In both example DTMCs, transitions indicate a change in the machine's execution state, with probabilities representing the reliabilities of the corresponding execution steps. These reliabilities are usually taken to be the probabilities that the

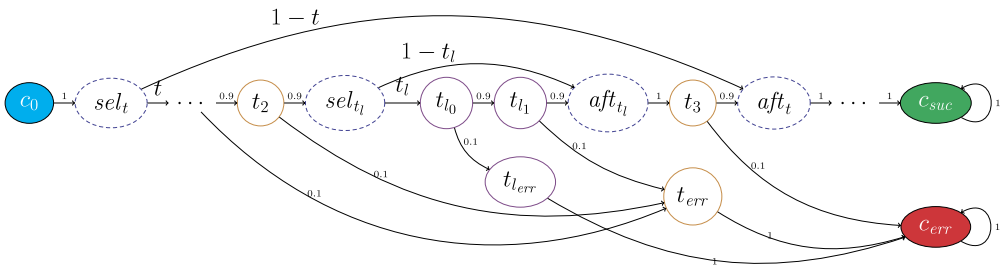


Fig. 5. Annotative PMC for the vending machine. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

software components responsible for each step will successfully produce the expected outcome. In this sense, one can notice most states have two outgoing transitions: one representing success and another representing failure. The states with only one outgoing transition may be seen as execution control handoffs. Also, to help us identify variation points, states are labeled according to the behavior they model and are correspondingly colored. Label c denotes *common* behavior (present in all products), while s and t denote behaviors introduced by features `Soda`, and `Tea`, respectively. States labeled t_l correspond to the behavior of adding lemon to tea, that is, they only exist in products derived by a feature selection with both features `Tea` and `Lemon`.

As with source code, the way variability is represented as PMCs and the way products (i.e., DTMCs) are generated from the resulting variable assets can be classified in two main categories: annotation-based (or annotative) and composition-based (or compositional) [29,1]. We now discuss both kinds of models, since each will play a role in the analysis strategies presented in Section 4. We also present a correspondence between compositional and annotative models in Section 4.4.

3.1. Annotative models

To represent the variable behavior of a product line in an annotative way, we use a PMC in which variables are interpreted as configuration-specific behavior selectors. Such a PMC for the vending machine product line is shown in Fig. 5, where we introduce blue dashed states to represent configuration-specific behavior selection. For instance, to represent the variability for `Tea`-related behavior, we introduce a state labeled sel_t, which transitions to t_0 (not shown) with probability 1, if it is present, or transitions to the point right after the same behavior (a state correspondingly labeled aft_t) with probability 1, if it is absent.[5] This mutually exclusive selection is represented by labeling transitions with the expressions t and $1-t$, such that evaluating t as 1 yields the expected "present" behavior, while evaluating it with 0 yields the "absent" behavior. The same approach is also applied to the behavior corresponding to adding lemon to tea. Some states of the model for serving tea, as well as the behaviors corresponding to `Soda` and its lemon-adding variant, are omitted for brevity. The whole model can be seen in Fig. A.1.

We generalize and formally define this annotative approach of variability representation as follows.

Definition 4 *(Annotative PMC).* An annotative PMC is a PMC $(S, s_0, X, \mathbf{P}, T)$ such that for all states $s \in S$, either:

1. $\forall_{s' \in S} \cdot \mathbf{P}(s, s') \in [0,1] \wedge \mathbf{P}(s, S) = 1$ (the probabilities of all outgoing transitions are constants that add up to 1); or
2. $\exists_{s_0, aft_s \in S}\ \exists_{x \in X} \cdot Succ(s) = \{s_0, aft_s\} \wedge \mathbf{P}(s, s_0) = x \wedge \mathbf{P}(s, aft_s) = 1 - x$ (there are exactly two outgoing transitions, whose probabilities are expressed as a single variable and its complement).

The states in Fig. 5 that fall in the second case are sel_t and sel_{t_l} (as well as sel_s and sel_{s_l}, which are not shown), while all others fall in the first case. Each variable of an annotative PMC denotes the presence of a given behavior in a product. The intended semantics is that the sets of states and transitions giving rise to the denoted behavior will be reachable within the model if, and only if, its corresponding variable evaluates to 1.

For such an annotative PMC to represent the variable behavior of a product line with feature model *FM*, we must be able to use it to derive the behavioral model of any product generated by a configuration $c \in FM$. However, the use of a PMC by itself does not help with restricting the possible evaluations to achieve that. Evaluating the introduced variables with values other than 0 and 1 may yield ill-formed DTMCs (e.g., violating the stochastic property). Also, a variable should evaluate to 1 if, and only if, the presence condition of the subsystem whose behavior is controlled by this variable is satisfied. Hence, we need to constrain evaluations of this annotative PMC to reflect the corresponding feature model and presence conditions.

The first step towards this goal is to formalize what presence conditions mean in the context of variable behavioral models. Thus, let p_x be the presence condition for the behavior identified by x. In our vending machine example, we would have $p_t = \texttt{Tea}$, $p_{t_l} = \texttt{Tea} \wedge \texttt{Lemon}$, $p_s = \texttt{Soda}$, and $p_{s_l} = \texttt{Soda} \wedge \texttt{Lemon}$. To precisely associate a variable to a presence

[5] The states sel_t and aft_t are analogous to the `#ifdef` and `#endif` macros of the C preprocessor, usually seen in preprocessor-based product lines.

condition, we define a higher-order function that maps a variable to a Boolean function over the features (see Section 2.1), which we call *presence function*.

Definition 5 *(Presence function).* Given a set X of variables and a feature model FM, a presence function is a function $p : X \to [\![FM]\!] \to \mathbb{B}$ such that, for all $x \in X$ and all $c \in [\![FM]\!]$,

$$p(x)(c) = \begin{cases} 1 & \text{if } c \models p_x \quad \textit{(presence condition is satisfied)} \\ 0 & \text{otherwise} \end{cases}$$

where p_x is the presence condition associated with the variable x and $c \models p_x$ means that the configuration c *satisfies* p_x.

Next, we must be able to use the feature model to define evaluations. For instance, the annotative PMC for the vending machine product line would allow serving both tea and soda, if both t and s were evaluated to 1. However, this behavior is forbidden by the feature model, which states that `Tea` and `Soda` are alternative features. By incorporating knowledge of the feature model to evaluations, we can model all variant behavior as if it were optional and enforce the constraints of alternative and OR features when evaluating the PMC. The solution to this problem are higher-order functions complying to the following definition of an *evaluation factory*.

Definition 6 *(Evaluation factory).* Given a feature model FM and a set X of variables, an evaluation factory $w : [\![FM]\!] \to X \to \mathbb{R}$ is a function that, for a given configuration $c \in [\![FM]\!]$, yields an evaluation $w(c) \in X \to \mathbb{R}$.

At this point we have defined what we mean by an annotative PMC as well as an abstract means to constrain possible evaluations to the ones that make sense in the context of a given product line. For the particular case of annotative PMCs, an evaluation factory must generate evaluations that interpret variables as presence values and according to the presence conditions. Thus, we need to interpret the set $\{0, 1\}$ of *numbers* as the set $\mathbb{B}$ of Boolean values and restrict the generated evaluations to have this set as image. With this in mind, we define an *annotative probabilistic model* as follows:

Definition 7 *(Annotative probabilistic model).* An annotative probabilistic model is a tuple $(\mathcal{P}, p, w, FM)$ such that:

- $\mathcal{P} = (S, s_0, X, \mathbf{P}, T)$ is an annotative PMC (Definition 4);
- FM is a feature model;
- $p : X \to [\![FM]\!] \to \mathbb{B}$ is a presence function (Definition 5); and
- w is an evaluation factory (Definition 6) such that, for all $c \in [\![FM]\!]$ and $x \in X$,

$$w(c)(x) = \begin{cases} 1 & \text{if } p(x)(c) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Remark 1 *(Pointwise definition of w).* For practical purposes, it is worth noting that the right-hand sides of the definitions of w (Definition 7) and of the presence function p (Definition 5) are the same. That is, one can operationalize w as $w(c)(x) = p(x)(c)$, so the annotative evaluation factory could be uniquely determined from an annotative PMC $\mathcal{P}$, a presence function p, and a feature model FM. Nonetheless, we keep w as part of the annotative model tuple for uniformity, since it is the annotative counterpart of the composition factory w' in a compositional probabilistic model (Definition 17). The definitions of the presence function and the annotative evaluation factory are only similar because the set of Real values in the image of the possible evaluations (i.e., $\{0, 1\}$) in the annotative case correspond to our Real encoding of Boolean values.

Starting with such an annotative model, the derivation of a specific behavioral model of a product with configuration $c \in [\![FM]\!]$ is then carried out by applying the evaluation $w(c)$ to the underlying PMC $\mathcal{P}$. Since PMC evaluation is not restricted to annotative PMCs, we define this process of DTMC derivation (which is the basis for product derivation) without resorting to the just defined concept of annotative models.

Definition 8 *(DTMC derivation).* Given a PMC $(S, s_0, X, \mathbf{P}, T)$, a feature model FM, and an evaluation factory $w : [\![FM]\!] \to X \to \mathbb{R}$, the DTMC derivation function $\pi : PMC_X \times \left([\![FM]\!] \to X \to \mathbb{R}\right) \times [\![FM]\!] \to DTMC$ is such that

$$\pi(\mathcal{P}, w, c) = \mathcal{P}_{w(c)}$$

where PMC_X is the set of PMCs with variables set X. For brevity, we can also note $[\![\mathcal{P}]\!]_c^w$ to mean $\pi(\mathcal{P}, w, c)$.

Note that the analysis methods we exploit in this work rely on evaluations being well-defined (Definition 2). This is where the restrictions we imposed on annotative models come into play: the evaluation factory of an annotative model always yields well-defined evaluations for the underlying annotative PMC.

Lemma 2 *(Evaluation well-definedness for annotative models). For every annotative model $(\mathcal{P}, p, w, FM)$, $w(c)$ is a well-defined evaluation for $\mathcal{P}$, for all $c \in [\![FM]\!]$.*

Proof. By definition of well-defined evaluation for a PMC $\mathcal{P} = (S, s_0, X, \mathbf{P}, T)$ (Definition 2), an evaluation u is well-defined iff $\mathcal{P}_u$ obeys the stochastic property and $\mathbf{P}_u$ assigns a valid probability value to each transition. That is, $\forall_{s \in S} \cdot \mathbf{P}_u(s, Succ(s)) = 1$ and $\forall_{s,s' \in S} \cdot \mathbf{P}_u(s, s') \in [0, 1]$.

From Definition 7, $\mathcal{P}$ is an annotative PMC (Definition 4), so states with no variability (Case 1) satisfy the needed properties by definition. For states s with variability (Case 2), it holds that

$$\exists_{s_1, s_2 \in S}\ \exists_{x \in X} \cdot Succ(s) = \{s_1, s_2\} \wedge \mathbf{P}(s, s_1) = x \wedge \mathbf{P}(s, s_2) = 1 - x$$

Let us consider each property whenever $u = w(c)$:

Stochastic property. By definition,

$$\begin{aligned}\sum_{s' \in Succ(s)} \mathbf{P}_{w(c)}(s, s') &= \mathbf{P}_{w(c)}(s, s_1) + \mathbf{P}_{w(c)}(s, s_2) \\ &= \mathbf{P}(s, s_1)[X/w(c)] + \mathbf{P}(s, s_2)[X/w(c)] \\ &= x[X/w(c)] + (1 - x)[X/w(c)] \\ &= w(c)(x) + (1 - w(c)(x)) \\ &= 1\end{aligned}$$

Valid probabilities. From Definition 7, we have that for every $c \in [\![FM]\!]$, the image of $w(c)$ is $\{0, 1\} \subseteq [0, 1]$. Hence, either $\mathbf{P}_{w(c)}(s, s_1) = 1 \wedge \mathbf{P}_{w(c)}(s, s_2) = 0$ or $\mathbf{P}_{w(c)}(s, s_1) = 0 \wedge \mathbf{P}_{w(c)}(s, s_2) = 1$. That is, all possible transition probabilities lie in the $[0, 1]$ interval.

As there is no other case to consider, $\mathcal{P}_{w(c)}$ satisfies the required properties. Thus, $w(c)$ is well-defined for $\mathcal{P}$. □

In summary, an annotative probabilistic model represents all products of the product line, relying on presence conditions to define which parts have to be removed to derive a concrete product model. Because of that, this type of model is also known as 150% model [24], metaproduct [48], variant simulator [45], or product simulator [2].

3.2. Compositional models

A compositional representation of variable configuration-specific behavior consists of a hierarchy of PMCs whose variables represent variation points, such that they can be composed with one another at predefined locations. To model a product line in this way, we start with a PMC comprising all common behavior, while abstracting all variable configuration-specific behavior. We then model each abstracted behavior as a DTMC, if it presents no further variability, or as another PMC, otherwise. In the latter case, we follow the same procedure to abstract inner variation points, until all behavior is modeled.

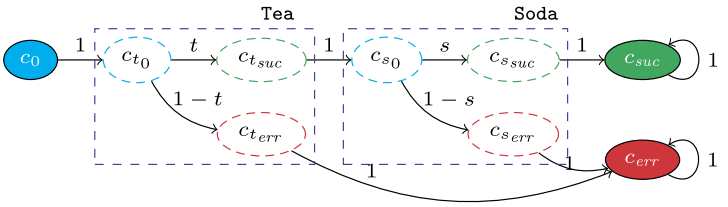
Fig. 6 illustrates this concept. For the vending machine example, the *top-level* PMC $\mathcal{P}_\top$ would be as in Fig. 6a. In this PMC, we introduce triples of dashed states that act as placeholders for the abstracted behavior. We call these states and corresponding transitions *slots*. For instance, the top-level PMC in Fig. 6a has two slots, abstracting the behaviors of serving tea and soda. The tea slot consists of two elements: (a) the set of states c_{t_0}, $c_{t_{suc}}$, and $c_{t_{err}}$, representing the initial, success, and error states in the abstracted behavior, respectively; and (b) two transitions, annotated with the expressions t and $1 - t$, denoting the probabilities of success and failure of this behavior, respectively. This way, we not only use the variable t as a slot identifier, but give it the possibility to be interpreted as the reliability of the tea behavior.

Note that, despite being alternatives, the behaviors of serving tea and soda are both represented in this PMC. This parametric model, by itself, does not prohibit the behavior of serving tea *and* soda subsequently. Like in the annotative representation of the vending machine (Fig. 5), we do not enforce the rules of the feature model in the PMC itself. Instead, we ensure valid combinations of features during the composition process, as we shall see later.

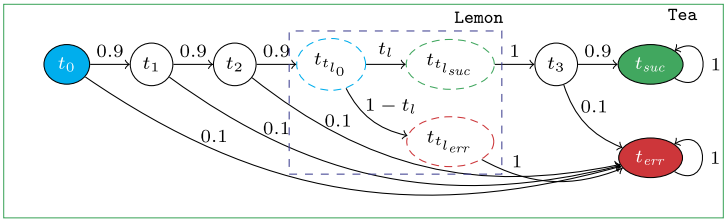
Fig. 6b shows the PMC $\mathcal{P}_t$ for the tea behavior, in which we use a slot to abstract the optional lemon-adding behavior, whose behavior is modeled by the PMC $\mathcal{P}_{t_l}$ in Fig. 6c. Since this tea-lemon PMC has no variability, it is in fact a regular DTMC. We omit the PMCs for serving soda ($\mathcal{P}_s$) and for adding lemon to soda ($\mathcal{P}_{s_l}$), for brevity, but the complete example can be seen in Fig. A.2 (Appendix A).

Formally, we define a compositional PMC as a PMC in which transition probabilities depend on the value of some probabilistic reachability property of other PMCs. For a PMC defined this way, possible evaluations map variables to real numbers within the interval $[0, 1]$, instead of the binary set $\{0, 1\}$ used for an annotative model (see Definition 6). To compose PMCs modeled this way with one another, we augment the definition of a PMC with explicit mentions of *success* and *error* states.

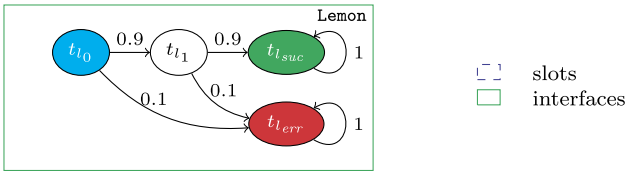
Definition 9 *(Compositional PMC).* A compositional PMC $\mathcal{P}$ is a tuple $(S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$, where:



(a) Top-level compositional PMC $\mathcal{P}_\top$ for the vending machine (common behavior and main variation points)



(b) Compositional PMC $\mathcal{P}_t$ for the behavior of serving tea



(c) Compositional PMC $\mathcal{P}_{t_l}$ for the behavior of adding lemon to tea

Fig. 6. Compositional PMCs for the vending machine.

- S is a set of states, $s_0 \in S$ is the initial state, X is a set of variables, and $\mathbf{P}$ is a transition probability matrix, such that $(S, s_0, X, \mathbf{P}, T)$ is an annotative PMC (see Definition 4).
- States $s_{suc}, s_{err} \in S$ are called *success* and *error* states, respectively. Together with the initial state, s_0, they define the *interface* of the compositional PMC: $\mathit{interface}(\mathcal{P}) = \{s_0, s_{suc}, s_{err}\}$ (solid box around PMCs in Fig. 6).
- $T = \{s_{suc}\}$. That is, s_{suc} is the only target state.
- The success and error states are the only *bottom strongly connected components* [5] in $\mathcal{P}$, that is:
  - once one of them is reached, no other state is ever reachable; and
  - they are the only states satisfying this property.

  This restriction ensures that we model all executions as either successful (if the success state is reached) or non-successful (if the error state is reached).

Definition 9 builds on Definition 4 to define the *structure* of compositional PMCs, but the intended semantics of variables in this type of parametric Markov chain is different from the corresponding semantics in an annotative PMC. In a compositional PMC, the condition that the outgoing transitions of a given node are either all constant or all variable (inherited from Definition 4) relates to the concept of *slots*, whereas annotative PMCs treat variable transitions as behavioral switches. Informally, a slot for the variable x (dashed boxes in Fig. 6) marks the part of a product's behavior where a configuration-specific behavior (identified by x) takes place. Note that there can be more than one slot for a given behavior.

Definition 10 *(Compositional PMC slot).* For a compositional PMC $\mathcal{P} = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$, a *slot* for $x \in X$ is a triple $(s_{x_0}, s_{x_{suc}}, s_{x_{err}})$, where:

- $s_{x_0}, s_{x_{suc}}, s_{x_{err}} \in S$;
- $\mathit{Succ}(s_{x_0}) = \{s_{x_{suc}}, s_{x_{err}}\}$;
- $\mathbf{P}(s_{x_0}, s_{x_{suc}}) = x \wedge \mathbf{P}(s_{x_0}, s_{x_{err}}) = 1 - x$.

The set of slots for x in $\mathcal{P}$ is denoted by $\mathit{slots}^{\mathcal{P}}(x)$, and the set of states belonging to any slot in $\mathit{slots}^{\mathcal{P}}(x)$ is given by $\mathit{slotStates}^{\mathcal{P}}(x) = \{s \in S \mid \exists_{\varsigma \in \mathit{slots}^{\mathcal{P}}(x)} \cdot s \in \varsigma\}$. We extend these definitions for the set of all slots in $\mathcal{P}$ for any variable in X ($\mathit{slots}^{\mathcal{P}}(X)$) and the set of states belonging to any slot in that set ($\mathit{slotStates}^{\mathcal{P}}(X)$).

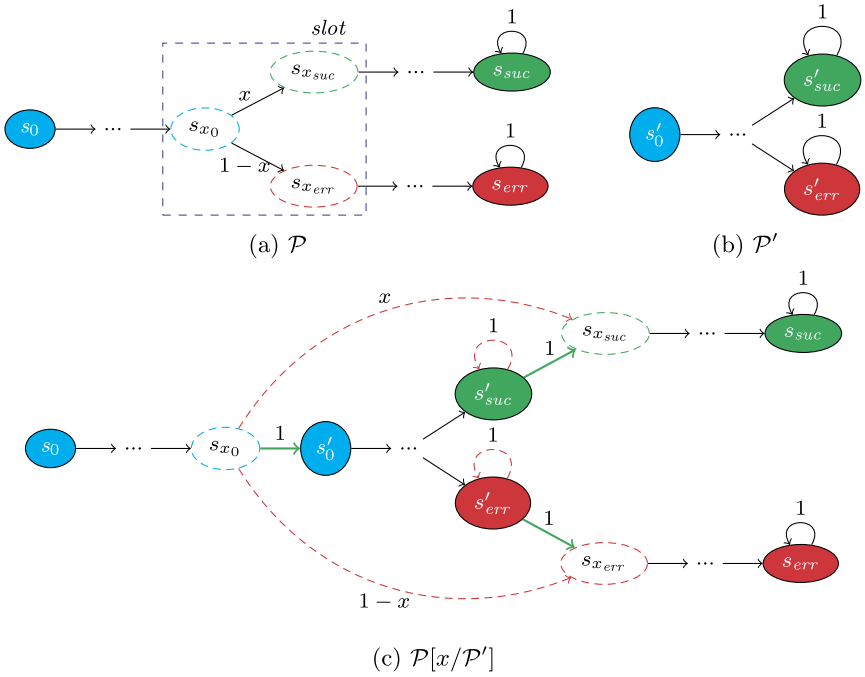


Fig. 7. Example of a partial composition of two PMCs. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

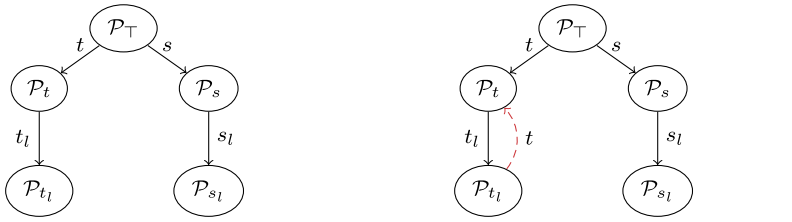
With compositional PMCs at hand, we need to be able to derive a DTMC, modeling the behavior of a given product of the product line, as in Section 3.1. Before we can handle the product-line aspect, we must define the mechanics of PMC composition. The intuition is that composition is achieved by connecting the interface (solid outer box) of a compositional PMC $\mathcal{P}'$ to the slots (dashed boxes) in a compositional PMC $\mathcal{P}$ that are meant to abstract the behavior in $\mathcal{P}'$, that is, $slots^{\mathcal{P}}(x)$ (see Fig. 7).

Definition 11 *(Partial PMC composition).* Given a compositional PMC $\mathcal{P} = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$ and a variable $x \in X$, assume that x occurs only once in $\mathcal{P}$, and let $\mathcal{P}' = (S', s'_0, s'_{suc}, s'_{err}, X', \mathbf{P}', T')$ be a compositional PMC to be composed on that single slot marked by x. The partial PMC composition $\mathcal{P}[x/P']$ is a compositional PMC $\mathcal{P}'' = (S'', s''_0, s''_{suc}, s''_{err}, X'', \mathbf{P}'', T'')$ such that:

- $S'' = S \uplus S'$, where $\uplus$ denotes the disjoint union operator (all states are disjointly merged);
- $s''_0 = s_0$, $s''_{suc} = s_{suc}$, and $s''_{err} = s_{err}$ (the interface of $\mathcal{P}$ is preserved);
- $X'' = X \setminus \{x\} \cup X'$ (the occurrence of x is replaced by a copy of $\mathcal{P}'$, whose variables are those of X');
- $T'' = T$ (target states of the base PMC are preserved);
- $\mathbf{P}''$ is such that
  - $\mathbf{P}''(s_{x_0}, s'_0) = 1$ (new transition from a slot's initial state to the initial state of the corresponding composed PMC)
  - $\mathbf{P}''(s'_{suc}, s_{x_{suc}}) = 1$ (new transition from the success state of a composed PMC to the success state of the corresponding slot)
  - $\mathbf{P}''(s'_{err}, s_{x_{err}}) = 1$ (new transition from the error state of a composed PMC to the error state of the corresponding slot)
  - $\mathbf{P}''(s_{x_0}, s_{x_{suc}}) = 0$ (slot's success transition is removed)
  - $\mathbf{P}''(s_{x_0}, s_{x_{err}}) = 0$ (slot's error transition is removed)
  - $\mathbf{P}''(s'_{suc}, s'_{suc}) = 0$ (success loops from composed PMCs are removed)
  - $\mathbf{P}''(s'_{err}, s'_{err}) = 0$ (error loops from composed PMCs are removed)
  - For all remaining combinations of $s_1, s_2 \in S''$:

$$\mathbf{P}''(s_1, s_2) = \begin{cases} \mathbf{P}(s_1, s_2) & \text{if } s_1, s_2 \in S \setminus slotStates^{\mathcal{P}}(x) \\ \mathbf{P}'(s_1, s_2) & \text{if } s_1, s_2 \in S' \\ 0 & \text{otherwise} \end{cases}$$

In summary, transitions among slot states of $\mathcal{P}$ are removed as well as the looping transitions from success and error absorbing states of $\mathcal{P}'$. Then, slot states are connected to respective interface states, yielding a partially composed PMC. This process is illustrated in Fig. 7c, which depicts the partial composition of the compositional PMC $\mathcal{P}'$ (Fig. 7b) into $\mathcal{P}$ (Fig. 7a) from the perspective of a single slot. New transitions are green bold, while red dashed transitions are the ones



(a) Dependency relation $\prec$ (b) Hypothetical cyclic dependency in $\prec$

Fig. 8. Dependency relation induced in the vending machine.

suppressed during composition. We say this transformation is partial because slots for variables other than x are not subject to composition.

Since there might be more than one slot for a given variable, we extend the concept of partial composition to mean the composition of n renamings of a given compositional PMC $\mathcal{P}'$ into each of the n slots for a single variable x in another compositional PMC $\mathcal{P}$. A full (*total*) composition is then obtained by composing PMCs over all slots in a given base compositional PMC at once. Such a composition relies on a *composition function*—a function $u' : X \to \mathscr{P}$ that yields a compositional PMC $\mathcal{P} \in \mathscr{P}$ to compose in the corresponding slots for any given variable. The detailed definitions of PMC renaming (Definition 32) and total PMC composition (Definition 33) are presented in Appendix B.3.

In a composition, slots mark locations where behavioral model fragments (i.e., other compositional PMCs) can be inserted to expand the base behavior. However, nothing so far prevents composition to happen at arbitrary slots (e.g., composing the behavior of adding lemon to soda in the slot for t, which was meant to represent the behavior of serving tea). Thus, we need a way to relate slots and the intended abstracted configuration-specific behaviors. We do so by naming compositional PMCs with the same variables that are used in the slots that mark their places, by means of an *identifying function*.

Definition 12 *(Identifying function).* Let $\mathscr{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ be a finite set of compositional PMCs $\mathcal{P}_i$, each with a set X_i of variables, where $i \in \{1, \ldots, n\}$. An identifying function is a bijection $idt : \mathscr{P} \to I$, where $I \supset \bigcup_{\mathcal{P}_i} X_i$ is a set of variables that contains all variables in the compositional PMCs $\mathcal{P}_i$.

Since idt is a bijection, the set I of identifiers must have the same cardinality as $\mathscr{P}$. In practical terms, we arbitrarily identify PMCs that do not directly correspond to an abstracted behavior (i.e., those that are not directly referred by variables in other PMCs). This is the case of top-level PMCs, which are mainly composed of states that are shared between the behaviors of all products. For the vending machine product line (Fig. A.2, summarized in Fig. 6), for which $\mathscr{P} = \{\mathcal{P}_\top, \mathcal{P}_t, \mathcal{P}_{t_l}, \mathcal{P}_s, \mathcal{P}_{s_l}\}$, we can define $I = \{\top, t, t_l, s, s_l\}$ and, correspondingly, $idt = \{\mathcal{P}_\top \mapsto \top, \mathcal{P}_t \mapsto t, \mathcal{P}_{t_l} \mapsto t_l, \mathcal{P}_s \mapsto s, \mathcal{P}_{s_l} \mapsto s_l\}$.

An identifying function induces a *dependency* relation over PMCs, based on their names and the variables they employ to abstract behavior in slots. If we denote this relation by $\prec$, in the vending machine example, we can say that $\mathcal{P}_{t_l} \prec \mathcal{P}_t \prec \mathcal{P}_\top$, meaning $\mathcal{P}_\top$ depends on $\mathcal{P}_t$, which, in turn, depends on $\mathcal{P}_{t_l}$. Also, $\mathcal{P}_{s_l} \prec \mathcal{P}_s \prec \mathcal{P}_\top$. Fig. 8a illustrates this dependency relation as a dependency graph, in which edges are labeled according to the variables identifying the respective dependencies. There should be no infinite descending chain under this relation, because otherwise one would infinitely compose PMCs and never get a DTMC as a result. This could happen as a modeling error, for instance, as introduced by the hypothetical dashed red cyclic dependency in Fig. 8b. Hence, we require the dependency relation among compositional PMCs to be *well-founded*, meaning there can be no infinite sequence $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \ldots$ such that $\forall_{i \geq 1} \cdot \mathcal{P}_{i+1} \prec \mathcal{P}_i$. This also prohibits cyclic dependencies, since they would allow infinite chains.

Definition 13 *(Dependency relation induced in compositional PMCs).* Given a finite set $\mathscr{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ of compositional PMCs $\mathcal{P}_i$, each with a set X_i of variables, and a corresponding identifying function $idt : \mathscr{P} \to I$, the binary relation $\prec : \mathscr{P} \times \mathscr{P}$ is the well-founded dependency relation induced by idt and by the use of variables in the $\mathcal{P}_i$. That is,

$$\forall_{\mathcal{P}_i, \mathcal{P}_j \in \mathscr{P}} \cdot idt(\mathcal{P}_j) \in X_i \Leftrightarrow \mathcal{P}_j \prec \mathcal{P}_i$$

We read $\mathcal{P}_j \prec \mathcal{P}_i$ as "$\mathcal{P}_i$ depends on $\mathcal{P}_j$".

A consequence of this definition is that, in a finite set of compositional PMCs with an identifying function, there must be, at least, one PMC that depends on no other and has no variability whatsoever (a *minimal* PMC), and, at least, one PMC on which no other depends (a *maximal* PMC). In the vending machine (Fig. A.2), the minimal PMCs are $\mathcal{P}_{t_l}$ and $\mathcal{P}_{s_l}$, while $\mathcal{P}_\top$ is the single maximal PMC.

Definition 14 *(Minimal and maximal compositional PMCs).* Given a set $\mathscr{P}$ of compositional PMCs, an identifying function idt, and the corresponding induced well-founded relation $\prec$, a compositional PMC $\mathcal{P} \in \mathscr{P}$ is called *minimal* iff

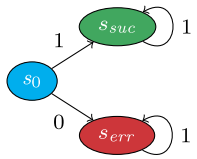


Fig. 9. Feature disabler compositional PMC $\mathcal{P}_\perp$.

$$\nexists_{\mathcal{P}' \in \mathscr{P}} \cdot \mathcal{P}' \prec \mathcal{P}$$

Conversely, $\mathcal{P} \in \mathscr{P}$ is called *maximal* iff

$$\nexists_{\mathcal{P}' \in \mathscr{P}} \cdot \mathcal{P} \prec \mathcal{P}'$$

Maximal PMCs can be seen as models of top-level behavior in a system, such as the main tasks usually represented by UML activity diagrams. In an automation software charged with managing different workflows, for instance, one could model each of the workflows as a separate behavior with internal variability, thus yielding as many maximal PMCs as there are tasks to accomplish. The number of maximal PMCs in a compositional model is mainly a modeling decision, and analyzing the whole product line amounts to analyzing each of these top-level behaviors. Thus, without loss of generality, we consider models that have only one maximal PMC,[6] which we denote by $\mathcal{P}_\top$.

After composition, the variability in a compositional PMC is replaced by the variabilities of the PMCs composed into it. That is to say, the set of variables of the resulting compositional PMC is given by $\bigcup_{i=1}^{k} X_i$, the set of variables in all composed PMCs. In the vending machine (Fig. 6), for instance, if we compose the tea PMC $\mathcal{P}_t$ (Fig. 6b) into the top-level PMC $\mathcal{P}_\top$ (Fig. 6a) using the slot $(c_{t_0}, c_{t_{suc}}, c_{t_{err}})$, the resulting compositional PMC $\mathcal{P}_\top[t/\mathcal{P}_t]$ will no longer have variable t, but will have a new variable t_l, stemming from $\mathcal{P}_t$. Consequently, to derive a product, one has to recursively perform the composition operation until a plain DTMC is returned.

This recursive approach to derive a product by composition relies on an identifying function *idt* to assign PMCs to slots corresponding to their identifiers. This composition depends upon satisfaction of a presence condition. Thus, before we can properly define this approach of *derivation by composition*, we must define how to proceed with composition in the case that the presence condition of a model to be composed is not satisfied. We achieve this result by composing the *feature disabler* compositional PMC, depicted in Fig. 9. This compositional PMC models an always successful behavior, so composing it would not affect the overall reliability of the base model.

Definition 15 *(Feature disabler compositional PMC).* The feature disabler compositional PMC, $\mathcal{P}_\perp = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$, is a compositional PMC such that:

- $S = \{s_0, s_{suc}, s_{err}\}$
- $X = \emptyset$
- $\mathbf{P}(s_0, s_{suc}) = 1$, $\mathbf{P}(s_{suc}, s_{suc}) = 1$, and $\mathbf{P}(s_{err}, s_{err}) = 1$. Otherwise, for $s, s' \in S$, $\mathbf{P}(s, s') = 0$
- $T = \{s_{suc}\}$

Similar to what we have achieved with evaluation factories (Definition 6), we need to constrain the possible compositions to ones that respect both: (a) satisfying presence conditions and (b) matching of slots and compositional PMCs via an identifying function. To enable this, we define a *composition factory* as a higher-order function that constrains compositions based on possible configurations of the modeled product line. This is the basis of product derivation.

Definition 16 *(Composition factory).* Given a set $\mathscr{P}$ of compositional PMCs, a set I of identifiers that is a superset of the variables used in slots, and a feature model $[\![FM]\!]$, a composition factory $w' : [\![FM]\!] \to I \to DTMC$ is a function that, for a given configuration $c \in [\![FM]\!]$, yields a *composition function* $w'(c) : I \to DTMC$.

To populate this definition with concrete composition factories, we fix the set I of identifiers as well as an identifying function, thus inducing a dependency relation that establishes which models should be composed to get a probabilistic model for a desired product. This way, a *compositional model* of a product line is a set of compositional PMCs closed under this dependency relation.

Definition 17 *(Compositional probabilistic model).* A compositional probabilistic model for a product line with feature model FM is a tuple $(\mathscr{P}, \prec, I, idt, p, w', FM)$, where:

[6] The existence of minimal and maximal PMCs follows from the well-foundedness of $\prec$. More details are available at Appendix B.1.

- $\mathscr{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ is a finite set of compositional PMCs $\mathcal{P}_i = (S_i, s_{i_0}, s_{i_{suc}}, s_{i_{err}}, X_i, \mathbf{P}_i, T_i)$ (Definition 9).
- I is a set of variables, such that $I \supset \bigcup_{\mathcal{P}_i} X_i$ and $|I| = |\mathscr{P}|$. These variables are a superset of all variables in the compositional PMCs in $\mathscr{P}$.
- $idt : \mathscr{P} \to I$ is an identifying function for $\mathscr{P}$ (Definition 12).
- $\prec : \mathscr{P} \times \mathscr{P}$ is the well-founded dependency relation induced by idt and by the use of variables in the compositional PMCs $\mathcal{P}_i$ (Definition 13).
- FM is a feature model.
- $p : I \to [\![FM]\!] \to \mathbb{B}$ is a presence function (Definition 5) denoting presence conditions satisfaction.
- w' is a composition factory (Definition 16) recursively defined as

$$w'(c)(x) = \begin{cases} \mathcal{P}_i[x_1/w'(c)(x_1), \ldots, x_k/w'(c)(x_k)] & \text{if } p(x)(c) = 1 \\ \mathcal{P}_\perp & \text{otherwise} \end{cases}$$

where $\mathcal{P}_i \in \mathscr{P}$, $idt(\mathcal{P}_i) = x \in I$, and $X_i = \{x_1, \ldots, x_k\}$.

This definition allows us to model the behavior of a product line in a compositional way. To leverage this model for product-line analysis, we define a way to derive a DTMC that is consistent with the behavior of a product generated using the same configuration.

Definition 18 *(Derivation by composition).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and a compositional PMC $\mathcal{P} \in \mathscr{P}$ with a set X of variables, the DTMC derivation by composition $\pi'(\mathcal{P}, w', c)$ is defined as

$$\pi'(\mathcal{P}, w', c) = \mathcal{P}[X/w'(c)]$$

The notation is overloaded from PMC evaluation, since both are model transformations that operate on variables. Since w' is defined recursively, we need to guarantee its execution terminates, which is why we require $\prec$ to be well-founded. The termination proof (Lemma 11) is presented in Appendix B.2.

4. Reliability analysis strategies

The scenario on which we focus is analyzing the reliability of all products of a product line using model checking of a probabilistic reachability property of Markov-chain models. For this task, one can choose a number of product-line analysis strategies [47]. Following the taxonomy of Thüm et al. [47], we discussed possible strategies for each of the variability representations (annotative and compositional) presented in Section 3.

Fig. 10 depicts these choices. Starting with a compositional (upper left corner) or an annotative model (upper right corner), one can follow any of the outgoing arrows while performing the respective analysis steps (abstracted as functions), until reliabilities are computed (either real-valued reliabilities or an ADD representing all possible values). These analysis steps can be feature-based (green solid arrows), product-based (blue dotted arrows), or family-based (red dashed arrows). Thus, the arrows form an "analysis path" (a function composition), which defines the employed analysis strategy. Furthermore, Fig. 10 is a commuting diagram (as we will demonstrate later in this section), meaning that different analysis paths are equivalent (i.e., they yield equal results) if they share the start and end points.

After choosing a variability representation, the analysis of any of the resulting models presents another choice: either variability-free models (i.e., DTMC) are derived for each configuration (function π) and then analyzed (function α), or variability-aware analysis is applied, using some form of parametric model checking (function $\hat{\alpha}$). The first choice yields a product-based strategy (Section 4.1), whereby each variant is independently analyzed. The second one leverages parametric model checking to produce expressions denoting the reliability of PMCs in terms of their variables (Section 2.2.2). These variables carry the semantics they had in the model-checked PMC, so we correspondingly classify the resulting expressions as annotative or compositional.

Evaluating these expressions provides another choice: to evaluate the expressions for each valid configuration (function σ), yielding feature-product-based (Section 4.3.1) and family-product-based (Section 4.2.1) strategies; or to interpret the expressions in terms of ADDs (function *lift*), effectively evaluating them for the whole family of models at once (function $\hat{\sigma}$)—a step we call *expression lifting*. The latter represents feature-family-based (Section 4.3.2) and family-based (Section 4.2.2) strategies.

As an example of walking through the choices of Fig. 10, suppose we start with a compositional model (upper-left corner), perform parametric model checking (move down), and then lift the resulting expressions (move down one more step) and evaluate them (move right), reaching a reliability ADD for the family as a whole. The arrows in this path are, respectively, green solid, red dashed, and red dashed, meaning the analysis strategy is feature-family-based.

In the remaining sections, we detail each of these strategies and analysis steps with the goal of making statements about their commuting relations. Section 4.1 presents product-based analysis strategies for both annotative and compositional models, with the goal of establishing a baseline for the remaining soundness proofs. Section 4.2 discusses family-product-based and family-based analyses of annotative models. Feature-product-based and feature-family-based analysis strategies

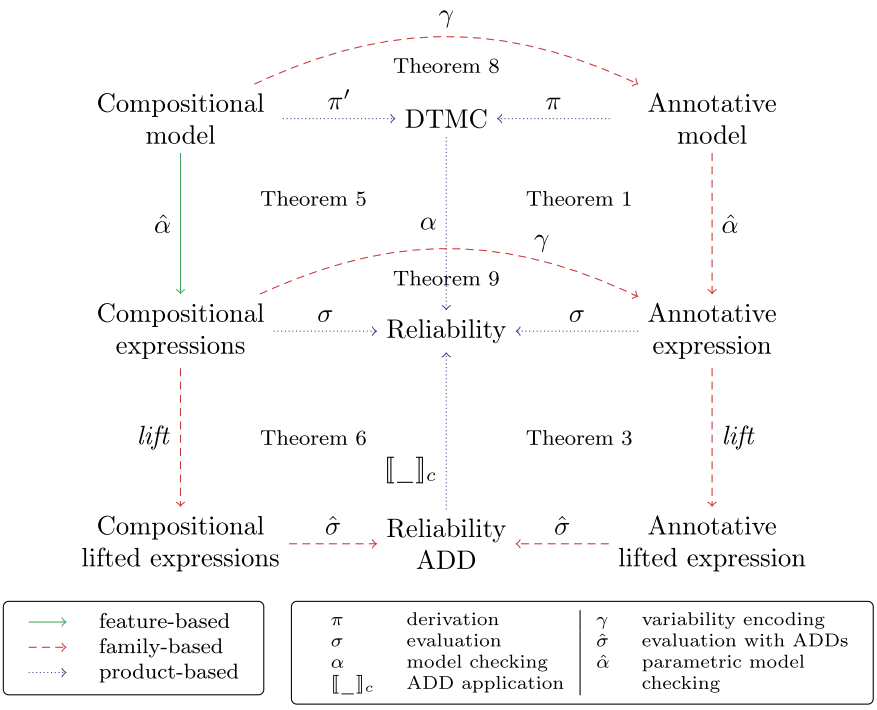


Fig. 10. Commutative diagram of product-line reliability analysis strategies. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

are the subject of Section 4.3, which focuses on compositional models. Then, Section 4.4 bridges the gap between analyses of annotative and compositional models (function γ in Fig. 10), establishing their commutativity. Finally, we leverage these results to present the novel feature-family-product-based strategy in Section 4.5.

4.1. Product-based strategies

Product-based analysis strategies are based on the analysis of generated products or models thereof [47]. In Section 3, we have discussed how to represent probabilistic behavioral models of product lines as PMCs, using both annotative and compositional approaches. There, we also described how to derive models of individual products, both for the annotative and the compositional approaches. The generated models are plain DTMCs, that is, their variability has been resolved at derivation time. Thus, to analyze the generated models, one only needs to model-check the non-parametric probabilistic reachability for every such model. We hereafter denote this non-parametric model checking analysis step by the following function α.

Definition 19 *(Non-parametric model checking).* The non-parametric model checking step $\alpha : DTMC \to [0, 1]$ consists of applying the algorithm by Hahn et al. [26]. For a DTMC $\mathcal{D} = (S, s_0, \mathbf{P}, T)$,

$$\alpha(\mathcal{D}) = Pr^{\mathcal{D}}(s_0, T)$$

Since a DTMC has no parameters, α yields constant functions, which we interpret as plain Real numbers.

Although there are more efficient algorithms for reliability model checking of regular (non-parametric) DTMCs, we use the algorithm by Hahn et al. [26] in the above definition for uniformity, which eases understanding. Since this algorithm is sound (Lemma 1), a working implementation of the presented theory is free to exploit another sound probabilistic reachability algorithm for performance reasons.

Now we are able to define product-based analysis for annotative and compositional models.

Strategy 1 *(Product-based analysis of annotative models).* Given an annotative model $(\mathcal{P}, p, w, FM)$, a product-based analysis yields, for all $c \in [\![FM]\!]$,

$$\alpha(\pi(\mathcal{P}, w, c))$$

or, alternatively,

$$\alpha([\![\mathcal{P}]\!]_c^w)$$

Strategy 2 *(Product-based analysis of compositional models).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$, a product-based analysis yields, for all $c \in [\![FM]\!]$,

$$\alpha(\pi'(\mathcal{P}_\top, w', c))$$

where $\mathcal{P}_\top$ is the maximal PMC in $\mathscr{P}$ under $\prec$.

So, a product-based analysis results in a mapping from configurations to respective reliability values, such as $\{c \mapsto \alpha(\pi(\mathcal{P}, w, c)) \mid c \in [\![FM]\!]\}$ for annotative models, for instance.

> Both analysis strategies presented in this section derive models for individual products of a given product line and then apply a single-product analysis technique as is. Since single-product analyses represent the base case upon which product-line analyses are built, the product-based strategies establish a baseline for proving the soundness of other strategies.

4.2. Family-based strategies

According to Thüm et al. [47], a family-based analysis strategy is one that (a) operates only on domain artifacts and that (b) incorporates the knowledge about valid feature combinations. In this section, we explore this kind of strategy in the context of annotative probabilistic models, because they encode the behavior of all products of a product line in a single PMC. It is also possible to perform family-based analyses on a compositional model by first transforming it into an annotative one, but this is discussed later in Section 4.4.

First, we show how to perform an analysis that yields a reliability expression, which can in turn be evaluated for each valid configuration of the product line. This characterizes a family-product-based strategy (Section 4.2.1). Then, the aforementioned analysis is leveraged to build a pure family-based (i.e., non-enumerative) strategy (Section 4.2.2). At first, it may seem counterintuitive to present the family-product-based approach before the family-based one. However, we shall see that our pure family-based approach builds upon concepts of the hybrid family-product-based approach, and that performing one or the other is a matter of choosing product-based or family-based analysis steps after a preliminary family-based step.

4.2.1. Family-product-based strategy

A family-product-based strategy is a family-based strategy followed by a product-based strategy over intermediate results [47]. The preliminary family-based step of our family-product-based analysis consists of applying parametric model checking of probabilistic reachability (Section 2.2.2) of the underlying PMC of the annotative model. This step is abstracted as a function $\hat{\alpha}$, where the $\hat{}$ symbol denotes that it is a variability-aware version of the non-parametric model checking function α (Definition 19).

Definition 20 *(Parametric model checking).* The parametric model checking analysis step $\hat{\alpha} : PMC_X \to \mathcal{F}_X$ consists of applying the algorithm by Hahn et al. [26] for probabilistic reachability, which yields a rational expression $\varepsilon \in \mathcal{F}_X$ for a PMC with variables set X. For a PMC $\mathcal{P} = (S, s_0, X, \mathbf{P}, T)$, the input target states of the algorithm are the ones in T.

After performing parametric model checking, the result of reachability analysis is an expression over the same variables as the annotative input PMC, denoting the PMC's reliability as a function of these variables. Hence, we expect this annotative reliability expression to be evaluated using the same evaluation functions that restricted the possible behaviors in the original model. This *expression evaluation*, which can be seen as model derivation applied to expressions, is captured in function σ.

Definition 21 *(Expression evaluation).* Given an expression ε over a set X of variables, an evaluation factory w, and a configuration $c \in [\![FM]\!]$, we define the expression evaluation function in a similar fashion as DTMC derivation:

$$\sigma(\varepsilon, w, c) = \varepsilon[X/w(c)]$$

Likewise, we can use $[\![\varepsilon]\!]_c^w$ to denote $\sigma(\varepsilon, w, c)$.

The function σ is applied to the reliability expression for all valid configurations of the product line, yielding the final product-based step. The resulting family-product-based approach for the analysis of annotative models is then defined as follows.

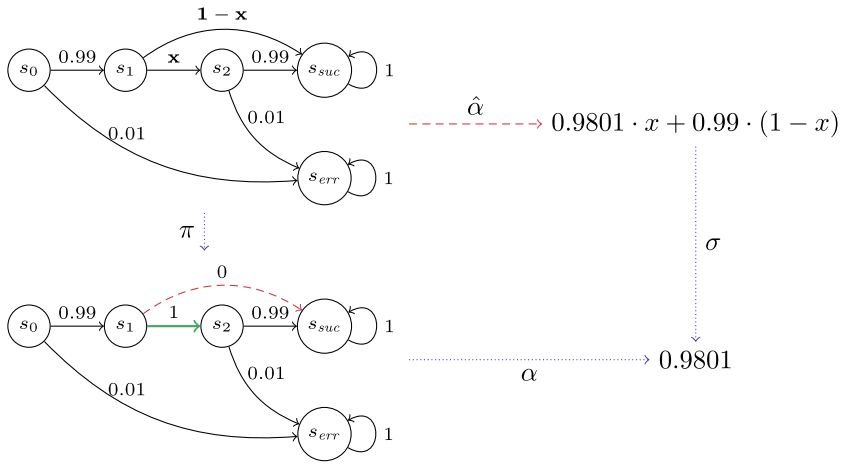


Fig. 11. Example of family-product-based analysis ($\hat{\alpha}$ followed by σ) in contrast to a product-based analysis (π followed by α) of an annotative PMC, for a configuration satisfying x's presence condition.

Strategy 3 *(Family-product-based analysis).* Given an annotative model $(\mathcal{P}, p, w, FM)$, the family-product-based analysis yields, for all $c \in [\![FM]\!]$,

$$\sigma(\hat{\alpha}(\mathcal{P}), w, c)$$

or, alternatively,

$$[\![\hat{\alpha}(\mathcal{P})]\!]_c^w$$

Fig. 11 illustrates the family-product-based strategy in contrast with the product-based one (Section 4.1), providing an intuition for why they commute. DTMC derivation π and expression evaluation σ are both performed for a configuration c such that $c \models p_x$. This way, $w(c)(x) = 1$ and the reliability is 0.9801. If x was absent (i.e., $c \not\models p_x$), then the reliability would be 0.99.

To be considered sound, a family-product-based analysis must be equivalent[7] to performing a product-based analysis of all products. This means that performing a parametric model checking step and then evaluating the resulting expression for each valid product must yield the same result as first deriving the original annotative model for each product and then performing non-parametric model checking on each resulting DTMC. To prove that this equivalence holds, we can leverage a more general result about PMCs and well-defined evaluations.

Lemma 3 *(Commutativity of PMC and expression evaluations). Given any PMC $\mathcal{P} = (S, s_0, X, \mathbf{P}, T)$ and a well-defined evaluation u, it holds that*

$$\alpha(\mathcal{P}[X/u]) = \hat{\alpha}(\mathcal{P})[X/u]$$

Proof.

$$\begin{aligned} \alpha(\mathcal{P}[X/u]) &= \alpha(\mathcal{P}_u) && \text{(syntax change)} \\ &= Pr^{\mathcal{P}_u}(s_0, T) && \text{(Definition 19)} \end{aligned}$$

and, since u is well-defined,

$$= \hat{\alpha}(\mathcal{P})[X/u] \qquad \text{(Lemma 1 and Definition 20)} \quad \square$$

Using this result, we are able to express the soundness of the family-product-based approach in the following theorem.

Theorem 1 *(Soundness of family-product-based analysis). Given an annotative model $(\mathcal{P}, p, w, FM)$, for all $c \in [\![FM]\!]$*

$$\alpha([\![\mathcal{P}]\!]_c^w) = [\![\hat{\alpha}(\mathcal{P})]\!]_c^w$$

Alternatively, $\alpha(\pi(\mathcal{P}, w, c)) = \sigma(\hat{\alpha}(\mathcal{P}), w, c)$.

[7] Whenever two analysis strategies yield equal reliability values, we say they are *r-equivalent*.

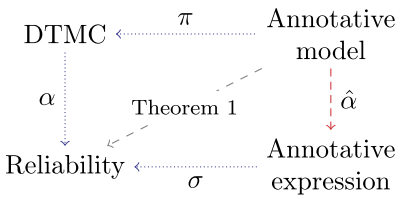


Fig. 12. Statement of Theorem 1.

Proof. Since $w(c)$ is a well-defined evaluation (Lemma 2), we can use it to instantiate u in Lemma 3. Thus, let $\mathcal{P} = (S, s_0, X, \mathbf{P}, T)$.

$$\begin{aligned}\alpha([\![\mathcal{P}]\!]_c^w) &= \alpha(\mathcal{P}[X/w(c)]) && \text{(Definition 8)}\\ &= \hat{\alpha}(\mathcal{P})[X/w(c)] && \text{(Lemmas 2 and 3)}\\ &= [\![\hat{\alpha}(\mathcal{P})]\!]_c^w && \text{(Definition 21)}\end{aligned}$$ □

As a major result, Theorem 1 states that the diagram in Fig. 12 commutes. This diagram corresponds to the upper right quadrant in Fig. 10.

4.2.2. Family-based strategy

The pure family-based strategy starts by applying parametric model checking to the given annotative model, as in the family-based step of the family-product-based strategy. However, instead of evaluating the resulting expression for each variant, we *lift* it to an ADD-based reliability expression, which can be evaluated for all variants at once. While an expression is evaluated with real values, a lifted expression is evaluated using ADDs, which represent Boolean functions from features to real values. Each of these ADDs encode the values that a variable can assume according to each possible configuration, also known as *variational data* [49]. Since this approach incorporates the knowledge of valid feature combinations, it is a family-based strategy.

Let us take the vending machine product line (Fig. A.1) as an example. Its reliability expression after parametric model checking has 8 terms, one of which is $0.124659 \cdot t \cdot t_l$. Starting from the evaluation factory w, we can derive functions ψ_x that, for each variable x, take a configuration $c \in [\![FM]\!]$ as input and output the corresponding value $w(c)(x)$. For t and t_l, for instance, these functions would be as follows:

$$\begin{aligned}\psi_t(\texttt{Tea}, \neg\texttt{Soda}, \neg\texttt{Lemon}) &= 1 & \psi_{t_l}(\texttt{Tea}, \neg\texttt{Soda}, \neg\texttt{Lemon}) &= 0\\ \psi_t(\texttt{Tea}, \neg\texttt{Soda}, \texttt{Lemon}) &= 1 & \psi_{t_l}(\texttt{Tea}, \neg\texttt{Soda}, \texttt{Lemon}) &= 1\\ \psi_t(\neg\texttt{Tea}, \texttt{Soda}, \neg\texttt{Lemon}) &= 0 & \psi_{t_l}(\neg\texttt{Tea}, \texttt{Soda}, \neg\texttt{Lemon}) &= 0\\ \psi_t(\neg\texttt{Tea}, \texttt{Soda}, \texttt{Lemon}) &= 0 & \psi_{t_l}(\neg\texttt{Tea}, \texttt{Soda}, \texttt{Lemon}) &= 0\end{aligned}$$

Having each of these functions represented by an ADD enables the efficient computation of the reliability expression as another ADD $\hat{r}$, representing a Boolean function that could be defined pointwise as $\hat{r}(c) = 0.124659 \cdot \psi_t(c) \cdot \psi_{t_l}(c)$ (we omit the remaining terms for simplicity).

We now formally define expression lifting, as well as the mechanics of generating ADD-based evaluations and evaluating lifted expressions.

Definition 22 *(Expression lifting).* For a given rational expression $\varepsilon \in \mathcal{F}_X$, whose semantics is a rational function $\mathbb{R}^{|X|} \to \mathbb{R}$, and a product line with k features, we define the lifted expression $lift(\varepsilon) = \hat{\varepsilon}$ as an expression which is syntactically equal to ε, but whose semantics is lifted to a rational function $(\mathbb{B}^k \to \mathbb{R})^{|X|} \to (\mathbb{B}^k \to \mathbb{R})$, such that:

- The function's inputs are k-ary ADDs.
- Polynomial coefficients are interpreted as constant ADDs (e.g., the number 5 becomes $c \in \mathbb{B}^k \mapsto 5$). We denote a constant a lifted to a constant ADD as $\hat{a}$, so that $\hat{a}(\bar{b}) = a$ (where $\bar{b}$ is a Boolean tuple).
- Arithmetic operators are lifted to their ADD-based counterparts.

Hence, the admitted evaluations for $\hat{\varepsilon}$ are of type $u : X \to (\mathbb{B}^k \to \mathbb{R})$, so that variables are properly replaced by k-ary ADDs.

By the above definition, lifted expressions are syntactically equal to their original (non-lifted) counterparts. However, instead of using Real arithmetics, we interpret operators, constants, and variables using ADDs and ADD arithmetics (Section 2.3). These semantically lifted expressions are sound in the sense that they denote functions that, when evaluated with a given configuration, yield the same results as if the variables of the original expressions would have been individually evaluated for the same configuration.

Lemma 4 *(Soundness of expression lifting). If ε is a rational expression over Real constants and variables $x_i \in X$, $|X| = n$, $A_1, \ldots, A_n$ are ADDs, and $\hat{\varepsilon} = lift(\varepsilon)$, then*

$$\hat{\varepsilon}[x_1/A_1, \ldots, x_n/A_n](\bar{b}) = \varepsilon[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})]$$

where $\bar{b}$ is a vector of k Booleans, corresponding to a selection of the k features in a given product line.

Proof. The proof is by induction on the structure of the rational expression ε. The base cases are constant expressions and single variables, for which the lemma holds. We then use induction and algebraic manipulation to prove for the arithmetic case (i.e., $\varepsilon = \varepsilon_1 \odot \varepsilon_2$, where $\odot \in \{+, -, \times, \div\}$) and for exponentiation. Proof details can be found in Appendix B.4. □

Note how a lifted expression demands a different type of evaluation, namely one that replaces variables with ADDs. To handle this interdependency, we correspondingly lift the evaluation factory.

Definition 23 *(Lifted evaluation factory).* Given an evaluation factory w defined over a feature model FM and a set X of variables, the factory's lifted counterpart is a function $\hat{w} : X \to (\mathbb{B}^{|FM|} \to \mathbb{R})$ that yields an ADD for a given variable. This function is such that, for every variable $x \in X$ and all $c \in [\![FM]\!]$,

$$\hat{w}(x)(c) = w(c)(x)$$

With a lifted evaluation factory, one can evaluate a lifted expression over the same set X in a variability-aware fashion. The intuition is that we valuate each variable with an ADD that encodes all the real values it may assume for any configuration of the product line.

Definition 24 *(Variability-aware expression evaluation).* Let $\hat{w}$ be a lifted evaluation factory and $\hat{\varepsilon}$ be a lifted expression. The variability-aware expression evaluation function, $\hat{\sigma}$, is defined as

$$\hat{\sigma}(\hat{\varepsilon}, \hat{w}) = \hat{\varepsilon}[X/\hat{w}]$$

Remark 2. This definition of variability-aware evaluation is not restricted to reliability analysis or to the specific definitions of probabilistic models presented in this text. Indeed, one can notice that it relies on the definitions of an expression with rational function semantics and of an evaluation factory with respect to a given feature model.

Thus, we are able to prove the following theorem, which applies to product line analysis strategies that are based on expression evaluation.

Theorem 2 *(Soundness of variability-aware expression evaluation). If ε is an expression and w is an evaluation factory with respect to a feature model FM, let $\hat{\varepsilon}$ and $\hat{w}$ be their respective lifted counterparts. Then, for all $c \in [\![FM]\!]$,*

$$\hat{\sigma}(\hat{\varepsilon}, \hat{w})(c) = \sigma(\varepsilon, w, c)$$

In other words, $\hat{\varepsilon}[X/\hat{w}](c) = \varepsilon[X/w(c)]$.

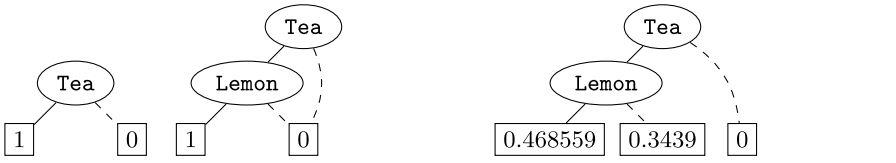
Proof. Using $\hat{w}$ as a substitution,

$$\hat{\varepsilon}[X/\hat{w}] = \hat{\varepsilon}[x_1/\hat{w}(x_1), \ldots, x_n/\hat{w}(x_n)]$$

Thus, for all $c \in [\![FM]\!]$,

$$\begin{aligned}
\hat{\sigma}(\hat{\varepsilon}, \hat{w})(c) &= \hat{\varepsilon}[X/\hat{w}](c) && \text{(Definition 24)} \\
&= \hat{\varepsilon}[x_1/\hat{w}(x_1), \ldots, x_n/\hat{w}(x_n)](c) \\
&= \varepsilon[x_1/\hat{w}(x_1)(c), \ldots, x_n/\hat{w}(x_n)(c)] && \text{(Lemma 4)} \\
&= \varepsilon[x_1/w(c)(x_1), \ldots, x_n/w(c)(x_n)] && \text{(Definition 23)} \\
&= \varepsilon[X/w(c)] \\
&= \sigma(\varepsilon, w, c) && \text{(Definition 21)}
\end{aligned}$$

□



(a) $\hat{p}(t)$ (b) $\hat{p}(t_l)$ (c) $lift(0.124659 \cdot t \cdot t_l + 0.3439 \cdot t)[t/\hat{p}(t), t_l/\hat{p}(t_l)]$

Fig. 13. Example of lifted expression evaluation using $\hat{p}$.

We have seen that, in a product line with feature model *FM*, the presence function p denotes a presence condition p_x as a Boolean function $p(x) : [\![FM]\!] \to \mathbb{B}$. Since this can be alternatively expressed as $p(x) : \mathbb{B}^{|FM|} \to \mathbb{B}$, the presence function can also be encoded by ADDs, denoted by $\hat{p}(x)$. We now resort to the pointwise definition of w as $w(c)(x) = p(x)(c)$ (Remark 1), to define a lifted evaluation factory $\hat{w}$, for evaluating the lifted version of expressions resulting from parametric model checking of an annotative model.

Lemma 5 *(Soundness of lifted annotative evaluation factory). Given an annotative model $(\mathcal{P}, p, w, FM)$ and a function $\hat{p} : X \to (\mathbb{B}^{|FM|} \to \mathbb{B})$ that encodes presence conditions for variables as ADDs, then $\hat{w} = \hat{p}$ is a lifted evaluation factory for w.*

Proof. From Definition 7, we have that

$$w(c)(x) = \begin{cases} 1 & \text{if } p(x)(c) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Thus, from Remark 1, $w(c)(x) = p(x)(c)$. Also, $p(x)(c) = \hat{p}(x)(c)$ by definition, so $w(c)(x) = \hat{p}(x)(c)$. □

Recalling the vending machine example, the presence conditions for the variables t and t_l are, respectively, `Tea` and `Tea` $\wedge$ `Lemon`. Then, the ADDs $\hat{p}(t)$ and $\hat{p}(t_l)$ are given by the Figs. 13a and 13b, where we use the notation presented in Section 2.3. If we evaluate a lifted version of the example expression $\varepsilon = 0.124659 \cdot t \cdot t_l + 0.3439 \cdot t$ (2 terms from the actual reliability expression for the vending machine annotative model in Fig. A.1) with $\hat{p}$, the resulting ADD will be $\hat{r} = 0.124659 \cdot \hat{p}(t) \cdot \hat{p}(t_l) + 0.3439 \cdot \hat{p}(t)$, as depicted in Fig. 13c. Hence, for a given configuration $c \in [\![FM]\!]$, if both `Tea` and `Lemon` are present (i.e., $\hat{p}(t)(c) = 1$ and $\hat{p}(t_l)(c) = 1$), then $\hat{r}(c) = 0.124659 \cdot 1 \cdot 1 + 0.3439 \cdot 1 = 0.468559$; if only `Tea` is present, then $\hat{r}(c) = 0.124659 \cdot 1 \cdot 0 + 0.3439 \cdot 1 = 0.3439$; and if both `Tea` and `Lemon` are absent, then $\hat{r}(c) = 0$.

Using the result from Lemma 5, we can now express the soundness of this family-based analysis step of evaluating lifted expressions.

Theorem 3 *(Soundness of expression evaluation using $\hat{p}$). Given an annotative model $(\mathcal{P}, p, w, FM)$, $\varepsilon = \hat{\alpha}(\mathcal{P})$, and $\hat{\varepsilon} = lift(\varepsilon)$, let $\hat{p}$ be the encoding of the presence condition function p to yield ADDs. If we use $\hat{p}$ as a lifted evaluation factory, then for all $c \in [\![FM]\!]$*

$$[\![\hat{\sigma}(\hat{\varepsilon}, \hat{p})]\!]_c = [\![\varepsilon]\!]_c^w$$

Alternatively, $\hat{\sigma}(lift(\varepsilon), \hat{p})(c) = \sigma(\varepsilon, w, c)$.

Proof. For a given annotative model, Lemma 5 states that $\hat{p}$ is a sound lifted counterpart of w. Hence, by Theorem 2, $\varepsilon[X/w(c)] = \hat{\varepsilon}[X/\hat{p}](c)$. In other words, $[\![\hat{\sigma}(\hat{\varepsilon}, \hat{p})]\!]_c = [\![\varepsilon]\!]_c^w$. □

Fig. 14 illustrates the main result from Theorem 3. The depicted diagram, which corresponds to the lower right quadrant in Fig. 10, is commutative because of this theorem.

Now that we have all analysis steps needed, we can formally define the family-based strategy.

Strategy 4 *(Family-based analysis).* Given an annotative model $(\mathcal{P}, p, w, FM)$, a family-based analysis yields

$$\hat{\sigma}\left(lift(\hat{\alpha}(\mathcal{P})), \hat{p}\right)$$

The result of a family-based analysis is a Boolean function encoded as an ADD. Such an analysis is sound if, and only if, it yields an ADD for which every valid configuration $c \in [\![FM]\!]$ results in the same probability as if the original annotative model had been subject to product-based analysis for the same configuration c.

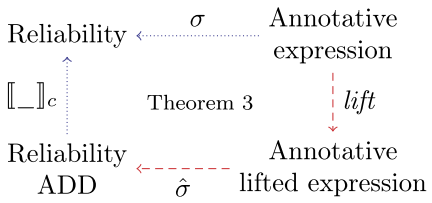


Fig. 14. Statement of Theorem 3.

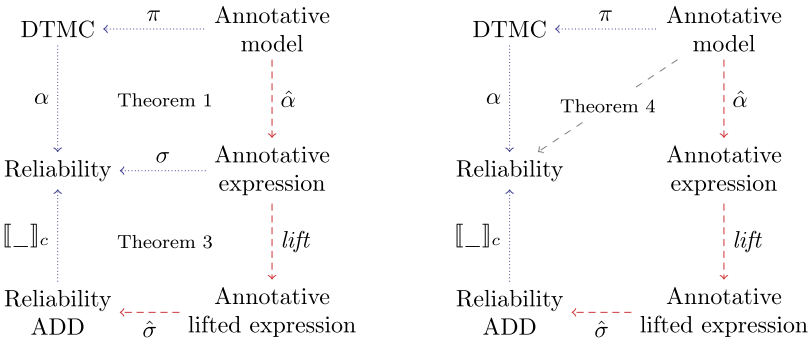


Fig. 15. Alternative views of the statement of Theorem 4.

Theorem 4 *(Soundness of family-based analysis). Given an annotative model* $(\mathcal{P}, p, w, FM)$*, for all* $c \in [\![FM]\!]$ *it holds that*

$$[\![\hat{\sigma}\left(lift(\hat{\alpha}(\mathcal{P})), \hat{p}\right)]\!]_c = \alpha([\![\mathcal{P}]\!]_c^w)$$

Proof. Follows from the successive application of Theorems 3 and 1:

$$\begin{aligned} [\![\hat{\sigma}\left(lift(\hat{\alpha}(\mathcal{P})), \hat{p}\right)]\!]_c &= [\![\hat{\alpha}(\mathcal{P})]\!]_c^w && \text{(Theorem 3)} \\ &= \alpha([\![\mathcal{P}]\!]_c^w) && \text{(Theorem 1)} \quad \square \end{aligned}$$

As a key result, Theorem 4 states that the diagrams in Fig. 15 commute. Both diagrams correspond to the right half of the one in Fig. 10.

4.3. Feature-based strategies

A feature-based analysis strategy is one that (a) operates only on domain artifacts and that (b) analyzes the artifacts belonging to each feature in isolation [47]. Compositional models describe modular behaviors that represent units of variability. A given PMC within a compositional model may represent the behavior associated with one or more features, or even model part of a given feature's behavior (in case of behavior scattering). In this sense, analyzing individual PMCs of a compositional model can be seen as analyzing features in isolation, which is why we use this kind of probabilistic model to discuss feature-based strategies. Moreover, since our focus is on reliability, which is highly influenced by feature interactions, we cannot use a pure feature-based strategy [47]. Thus, we concentrate on feature-product-based and feature-family-based analysis strategies.

Similar to what happens with family-based strategies (Section 4.2), the feature-family-based approach builds upon concepts used by the feature-product-based strategy, and performing one or the other is a matter of choosing product-based or family-based analysis steps after a preliminary feature-based step. Because of that, we first discuss the feature-product-based strategy (Section 4.3.1), focusing on the feature-based step of applying parametric model checking to each compositional PMC to generate corresponding compositional expressions. These reliability expressions can be evaluated for every possible configuration, yielding a product-based step and giving rise to a feature-product-based strategy. Alternatively, we can lift each expression and evaluate them using ADDs, in a similar fashion to what we did for the family-based strategy (Section 4.2.2). This leads to an overall feature-family-based strategy, which we discuss in Section 4.3.2.

4.3.1. Feature-product-based strategy

A product-line analysis strategy is feature-product-based (a) if it consists of a feature-based analysis followed by a product-based analysis and (b) if the analysis results of the feature-based analysis are used in the product-based analysis [47]. The preliminary feature-based analysis step consists of applying the parametric model checking function $\hat{\alpha}$ to each PMC in a compositional model, yielding corresponding reliability expressions. These resulting expressions preserve the dependency relation, since each of them is defined in terms of the same variables as its originating PMC and can be assigned the same identifier.

As an example, the compositional model of the vending machine product line (Fig. 6) yields the following expressions after the feature-based analysis step: $\hat{\alpha}(\mathcal{P}_\top) = 1 \cdot t \cdot s$, $\hat{\alpha}(\mathcal{P}_t) = 0.6561 \cdot t_l$, and $\hat{\alpha}(\mathcal{P}_{t_l}) = 0.81$. Also, $\hat{\alpha}(\mathcal{P}_s) = 0.729 \cdot s_l$ and $\hat{\alpha}(\mathcal{P}_{s_l}) = 0.81$ for the remaining PMCs in Fig. A.2.

A bottom-up evaluation of variables can be applied for each valid configuration, giving rise to the product-based analysis step. This procedure consists of *compositional expression evaluation*, that is, expression evaluation using a *compositional evaluation factory* derived from the composition factory used for the corresponding PMCs.

Definition 25 *(Compositional evaluation factory).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$, a compositional evaluation factory is defined as an evaluation factory (Definition 6) $w : [\![FM]\!] \to I \to \mathbb{R}$, such that for all $c \in [\![FM]\!]$ and $x \in I$,

$$w(c)(x) = \begin{cases} \sigma(\hat{\alpha}(\mathcal{P}), w, c) & \text{if } p(x)(c) = 1 \\ 1 & \text{otherwise} \end{cases}$$

where $idt(\mathcal{P}) = x$. Alternatively, we can write

$$w(c)(x) = \begin{cases} [\![\hat{\alpha}(\mathcal{P})]\!]_c^w & \text{if } p(x)(c) = 1 \\ 1 & \text{otherwise} \end{cases}$$

In other words, whereas a composition factory composes a recursively derived version of PMC $\mathcal{P}'$ into slots identified by a variable x of a PMC $\mathcal{P}$, a compositional evaluation factory composes a recursively evaluated version of $\hat{\alpha}(\mathcal{P}')$ in every occurrence of the variable x in $\hat{\alpha}(\mathcal{P})$. This recursion always terminates, because $\prec$ is a well-founded relation (see Lemma 12, in Appendix B.2).

We define the feature-product-based analysis of compositional models as a recursive evaluation of the expressions obtained from the feature-based step, using the compositional evaluation factory shown above. This recursion starts from the maximal PMC in the compositional model, traversing the dependency graph induced by $\prec$ (Fig. 8a) in a depth-first fashion.

For the vending machine product line (Fig. 6), for instance, the computation for configuration $c = \{\texttt{Tea}, \texttt{Lemon}\}$ would be as follows: Starting with $\hat{\alpha}(\mathcal{P}_\top)$, we evaluate the presence conditions for its variables, t and s. Since $p_s = \texttt{Soda}$ is not satisfied, s is evaluated to 1, ending the computation for this branch. On the other hand, $p_t = \texttt{Tea}$ is satisfied, so we step into this branch to compute $\hat{\alpha}(\mathcal{P}_t)$ under c. The only variable in this expression, t_l, has its presence condition satisfied by c, so we step further into this branch to compute $\hat{\alpha}(\mathcal{P}_{t_l})$ under c. Since this expression denotes a constant value, we return this value and the recursion terminates, yielding the following constant expression:

$$[\![\hat{\alpha}(\mathcal{P}_\top)]\!]_c = 1 \cdot \underbrace{\Big(0.6561 \cdot \overbrace{(0.81)}^{[\![\hat{\alpha}(\mathcal{P}_{t_l})]\!]_c}\Big)}_{[\![\hat{\alpha}(\mathcal{P}_t)]\!]_c} \cdot \underbrace{(1)}_{[\![\hat{\alpha}(\mathcal{P}_s)]\!]_c}$$

We generalize and formalize this procedure as follows.

Strategy 5 *(Feature-product-based analysis).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and the compositional evaluation factory w, derived from the composition factory w', the feature-product-based analysis yields, for all $c \in [\![FM]\!]$,

$$\sigma(\hat{\alpha}(\mathcal{P}_\top), w, c)$$

or, alternatively,

$$[\![\hat{\alpha}(\mathcal{P}_\top)]\!]_c^w$$

where $\mathcal{P}_\top$ is the maximal PMC in $\mathscr{P}$ under the dependency relation $\prec$.

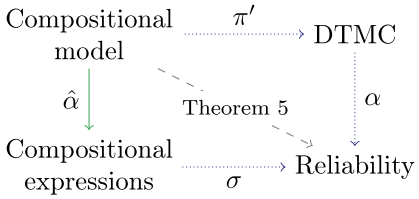


Fig. 16. Statement of Theorem 5.

To establish the soundness of the feature-product-based strategy, we need to compare it to the product-based strategy for compositional models. We state this result in the following theorem.

Theorem 5 *(Soundness of feature-product-based analysis). Given a compositional model* $(\mathscr{P}, \prec, I, idt, p, w', FM)$*, for all configurations* $c \in [\![FM]\!]$*, it holds that*

$$\sigma(\hat{\alpha}(\mathcal{P}), w, c) = \alpha(\pi'(\mathcal{P}, w', c))$$

or, alternatively,

$$[\![\hat{\alpha}(\mathcal{P})]\!]_c^w = \alpha([\![\mathcal{P}]\!]_c^{w'})$$

where $\mathcal{P} \in \mathscr{P}$ *and* w *is the compositional evaluation factory (Definition 25) derived from the composition factory* w'*.*

Proof. We use well-founded induction. The base of the induction is when $\mathcal{P}$ is minimal with respect to $\prec$. Since minimal PMCs have empty sets of variables, $\pi'(\mathcal{P}, w', c) = \mathcal{P}$ and $\hat{\alpha}(\mathcal{P}) = \alpha(\mathcal{P})$. Thus, the statement holds for the base case.

The general case is proved by expanding definitions in the proof goal and applying the induction hypothesis and Lemma 3. The complete proof is presented in Appendix B.3. □

As a further major result, Theorem 5 states that the diagram in Fig. 16 commutes. This diagram relates to the upper left quadrant in Fig. 10.

4.3.2. Feature-family-based strategy

Similar to the family-based strategy (Section 4.2.2), the feature-family-based strategy leverages ADDs to store and reason about variational data. Since the preceding feature-based analysis yields expressions over reliabilities, this variational data is made of Real values corresponding to the reliabilities of the products of a product line. Again, lifting expressions involves lifting the corresponding evaluation factory. In this process, the presence conditions are encoded in ADDs to represent the variability under feature selection. This encoding is achieved by the ADD operator `ITE` (*if-then-else*).

Let us revisit expression evaluation in the vending machine example (Fig. 6). We have seen the expression for t_l is the constant 0.81, so its lifted version is the constant ADD $\widehat{0.81}$ (according to the notation introduced in Definition 22). The expression for t, $\hat{\alpha}(\mathcal{P}_t) = 0.6561 \cdot t_l$, has the variable t_l. Thus, if the presence condition $p_{t_l} = \texttt{Tea} \wedge \texttt{Lemon}$ is satisfied, this variable must be evaluated to the constant value 0.81, assuming the value 1 otherwise. Thus, the lifted expression $\widehat{\hat{\alpha}(\mathcal{P}_t)}$ is evaluated with an ADD encoding this choice, given by $\varphi(t_l) = \texttt{ITE}(\widehat{\hat{p}(t_l)}, \widehat{0.81}, \widehat{1})$ and depicted in Fig. 17a. The evaluated lifted expression $\widehat{\hat{\alpha}(\mathcal{P}_t)}[t_l/\varphi(t_l)]$ is the ADD product of the constant $\widehat{0.6561}$ and $\varphi(t_l)$, shown in Fig. 17b. The procedure is repeated for every composition, so that the variable t in the expression $\widehat{\hat{\alpha}(\mathcal{P}_\top)}$ would be replaced by the ADD in Fig. 17c, which already encodes the combined presence conditions for t and t_l.

The function φ shown in the example is the lifted version of the compositional evaluation factory w. We first present a formal definition of φ and then proceed to proving its soundness. Soundness of the feature-family-based strategy follows from this result and from the soundness of the feature-product-based strategy (Section 4.3.1).

Definition 26 *(Lifted compositional evaluation factory).* Given a compositional probabilistic model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and the compositional evaluation factory w, derived from the composition factory w', the lifted evaluation factory $\varphi : I \to (\mathbb{B}^{|FM|} \to \mathbb{R})$ is a function that, for any $x \in I$, yields an ADD $\varphi(x)$ such that:

$$\varphi(x) = \texttt{ITE}(\hat{p}(x), \widehat{\hat{\alpha}(\mathcal{P})}[X/\varphi], \hat{\mathbf{1}})$$

where $\mathcal{P} \in \mathscr{P}$, $idt(\mathcal{P}) = x$, $\widehat{\hat{\alpha}(\mathcal{P})} = lift(\hat{\alpha}(\mathcal{P}))$ and $\hat{\mathbf{1}}$ is the constant ADD corresponding to the function $(c \in [\![FM]\!]) \mapsto 1$.

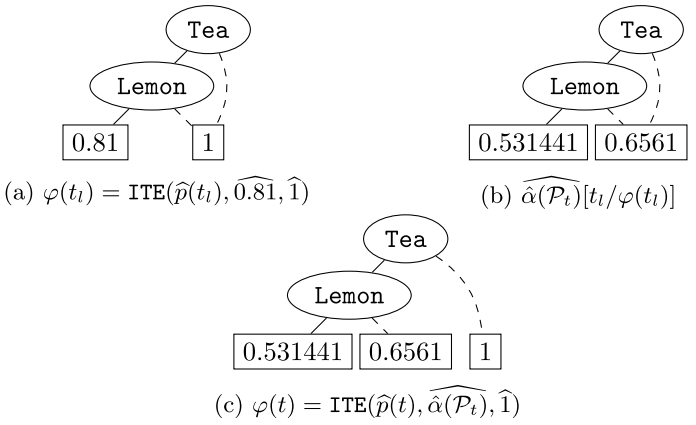


Fig. 17. Example of lifted compositional expression evaluation.

The next lemma, which is the compositional counterpart of Lemma 5, states this function φ is indeed a lifted version of w.

Lemma 6 *(Soundness of lifted compositional evaluation factory). Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and the compositional evaluation factory w, derived from the composition factory w' (Definition 25), for all $x \in I$ and all $c \in [\![FM]\!]$ it holds that*

$$\varphi(x)(c) = w(c)(x)$$

Proof. We first expand the definitions of φ (Definition 26) and w (Definition 25), then proceed to compare corresponding cases. The cases in which the presence condition is not satisfied are trivially equal; for the complementary case, we use well-founded induction on the dependency relation $\prec$, along with the soundness result for expression lifting (Lemma 4). The reader is invited to follow the complete proof in Appendix B.4. □

This way, the ADDs yielded by function φ from Definition 26 correctly encode the variation in values returned by the compositional evaluation factory w. An immediate consequence is that the expressions resulting from the feature-based analysis step can, indeed, be lifted and then evaluated using φ, and this gives us the same results as the corresponding (i.e., for the same configurations) product-based evaluations. This is expressed by the following theorem.

Theorem 6 *(Soundness of expression evaluation using φ). Given a compositional probabilistic model $(\mathscr{P}, \prec, I, idt, p, w', FM)$, the compositional evaluation factory w, derived from the composition factory w', and $x \in I$, let $\mathcal{P} = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$ be such that $idt(\mathcal{P}) = x$, $\mathcal{P} \in \mathscr{P}$. If $\varepsilon = \hat{\alpha}(\mathcal{P})$, $\hat{\varepsilon} = lift(\varepsilon)$, and φ is the lifted compositional evaluation factory obtained from w (Definition 26), then, for all $c \in [\![FM]\!]$, it holds that*

$$\hat{\varepsilon}[X/\varphi](c) = \varepsilon[X/w(c)]$$

Proof. For the given compositional probabilistic model, Lemma 6 states φ is a sound lifted counterpart of w. Hence, by Theorem 2, $\varepsilon[X/w(c)] = \hat{\varepsilon}[X/\varphi](c)$. In other words, $[\![\hat{\sigma}(\hat{\varepsilon}, \varphi)]\!]_c = [\![\varepsilon]\!]_c^w$. □

So, Theorem 6 states that the diagram in Fig. 18 commutes. This diagram corresponds to the lower left quadrant in Fig. 10.

The feature-family-based analysis strategy leverages the preceding results to yield an ADD encoding all reliabilities for valid configurations of the product line. This process is formally defined as follows.

Strategy 6 *(Feature-family-based analysis).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and the lifted compositional evaluation factory φ, derived from w', the feature-family-based strategy yields

$$\hat{\sigma}\left(lift(\hat{\alpha}(\mathcal{P}_\top)), \varphi\right)$$

where $\mathcal{P}_\top$ is the maximal PMC in $\mathscr{P}$ under the dependency relation $\prec$.

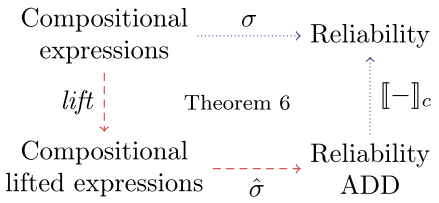


Fig. 18. Statement of Theorem 6.

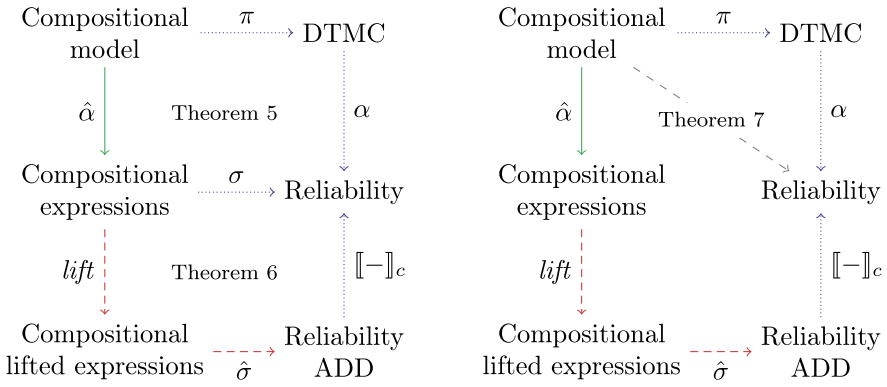


Fig. 19. Alternative views of the statement of Theorem 7.

Similar to the family-based strategy, the feature-family-based strategy is sound if this ADD is such that applying it to every valid configuration $c \in [\![FM]\!]$ results in the same probability as if the original compositional model had been derived for c and the resulting DTMC had been model-checked for probabilistic reachability (product-based strategy). The difference is that, in the feature-family-based case, this statement holds for every PMC in the compositional model.

Theorem 7 *(Soundness of feature-family-based analysis). Given a compositional model* $(\mathscr{P}, \prec, I, idt, p, w', FM)$ *and the lifted compositional evaluation factory* φ*, derived from* w'*, for every PMC* $\mathcal{P} \in \mathscr{P}$ *and for all configurations* $c \in [\![FM]\!]$ *it holds that*

$$[\![\hat{\sigma}\left(lift(\hat{\alpha}(\mathcal{P}_\top)), \varphi\right)]\!]_c = \alpha([\![\mathcal{P}]\!]_c^{w'})$$

Proof. Let w be the compositional evaluation factory derived from the composition factory w'. The proof follows from successive application of Theorems 6, 5:

$$\begin{aligned} [\![\hat{\sigma}\left(lift(\hat{\alpha}(\mathcal{P})), \varphi\right)]\!]_c &= [\![\hat{\alpha}(\mathcal{P})]\!]_c^{w} && \text{(Theorem 6)} \\ &= \alpha([\![\mathcal{P}]\!]_c^{w'}) && \text{(Theorem 5)} \quad \square \end{aligned}$$

As a key result, Theorem 7 states that the diagrams in Fig. 19 commute. Both diagrams correspond to the left half of the one in Fig. 10.

4.4. Bridging compositional and annotative models

Thus far, we have discussed family-based analysis strategies applied to annotative models and feature-based analysis strategies applied to compositional models. We now present a technique to transform any composition-based model into an r-equivalent annotation-based model. This ability may be useful in the case that the reliability analysis of a given product line is predictably more efficient if performed using a strategy suited for annotative models, such as our family-product-based and family-based approaches. This transformation of models resembles *variability encoding* techniques, that is, the rewriting of compile-time variability as load-time or run-time variability [1,43,45,2].

Although the concepts of compilation and execution are not defined for Markov chains, variability encoding, as established in the literature, has the main goal of creating artifacts that can be analyzed by off-the-shelf tools. Correspondingly, we are able to transform a compositional model, which cannot be directly model-checked (because it is split into a number

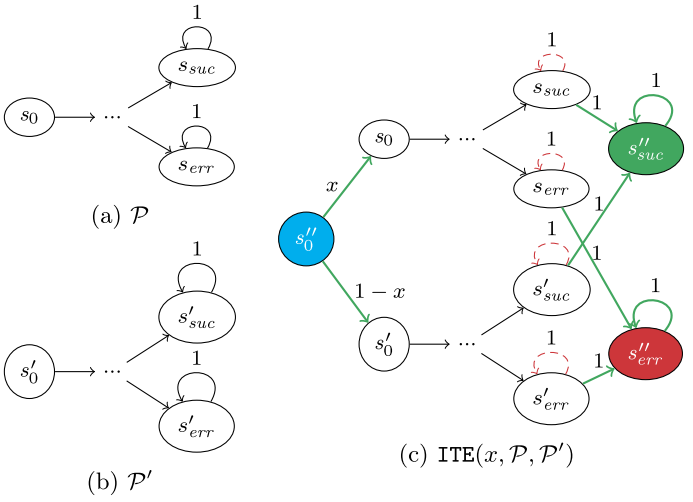


Fig. 20. Example ITE operator for PMCs. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

of PMCs), into an annotative model, which can be immediately issued to a parametric model checker. Thus, we address the transformation of compositional models into annotative ones in terms of two *variability encoding functions*: one operating on PMCs (Section 4.4.1) and the other for handling expressions (Section 4.4.2).

4.4.1. Variability encoding of PMCs

In terms of Markov chains, variability encoding can be realized by turning compositional models into annotative ones. This means transforming both the underlying compositional PMCs and the composition factory w' into a single annotative PMC with a corresponding evaluation factory. To accomplish this, we propose an *if-then-else* operator for PMCs that switches between possible states with a Boolean variable.

For brevity, the formal definition of this operator (Definition 34) is available in Appendix B.5.1. We rely on Fig. 20 for intuition. Again, green bold arrows represent new transitions, whereas red dashed ones are removed. Intuitively, an evaluation that maps x to 1 yields a PMC with the same behavior as $\mathcal{P}$ (consequent), while an evaluation that maps x to 0 yields a PMC with the same behavior as $\mathcal{P}'$ (alternative). We formalize this behavioral switching in terms of r-equivalence.

Lemma 7 *(R-equivalence for ITE). Given two compositional PMCs, $\mathcal{P} = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$ and $\mathcal{P}' = (S', s'_0, s'_{suc}, s'_{err}, X', \mathbf{P}', T')$, and a variable $x \notin X \cup X'$, let $\mathcal{P}'' = \texttt{ITE}(x, \mathcal{P}, \mathcal{P}')$. If $(\mathcal{P}'', p, w, FM)$ is an annotative model with $\mathcal{P}''$ as its underlying PMC,[8] where p, w, and FM are arbitrarily chosen, then, for every $c \in [\![FM]\!]$,*

$$\alpha([\![\texttt{ITE}(x, \mathcal{P}, \mathcal{P}')]\!]_c^w) = \begin{cases} \alpha([\![\mathcal{P}]\!]_c^w) & \text{if } p(x)(c) = 1 \\ \alpha([\![\mathcal{P}']\!]_c^w) & \text{otherwise} \end{cases}$$

Proof. We are interested in computing the probability of reaching s''_{suc} from s''_0 in $\mathcal{P}'' = \texttt{ITE}(x, \mathcal{P}, \mathcal{P}')$ under evaluation $w(c)$. Using the formal definition of ITE (Definition 34) and Definition 1, we are able to derive a reachability expression with only two terms, each corresponding to the "activated" PMC ($\mathcal{P}$ or $\mathcal{P}'$). The complete proof can be found in Appendix B.5.1. □

The above lemma establishes the ITE operator has the effect of alternating behaviors if the resulting PMC is evaluated by replacing the switching variable x with 0 or 1. With this result, we define the variability encoding of PMCs as a composition of PMCs using the ITE operator in a recursive way, with minimal PMCs as the base case. The alternative choice (second argument to ITE) is always the feature disabler PMC $\mathcal{P}_\perp$ (Definition 15), meaning no probabilistic behavior is actually added if the presence condition is not satisfied. This is coherent with the corresponding case in a composition factory (see Definition 17).

Definition 27 *(Variability encoding function for PMCs).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_k \in \mathscr{P}$ such that $\mathcal{P}_i \prec \mathcal{P}$ and $x_i = idt(\mathcal{P}_i)$ for $i \in \{1, \ldots, k\}$, the variability encoding function γ is defined as the following derivation by composition (Definition 18):

$$\gamma(\mathcal{P}) = \mathcal{P}[x_1/\texttt{ITE}(x_1, \gamma(\mathcal{P}_1), \mathcal{P}_\perp), \ldots, x_k/\texttt{ITE}(x_k, \gamma(\mathcal{P}_k), \mathcal{P}_\perp)]$$

[8] By Definition 9, any compositional PMC is also an annotative PMC (Definition 4). Thus, a compositional PMC can be the underlying PMC of an annotative model.

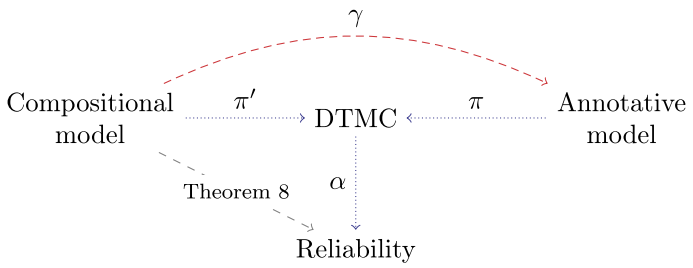


Fig. 21. Statement of Theorem 8.

This recursion terminates, since the arguments to the recursive calls involved are less than the input with respect to the well-founded relation $\prec$ (Lemma 11). Nonetheless, each variable x_i, which was meant as a slot marker, is replaced by a variable with the same name, but different meaning (i.e., intended to be evaluated with presence values). Since all variables in the PMC yielded by γ have this issue, the composition factory from the original compositional model will no longer be suitable. Thus, we must broaden the scope of variability encoding to also transform the composition factory w' into an annotative evaluation factory.

Definition 28 *(Variability encoding of PMCs).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$, let $\mathcal{P} \in \mathscr{P}$ be a PMC. Then, $(\gamma(\mathcal{P}), p, w, FM)$ is an annotative model that encodes $\mathcal{P}$'s variability, where w is an evaluation factory as in Definition 7.

The main goal of variability encoding is to transform a compositional model into an annotative one, but this technique can only be exploited if the reliability analysis of both the original and the transformed models yields the same results. This fact is established by the following theorem.

Theorem 8 *(R-equivalence of variability encoding and derivation by composition). Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and $\mathcal{P} \in \mathscr{P}$, let $(\gamma(\mathcal{P}), p, w, FM)$ be its variability-encoded annotative model. Then, for all $c \in [\![FM]\!]$,*

$$\alpha([\![\gamma(\mathcal{P})]\!]_c^w) = \alpha(\pi'(\mathcal{P}, w', c))$$

Proof. We use well-founded induction. For minimal PMCs (base of induction), $\gamma(\mathcal{P}) = \mathcal{P}$, so $[\![\gamma(\mathcal{P})]\!]_c^w = \mathcal{P}$. Likewise, $\pi'(\mathcal{P}, w', c) = \mathcal{P}$, so the proposition holds trivially.

As induction hypothesis, we have that $\alpha([\![\gamma(\mathcal{P}_i)]\!]_c^w) = \alpha(\pi'(\mathcal{P}_i, w', c))$ for all $\mathcal{P}_i \in \mathscr{P}$ such that $\mathcal{P}_i \prec \mathcal{P}$. Expanding $\alpha([\![\gamma(\mathcal{P})]\!]_c^w)$ and using previous soundness and r-equivalence results, we leverage this induction hypothesis to reach $\alpha(\pi'(\mathcal{P}, w', c))$.

The detailed proof can be found in Appendix B.5.1. □

In summary, Theorem 8 establishes the commuting diagram in Fig. 21, which corresponds to the upper arc in Fig. 10. Note that the derived DTMCs are not necessarily equal—this theorem only states α computes the same reliability for both models.

4.4.2. Variability encoding of expressions

Aside from encoding variability in Markov chains, we can also encode variability in reliability expressions (represented by the arc in the middle row of Fig. 10). Expressions derived from a compositional model can be combined to form a single larger expression (in terms of operands). Applying such a transformation can be useful in cases where parsing and evaluating each compositional expression is less efficient than doing so for the single variability-encoded expression. As with PMCs, variability encoding of expressions can be defined in terms of a dedicated *if-then-else* operator for expressions.

Definition 29 *(`ITE` operator for expressions).* Given two expressions ε and ε' over the sets X and X' of variables, respectively, and a variable x, the *if-then-else* operator for expressions is defined as

$$\texttt{ITE}(x, \varepsilon, \varepsilon') = x \cdot \varepsilon + (1 - x) \cdot \varepsilon'$$

The set of variables of the resulting expression is $X'' = X \cup X' \cup \{x\}$. Additionally, x is expected to be evaluated with a Boolean value, that is, 0 or 1. Procedures that do not affect the semantics of expressions, such as distributing the terms over the switching variable x and simplifying the resulting expression, can be leveraged in working implementations.

This *if-then-else* operator merges two expressions to form a third one that uses a new variable to represent a choice and satisfies the following lemma.

Lemma 8 *(Extensional equality for expression* ITE*). Given two expressions ε and ε' over the sets X and X' of variables, respectively, and a variable x, let $X'' = X \cup X' \cup \{x\}$ and $u : X'' \to [0, 1]$ be an evaluation function such that $u(x) \in \mathbb{B}$. Then,*

$$\texttt{ITE}(x, \varepsilon, \varepsilon')[X''/u] = \begin{cases} \varepsilon[X/u] & \text{if } u(x) = 1 \\ \varepsilon'[X'/u] & \text{if } u(x) = 0 \end{cases}$$

Proof. We prove this by expanding the definition of ITE and performing algebraic manipulation. The complete proof can be found in Appendix B.5.2. □

The above lemma establishes that the ITE operator has the effect of alternating the semantics of the resulting expression between the ones of its arguments, but only if this resulting expression is evaluated with an evaluation that replaces the switching variable x by 0 or 1. Similar to the ITE operator for PMCs, we define variability encoding of expressions as a composition of expressions using the ITE operator in a recursive way, with constant expressions (i.e., reliabilities of minimal PMCs) as the base case.

Definition 30 *(Variability encoding function for expressions).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_k \in \mathscr{P}$ such that $\mathcal{P}_i \prec \mathcal{P}$ and $x_i = idt(\mathcal{P}_i)$ for $i \in \{1, \ldots, k\}$, let $\varepsilon = \hat{\alpha}(\mathcal{P})$ and $\varepsilon_i = \hat{\alpha}(\mathcal{P}_i)$. The variability encoding function γ is overloaded for expressions as

$$\gamma(\varepsilon) = \varepsilon[x_1/\texttt{ITE}(x_1, \gamma(\varepsilon_1), \mathbf{1}), \ldots, x_k/\texttt{ITE}(x_k, \gamma(\varepsilon_k), \mathbf{1})]$$

This recursion terminates, since the arguments to the recursive calls involved are less than the input with respect to the well-founded relation $\prec$ (see Lemma 11).

Similar to variability encoding of PMCs, the new variables after encoding have the same names as the previous ones, but different meaning. Thus, we also transform the compositional evaluation factory w (Definition 25) into an annotative evaluation factory (see Definition 7). This way, we ensure variables, which have all been transformed into conditionals, are evaluated as expected of the ITE semantics.

Definition 31 *(Variability encoding of expressions).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$, and the compositional evaluation factory w, derived from the composition factory w', let w_p be an annotative evaluation factory (w in Definition 7) with the same presence conditions as w. That is, for all $c \in [\![FM]\!]$,

$$w_p(c)(x) = \begin{cases} 1 & \text{if } p(x)(c) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Then, for any $\mathcal{P} \in \mathscr{P}$ and $\varepsilon = \hat{\alpha}(\mathcal{P})$, $\gamma(\varepsilon)$ encodes ε's variability under the evaluation w_p.

We state the soundness of variability encoding for expressions in terms of r-equivalence. For any configuration $c \in [\![FM]\!]$, a variability-encoded expression and its corresponding evaluation factory must yield the same reliabilities as the original compositional expressions and the corresponding compositional evaluation factory.

Theorem 9 *(Soundness of variability encoding for expressions). Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_k \in \mathscr{P}$ such that $\mathcal{P}_i \prec \mathcal{P}$ and $x_i = idt(\mathcal{P}_i)$ for $i \in \{1, \ldots, k\}$, let $\varepsilon = \hat{\alpha}(\mathcal{P})$. Let also w be the compositional evaluation factory derived from w' (Definition 25) and w_p be the annotative evaluation factory obtained from w (Definition 31). Then, for all $c \in [\![FM]\!]$ it holds that*

$$\sigma(\gamma(\varepsilon), w_p, c) = \sigma(\varepsilon, w, c)$$

Proof. We use well-founded induction. For a minimal PMC $\mathcal{P}$ (base of induction), $\hat{\alpha}(\mathcal{P}) = \varepsilon$ has no variables. This way, $\gamma(\varepsilon) = \varepsilon$ and $\sigma(\varepsilon, u) = \varepsilon$ for any evaluation u. Thus, both sides of the equality evaluate to ε and the proposition holds trivially.

As induction hypothesis, we have that $\sigma(\gamma(\varepsilon_i), w_p, c) = \sigma(\varepsilon_i, w, c)$ for all $\varepsilon_i = \hat{\alpha}(\mathcal{P}_i)$ such that $\mathcal{P}_i \prec \mathcal{P}$. Expanding $\sigma(\gamma(\varepsilon), w_p, c)$ and using previous soundness and extensional equality results, we leverage this induction hypothesis to reach $\sigma(\varepsilon, w, c)$.

The detailed proof can be found in Appendix B.5.2. □

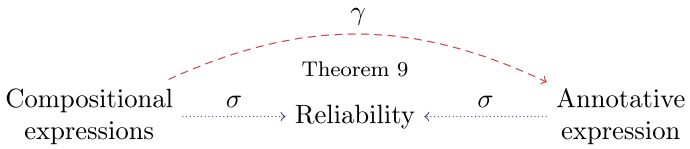


Fig. 22. Statement of Theorem 9.

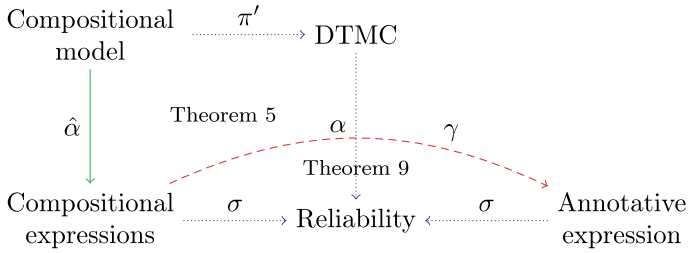


Fig. 23. Commuting diagram leading to the feature-family-product-based strategy.

As a further key result, Theorem 9 establishes the commuting diagram in Fig. 22. This diagram corresponds to the arc in the middle section of Fig. 10.

4.5. Feature-family-product-based strategy

So far, we have proved that all compositions of analysis steps leading up to reliabilities in Fig. 10 are r-equivalent. That is, these analysis steps commute, and, consequently, any path in this diagram can be equally taken to reach the same reliability value. By reflecting over the results condensed in this commuting diagram, we noticed a possible path that had not yet been exploited. This "unbeaten path", presented in Fig. 23 as an excerpt from Fig. 10, led us to derive a novel feature-family-product-based analysis strategy:

1. Starting from a compositional model (upper left corner), we apply parametric model checking ($\hat{\alpha}$) to obtain compositional expressions (*feature-based step*);
2. The resulting compositional expressions (lower left corner) are variability-encoded (γ) into a single annotative expression (*family-based step*); and
3. The annotative expression (lower right corner) is analyzed for each configuration $c \in [\![FM]\!]$ of the product line (*product-based step*).

The existence of a feature-family-product-based class of analyses was foreshadowed in a recent survey, but no instance has been found in the literature [47]. Thus, to the best of our knowledge, this is the first feature-family-product-based analysis to be presented, either formally or informally. The precise conditions under which this approach outperforms the others still need to be characterized by empirical studies. However, we believe it is an alternative to the family-product-based approach for cases in which (a) the model at hand is compositional and (b) applying variability encoding to the PMCs themselves is infeasible (e.g., the resulting annotative model is too big to be efficiently analyzed).

The novel strategy can be formally described as follows:

Strategy 7 *(Feature-family-product-based analysis).* Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and the compositional evaluation factory w, derived from the composition factory w', the feature-family-product-based analysis yields, for all $c \in [\![FM]\!]$,

$$\sigma\left(\gamma(\hat{\alpha}(\mathcal{P}_\top)), w_p, c\right)$$

or, alternatively,

$$[\![(\gamma \circ \hat{\alpha})(\mathcal{P}_\top)]\!]_c^{w_p}$$

where $\mathcal{P}_\top$ is the maximal PMC in $\mathscr{P}$ under the dependency relation $\prec$, and w_p is the variability-encoded annotative evaluation factory obtained from w (Definition 31).

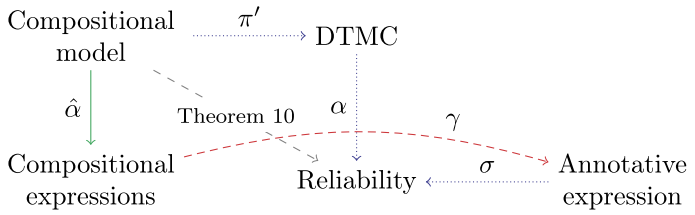


Fig. 24. Statement of Theorem 10.

Since the diagram in Fig. 10 commutes, this analysis is sound with respect to the product-based analysis of the same compositional model (Definition 2). This soundness property is established by the following theorem:

Theorem 10 *(Soundness of feature-family-product-based analysis). Given a compositional model* $(\mathscr{P}, \prec, I, idt, p, w', FM)$ *and a compositional evaluation factory* w*, derived from the composition factory* w'*, for every PMC* $\mathcal{P} \in \mathscr{P}$ *and for all configurations* $c \in [\![FM]\!]$ *it holds that*

$$\sigma\big(\gamma(\hat{\alpha}(\mathcal{P})), w_p, c\big) = \alpha\big(\pi'(\mathcal{P}, w', c)\big)$$

where w_p *is the variability-encoded annotative evaluation factory obtained from* w *(Definition 31).*

Proof. The proof follows from successive application of other commutativity theorems.

$$\begin{aligned}\sigma\big(\gamma(\hat{\alpha}(\mathcal{P})), w_p, c\big) &= \sigma\big(\hat{\alpha}(\mathcal{P}), w, c\big) && \text{(Theorem 9)}\\ &= \alpha\big(\pi'(\mathcal{P}, w', c)\big) && \text{(Theorem 5)}\end{aligned}$$ □

In summary, Theorem 10 states that the diagram in Fig. 24 commutes. This diagram corresponds to the upper left quadrant and the middle arc in Fig. 10.

Together, the theorems demonstrated in this section constitute the main contribution of this work. Intermediate steps of the presented analysis techniques commute, making the diagram in Fig. 10 fully commutative. Thus, any path constructed by following the arrows in that diagram yields an analysis that is equivalent to the one yielded by any other path that shares the same starting and ending points. This way, we guarantee all product-line reliability analysis techniques presented in this work yield the same results if given the same input models. Furthermore, we formally described the different analysis strategies in terms of reusable functions, making them comparable to one another.

5. Related work

Efficient analysis of software product lines is a relevant problem that has been tackled from many different perspectives, as pointed out by a recent survey [47]. In particular, several model checking techniques have been successfully lifted to work with product lines [11–13,22,21,9,32,3,40]. In contrast to existing research, our work presents different analysis techniques, covering all groups identified in the taxonomy by Thüm et al. [47], and relates these techniques to one another. Moreover, we present what is—to the best of our knowledge—the first feature-family-product-based analysis strategy in the literature. Hence, we discuss the closest related work according to different criteria.

PMC-based analysis of product lines: Ghezzi and Molzam Sharifloo [22] propose a model-based approach to analyze non-functional properties of product lines, illustrated by reliability and energy-consumption analysis. Their technique models probabilistic behavior by organizing parametric Markov chains in a hierarchical data structure, derived from nested UML sequence diagrams, annotated with the reliability of individual operations. Then, they employ parametric model checking in a bottom-up fashion, yielding a hierarchy of reliability expressions that are evaluated for each product configuration of interest. Although Ghezziand and Molzam Sharifloo also deal with modeling issues, their analysis technique can be seen as an instance of the *feature-product-based* reliability analysis in our framework, where the PMCs obtained from the nested sequence diagrams form the set $\mathscr{P}$ of compositional PMCs, and the decomposition tree induces the dependency relation $\prec$. For that reason, our work provides formal evidence of the soundness of their approach.

Rodrigues et al. [46] introduced *Featured Discrete-Time Markov Chains* (FDTMC), an extension of DTMCs to cope with variability and to represent the probabilistic behavior of product lines. This formalism, which is not restricted to reliability, enables verification of any probabilistic property that can be expressed using *Probabilistic Computation Tree Logic* (PCTL) [27]. The authors present three *family-based* approaches to conduct such analyses, one of which relies on an encoding of an FDTMC as a PMC to leverage off-the-shelf model checkers. Our work, in contrast, relies on models specifically tailored

to reliability analysis (a probabilistic reachability property), but incorporates different strategies to perform this analysis, covering the currently accepted product-line analysis taxonomy [47] in its entirety. Furthermore, Rodrigues et al. do not formally argue about the soundness of their approaches.

The framework we present can be leveraged to represent FDTMCs, provided that the reliability-specific constraints to PMCs are relaxed. We can say that any PMC $(S, s_0, X, \mathbf{P}, T)$, along with an evaluation factory w and a feature model FM, represents an FDTMC (S, ν, FM, Π) such that, for all $s, s' \in S$ and $c \in [\![FM]\!]$:

- $\Pi(s, s')(c) = \mathbf{P}(s, s')[X/w(c)]$; and
- $\nu(s) = \begin{cases} 1 & \text{if } s = s_0 \\ 0 & \text{otherwise} \end{cases}$

Feature-based model checking: Li et al. [35] and Liu et al. [37] have proposed feature-based approaches to the analysis of non-probabilistic temporal properties of product lines. Using models of feature behavior based on transition systems and required properties expressed with Computation Tree Logic (CTL) [10], they analyze each feature in isolation and generate partial results that can be later reused. The composition of features in their proposed models relies on interface states, a concept that we leveraged to define PMC interfaces and slots. However, the interfaces defined by Li et al. [35] can have an arbitrary number of outgoing states, and Liu et al. [37] extended them to support inter-feature cycles. Our use of interfaces, in contrast, is focused on reliability analysis (a probabilistic existence property expressed in PCTL), allowing us to define two outgoing states to abstract success and error conditions, while also ruling out the existence of cycles. Moreover, both Li et al. [35] and Liu et al. [37] treat feature modules as open systems, so they aggregate partial analysis results and CTL obligations to the interfaces themselves. Since we focus on a compositional model of a single product line, we use a separate model for intermediate feature reliability expressions. Because of these differences in modeling and in the nature of analyzed properties, we see their work and our own as complementary.

Family-based model checking: Dubslaff et al. [21] created a framework for modeling probabilistic and non-deterministic properties of dynamic product lines. This framework consists of modeling the behaviors of features in isolation, yielding models that are later composed into a family-based model. The models and their compositions are established in terms of *Markov Decision Processes* (MDP), enabling their representation in a way that allows the composed model to be model-checked using off-the-shelf tools [9]. The focus of their work is on modeling probabilistic behavior of product lines in a way that existing model checking techniques can be exploited. In contrast, our goal is to prove soundness of alternative analysis strategies, leaving modeling issues out of scope. Although their modeling and analysis technique is sufficiently general to enable reliability analysis of static product lines, which are our focus, it enables only family-based and product-based strategies (which the authors call, respectively, *all-in-one* and *one-by-one* [21]), whereas our work also includes the feature-based dimension. Nonetheless, their family-based technique is an alternative to ours, since it encodes the feature model's constraints in the behavioral model itself.

Kowal et al. [31] presented a formalism to describe performance models of product lines in a compositional fashion, based on performance-annotated activity diagrams described in a delta-oriented language. Similar to our work, they provide formal definitions and provide theorems stating the soundness of their approach (although proofs are not provided in the paper). However, similar to Dubslaff et al. [21], they only address family-based analysis of a model derived from the delta modules. Another difference to our work is that the semantics of their diagrams is expressed by continuous-time Markov chains (CTMC), which are more appropriate to performance analysis than DTMCs. Because of that, the two pieces of work complement each other. Future work could investigate the feasibility of defining alternative analysis strategies using their models and an approach similar to ours.

Variability encoding: Previous research has exploited variability encoding (also called configuration lifting) as a technique to produce family-based model checking of product lines [32,2,3,43]. von Rhein et al. [45] formalize variability encoding in the context of programming languages, that is, the transformation of compile-time variability into load-time variability. This transformation is realized using *if-then-else* operations and an encoding of features as control variables in the resulting program, which the authors call a variant simulator. They prove their transformation preserves the behavior of variants in the variability-encoded program for corresponding configurations. The concept of encoding variability in a simulator, as mentioned before, inspired our definitions of variability encoding for PMCs and expressions. Furthermore, their overall proof strategy resembles the one used throughout our work (i.e., comparison of results for corresponding configurations). However, whereas von Rhein et al. [45] use trace semantics and a weak bisimulation relation to correlate behaviors, we perform this task using structural analysis of the behavioral models. Despite being less general, structural analysis is sufficiently strong for the purpose of proving that reliability is preserved, which is the main focus of our work.

Formal approaches to variability-aware analysis: The definition of product-line analysis techniques that are sound by construction has been investigated recently [39,6,8,7], although not specifically in the context of model checking. Midtgaard et al. [39] presented a methodology to derive family-based static analyses from single-product analyses based on *abstract interpretation*. This approach enables the lifting of existing analyses to work with product lines, yielding variability-aware analyses that are correct by construction. Although the authors only walked through a data-flow analysis scenario, they claim the methodology could be applied to other analyses, including model checking. Similar to their work, we provide soundness proofs of product-line analyses, conditioned on the soundness of a given single-product analysis. However, we

do not provide a framework for derivation of analysis strategies in general; instead, we focus on providing formal evidence that a set of alternative strategies for reliability analysis are sound, while also highlighting the relations between their intermediate steps. Moreover, whereas Midtgaard et al. handle only the family-based dimension of analysis, we also address the feature-based dimension. In this sense, our work can also be seen as a preliminary investigation on deriving alternative strategies to perform a given analysis.

Brabrand et al. [7] proposed a technique to automatically lift intraprocedural data-flow analyses to handle variability in product lines. Similar to our work, the authors propose alternative analysis strategies, which are derived by gradually introducing variability awareness in different components of an existing analysis. Brabrand et al. [7] also present a soundness proof for the proposed strategies, whereby all of them are guaranteed to compute the same result as the base analysis. The presented simultaneous and consecutive analysis strategies are similar to our family-based and family-product-based ones, respectively, even though different properties are analyzed. However, Brabrand et al. [7] do not consider feature-based analyses. Furthermore, our work breaks down analysis strategies in intermediate steps that can be composed in different ways, enabling reuse of proofs.

Comparison of analysis dimensions: Kolesnikov et al. [30] empirically compared family-based, feature-based, and product-based type checking of Java-based product lines. Their work was the first empirical study covering all three dimensions of analysis, providing guidance to practitioners over which type checking strategy to apply for a given product line. In a sense, their research and our own are complementary, since each one deals with a different analysis type (type checking and model checking). However, in contrast with their work, our focus is on the formal aspects of analysis—although we argue our techniques can be implemented in a tool to perform empirical studies. Furthermore, Kolesnikov et al. neither investigate combined strategies nor prove the soundness of the implemented type checkers.

von Rhein et al. [44] proposed a model for classification and comparison of product-line analyses (the *PLA model*), whereby existing analyses are broken down into intermediate steps. This model abstracts possible steps as four operators for composing features, encoding variability, resolving variability, and generic processing of artifacts. As stated by the authors themselves, the PLA model is helpful when describing complex analyses and designing new ones. Indeed, the PLA model was a source of inspiration for designing our analysis techniques as reusable analysis steps. However, we found the proposed operators to be too generic to be useful in our formal setup. In this sense, our work complements the work by von Rhein et al. [44] with a formally defined relation among analyses and intermediate steps, albeit restricted to reliability analysis.

Conceptual models and taxonomy: Thüm et al. [47] established the taxonomy for product-line analyses upon which we based our work, that is, the classification of analysis techniques in three basic strategies (product-based, feature-based, and family-based) and combinations thereof. von Rhein et al. [44] laid these strategies as dimensions in a cube, meaning analysis strategies can be expressed as a combination of the number of analyzed products (*sampling* dimension), the granularity of feature combinations (*feature grouping* dimension), and the extent to which variability is preserved or resolved during analysis (*variability encoding* dimension). Since our soundness proofs for variability encoding and feature composition apply to single features (not necessarily maximal PMCs), our techniques range over the PLA plane of feature grouping and variability encoding dimensions. Furthermore, given that sampling is a matter of restricting possible configurations and that we prove that our techniques are sound configuration-wise, our work also covers the sampling dimension.

Meinicke et al. [38] recently surveyed existing product-line analysis tools and categorized them along four criteria: product-line implementation technique (annotation-based *versus* composition-based approach), analysis technique (e.g., testing, type checking, model checking), strategies for product-line analysis (i.e., the analysis strategies taxonomy by Thüm et al. [47]), and strategy of the tool (product-based, variability-aware, and variability-encoding). Using this taxonomy, an implementation of our techniques would cover all possibilities on the dimensions of implementation technique, strategies for product-line analysis, and strategy of the tool, while the dimension of analysis technique would be fixed to reliability analysis.

6. Conclusion

We formally presented seven approaches to reliability analysis of product lines, covering all strategies in the taxonomy by Thüm et al. [47]. To the best of our knowledge, this is the first work to address all three dimensions of product-line analysis (product-based, family-based, and feature-based) in the context of model checking, and also the first to present an instance of feature-family-product-based analysis strategy. The soundness of our analysis techniques is established by results on the commutativity of their intermediate steps, summarized by the commuting diagram in Fig. 10. This constitutes formal evidence that, given a product line, each of the presented approaches yields the same results as the others, enabling practitioners to choose among analysis strategies based on their space and time trade-offs. Future work can build on this formal foundation to compare techniques in search for selection criteria.

The input models for our analysis approaches are based on the formalism of parametric Markov chains, meaning they can be represented using the input language of parametric model checkers such as PRISM [33] and PARAM [25]. Indeed, the parametric probabilistic reachability algorithm by Hahn et al. [26], used throughout this work as an instance of variability-aware analysis function ($\hat{\alpha}$), is implemented by these tools. Therefore, we argue that our analysis approaches are feasible, and that they can be implemented by a program that coordinates calls to an off-the-shelf parametric model checker accord-

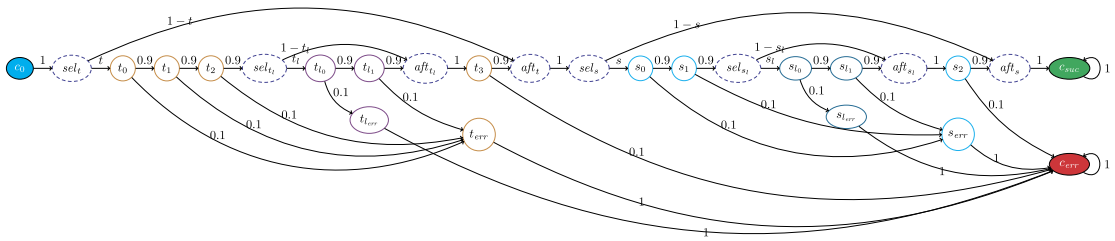


Fig. A.1. Complete annotative PMC for the vending machine.

ing to variability information. The product-based techniques and variability encoding can be implemented by manipulating the PMCs themselves.

Although our theory is focused on reliability analysis, we were able to prove a general result on lifting rational functions over the Real numbers to work with ADDs (Lemma 4), which can be leveraged to evaluate algebraic expressions in the context of product lines. Future work may also extend our analysis theory with product-line analyses other than reliability, seeking commonalities in definitions and soundness proofs. As suggested by Fig. 10, we also believe that category theory can be leveraged to analyze and describe such extended theories, as a means towards the broader goal of finding a set of general principles relating different dimensions of product-line analysis.

Acknowledgements

We thank the anonymous reviewers for the useful suggestions for improvement. This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES)[9], funded by CNPq [grant 465614/2014-0] and FACEPE [grant APQ-0388-1.03/14]. Thiago Castro acknowledges support from the Science and Technology Department of the Brazilian Army. Vander Alves would like to thank for the research grant CAPES ref. BEX 0557-16-1 / Alexander von Humboldt ref. 3.2-1190844-BRA-HFSTCAPES-E. Leopoldo Teixeira is supported by FACEPE [grant APQ-0570-1.03/14] and CNPq [grant 409335/2016-9]. Sven Apel is supported by the German Research Foundation (AP 206/4 and AP 206/6).

Appendix A. Probabilistic models

This section presents the probabilistic models of the beverage machine product line example (Section 3) in their entirety. Fig. A.1 contains the annotative model, and the compositional model is depicted by Fig. A.2.

Appendix B. Additional proofs

This section contains formal definitions and proofs that were omitted from the main body of the paper to avoid digressions.

B.1. Existence of minimal and maximal PMCs

Lemma 9 *(Existence of minimal PMCs). Given a set $\mathscr{P}$ of compositional PMCs, an identifying function idt, and the corresponding induced well-founded relation $\prec$, there exists, at least, one* minimal *PMC $\mathcal{P} = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$. Furthermore, $X = \emptyset$, that is, minimal PMCs are, in fact, DTMCs with defined interfaces and only two bottom strongly connected components (cf. Definition 9).*

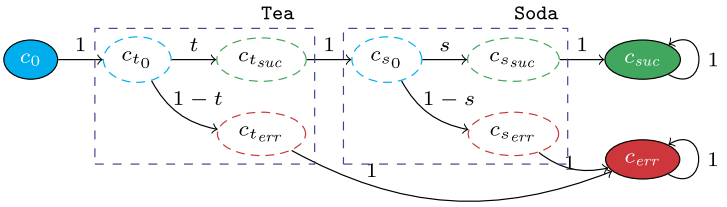
Proof. The existence of minimal PMCs follows directly from the fact that the induced relation $\prec$ is well-founded: otherwise, all descending chains would be infinite.

Now suppose $X \neq \emptyset$. Then, it has, at least, one element x. Since the set I of identifiers (image of idt) is a superset of all X_i, $x \in I$. By definition, function idt is bijective, so there must be a compositional PMC $\mathcal{P}' \in \mathscr{P}$ such that $idt(\mathcal{P}') = x$. But $idt(\mathcal{P}') \in X \Rightarrow \mathcal{P}' \prec \mathcal{P}$. Since $\mathcal{P}$ is minimal by hypothesis, this is a contradiction. □

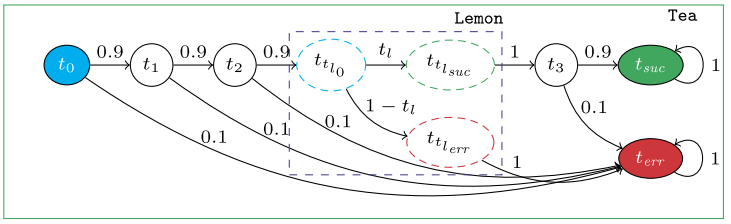
Lemma 10 *(Existence of maximal PMCs). Given a set $\mathscr{P}$ of compositional PMCs, an identifying function idt, and the corresponding induced well-founded relation $\prec$, there exists, at least, one* maximal *PMC $\mathcal{P} = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$.*

Proof. The proof is by contraposition. Nonexistence of such maximal PMC means there are infinite *ascending* chains $\mathcal{P}_1 \prec \mathcal{P}_2 \prec \mathcal{P}_3 \prec \ldots$ for $\mathcal{P}_i \in \mathscr{P}$. Since $\mathscr{P}$ is finite, such infinite chain implies the existence of cycles, that is, at least, one $\mathcal{P}_i$ transitively depending on itself. But cycles imply both ascending and descending infinite chains, contradicting the

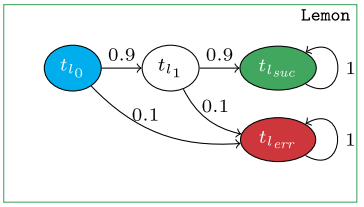
[9] http://www.ines.org.br.



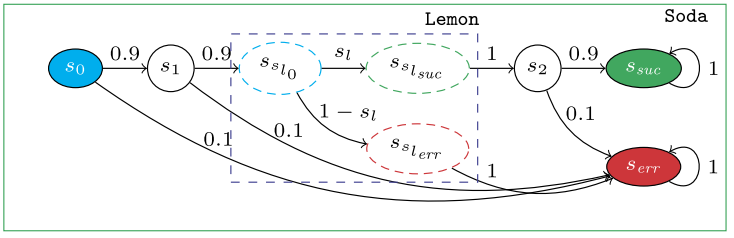
(a) Top-level compositional PMC for the vending machine (common behavior and main variation points)



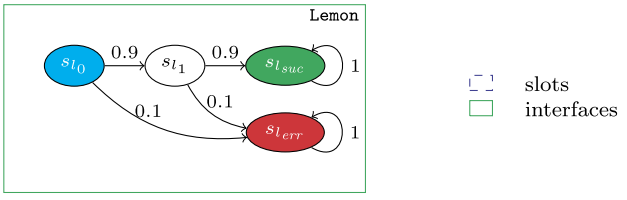
(b) Compositional PMC for the behavior of serving tea



(c) Compositional PMC for the behavior of adding lemon to tea



(d) Compositional PMC for the behavior of serving soda



(e) Compositional PMC for the behavior of adding lemon to soda

Fig. A.2. Compositional PMCs for the vending machine.

well-foundedness of $\prec$. Hence, there are no infinite ascending chains under $\prec$ and, by contraposition, there is, at least, one maximal PMC. □

B.2. Termination lemmas

The following lemma states the termination of the recursive definitions of the composition factory w' (Definition 17).

Lemma 11 *(Derivation by composition terminates). For a compositional model* $(\mathscr{P}, \prec, I, idt, p, w', FM)$*, for all configurations* $c \in [\![FM]\!]$*, the composition function* $w'(c)$ *terminates.*

Proof. Let $idt^{-1} : I \to \mathscr{P}$ be the inverse function of idt. To prove $w'(c)$ terminates, we note that the arguments in the recursive calls in the definition of w' (Definition 17) strictly decrease if we use idt^{-1} as a measure function into the well-founded set $\mathscr{P}$.

Without loss of generality, let $x = idt(\mathcal{P})$ for some $\mathcal{P} \in \mathscr{P}$ with variables set $X = \{x_1, \ldots, x_k\}$. The right-hand side of $w'(c)(x)$ evaluates to either $\mathcal{P}_\perp$ (the feature disabler PMC) or $\mathcal{P}[x_1/w'(c)(x_1), \ldots, x_k/w'(c)(x_k)]$. In the first case, it trivially terminates, since $\mathcal{P}_\perp$ have no slots; in the latter, the arguments to each recursive call are the variables $x_i \in X$. By definition, $x_i = idt(\mathcal{P}_i)$ for some $\mathcal{P}_i \in \mathscr{P}$ such that $\mathcal{P}_i \prec \mathcal{P}$. Thus, $idt^{-1}(x_i) \prec idt^{-1}(x)$. Since $\prec$ is well-founded, $w'(c)$ terminates. □

The following lemma states that the recursion in Definition 25 terminates.

Lemma 12 *(Compositional evaluation terminates). For a compositional model* $(\mathscr{P}, \prec, I, idt, p, w', FM)$*, for all configurations* $c \in [\![FM]\!]$*, the compositional evaluation* $w(c)$ *terminates.*

Proof. Let $idt^{-1} : I \to \mathscr{P}$ be the inverse function of idt. To prove $w(c)$ terminates, we note that the arguments in recursive calls to $w(c)$ (Definition 25) strictly decrease if we use idt^{-1} as a measure function into the well-founded set $\mathscr{P}$.

Indeed, without loss of generality, let $x = idt(\mathcal{P})$ for some $\mathcal{P} \in \mathscr{P}$ with variables set $X = \{x_1, \ldots, x_k\}$. By definition of σ, the right-hand side of $w(c)(x)$ evaluates to either 1 or $\hat{\alpha}(\mathcal{P})[x_1/w(c)(x_1), \ldots, x_k/w(c)(x_k)]$. In the first case, it trivially terminates; in the second, the arguments to each recursive call are the variables $x_i \in X$. By definition, $x_i = idt(\mathcal{P}_i)$ for some $\mathcal{P}_i \in \mathscr{P}$ such that $\mathcal{P}_i \prec \mathcal{P}$. Thus, $idt^{-1}(x_i) \prec idt^{-1}(x)$. Since $\prec$ is well-founded, $w(c)$ terminates. □

B.3. Soundness of feature-product-based analysis

We first state formally what we mean by PMC renaming, which is a key concept in PMC composition.

Definition 32 *(Compositional PMC renaming).* Given a compositional PMC $\mathcal{P} = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$, the i-th renaming of $\mathcal{P}$, $\mathcal{P}^i = (S^i, s_0^i, s_{suc}^i, s_{err}^i, X^i, \mathbf{P}^i, T^i)$, is an isomorphic compositional PMC with renamed states. That is, $\mathcal{P}^i$ is such that:

- $S^i \cap S = \emptyset$.
- $\forall_{i,j \in \mathbb{N}} \cdot i \neq j \implies S^i \cap S^j = \emptyset$.
- There exists a bijective mapping $\_^i : S \to S^i$ from each state $s_j \in S$ to a state $s_j^i \in S^i$.
- $X^i = X$.
- $\forall_{s_1, s_2 \in S} \cdot \mathbf{P}^i(s_1^i, s_2^i) = \mathbf{P}(s_1, s_2)$.
- $T^i = \{s^i \mid s \in T\}$.

With the formal definition of PMC renaming, we are able to present a precise definition of a total composition, obtained by composing PMCs over all slots in a given base compositional PMC at once.

Definition 33 *(Total PMC composition).* Given a compositional PMC $(S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$ with k variables $x_1, \ldots, x_k$, and a set $\mathscr{P}$ of k compositional PMCs $(S_i, s_{i_0}, s_{i_{suc}}, s_{i_{err}}, X_i, \mathbf{P}_i, T_i)$, $i \in \{1, \ldots, k\}$, let $u' : X \to \mathscr{P}$ be a function that yields a compositional PMC $\mathcal{P} \in \mathscr{P}$ to compose in the corresponding slots for any given variable. Let also $n_i = |slots^{\mathcal{P}}(x_i)|$ for $i \in 1, \ldots, k$, and $\mathcal{P}_i^j = (S_i^j, s_{i_0}^j, s_{i_{suc}}^j, s_{i_{err}}^j, X_i^j, \mathbf{P}_i^j, T_i^j)$ for $j \in 1, \ldots, n_i$ be the j-th renaming of $\mathcal{P}_i$ (Definition 32). The total PMC composition $\mathcal{P}[X/u']$, also denoted by $\mathcal{P}[x_1/u'(x_1), \ldots, x_k/u'(x_k)]$, is a compositional PMC $\mathcal{P}' = (S', s_0', s_{suc}', s_{err}', X', \mathbf{P}', T')$ such that:

- $S' = S \uplus \biguplus_{j=1}^{n_1} S_1^j \uplus \cdots \uplus \biguplus_{j=1}^{n_k} S_k^j$, where $\uplus$ denotes the disjoint union operator (all states are disjointly merged);
- $s_0' = s_0$, $s_{suc}' = s_{suc}$, and $s_{err}' = s_{err}$ (the interface of $\mathcal{P}$ is preserved);
- $X' = \bigcup_{i=1}^{k} X_i$ (each occurrence of x_i is replaced by a copy of $\mathcal{P}_i$, whose variables are those of X_i);
- $T' = T$ (target states of the base PMC are preserved);
- $\mathbf{P}'$ is such that, for all slots $(s_{x_{i_0}^j}, s_{x_{i_{suc}}^j}, s_{x_{i_{err}}^j})$ of the base PMC $\mathcal{P}$ and interfaces $(s_{i_0}^j, s_{i_{suc}}^j, s_{i_{err}}^j)$ of the renamed PMCs $\mathcal{P}_i^j$ (where $i \in 1, \ldots, k$ and $j \in 1, \ldots, n_i$),
  - $\mathbf{P}'(s_{x_{i_0}^j}, s_{i_0}^j) = 1$ (new transition from a slot's initial state to the initial state of the corresponding composed PMC)
  - $\mathbf{P}'(s_{i_{suc}}^j, s_{x_{i_{suc}}^j}) = 1$ (new transition from the success state of a composed PMC to the success state of the corresponding slot)
  - $\mathbf{P}'(s_{i_{err}}^j, s_{x_{i_{err}}^j}) = 1$ (new transition from the error state of a composed PMC to the error state of the corresponding slot)
  - $\mathbf{P}'(s_{x_{i_0}^j}, s_{x_{i_{suc}}^j}) = 0$ (slot's success transition is removed)

- $\mathbf{P}'(s_{x_{i_0}^j}, s_{x_{i_{err}}^j}) = 0$ (slot's error transition is removed)
- $\mathbf{P}'(s_{i_{suc}}^j, s_{i_{suc}}^j) = 0$ (success loops from composed PMCs are removed)
- $\mathbf{P}'(s_{i_{err}}^j, s_{i_{err}}^j) = 0$ (error loops from composed PMCs are removed)
- For all remaining combinations of $s_1, s_2 \in S'$:

$$\mathbf{P}'(s_1, s_2) = \begin{cases} \mathbf{P}(s_1, s_2) & \text{if } s_1, s_2 \in S \setminus \mathit{slotStates}^{\mathcal{P}}(X) \\ \mathbf{P}_i^j(s_1, s_2) & \text{if } s_1, s_2 \in S_i^j \\ 0 & \text{otherwise} \end{cases}$$

The function u' is called a *composition function*.

To establish the soundness of the feature-product-based strategy, we need to compare it to the product-based strategy for compositional models. However, the latter relies on PMC composition, while the former is based on compositional evaluation of expressions. To bridge this gap, we first note that, as far as reliability analysis is concerned, composing a PMC $\mathcal{P}'$ into a slot of another PMC $\mathcal{P}$ is equivalent to evaluating the corresponding variable in $\mathcal{P}$ with the reliability expression of $\mathcal{P}'$ (i.e., $\hat{\alpha}(\mathcal{P}')$).

Lemma 13 *(R-equivalence of total composition and evaluation). Let $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_k$ be compositional parametric Markov chains, and $X = \{x_1, \ldots, x_k\}$ be $\mathcal{P}$'s set of variables. Then,*

$$\hat{\alpha}(\mathcal{P}[x_1/\mathcal{P}_1, \ldots, x_k/\mathcal{P}_k]) = \hat{\alpha}(\mathcal{P}[x_1/\hat{\alpha}(\mathcal{P}_1), \ldots, x_k/\hat{\alpha}(\mathcal{P}_k)])$$

where the equals sign denotes extensional *equality. In other words, the two expressions (i.e., syntactic objects) are not necessarily equal in a syntactical sense, but their corresponding rational functions (i.e., semantic objects) always yield equal values if given equal inputs.*

Proof. The main argument for this proof is the case where $\mathcal{P}$ has only one variable, that is, $X = \{x\}$. This way, we start by proving that $\hat{\alpha}(\mathcal{P}[x/\mathcal{P}']) = \hat{\alpha}(\mathcal{P}[x/\hat{\alpha}(\mathcal{P}')])$ for a given compositional PMC $\mathcal{P}' = (S', s'_0, s'_{suc}, s'_{err}, X', \mathbf{P}', T')$. Then, we extend this to the general case where $\mathcal{P}$ has an arbitrary number of variables.

A generic illustration of $\mathcal{P}$ and $\mathcal{P}'$ is given by Figs. 7a and 7b, respectively. Let $\mathcal{P}_e = \mathcal{P}[x/\hat{\alpha}(\mathcal{P}')]$ be the PMC resulting from evaluation, denoted by $(S_e, s_{e_0}, s_{e_{suc}}, s_{e_{err}}, X_e, \mathbf{P}_e, T_e)$, and $\mathcal{P}_c = \mathcal{P}[x/\mathcal{P}']$ be the PMC obtained by composition, denoted by the tuple $(S_c, s_{c_0}, s_{c_{suc}}, s_{c_{err}}, X_c, \mathbf{P}_c, T_c)$. Figs. B.3a and B.3b represent these PMCs and serve as a visual aid to the proof.

Since $\hat{\alpha}$ computes the probabilistic reachability property, we base this proof on the algorithm by Hahn et al. [26]. This algorithm consists of successive eliminations of states, with the transition probability matrix being updated at each step. A useful property, which Hahn et al. use to prove that the algorithm is sound, is that the probability of reaching the target states in the input PMC is an invariant, that is, it remains the same throughout elimination steps.

Let us apply the algorithm by Hahn et al. [26] to $\mathcal{P}_c$. For brevity, we show the composition via a single slot. In the case where more slots exist, the following argument can be applied sequentially to each slot and corresponding renaming of $\mathcal{P}'$.

Since the order in which states are eliminated is not fixed, we first eliminate states $s' \in S' \setminus \mathit{interface}(\mathcal{P}')$. The intermediate PMC at this point is given by Fig. B.3c. These eliminations are restricted to states in S', because the only transitions in $\mathbf{P}_c$ between states in S and states in S' are the ones connecting interface and slot (by construction—see Definition 11).

Now, we eliminate the interface states. Performing a single step of the algorithm by Hahn et al. (Definition 3), we eliminate s'_0 and update $\mathbf{P}_c$ so that

$$\begin{aligned} \mathbf{P}_c(s_{x_0}, s'_{suc}) &= \mathbf{P}_c(s_{x_0}, s'_{suc}) + \mathbf{P}_c(s_{x_0}, s'_0) \cdot \frac{1}{1 - \mathbf{P}_c(s'_0, s'_0)} \cdot \mathbf{P}_c(s'_0, s'_{suc}) \\ &= 0 + 1 \cdot \frac{1}{1 - 0} \cdot Pr^{\mathcal{P}'}(s'_0, s'_{suc}) \\ &= Pr^{\mathcal{P}'}(s'_0, s'_{suc}) \end{aligned}$$

Similarly, $\mathbf{P}_c(s_{x_0}, s'_{err}) = Pr^{\mathcal{P}'}(s'_0, s'_{err})$. Repeating these steps for s'_{suc} and s'_{err}, $\mathbf{P}_c$ is updated to have $\mathbf{P}_c(s_{x_0}, s_{x_{suc}}) = Pr^{\mathcal{P}'}(s'_0, s'_{suc})$ and $\mathbf{P}_c(s_{x_0}, s_{x_{err}}) = Pr^{\mathcal{P}'}(s'_0, s'_{err})$ (see Fig. B.3d).

At this stage, all states $s' \in S'$ have been eliminated, so that $S_c = S = S_e$. Furthermore, for all $s_1, s_2 \in S \setminus \mathit{slotStates}^{\mathcal{P}}(x)$, the transition probability matrices are such that $\mathbf{P}_c(s_1, s_2) = \mathbf{P}(s_1, s_2) = \mathbf{P}_e(s_1, s_2)$ (Definition 11). Thus, the only difference between $\mathcal{P}_c$ and $\mathcal{P}_e$ are the transitions for slot states: $(s_{x_0}, s_{x_{suc}})$ and $(s_{x_0}, s_{x_{err}})$.

For the "success" slot, $\mathbf{P}_c(s_{x_0}, s_{x_{suc}}) = Pr^{\mathcal{P}'}(s'_0, s'_{suc})$, which is *syntactically* equal to $\mathbf{P}_e(s_{x_0}, s_{x_{suc}})$. So, we must prove that the "error" transitions, $\mathbf{P}_c(s_{x_0}, s_{x_{err}})$ and $\mathbf{P}_e(s_{x_0}, s_{x_{err}})$, are extensionally equal. But s'_{suc} and s'_{err} are the only two bottom strongly connected components of the underlying digraph of $\mathcal{P}'$ (Definition 9). Thus, by Theorem 10.27 of Baier and Katoen [5],

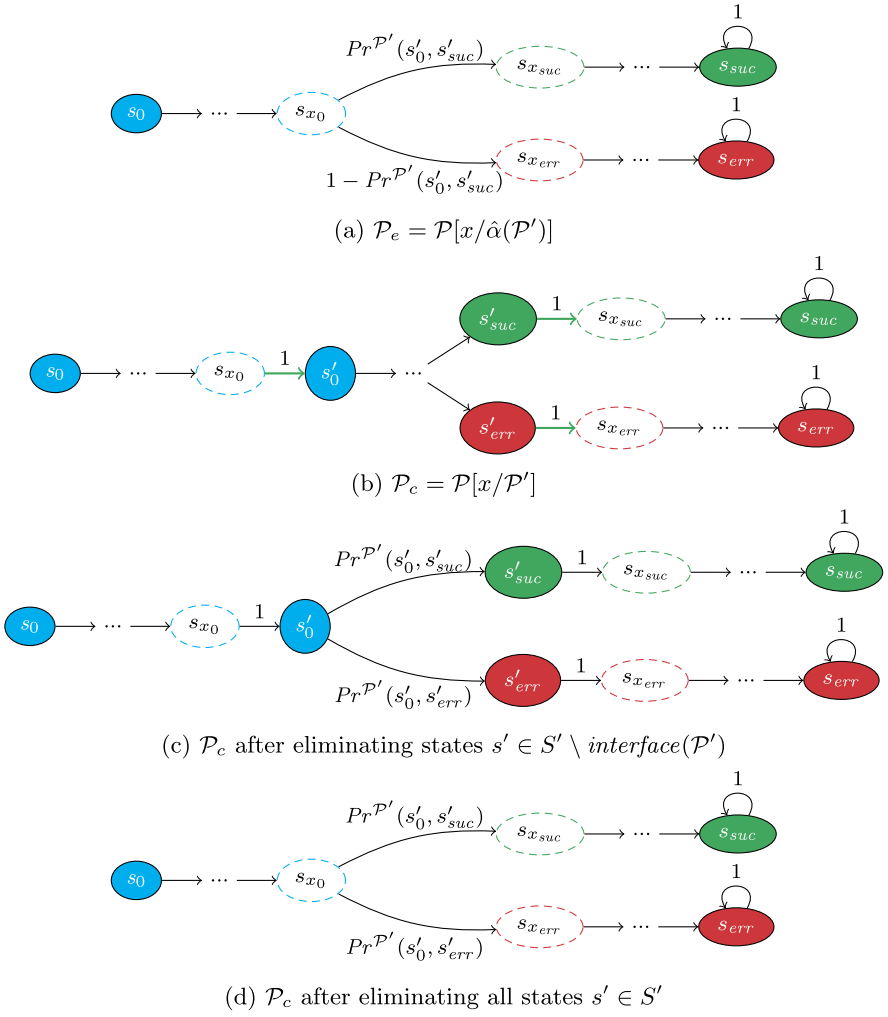


Fig. B.3. Generic PMCs in Lemma 13.

$Pr^{\mathcal{P}'_u}(s'_0, s'_{suc}) + Pr^{\mathcal{P}'_u}(s'_0, s'_{err}) = 1$, where $\mathcal{P}'_u$ is the DTMC obtained by applying some well-defined evaluation u to $\mathcal{P}'$. Since the choice of u is arbitrary, $\mathbf{P}_c(s_{x_0}, s_{x_{err}})$ is extensionally equal to $\mathbf{P}_e(s_{x_0}, s_{x_{err}})$.

This means that, at the current point of application of the probabilistic reachability algorithm to $\mathcal{P}_c$, $\mathbf{P}_c = \mathbf{P}_e$. $\mathcal{P}_e$ and the partially analyzed $\mathcal{P}_c$ have the same probability of reaching the target state s_{suc}. Moreover, since the algorithm preserves this probability at each step, the probabilistic reachability in $\mathcal{P}_c$ is the same at this point as before the algorithm started, and will remain the same until the algorithm stops. Hence, $\hat{\alpha}(\mathcal{P}[x/\mathcal{P}']) = \hat{\alpha}(\mathcal{P}[x/\hat{\alpha}(\mathcal{P}')])$.

To extend this proof to the case where $\mathcal{P}$ has an arbitrary number of variables, we repeat the argument that the choice of states for elimination is arbitrary. Let us assume, as induction hypothesis, that the lemma holds for a PMC with n variables. If $\mathcal{P}$ has $n+1$ variables, we apply the same reasoning as in the single-variable case for one of $\mathcal{P}$'s slots, $(s_{x_{n+1_0}}, s_{x_{n+1_{suc}}}, s_{x_{n+1_{err}}})$. After eliminating *only* the states corresponding to a composition at the given slot, we have the following extensional equalities: $\mathbf{P}_c(s_{x_{n+1_0}}, s_{x_{n+1_{suc}}}) = \mathbf{P}_e(s_{x_{n+1_0}}, s_{x_{n+1_{suc}}})$ and $\mathbf{P}_c(s_{x_{n+1_0}}, s_{x_{n+1_{err}}}) = \mathbf{P}_e(s_{x_{n+1_0}}, s_{x_{n+1_{err}}})$. Also, the resulting PMC $\mathbf{P}_c$ has n remaining slots, one for each variable. By the induction hypothesis, after eliminating the states corresponding to all compositions in $\mathbf{P}_c$, we have that $\mathbf{P}_c$ and $\mathbf{P}_e$ are extensionally equal. Hence, $\hat{\alpha}(\mathcal{P}[x_1/\mathcal{P}_1, \ldots, x_{n+1}/\mathcal{P}_{n+1}]) = \hat{\alpha}(\mathcal{P}[x_1/\hat{\alpha}(\mathcal{P}_1), \ldots, x_{n+1}/\hat{\alpha}(\mathcal{P}_{n+1})])$. □

Furthermore, since a composition of only DTMCs into a PMC yields another DTMC, both parametric and non-parametric model checking of this resulting chain (which has no variability) produce the same result. Thus, we have the following corollary of Lemma 13.

Corollary 1 *(R-equivalence of total composition with DTMCs and evaluation). Let $\mathcal{P}$ be a compositional PMC, $\mathcal{D}_1, \ldots, \mathcal{D}_k$ be DTMCs, and $X = \{x_1, \ldots, x_k\}$ be $\mathcal{P}$'s variables set. Then,*

$$\alpha(\mathcal{P}[x_1/\mathcal{D}_1, \ldots, x_k/\mathcal{D}_k]) = \alpha(\mathcal{P}[x_1/\alpha(\mathcal{D}_1), \ldots, x_k/\alpha(\mathcal{D}_k)])$$

Now we have the tools to prove that our feature-product-based analysis is sound. We recall Theorem 5:

Theorem 5 *(Soundness of feature-product-based analysis). Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$, for all configurations $c \in [\![FM]\!]$, it holds that*

$$\sigma(\hat{\alpha}(\mathcal{P}), w, c) = \alpha(\pi'(\mathcal{P}, w', c))$$

or, alternatively,

$$[\![\hat{\alpha}(\mathcal{P})]\!]_c^w = \alpha([\![\mathcal{P}]\!]_c^{w'})$$

where $\mathcal{P} \in \mathscr{P}$ and w is the compositional evaluation factory (Definition 25) derived from the composition factory w'.

Complete proof. We use well-founded induction. The base of the induction is when $\mathcal{P}$ is minimal with respect to $\prec$. In this case, $X = \emptyset$, so $\pi'(\mathcal{P}, w', c) = \mathcal{P}$, that is, $\alpha(\pi'(\mathcal{P}, w', c)) = \alpha(\mathcal{P})$. Likewise, $\hat{\alpha}(\mathcal{P}) = \alpha(\mathcal{P})$, so that $\sigma(\hat{\alpha}(\mathcal{P}), w, c) = \sigma(\alpha(\mathcal{P}), w, c) = \alpha(\mathcal{P})$. Thus, for the base case, $\sigma(\hat{\alpha}(\mathcal{P}), w, c) = \alpha(\pi'(\mathcal{P}, w', c))$.

We now have to prove that $\sigma(\hat{\alpha}(\mathcal{P}), w, c) = \alpha(\pi'(\mathcal{P}, w', c))$ for an arbitrary $\mathcal{P} \in \mathscr{P}$. Our induction hypothesis is that $\sigma(\hat{\alpha}(\mathcal{P}_i), w, c) = \alpha(\pi'(\mathcal{P}_i, w', c))$ for all $\mathcal{P}_i \in \mathscr{P}$ such that $\mathcal{P}_i \prec \mathcal{P}$. Thus, let $x_i = idt(\mathcal{P}_i)$, $i \in \{1, \ldots, k\}$. By Definition 21, we have:

$$\sigma(\hat{\alpha}(\mathcal{P}), w, c) = \hat{\alpha}(\mathcal{P})[x_1/w(c)(x_1), \ldots, x_k/w(c)(x_k)]$$

For each x_i, from the definition of the compositional evaluation factory w (Definition 25),

$$w(c)(x_i) = \begin{cases} \sigma(\hat{\alpha}(\mathcal{P}_i), w, c) & \text{if } p(x_i)(c) = 1 \\ 1 & \text{otherwise} \end{cases}$$

$$= \begin{cases} \alpha(\pi'(\mathcal{P}_i, w', c)) & \text{if } p(x_i)(c) = 1 \quad \text{(by induction hypothesis)} \\ 1 & \text{otherwise} \end{cases}$$

But, from the definition of the composition factory w' (Definition 17),

$$w'(c)(x_i) = \begin{cases} \mathcal{P}_i[X_i/w'(c)] & \text{if } p(x_i)(c) = 1 \\ \mathcal{P}_\perp & \text{otherwise} \end{cases}$$

$$= \begin{cases} \pi'(\mathcal{P}_i, w', c)) & \text{if } p(x_i)(c) = 1 \quad \text{(Definition 18)} \\ \mathcal{P}_\perp & \text{otherwise} \end{cases}$$

Applying α to both sides,

$$\alpha(w'(c)(x_i)) = \begin{cases} \alpha(\pi'(\mathcal{P}_i, w', c)) & \text{if } p(x_i)(c) = 1 \\ \alpha(\mathcal{P}_\perp) & \text{otherwise} \end{cases}$$

and, since $\alpha(\mathcal{P}_\perp) = 1$,

$$= \begin{cases} \alpha(\pi'(\mathcal{P}_i, w', c)) & \text{if } p(x_i)(c) = 1 \\ 1 & \text{otherwise} \end{cases}$$

$$= w(c)(x_i)$$

Thus, $w(c)(x_i) = \alpha(w'(c)(x_i))$ and we have the following:

$$\begin{aligned} \sigma(\hat{\alpha}(\mathcal{P}), w, c) &= \hat{\alpha}(\mathcal{P})[x_1/w(c)(x_1), \ldots, x_k/w(c)(x_k)] \\ &= \hat{\alpha}(\mathcal{P})[x_1/\alpha(w'(c)(x_1)), \ldots, x_k/\alpha(w'(c)(x_k))] \\ &= \alpha(\mathcal{P}[x_1/\alpha(w'(c)(x_1)), \ldots, x_k/\alpha(w'(c)(x_k))]) && \text{(Lemma 3)} \\ &= \alpha(\mathcal{P}[x_1/w'(c)(x_1), \ldots, x_k/w'(c)(x_k)]) && \text{(Corollary 1)} \\ &= \alpha(\pi'(\mathcal{P}, w', c)) && \text{(Definition 18)} \quad \square \end{aligned}$$

B.4. Lifting lemmas

This appendix covers details of lemmas related to lifting of expressions and of compositional evaluation factories.

Lemma 4 *(Soundness of expression lifting). If ε is a rational expression over Real constants and variables $x_i \in X$, $|X| = n$, $A_1, \ldots, A_n$ are ADDs, and $\hat{\varepsilon} = lift(\varepsilon)$, then*

$$\hat{\varepsilon}[x_1/A_1, \ldots, x_n/A_n](\bar{b}) = \varepsilon[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})]$$

where $\bar{b}$ is a vector of k Booleans, corresponding to a selection of the k features in a given product line.

Complete proof. The proof is by structural induction on the expression ε. The base cases are constant expressions and single variables:

- $\varepsilon = c$, where $c \in \mathbb{R}$:
  In this case, $\hat{\varepsilon} = \hat{c}$. Since ε has no variables (and neither has $\hat{\varepsilon}$), we apply the empty evaluation [ ]. Thus, $\hat{\varepsilon}[\,](\bar{b}) = \hat{c}(\bar{b}) = c = \varepsilon = \varepsilon[\,]$.
- $\varepsilon = x$:
  In this case, $\hat{\varepsilon} = x$. If A is an arbitrary ADD, then: $\hat{\varepsilon}[x/A](\bar{b}) = A(\bar{b}) = \varepsilon[x/A(\bar{b})]$.

Now we have to prove the statement holds for $\varepsilon = \varepsilon_1 \odot \varepsilon_2$ (where $\odot \in \{+, -, \times, \div\}$) and for $\varepsilon = \varepsilon_1^i$ (where $i \in \mathbb{N}$). As induction hypothesis, assume that the following holds for the expressions ε_1 and ε_2:

$$\hat{\varepsilon}[x_1/A_1, \ldots, x_n/A_n](\bar{b}) = \varepsilon[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})] \tag{I.H.}$$

Let $u : X \to (\mathbb{B}^k \to \mathbb{R})$ be a lifted evaluation such that $u(x_i) = A_i$ is an ADD. We then have the following:

- $\varepsilon = \varepsilon_1 \odot \varepsilon_2$, where $\odot \in \{+, -, \times, \div\}$:
  In this case, $\hat{\varepsilon} = \hat{\varepsilon_1} \odot \hat{\varepsilon_2}$. Hence,

$$\begin{aligned}
\hat{\varepsilon}[X/u](\bar{b}) &= \big(\hat{\varepsilon_1} \odot \hat{\varepsilon_2}\big)[X/u](\bar{b}) \\
&= \big(\hat{\varepsilon_1}[X/u] \odot \hat{\varepsilon_2}[X/u]\big)(\bar{b}) && \text{(evaluation)} \\
&= \hat{\varepsilon_1}[X/u](\bar{b}) \odot \hat{\varepsilon_2}[X/u](\bar{b}) && \text{(ADD arithmetics)} \\
&= \hat{\varepsilon_1}[x_1/A_1, \ldots, x_n/A_n](\bar{b}) \\
&\quad \odot \hat{\varepsilon_2}[x_1/A_1, \ldots, x_n/A_n](\bar{b}) && \text{(expanding } u\text{)} \\
&= \varepsilon_1[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})] \\
&\quad \odot \varepsilon_2[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})] && \text{(induction hypothesis)} \\
&= \big(\varepsilon_1 \odot \varepsilon_2\big)[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})] && \text{(evaluation)} \\
&= \varepsilon[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})]
\end{aligned}$$

- $\varepsilon = \varepsilon_1^i$, where $i \in \mathbb{N}$:
  In this case, $\hat{\varepsilon} = \hat{\varepsilon_1}^i$. Hence,

$$\begin{aligned}
\hat{\varepsilon}[X/u](\bar{b}) &= \hat{\varepsilon_1}^i[X/u](\bar{b}) \\
&= \hat{\varepsilon_1}[X/u]^i(\bar{b}) && \text{(evaluation)} \\
&= \hat{\varepsilon_1}[X/u](\bar{b})^i && \text{(ADD arithmetics)} \\
&= \hat{\varepsilon_1}[x_1/A_1, \ldots, x_n/A_n](\bar{b})^i && \text{(expanding } u\text{)} \\
&= \varepsilon_1[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})]^i && \text{(induction hypothesis)} \\
&= \varepsilon_1^i[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})] && \text{(evaluation)} \\
&= \varepsilon[x_1/A_1(\bar{b}), \ldots, x_n/A_n(\bar{b})] && \square
\end{aligned}$$

The soundness of lifted compositional evaluation factories is now presented in its complete form. First, we recall the corresponding lemma's statement.

Lemma 6 *(Soundness of lifted compositional evaluation factory). Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and the compositional evaluation factory w, derived from the composition factory w' (Definition 25), for all $x \in I$ and all $c \in [\![FM]\!]$ it holds that*

$$\varphi(x)(c) = w(c)(x)$$

Complete proof. If $\mathcal{P} \in \mathscr{P}$ is such that $idt(\mathcal{P}) = x$, then

$$\begin{aligned}\varphi(x)(c) &= \texttt{ITE}(\hat{p}(x), \widehat{\hat{\alpha}(\mathcal{P})}[X/\varphi], \hat{\mathbf{1}})(c)\\ &= \begin{cases}\widehat{\hat{\alpha}(\mathcal{P})}[X/\varphi](c) & \text{if } \hat{p}(x)(c) \neq 0\\ \hat{\mathbf{1}}(c) & \text{if } \hat{p}(x)(c) = 0\end{cases}\end{aligned}$$

By Lemma 4, $\widehat{\hat{\alpha}(\mathcal{P})}[X/\varphi](c) = \hat{\alpha}(\mathcal{P})[x_1/\varphi(x_1)(c), \ldots, x_k/\varphi(x_k)(c)]$. Also, $\forall_{c \in [\![FM]\!]} \cdot \hat{\mathbf{1}}(c) = 1$. Thus,

$$\varphi(x)(c) = \begin{cases}\hat{\alpha}(\mathcal{P})[x_1/\varphi(x_1)(c), \ldots, x_k/\varphi(x_k)(c)] & \text{if } \hat{p}(x)(c) \neq 0\\ 1 & \text{if } \hat{p}(x)(c) = 0\end{cases} \tag{2}$$

On the other hand, w is defined (Definition 25) as

$$w(c)(x) = \begin{cases}[\![\hat{\alpha}(\mathcal{P})]\!]_c^w & \text{if } p(x)(c) \neq 0\\ 1 & \text{if } p(x)(c) = 0\end{cases}$$

Expanding the definition of $[\![\hat{\alpha}(\mathcal{P})]\!]_c^w$, we have

$$w(c)(x) = \begin{cases}\hat{\alpha}(\mathcal{P})[x_1/w(c)(x_1), \ldots, x_k/w(c)(x_k)] & \text{if } p(x)(c) \neq 0\\ 1 & \text{if } p(x)(c) = 0\end{cases} \tag{3}$$

Since $\hat{p}(x)(c) = p(x)(c)$, we compare corresponding cases in the Equations (2), (3). The cases in which $p(x)(c) = 0$ are trivially equal. Otherwise, we use well-founded induction.

The base of our induction are minimal PMCs. A minimal PMC $\mathcal{P}$ has no variables ($X = \emptyset$), so $\hat{\alpha}(\mathcal{P})[X/u] = \alpha(\mathcal{P})$ for any evaluation u. Since $w(c)$ is an evaluation, and considering $\varphi(x)(c)$ takes a variable x to a Real number (thus, also being an evaluation), we have that $\widehat{\hat{\alpha}(\mathcal{P})}[X/\varphi](c) = \hat{\alpha}(\mathcal{P})[X/w(c)]$ in this case. For non-minimal PMCs, assume, as induction hypothesis, that $\widehat{\hat{\alpha}(\mathcal{P}_j)}[X_j/\varphi](c) = \hat{\alpha}(\mathcal{P}_j)[X_j/w(c)]$ for all $\mathcal{P}_j \prec \mathcal{P}$, where $j \in \{1, \ldots, k\}$. Then, for any $x_j \in X$,

$$\varphi(x_j)(c) = \begin{cases}\widehat{\hat{\alpha}(\mathcal{P}_j)}[X_j/\varphi](c) & \text{if } \hat{p}(x_j)(c) \neq 0\\ 1 & \text{if } \hat{p}(x_j)(c) = 0\end{cases} \tag{4}$$

$$w(c)(x_j) = \begin{cases}\hat{\alpha}(\mathcal{P}_j)[X_j/w(c)] & \text{if } p(x_j)(c) \neq 0\\ 1 & \text{if } p(x_j)(c) = 0\end{cases} \tag{5}$$

However, the induction hypothesis implies the right-hand sides of the Equations (4), (5) are equal. Thus, $\varphi(x_j)(c) = w(c)(x_j)$ for all $x_j \in X$, which means

$$\hat{\alpha}(\mathcal{P})[x_1/\varphi(x_1)(c), \ldots, x_k/\varphi(x_k)(c)] = \hat{\alpha}(\mathcal{P})[x_1/w(c)(x_1), \ldots, x_k/w(c)(x_k)]$$

and, by well-founded induction, the cases where $p(x)(c) = 1$ in the Equations (2), (3) are also equal. Hence, $\varphi(x)(c) = w(c)(x)$. □

B.5. Variability encoding

This appendix deals with formal definitions and complete proofs related to variability encoding of PMCs and of rational expressions.

B.5.1. Variability encoding of PMCs

We start by formally defining the ITE operator for PMCs, which was only presented as an intuition in the main body of the paper.

Definition 34 *(ITE operator for PMCs).* Given two compositional PMCs, $\mathcal{P} = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$ and $\mathcal{P}' = (S', s'_0, s'_{suc}, s'_{err}, X', \mathbf{P}', T')$, and a variable $x \notin X \cup X'$, the *if-then-else* operator for PMCs is defined as

$$\texttt{ITE}(x, \mathcal{P}, \mathcal{P}') = \mathcal{P}''$$

where $\mathcal{P}'' = (S'', s''_0, s''_{suc}, s''_{err}, X'', \mathbf{P}'', T'')$ is a compositional PMC such that:

- $S'' = S \cup S' \cup \{s''_0, s''_{suc}, s''_{err}\}$
- The state s''_0 is the new initial one, s''_{suc} is the new success state, and s''_{err} is the new error state.
- $X'' = X \cup X' \cup \{x\}$
- $T'' = \{s''_{suc}\}$
- $\mathbf{P}''$ is such that:
  - $\mathbf{P}''(s''_0, s_0) = x$
  - $\mathbf{P}''(s''_0, s'_0) = 1 - x$
  - $\mathbf{P}''(s_{suc}, s''_{suc}) = \mathbf{P}''(s'_{suc}, s''_{suc}) = \mathbf{P}''(s''_{suc}, s''_{suc}) = 1$
  - $\mathbf{P}''(s_{suc}, s_{suc}) = \mathbf{P}''(s'_{suc}, s'_{suc}) = 0$
  - $\mathbf{P}''(s_{err}, s''_{err}) = \mathbf{P}''(s'_{err}, s''_{err}) = \mathbf{P}''(s''_{err}, s''_{err}) = 1$
  - $\mathbf{P}''(s_{err}, s_{err}) = \mathbf{P}''(s'_{err}, s'_{err}) = 0$
  - For all remaining combinations of $s_1, s_2 \in S''$:

$$\mathbf{P}''(s_1, s_2) = \begin{cases} \mathbf{P}(s_1, s_2) & \text{if } s_1, s_2 \in S \\ \mathbf{P}'(s_1, s_2) & \text{if } s_1, s_2 \in S' \\ 0 & \text{otherwise} \end{cases}$$

This ITE operator is mainly useful because of its r-equivalence property. We recall Lemma 7 and present its complete proof:

Lemma 7 *(R-equivalence for ITE). Given two compositional PMCs, $\mathcal{P} = (S, s_0, s_{suc}, s_{err}, X, \mathbf{P}, T)$ and $\mathcal{P}' = (S', s'_0, s'_{suc}, s'_{err}, X', \mathbf{P}', T')$, and a variable $x \notin X \cup X'$, let $\mathcal{P}'' = \mathtt{ITE}(x, \mathcal{P}, \mathcal{P}')$. If $(\mathcal{P}'', p, w, FM)$ is an annotative model with $\mathcal{P}''$ as its underlying PMC,[10] where p, w, and FM are arbitrarily chosen, then, for every $c \in [\![FM]\!]$,*

$$\alpha([\![\mathtt{ITE}(x, \mathcal{P}, \mathcal{P}')]\!]^w_c) = \begin{cases} \alpha([\![\mathcal{P}]\!]^w_c) & \text{if } p(x)(c) = 1 \\ \alpha([\![\mathcal{P}']\!]^w_c) & \text{otherwise} \end{cases}$$

Complete proof. We are interested in computing the probability of reaching s''_{suc} from s''_0 in $\mathcal{P}'' = \mathtt{ITE}(x, \mathcal{P}, \mathcal{P}')$ under evaluation $w(c)$. In $\mathcal{P}''$, $s''_0 \neq s''_{suc}$ and s''_{suc} is reachable from s''_0 (since s''_{suc} is, by definition, reachable from s_{suc} and s'_{suc}). Hence, the reachability of s''_{suc} from s''_0 satisfies Definition 1, by which the probability of reaching state s_2 from state s_1 in a DTMC $\mathcal{D} = (S, s_0, \mathbf{P}, T)$ is given by

$$Pr^{\mathcal{D}}(s_1, s_2) = \sum_{s' \in S \setminus \{s_2\}} \mathbf{P}(s_1, s') \cdot Pr^{\mathcal{D}}(s', s_2) + \mathbf{P}(s_1, s_2)$$

By Definition 34, $\mathbf{P}''(s''_0, s_0) = x$, $\mathbf{P}''(s''_0, s'_0) = 1 - x$, and $\mathbf{P}''(s''_0, s'') = 0$ for all other $s'' \in S''$. Thus,

$$\begin{aligned} \alpha([\![\mathcal{P}'']\!]^w_c) &= Pr^{\mathcal{P}''_{w(c)}}(s''_0, s''_{suc}) \\ &= \sum_{s'' \in S'' \setminus \{s''_{suc}\}} \mathbf{P}''_{w(c)}(s''_0, s'') \cdot Pr^{\mathcal{P}''_{w(c)}}(s'', s''_{suc}) + \mathbf{P}''_{w(c)}(s''_0, s''_{suc}) \\ &= \sum_{s'' \in S'' \setminus \{s''_{suc}\}} \mathbf{P}''_{w(c)}(s''_0, s'') \cdot Pr^{\mathcal{P}''_{w(c)}}(s'', s''_{suc}) + 0 \\ &= \mathbf{P}''_{w(c)}(s''_0, s_0) \cdot Pr^{\mathcal{P}''_{w(c)}}(s_0, s''_{suc}) + \mathbf{P}''_{w(c)}(s''_0, s'_0) \cdot Pr^{\mathcal{P}''_{w(c)}}(s'_0, s''_{suc}) \\ &= w(c)(x) \cdot Pr^{\mathcal{P}''_{w(c)}}(s_0, s''_{suc}) + (1 - w(c)(x)) \cdot Pr^{\mathcal{P}''_{w(c)}}(s'_0, s''_{suc}) \end{aligned}$$

Since $w(c)(x)$ equals 1 if $p(x)(c) = 1$ and 0 otherwise (Definition 7),

$$\alpha([\![\mathcal{P}'']\!]^w_c) = \begin{cases} Pr^{\mathcal{P}''_{w(c)}}(s_0, s''_{suc}) & \text{if } p(x)(c) = 1 \\ Pr^{\mathcal{P}''_{w(c)}}(s'_0, s''_{suc}) & \text{otherwise} \end{cases}$$

But, since $s_0 \in S$ and the only state in S that can reach s''_{suc} is s_{suc} (Definition 34), the probability of reaching s''_{suc} from s_0 is the probability of reaching s_{suc} from s_0 multiplied by the transition probability from s_{suc} to s''_{suc}:

[10] By Definition 9, any compositional PMC is also an annotative PMC (Definition 4). Thus, a compositional PMC can be the underlying PMC of an annotative model.

$$\begin{aligned} Pr^{\mathcal{P}''_{w(c)}}(s_0, s''_{suc}) &= Pr^{\mathcal{P}''_{w(c)}}(s_0, s_{suc}) \cdot \mathbf{P}''_{w(c)}(s_{suc}, s''_{suc}) \\ &= Pr^{\mathcal{P}_{w(c)}}(s_0, s_{suc}) \cdot 1 \\ &= Pr^{\mathcal{P}_{w(c)}}(s_0, s_{suc}) \\ &= \alpha([\![\mathcal{P}]\!]_c^w) \end{aligned}$$

Similar reasoning applied to S' leads to $Pr^{\mathcal{P}''_{w(c)}}(s'_0, s''_{suc}) = \alpha([\![\mathcal{P}']\!]_c)$. Hence,

$$\alpha([\![\mathcal{P}'']\!]_c^w) = \begin{cases} \alpha([\![\mathcal{P}]\!]_c^w) & \text{if } p(x)(c) = 1 \\ \alpha([\![\mathcal{P}']\!]_c^w) & \text{otherwise} \end{cases} \quad \square$$

The above lemma establishes the `ITE` operator has the effect of alternating behaviors if the resulting PMC is evaluated by replacing the switching variable x with 0 or 1. However, the PMC operands of `ITE` are part of a compositional model, so their own variables are interpreted as placeholders to be used during composition, instead (see Section 3.2). To cope with this mismatch, we only use the `ITE` operator with PMCs that are either plain DTMCs or that result themselves from variability encoding.

The resulting theorem stating the soundness of this variability encoding for PMCs is recalled and proved next.

Theorem 8 *(R-equivalence of variability encoding and derivation by composition). Given a compositional model* $(\mathscr{P}, \prec, I, idt, p, w', FM)$ *and* $\mathcal{P} \in \mathscr{P}$*, let* $(\gamma(\mathcal{P}), p, w, FM)$ *be its variability-encoded annotative model. Then, for all* $c \in [\![FM]\!]$*,*

$$\alpha([\![\gamma(\mathcal{P})]\!]_c^w) = \alpha(\pi'(\mathcal{P}, w', c))$$

Complete proof. We use well-founded induction. For minimal PMCs (base of induction), $\gamma(\mathcal{P}) = \mathcal{P}$, so $[\![\gamma(\mathcal{P})]\!]_c^w = \mathcal{P}$. Likewise, $\pi'(\mathcal{P}, w', c) = \mathcal{P}$, so the proposition holds trivially.

As induction hypothesis, we have that $\alpha([\![\gamma(\mathcal{P}_i)]\!]_c^w) = \alpha(\pi'(\mathcal{P}_i, w', c))$ for all $\mathcal{P}_i \in \mathscr{P}$ such that $\mathcal{P}_i \prec \mathcal{P}$. For brevity, in the following equations, we use Λ_i to denote $\texttt{ITE}(x_i, \gamma(\mathcal{P}_i), \mathcal{P}_\perp)$.

$$\begin{aligned} \alpha([\![\gamma(\mathcal{P})]\!]_c^w) &= [\![\hat{\alpha}(\gamma(\mathcal{P}))]\!]_c^w && \text{(Theorem 1)} \\ &= [\![\hat{\alpha}(\mathcal{P}[x_1/\Lambda_1, \ldots, x_k/\Lambda_k])]\!]_c^w && \text{(Definition 27)} \\ &= [\![\hat{\alpha}(\mathcal{P}[x_1/\hat{\alpha}(\Lambda_1), \ldots, x_k/\hat{\alpha}(\Lambda_k)])]\!]_c^w && \text{(Lemma 13)} \\ &= [\![\hat{\alpha}(\mathcal{P})[x_1/\hat{\alpha}(\Lambda_1), \ldots, x_k/\hat{\alpha}(\Lambda_k)]]\!]_c^w && \text{(Lemma 3)} \\ &= \hat{\alpha}(\mathcal{P})[x_1/\hat{\alpha}(\Lambda_1), \ldots, x_k/\hat{\alpha}(\Lambda_k)][X/w(c)] && \text{(Definition 21)} \\ &= \hat{\alpha}(\mathcal{P})[x_1/\hat{\alpha}(\Lambda_1)[X/w(c)], \ldots, \\ &\qquad \ldots, x_k/\hat{\alpha}(\Lambda_k)[X/w(c)]] && \text{(Equation (1))} \\ &= \hat{\alpha}(\mathcal{P})[x_1/[\![\hat{\alpha}(\Lambda_1)]\!]_c^w, \ldots, x_k/[\![\hat{\alpha}(\Lambda_k)]\!]_c^w] && \text{(Definition 21)} \\ &= \hat{\alpha}(\mathcal{P})[x_1/\alpha([\![\Lambda_1]\!]_c^w), \ldots, x_k/\alpha([\![\Lambda_k]\!]_c^w)] && \text{(Theorem 1)} \\ &= \alpha(\mathcal{P}[x_1/\alpha([\![\Lambda_1]\!]_c^w), \ldots, x_k/\alpha([\![\Lambda_k]\!]_c^w)]) && \text{(Lemma 3)} \end{aligned}$$

leaving us with the following partial result:

$$\alpha([\![\gamma(\mathcal{P})]\!]_c^w) = \alpha(\mathcal{P}[x_1/\alpha([\![\Lambda_1]\!]_c^w), \ldots, x_k/\alpha([\![\Lambda_k]\!]_c^w)]) \tag{6}$$

Each variable substitution expands to two different cases, corresponding to whether c satisfies the presence condition associated with x_i or not. Let us examine the substitution for a given x_i:

$$\begin{aligned} \alpha([\![\Lambda_i]\!]_c^w) &= \alpha([\![\texttt{ITE}(x_i, \gamma(\mathcal{P}_i), \mathcal{P}_\perp)]\!]_c^w) \\ &= \begin{cases} \alpha([\![\gamma(\mathcal{P}_i)]\!]_c^w) & \text{if } p(x_i)(c) = 1 \\ \alpha([\![\mathcal{P}_\perp]\!]_c^w) & \text{otherwise} \end{cases} && \text{(Lemma 7)} \\ &= \begin{cases} \alpha(\pi'(\mathcal{P}_i, w', c)) & \text{if } p(x_i)(c) = 1 \\ \alpha([\![\mathcal{P}_\perp]\!]_c^{w'}) & \text{otherwise} \end{cases} && \text{(by induction hypothesis)} \\ &= \alpha(w'(c)(x_i)) && \text{(Definitions 17 and 18)} \end{aligned}$$

that is,

$$\alpha([\![\Lambda_i]\!]_c^w) = \alpha(w'(c)(x_i)) \tag{7}$$

Hence, we can substitute Equation (7) into Equation (6):

$$\begin{aligned}
\alpha([\![\gamma(\mathcal{P})]\!]_c^w) &= \alpha(\mathcal{P}[x_1/\alpha([\![\Lambda_1]\!]_c^w), \ldots, x_k/\alpha([\![\Lambda_k]\!]_c^w)]) && \text{(Equation (6))}\\
&= \alpha(\mathcal{P}[x_1/\alpha(w'(c)(x_1)), \ldots, x_k/\alpha(w'(c)(x_k))]) && \text{(Equation (7))}\\
&= \alpha(\mathcal{P}[x_1/w'(c)(x_1), \ldots, x_k/w'(c)(x_k)]) && \text{(Corollary 1)}\\
&= \alpha(\pi'(\mathcal{P}, w', c)) && \text{(Definition 18)}\ \square
\end{aligned}$$

B.5.2. Variability encoding of expressions

We start by proving that the ITE operator for expressions has the intended semantics. This result is expressed by Lemma 8, which we now recall.

Lemma 8 *(Extensional equality for expression ITE). Given two expressions ε and ε' over the sets X and X' of variables, respectively, and a variable x, let $X'' = X \cup X' \cup \{x\}$ and $u : X'' \to [0, 1]$ be an evaluation function such that $u(x) \in \mathbb{B}$. Then,*

$$\mathtt{ITE}(x, \varepsilon, \varepsilon')[X''/u] = \begin{cases} \varepsilon[X/u] & \text{if } u(x) = 1 \\ \varepsilon'[X'/u] & \text{if } u(x) = 0 \end{cases}$$

Complete proof. The proof is mainly algebraic. Expanding the definition of ITE, we have:

$$\begin{aligned}
\mathtt{ITE}(x, \varepsilon, \varepsilon')[X''/u] &= (x \cdot \varepsilon + (1 - x) \cdot \varepsilon')[X''/u]\\
&= (x \cdot \varepsilon)[X''/u] + ((1 - x) \cdot \varepsilon')[X''/u]\\
&= x[X''/u] \cdot \varepsilon[X''/u] + (1 - x)[X''/u] \cdot \varepsilon'[X''/u]\\
&= u(x) \cdot \varepsilon[X''/u] + (1 - u(x)) \cdot \varepsilon'[X''/u]\\
&= \begin{cases} \varepsilon[X''/u] & \text{if } u(x) = 1 \\ \varepsilon'[X''/u] & \text{if } u(x) = 0 \end{cases}
\end{aligned}$$

which, considering that the sets of variables in ε and ε' are X and X', respectively, and that these sets are subsets of X'', leads to

$$\mathtt{ITE}(x, \varepsilon, \varepsilon')[X''/u] = \begin{cases} \varepsilon[X/u] & \text{if } u(x) = 1 \\ \varepsilon'[X'/u] & \text{if } u(x) = 0 \end{cases} \quad \square$$

Using this result and the definitions in the main body of the paper, we can prove that variability encoding for expressions is sound.

Theorem 9 *(Soundness of variability encoding for expressions). Given a compositional model $(\mathscr{P}, \prec, I, idt, p, w', FM)$ and $\mathcal{P}, \mathcal{P}_1, \ldots, \mathcal{P}_k \in \mathscr{P}$ such that $\mathcal{P}_i \prec \mathcal{P}$ and $x_i = idt(\mathcal{P}_i)$ for $i \in \{1, \ldots, k\}$, let $\varepsilon = \hat{\alpha}(\mathcal{P})$. Let also w be the compositional evaluation factory derived from w' (Definition 25) and w_p be the annotative evaluation factory obtained from w (Definition 31). Then, for all $c \in [\![FM]\!]$ it holds that*

$$\sigma(\gamma(\varepsilon), w_p, c) = \sigma(\varepsilon, w, c)$$

Complete proof. We use well-founded induction. For a minimal PMC $\mathcal{P}$ (base of induction), $\hat{\alpha}(\mathcal{P}) = \varepsilon$ has no variables. This way, $\gamma(\varepsilon) = \varepsilon$ and $\sigma(\varepsilon, u) = \varepsilon$ for any evaluation u. Thus, both sides of the equality evaluate to ε and the proposition holds trivially.

As induction hypothesis, we have that $\sigma(\gamma(\varepsilon_i), w_p, c) = \sigma(\varepsilon_i, w, c)$ for all $\varepsilon_i = \hat{\alpha}(\mathcal{P}_i)$ such that $\mathcal{P}_i \prec \mathcal{P}$. For brevity, we use Λ_i to denote $\mathtt{ITE}(x_i, \gamma(\varepsilon_i), \mathbf{1})$ in the following equations.

$$\begin{aligned}
\sigma(\gamma(\varepsilon), w_p, c) &= \sigma(\varepsilon[x_1/\Lambda_1, \ldots, x_k/\Lambda_k], w_p, c) && \text{(Definition 30)}\\
&= \varepsilon[x_1/\Lambda_1, \ldots, x_k/\Lambda_k][X/w_p(c)] && \text{(Definition 21)}\\
&= \varepsilon[x_1/\Lambda_1[X/w_p(c)], \ldots, x_k/\Lambda_k[X/w_p(c)]] && \text{(Equation (1))}
\end{aligned}$$

yielding the following equation:

$$\sigma(\gamma(\varepsilon), w_p, c) = \varepsilon[x_1/\Lambda_1[X/w_p(c)], \ldots, x_k/\Lambda_k[X/w_p(c)]] \tag{8}$$

Each variable substitution expands to two different cases, corresponding to whether c satisfies the presence condition associated with x_i or not. Let us examine the substitution for a given x_i:

$$\begin{aligned}
\Lambda_i[X/w_p(c)] &= \mathtt{ITE}(x_i, \gamma(\varepsilon_i), \mathbf{1})[X/w_p(c)] \\
&= \begin{cases} \gamma(\varepsilon_i)[X/w_p(c)] & \text{if } p(x_i)(c) = 1 \ (w_p(c)(x_i) = 1) \quad \text{(Lemma 8)} \\ \mathbf{1}[X/w_p(c)] & \text{otherwise } (w_p(c)(x_i) = 0) \end{cases} \\
&= \begin{cases} \sigma(\gamma(\varepsilon_i), w_p, c) & \text{if } p(x_i)(c) = 1 \quad \text{(Definition 21)} \\ 1 & \text{otherwise} \end{cases} \\
&= \begin{cases} \sigma(\varepsilon_i, w, c) & \text{if } p(x_i)(c) = 1 \quad \text{(by induction hypothesis)} \\ 1 & \text{otherwise} \end{cases} \\
&= w(c)(x_i) \quad \text{(Definition 25)}
\end{aligned}$$

that is,

$$\Lambda_i[X/w_p(c)] = w(c)(x_i) \tag{9}$$

Hence, substituting Equation (9) into Equation (8), we have

$$\begin{aligned}
\sigma(\gamma(\varepsilon), w_p, c) &= \varepsilon[x_1/\Lambda_1[X/w_p(c)], \ldots, x_k/\Lambda_k[X/w_p(c)]] \quad \text{(Equation (8))} \\
&= \varepsilon[x_1/w(c)(x_1), \ldots, x_k/w(c)(x_k)] \quad \text{(Equation (9))} \\
&= \varepsilon[X/w(c)] \\
&= \sigma(\varepsilon, w, c) \quad \text{(Definition 21)} \ \square
\end{aligned}$$

References

[1] S. Apel, D.S. Batory, C. Kästner, G. Saake, Feature-Oriented Software Product Lines – Concepts and Implementation, Springer, 2013.
[2] S. Apel, H. Speidel, P. Wendler, A. von Rhein, D. Beyer, Detection of feature interactions using feature-aware verification, in: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE Computer Society, 2011, pp. 372–375.
[3] S. Apel, A. Von Rhein, P. Wendler, A. Groslinger, D. Beyer, Strategies for product-line verification: case studies and experiments, in: Proceedings of the International Conference on Software Engineering, ICSE, IEEE Press, 2013, pp. 482–491.
[4] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, F. Somenzi, Algebraic decision diagrams and their applications, Form. Methods Syst. Des. 10 (1997) 171–206, https://doi.org/10.1023/A:1008699807402.
[5] C. Baier, J.P. Katoen, Principles of Model Checking, Representation and Mind Series, The MIT Press, 2008.
[6] E. Bodden, T. Tolêdo, M. Ribeiro, C. Brabrand, P. Borba, M. Mezini, SPL^{LIFT}: statically analyzing software product lines in minutes instead of years, in: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI, 2013, pp. 355–364.
[7] C. Brabrand, M. Ribeiro, T. Tolêdo, J. Winther, P. Borba, Intraprocedural dataflow analysis for software product lines, in: Transactions on Aspect-Oriented Software Development X, Springer, 2013, pp. 73–108.
[8] S. Chen, M. Erwig, Type-based parametric analysis of program families, ACM SIGPLAN Not. 49 (2014) 39–51, https://doi.org/10.1145/2692915.2628155.
[9] P. Chrszon, C. Dubslaff, S. Klüppelholz, C. Baier, Family-based modeling and analysis for probabilistic systems – featuring ProFeat, in: Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering, FASE, Springer, 2016, pp. 287–304.
[10] E.M. Clarke, E.A. Emerson, Design and synthesis of synchronization skeletons using branching-time temporal logic, in: Logic of Programs, Workshop, Springer, 1982, pp. 52–71.
[11] A. Classen, M. Cordy, P. Heymans, A. Legay, P.Y. Schobbens, Formal semantics, modular specification, and symbolic verification of product-line behaviour, Sci. Comput. Program. 80 (Part B) (2014) 416–439, https://doi.org/10.1016/j.scico.2013.09.019.
[12] A. Classen, M. Cordy, P.Y. Schobbens, P. Heymans, A. Legay, J.F. Raskin, Featured transition systems: foundations for verifying variability-intensive systems and their application to LTL model checking, IEEE Trans. Softw. Eng. 39 (2013) 1069–1089, https://doi.org/10.1109/TSE.2012.86.
[13] A. Classen, P. Heymans, P. Schobbens, A. Legay, Symbolic model checking of software product lines, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE, ACM, 2011, pp. 321–330.
[14] A. Classen, P. Heymans, P.Y. Schobbens, A. Legay, J.F. Raskin, Model checking lots of systems, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ICSE, ACM Press, 2010, p. 335.
[15] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison–Wesley Professional, 2001.
[16] K. Czarnecki, U.W. Eisenecker, Generative Programming: Methods, Tools, and Applications, ACM Press/Addison–Wesley Publishing Co., 2000.
[17] K. Czarnecki, K. Pietroszek, Verifying feature-based model templates against well-formedness OCL constraints, in: Proceedings of the 5th International Conference on Generative Programming and Component Engineering, GPCE, ACM, 2006, pp. 211–220.
[18] C. Daws, Symbolic and parametric model checking of discrete-time Markov chains, in: Proceedings of the First International Conference on Theoretical Aspects of Computing, ICTAC, Springer, 2005, pp. 280–294.
[19] D. Domis, R. Adler, M. Becker, Integrating variability and safety analysis models using commercial UML-based tools, in: Proceedings of the 19th International Software Product Line Conference, SPLC, ACM, 2015, pp. 225–234.
[20] F. Dordowsky, R. Bridges, H. Tschöpe, Implementing a software product line for a complex avionics system, in: Proceedings of the 15th International Conference on Software Product Lines, SPLC, IEEE, 2011, pp. 241–250.
[21] C. Dubslaff, C. Baier, S. Kluppelholz, Probabilistic model checking for feature-oriented systems, in: Transactions on Aspect-Oriented Software Development XII, in: Lect. Notes Comput. Sci., vol. 8989, Springer, 2015, pp. 180–220.
[22] C. Ghezzi, A. Molzam Sharifloo, Model-based verification of quantitative non-functional properties for software product lines, Inf. Softw. Technol. 55 (2013) 508–524, https://doi.org/10.1016/j.infsof.2012.07.017.

[23] L. Grunske, Specification patterns for probabilistic quality properties, in: Proceedings of the International Conference on Software Engineering, ICSE, ACM, 2008, pp. 31–40.
[24] A. Haber, K. Hölldobler, C. Kolassa, M. Look, B. Rumpe, K. Müller, I. Schaefer, Engineering delta modeling languages, in: Proceedings of the 17th International Software Product Line Conference on, SPLC, ACM Press, 2013, p. 22.
[25] E.M. Hahn, H. Hermanns, B. Wachter, L. Zhang, Param: a model checker for parametric Markov models, in: Proceedings of the 22nd International Conference on Computer Aided Verification, CAV, Springer, 2010, pp. 660–664.
[26] E.M. Hahn, H. Hermanns, L. Zhang, Probabilistic reachability for parametric Markov models, Int. J. Softw. Tools Technol. Transf. 13 (2011) 3–19, https://doi.org/10.1007/s10009-010-0146-x.
[27] H. Hansson, B. Jonsson, A logic for reasoning about time and reliability, Form. Asp. Comput. 6 (1994) 512–535, https://doi.org/10.1007/BF01211866.
[28] R. Heradio, H. Perez-Morago, D. Fernandez-Amoros, F.C. Javier, E. Herrera-Viedma, A bibliometric analysis of 20 years of research on software product lines, Inf. Softw. Technol. 72 (2016) 1–15, https://doi.org/10.1016/j.infsof.2015.11.004.
[29] C. Kästner, S. Apel, M. Kuhlemann, Granularity in software product lines, in: Proceedings of the 13th International Conference on Software Engineering, ICSE, ACM Press, 2008, p. 311.
[30] S. Kolesnikov, A. von Rhein, C. Hunsen, S. Apel, A comparison of product-based, feature-based, and family-based type checking, in: Proceedings of the 12th International Conference on Generative Programming, GPCE, ACM, 2013, pp. 115–124.
[31] M. Kowal, I. Schaefer, M. Tribastone, Family-based performance analysis of variant-rich software systems, in: Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering, vol. 8411, Springer, 2014, pp. 94–108.
[32] M. Kowal, M. Tschaikowski, M. Tribastone, I. Schaefer, Scaling size and parameter spaces in variability-aware software performance models, in: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, ASE, 2015, pp. 407–417.
[33] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: verification of probabilistic real-time systems, in: Proceedings of the 23rd International Conference on Computer Aided Verification, CAV, Springer, 2011, pp. 585–591.
[34] J.T. Lanman, R. Darbin, J. Rivera, P.C. Clements, C.W. Krueger, The challenges of applying service orientation to the U.S. Army's live training software product line, in: Proceedings of the 17th International Software Product Line Conference, SPLC, ACM, 2013, pp. 244–253.
[35] H.C. Li, S. Krishnamurthi, K. Fisler, Modular verification of open features using three-valued model checking, Autom. Softw. Eng. 12 (2005) 349–382, https://doi.org/10.1007/s10515-005-2643-9.
[36] F.J. van der Linden, K. Schmid, E. Rommes, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, Springer, 2007.
[37] J. Liu, S. Basu, R.R. Lutz, Compositional model checking of software product lines using variation point obligations, Autom. Softw. Eng. 18 (2010) 39–76, https://doi.org/10.1007/s10515-010-0075-7.
[38] J. Meinicke, T. Thüm, R. Schröter, F. Benduhn, G. Saake, An overview on analysis tools for software product lines, in: Proceedings of the 18th International Software Product Line Conference, SPLC, ACM Press, 2014, pp. 94–101.
[39] J. Midtgaard, A.S. Dimovski, C. Brabrand, A. Wąsowski, Systematic derivation of correct variability-aware program analyses, Sci. Comput. Program. 105 (2015) 145–170, https://doi.org/10.1016/j.scico.2015.04.005.
[40] V. Nunes, P. Fernandes, V. Alves, G. Rodrigues, Variability management of reliability models in software product lines: an expressiveness and scalability analysis, in: Proceedings of the Sixth Brazilian Symposium on Software Components Architectures and Reuse, SBCARS, 2012, pp. 51–60.
[41] V. Nunes, D. Mendonça, G. Rodrigues, V. Alves, Towards compositional approach for parametric model checking in software product lines, in: Proceedings of the International Workshop on Architecting Dependable Systems, WDAS, SBC, 2013.
[42] K. Pohl, G. Böckle, F.J. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer, 2005.
[43] H. Post, C. Sinz, Configuration lifting: verification meets software configuration, in: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE Computer Society, 2008, pp. 347–350.
[44] A. von Rhein, S. Apel, C. Kästner, T. Thüm, I. Schaefer, The PLA model, in: Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS, ACM Press, 2013, p. 1.
[45] A. von Rhein, T. Thüm, I. Schaefer, J. Liebig, S. Apel, Variability encoding: from compile-time to load-time variability, J. Log. Algebraic Methods Program. 85 (2016) 125–145, https://doi.org/10.1016/j.jlamp.2015.06.007.
[46] G.N. Rodrigues, V. Alves, V. Nunes, A. Lanna, M. Cordy, P. Schobbens, A.M. Sharifloo, A. Legay, Modeling and verification for probabilistic properties in software product lines, in: Proceedings of the 16th IEEE International Symposium on High Assurance Systems Engineering, HASE, IEEE Computer Society, 2015, pp. 173–180.
[47] T. Thüm, S. Apel, C. Kästner, I. Schaefer, G. Saake, A classification and survey of analysis strategies for software product lines, ACM Comput. Surv. 47 (2014) 1–45, https://doi.org/10.1145/2580950.
[48] T. Thüm, I. Schaefer, S. Apel, M. Hentschel, Family-based deductive verification of software product lines, ACM SIGPLAN Not. 48 (2013) 11–20, https://doi.org/10.1145/2480361.2371404.
[49] E. Walkingshaw, C. Kästner, M. Erwig, S. Apel, E. Bodden, Variational data structures, in: Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward!), ACM Press, 2014, pp. 213–226.
[50] D.M. Weiss, The product line hall of fame, in: Proceedings of the 12th International Software Product Line Conference, SPLC, IEEE Computer Society, 2008, p. 395.