Family-Based Model Checking with mCRL2

Maurice H. ter Beek[1], Erik P. de Vink[2], and Tim A. C. Willemse[2]

[1] ISTI–CNR, Pisa, Italy
[2] TU/e, Eindhoven, The Netherlands

Abstract. Family-based model checking targets the simultaneous verification of multiple system variants, a technique to handle feature-based variability that is intrinsic to software product lines (SPLs). We present an approach for family-based verification based on the feature μ-calculus μL_f, which combines modalities with feature expressions. This logic is interpreted over featured transition systems, a well-accepted model of SPLs, which allows one to reason over the collective behavior of a number of variants (a family of products). Via an embedding into the modal μ-calculus with data, underpinned by the general-purpose `mCRL2` toolset, off-the-shelf tool support for μL_f becomes readily available. We illustrate the feasibility of our approach on an SPL benchmark model and show the runtime improvement that family-based model checking with `mCRL2` offers with respect to model checking the benchmark product-by-product.

1 Introduction

Many software systems are configurable systems whose variants differ by the *features* they provide, i.e. the functionality that is relevant for an end-user, and are therefore referred to as software product lines (SPLs) or *product families*. SPLs challenge existing formal methods and analysis tools by the potentially high number of different products, each giving rise to a large behavioral state space in general. SPLs are popular in the embedded and critical systems domain. Therefore, analysis techniques for proving the correctness of SPL models are widely studied (cf. [1] for a survey).

Because for larger SPL models enumerative product-by-product approaches become unfeasible, dedicated *family-based* techniques have been developed, exploiting variability in product families in terms of features (cf., e.g., [2–8]). In this paper, we contribute to the field of *family-based model checking*. Over the past decades, model checking has seen significant progress [9]. However, state-space explosion remains an issue, amplified for SPL models by the accumulation of possible variants and configurations. To mitigate these problems, family-based model checking was proposed as a means to simultaneously verify multiple variants in a single run (cf. [1]). To make SPL models amenable to family-based reasoning, feature-based variability was introduced in many behavioral models, e.g. based on process calculi [2, 10–12] and labeled transition systems (LTSs) [3, 13–15].

Arguably the most widely used behavioral SPL models are *featured transition systems* (FTSs) [13]. An FTS compactly represents multiple behaviors in a single transition system by exploiting transitions guarded by *feature expressions*.

A transition of a given product can be taken if the product fulfills the feature expression associated with the transition. Thus, an FTS incorporates the behavior of all eligible products, while individual behavior can be extracted as LTSs. Properties of such models can be verified with dedicated SPL model checkers [16–18] or, to a certain degree, with single system model checkers [7, 12, 14].

As far as we know, none of the existing tools can verify modal μ-calculus properties over FTSs in a family-based manner. However, there have been earlier proposals for using the μ-calculus to analyze SPLs (cf., e.g., [2,10,12,19,20]). In [19], for instance, `mCRL2` and its toolset [21, 22] were used for *product-based* model checking. The flexibility of `mCRL2`'s data language allowed to model and select valid product configurations and to model and check the behavior of individually generated products. While the SPL models of [19] have an FTS-like semantics, to actually perform family-based model checking also the supporting logic must be able to handle the specificity of FTSs, viz. transitions labeled with feature expressions. In [20], we generalized the approach that led to the feature-oriented variants `fLTL` [13] and `fCTL` [14] of LTL and CTL to the modal μ-calculus by defining μL_f, a *feature-oriented* variant of μL with an FTS semantics obtained by incorporating feature expressions. While μL_f paves the way for family-based model checking, so far the logic was without tool support, and it remained unclear whether it could be used effectively to model check FTSs.

Contributions. In this paper, we show how to effectively perform family-based model checking for μL_f by exploiting the `mCRL2` toolset *as-is*, i.e. avoiding the implementation of a dedicated SPL-oriented verifier. We first show how to solve the family-based model-checking problem via an embedding of μL_f into `mCRL2`'s modal μ-calculus with data. Then we define a partitioning procedure for μL_f that allows us to apply our results from [20]. Next, we evaluate our approach by verifying a number of representative properties over an `mCRL2` specification of the minepump SPL benchmark model [8, 13, 23–25]. We verify typical linear-time and branching-time properties. We also verify properties involving more than one feature modality in a single formula, which is a novelty that allows to check different behavior for different variants *at once*. Finally, we discuss the improvement in runtime that results from using `mCRL2` for family-based model checking as opposed to product-based model checking.

Further Related Work. There is a growing body of research on customizing model-checking techniques for SPLs. Like our FTS-based proposals [19, 20], the CCS-based proposals PL-CCS [2, 10] and DeltaCCS [12] are grounded in the μ-calculus. In [26], PL-CCS was proven to be less expressive (in terms of the sets of definable products) than FTSs, while DeltaCCS allows only limited family-based model checking (viz. verifying family-wide invariants for entire SPLs). DeltaCCS does provide efficient *incremental* model checking, a technique that improves product-based model checking by partially reusing verification results obtained for previously considered products. The state-of-the-art by the end of 2013 is summarized in [1], which also discusses type checking, static analysis, and theorem proving tailored for SPLs, as well as *software* model checking.

In a broader perspective, also *probabilistic* model checking was applied to SPLs recently, e.g. on feature-oriented (parametric) Markov chains [27–29] or Markov decision processes [30], and via a feature-oriented extension of the input language of the probabilistic model checker `Prism` [31], making the tool amenable to (family-based) SPL model checking [32]. Most recently, also *statistical* model checking was applied to SPLs [33, 34], based on a probabilistic extension of the feature-oriented process calculus of [11].

2 A Feature μ-Calculus μL_f over FTSs

The μ-calculus is an extension of modal logic with fixpoint operators whose formulas are interpreted over LTSs (cf. [35]). We fix a set of actions $\mathcal{A}$, ranged over by $a, b, \ldots$, and a set of variables $\mathcal{X}$, ranged over by $X, Y, \ldots$.

Definition 1. *The μ-calculus μL over $\mathcal{A}$ and $\mathcal{X}$ is given by*

$$\varphi ::= \bot \mid \top \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \langle a\rangle\varphi \mid [a]\varphi \mid X \mid \mu X.\varphi \mid \nu X.\varphi$$

where for $\mu X.\varphi$ and $\nu X.\varphi$ all free occurrences of X in φ are in the scope of an even number of negations (to guarantee well-definedness of the fixpoint semantics). □

Next to the Boolean constants falsum and verum, μL contains the connectives $\neg$, $\vee$ and $\wedge$ of propositional logic and the diamond and box operators $\langle\,\rangle$ and $[\,]$ of modal logic. The least and greatest fixpoint operators μ and ν provide recursion used for 'finite' and 'infinite' looping, respectively.

Definition 2. *An LTS over $\mathcal{A}$ is a triple $L = (S, \rightarrow, s_*)$, with states from S, transition relation $\rightarrow \subseteq S \times \mathcal{A} \times S$, and initial state $s_* \in S$.* □

Definition 3. *Let L be an LTS with set of states S. Let $sSet = 2^S$ be the set of state sets with typical element U and $sEnv = \mathcal{X} \to sSet$ the set of state-based environments. The semantics $[\![\cdot]\!]_L : \mu L \to sEnv \to sSet$ is given by*

$$\begin{array}{ll}
[\![\bot]\!]_L(\varepsilon) = \varnothing & [\![\langle a\rangle\varphi]\!]_L(\varepsilon) = \{\, s \mid \exists t\colon s \xrightarrow{a} t \wedge t \in [\![\varphi]\!]_L(\varepsilon) \,\} \\
[\![\top]\!]_L(\varepsilon) = S & [\![[a]\varphi]\!]_L(\varepsilon) = \{\, s \mid \forall t\colon s \xrightarrow{a} t \Rightarrow t \in [\![\varphi]\!]_L(\varepsilon) \,\} \\
[\![\neg\varphi]\!]_L(\varepsilon) = S \setminus [\![\varphi]\!]_L(\varepsilon) & [\![X]\!]_L(\varepsilon) = \varepsilon(X) \\
[\![(\varphi \vee \psi)]\!]_L(\varepsilon) = [\![\varphi]\!]_L(\varepsilon) \cup [\![\psi]\!]_L(\varepsilon) & [\![\mu X.\varphi]\!]_L(\varepsilon) = lfp(\,\lambda U.[\![\varphi]\!]_L(\varepsilon[U/X])\,) \\
[\![(\varphi \wedge \psi)]\!]_L(\varepsilon) = [\![\varphi]\!]_L(\varepsilon) \cap [\![\psi]\!]_L(\varepsilon) & [\![\nu X.\varphi]\!]_L(\varepsilon) = gfp(\,\lambda U.[\![\varphi]\!]_L(\varepsilon[U/X])\,)
\end{array}$$

where $\varepsilon[U/X]$, for $\varepsilon \in sEnv$, denotes the environment which yields $\varepsilon(Y)$ for variables Y different from the variable X and the set $U \in sSet$ for X itself. □

As typical for model checking, we only consider closed μL-formulas whose interpretation is independent of the environment. In such case we write $[\![\varphi]\!]_L$ for the interpretation of φ. Given a state s of an LTS L, we set $s \models_L \varphi$ iff $s \in [\![\varphi]\!]_L$.

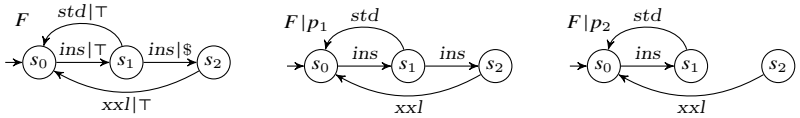
We next fix a finite non-empty set $\mathcal{F}$ of features, with f as typical element. Let $\mathbb{B}[\mathcal{F}]$ denote the set of Boolean expressions over $\mathcal{F}$. Elements χ and γ of $\mathbb{B}[\mathcal{F}]$ are referred to as feature expressions. A product is a set of features, and $\mathcal{P}$ denotes the set of products, thus $\mathcal{P} \subseteq \mathbf{2}^{\mathcal{F}}$, with $p, q, \ldots$ ranging over $\mathcal{P}$.

A subset $P \subseteq \mathcal{P}$ is referred to as a family of products. A feature expression γ, as Boolean expression over $\mathcal{F}$, can be interpreted as a set of products Q_γ, viz. the products p for which the induced truth assignment (**true** for $\mathsf{f} \in p$, **false** for $\mathsf{f} \notin p$) validates γ. Reversely, for each family $P \subseteq \mathcal{P}$ we fix a feature expression γ_P to represent it. The constant $\top$ denotes the feature expression that is always true. We now recall FTSs from [13] as a model for SPLs, using the notation of [20].

Definition 4. *An FTS over $\mathcal{A}$ and $\mathcal{F}$ is a triple $F=(S, \theta, s_*)$, with states from S, transition constraint function $\theta\colon S \times \mathcal{A} \times S \to \mathbb{B}[\mathcal{F}]$, and initial state $s_* \in S$.* □

For states $s, t \in S$, we write $s \xrightarrow{a|\gamma}_F t$ if $\theta(s,a,t) = \gamma$ and $\gamma \not\equiv \bot$. The projection of an FTS $F = (S, \theta, s_*)$ onto a product $p \in \mathcal{P}$ is the LTS $F|p = (S, \rightarrow_{F|p}, s_*)$ over $\mathcal{A}$ with $s \xrightarrow{a}_{F|p} t$ iff $p \in Q_\gamma$ for a transition $s \xrightarrow{a|\gamma}_F t$ of F.

Example 1. Let P be a product line of (four) coffee machines, with independent features $\{\$, €\}$ representing the presence of a coin slot accepting dollars or euros.



FTS F models its family behavior, with actions to insert coins (*ins*) and to pour standard (*std*) or extra large (*xxl*) coffee. Each coffee machine accepts either dollars or euros. Extra large coffee is exclusively available for two dollars. LTSs $F|p_1$ and $F|p_2$ model the behavior of products $p_1 = \{\$\}$ and $p_2 = \{€\}$. Note that $F|p_2$ lacks the transition from s_1 to s_2 that requires feature $\$$. □

In [20], we introduced μL_f, an extension with features of the μ-calculus μL, interpreted over FTSs rather than LTSs.

Definition 5. *The feature μ-calculus μL_f over $\mathcal{A}$, $\mathcal{F}$ and $\mathcal{X}$, is given by*

$$\varphi_f ::= \bot \mid \top \mid \neg\varphi_f \mid \varphi_f \vee \psi_f \mid \varphi_f \wedge \psi_f \mid \langle a|\chi\rangle\varphi_f \mid [a|\chi]\varphi_f \mid X \mid \mu X.\varphi_f \mid \nu X.\varphi_f$$

where for $\mu X.\varphi_f$ and $\nu X.\varphi_f$ all free occurrences of X in φ_f are in the scope of an even number of negations. □

Also for μL_f we mainly consider closed formulas. The logic μL_f replaces the binary operators $\langle a\rangle\varphi$ and $[a]\varphi$ of μL by ternary operators $\langle a|\chi\rangle\varphi_f$ and $[a|\chi]\varphi_f$, respectively, where χ is a feature expression.

A Product-Based Semantics. In [20], we gave a semantics $(\!|\varphi_f|\!)_F$ for closed μL_f-formulas φ_f with subsets of $S \times \mathcal{P}$ as denotations. We showed that this product-based semantics can be characterized as follows

$$(\!|\varphi_f|\!)_F = \{\, (s,p) \in S \times \mathcal{P} \mid s \in [\![\, pr(\varphi_f, p) \,]\!]_{F|p} \,\}$$

where the projection function $pr\colon \mu L_f \times \mathcal{P} \to \mu L$ is given by

$$\begin{array}{rrrr} pr(\bot,p) = \bot & pr(\top,p) = \top & pr(X,p) = X & pr(\neg\varphi_f,p) = \neg pr(\varphi_f,p) \\ pr(\varphi_f \vee \psi_f,p) = pr(\varphi_f,p) \vee pr(\psi_f,p) & & pr(\mu X.\varphi_f,p) = \mu X.pr(\varphi_f,p) & \\ pr(\varphi_f \wedge \psi_f,p) = pr(\varphi_f,p) \wedge pr(\psi_f,p) & & pr(\nu X.\varphi_f,p) = \nu X.pr(\varphi_f,p) & \end{array}$$

$$pr(\langle a|\chi\rangle\varphi_f,p) = \textbf{if } p \in Q_\chi \textbf{ then } \langle a\rangle pr(\varphi_f,p) \textbf{ else } \bot \textbf{ end}$$

$$pr([a|\chi]\varphi_f,p) = \textbf{if } p \in Q_\chi \textbf{ then } [a]\, pr(\varphi_f,p) \textbf{ else } \top \textbf{ end}$$

Thus, for a formula $\varphi_f \in \mu L_f$ and a product $p \in \mathcal{P}$, a μL-formula $pr(\varphi_f, p)$ is obtained from φ_f, by replacing subformulas $\langle a|\chi\rangle\psi_f$ by $\bot$ and $[a|\chi]\psi_f$ by $\top$, respectively, in case $p \notin Q_\chi$, while omitting χ otherwise. Formulas of μL_f permit reasoning about the behavior of products, as illustrated below.

Example 2. Formulas of μL_f for Example 1 include the following.

(a) $\varphi_f = \langle ins|\top\rangle\,(\,[ins|\Subset]\bot \wedge \langle std|\top\rangle\top\,)$ characterizes the family of products P that can execute *ins*, after which *ins* cannot be executed by products satisfying $\Subset$, while *std* can be executed by all products of P.
(b) $\psi_f = \nu X.\mu Y.((\,[ins|\Subset]Y \wedge [xxl|\Subset]Y) \wedge [std|\Subset]X)$ characterizes the (sub)family of products which, when having feature $\Subset$, action *std* occurs infinitely often on all infinite runs over *ins*, *xxl*, and *std*. □

In practice, we are often interested in deciding whether a family of products P satisfies a formula φ_f. The semantics of μL_f, however, does not allow for doing so in a family-based manner as it is product-oriented. For that reason, we introduced in [20] a second semantics $[\![\cdot]\!]_F$ for μL_f (cf. Definition 6 below) providing a stronger interpretation for the modalities to enable family-based reasoning. We stress that this second, family-based interpretation was designed to specifically support efficient model checking; the product-oriented $(\!|\cdot|\!)_F$ remains the semantic reference. The correspondence between the two interpretations was studied in detail in [20]. We next summarize the most important results.

A Family-Based Semantics. In our family-based interpretation, the ternary operator $\langle a|\chi\rangle\varphi_f$ holds for a family P with respect to an FTS F in a state s, if all products in P satisfy the feature expression χ and there is an a-transition, *shared among all products in* P, that leads to a state where φ_f holds for P (i.e. the products in P can collectively execute a). The $[a|\chi]\varphi_f$ modality holds in a state of F for a set of products P, if for each subset P' of P for which an a-transition is possible, φ_f holds for P' in the target state of that a-transition. While under the product-based interpretation $(\!|\cdot|\!)_F$ of μL_f, the two modalities in μL_f are, like in μL, each other's dual, this is no longer the case under the family-based interpretation $[\![\cdot]\!]_F$ below.

Definition 6. *Let $F = (S, \theta, s_*)$ be an FTS. Let $sPSet = 2^{S\times 2^{\mathcal{P}}}$ be the set of state-family pairs with typical element W and $sPEnv = \mathcal{X} \to sPSet$ the set of state-family environments. The semantics $[\![\cdot]\!]_F : \mu L_f \to sPEnv \to sPSet$ is given by*

$$[\![\bot]\!]_F(\zeta) = \varnothing \qquad [\![\top]\!]_F(\zeta) = S \times 2^{\mathcal{P}}$$
$$[\![\neg\varphi_f]\!]_F(\zeta) = (S \times 2^{\mathcal{P}}) \setminus [\![\varphi_f]\!]_F(\zeta) \qquad [\![X]\!]_F(\zeta) = \zeta(X)$$
$$[\![(\varphi_f \vee \psi_f)]\!]_F(\zeta) = [\![\varphi_f]\!]_F(\zeta) \cup [\![\psi_f]\!]_F(\zeta) \qquad [\![\mu X.\varphi_f]\!]_F(\zeta) = lfp(\lambda W.[\![\varphi_f]\!]_F(\zeta[W/X]))$$
$$[\![(\varphi_f \wedge \psi_f)]\!]_F(\zeta) = [\![\varphi_f]\!]_F(\zeta) \cap [\![\psi_f]\!]_F(\zeta) \qquad [\![\nu X.\varphi_f]\!]_F(\zeta) = gfp(\lambda W.[\![\varphi_f]\!]_F(\zeta[W/X]))$$
$$[\![\langle a|\chi\rangle\varphi_f]\!]_F(\zeta) = \{\,(s,P) \mid P \subseteq Q_\chi \wedge \exists \gamma, t : s \xrightarrow{a|\gamma}_F t \wedge P \subseteq Q_\gamma \wedge (t, P \cap Q_\chi \cap Q_\gamma) \in [\![\varphi_f]\!]_F(\zeta)\,\}$$
$$[\![[a|\chi]\varphi_f]\!]_F(\zeta) = \{\,(s,P) \mid \forall \gamma, t : s \xrightarrow{a|\gamma}_F t \wedge P \cap Q_\chi \cap Q_\gamma \neq \varnothing \Rightarrow (t, P \cap Q_\chi \cap Q_\gamma) \in [\![\varphi_f]\!]_F(\zeta)\,\}$$

where $\zeta[W/X]$, for $\zeta \in sPEnv$, denotes the environment which yields $\zeta(Y)$ for variables Y different from X and the set $W \in sPSet$ for X. □

The interpretation of a closed μL_f formula φ_f is independent of the environment and we therefore again simply write $[\![\varphi_f]\!]_F$. Given a state s of an FTS F, and a set of products $P \subseteq \mathcal{P}$, we write $s, P \models_F \varphi_f$ iff $(s, P) \in [\![\varphi_f]\!]_F$.

The theorem below summarizes the main results of [20], relating the family-based interpretation of μL_f to the LTS semantics of μL (and by extension, μL_f's product-based interpretation).

Theorem 1. *Let F be an FTS, and let $\mathcal{P}$ be a set of products.*

(a) For each formula $\varphi_f \in \mu L_f$, state $s \in S$, and individual product $p \in \mathcal{P}$: $s, \{p\} \models_F \varphi_f \iff s \models_{F|p} pr(\varphi_f, p)$.

(b) For negation-free formula $\varphi_f \in \mu L_f$, state $s \in S$, and product family $P \subseteq \mathcal{P}$: $s, P \models_F \varphi_f \implies \forall p \in P\colon s \models_{F|p} pr(\varphi_f, p)$ □

Note that in general, $s, P \not\models_F \varphi_f$ does not imply $s \not\models_{F|p} pr(\varphi_f, p)$ for *all* products in the family P. In the next section, we discuss how the above results can be exploited for family-based model checking of μL_f-formulas.

3 Family-Based Model Checking with mCRL2

In this section, we show how to obtain a decision procedure for $s_*, P \models_F \varphi_f$ via a mapping into the first-order μ-calculus μL_{FO} and solving the corresponding model-checking problem. Our approach consists of two steps: (i) translation of the μL_f-formula at hand; (ii) translation of the FTS describing the family behavior into an LTS with parametrized actions. Since μL_{FO} is a fragment of the logic from [36,37], we can use off-the-shelf tools such as the `mCRL2` toolset [21,22] to perform family-based model checking of properties expressed in μL_f. We first review μL_{FO} before we proceed to describe the above translations.

3.1 The First-Order μ-Calculus μL_{FO}

The first-order μ-calculus with data of [36,37] is given in the context of a data signature $\Sigma = (S, O)$, with set of sorts S and set of operations O, and of a set of 'sorted' actions $\mathcal{A}$. μL_{FO} is essentially a fragment of the logic of [36,37] in which S is the single sort FExp, with typical elements β, χ, γ, representing Boolean expressions over features and data variables taken from a set $\mathcal{V}$, with typical element v. In toolsets such as `mCRL2`, FExp can be formalized as an abstract data type defining BDDs ranging over features (cf. [38]). We fix a set of recursion variables $\tilde{X}$. Formulas $\varphi \in \mu L_{\text{FO}}$ are then given by

$$\varphi ::= \bot \mid \top \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \gamma_1 \Rightarrow \gamma_2 \mid \exists v.\varphi \mid \forall v.\varphi \mid$$
$$\langle a(v) \rangle \varphi \mid [a(v)]\varphi \mid \tilde{X}(\gamma) \mid \mu\tilde{X}(v_{\tilde{X}} := \gamma).\varphi \mid \nu\tilde{X}(v_{\tilde{X}} := \gamma).\varphi$$

where for $\mu\tilde{X}(v_{\tilde{X}} := \gamma).\varphi$ and $\nu\tilde{X}(v_{\tilde{X}} := \gamma).\varphi$ all free occurrences of $\tilde{X}$ in φ are in the scope of an even number of negations, and each variable $\tilde{X}$ is bound at most once. To each recursion variable $\tilde{X}$ a unique data variable $v_{\tilde{X}}$ is associated, but we often suppress this association and simply write $\mu\tilde{X}(v := \gamma).\varphi$ and $\nu\tilde{X}(v := \gamma).\varphi$ instead. The language construct $\gamma_1 \Rightarrow \gamma_2$ is used to express that the set of products characterized by γ_1 is a subset of those characterized by γ_2.

We interpret μL_{FO} over LTSs whose actions carry closed feature expressions.

Definition 7. *A parametrized LTS over $\mathcal{A}$ and $\mathcal{F}$ is a triple $(S, \to, s_*)$ with states from S, transition relation $\to \subseteq S \times \mathcal{A}[\mathcal{F}] \times S$ where $\mathcal{A}[\mathcal{F}] = \{ a(\gamma) \mid a \in \mathcal{A}, \gamma \in \mathbb{B}[\mathcal{F}] \}$, and initial state $s_* \in S$.* □

In the presence of variables ranging over feature expressions, we distinguish two sets of environments, viz. data environments $\theta \in VEnv = \mathcal{V} \to 2^{\mathcal{P}}$ and recursion variable environments $\xi \in XEnv = \tilde{\mathcal{X}} \to 2^{\mathcal{P}} \to 2^S$. The semantics $[\![\cdot]\!]_{FO}$ is then of type $\mu L_{\text{FO}} \to XEnv \to VEnv \to 2^S$. To comprehend our translation of μL_f into μL_{FO}, we address the semantics for the non-trivial constructs of μL_{FO}. The full semantics can be found in [36, 37].

$$[\![\exists v.\varphi]\!]_{FO}(\xi)(\theta) = \bigcup \{ [\![\varphi]\!]_{FO}(\xi)(\theta[Q/v]) \mid Q \subseteq \mathcal{P} \}$$
$$[\![\langle a(v) \rangle \varphi]\!]_{FO}(\xi)(\theta) = \{ s \mid \exists \gamma, t\colon s \xrightarrow{a(\gamma)} t \wedge \theta(v) = Q_\gamma \wedge t \in [\![\varphi]\!]_{FO}(\xi)(\theta) \}$$
$$[\![\mu \tilde{X}(v{:=}\gamma).\varphi]\!]_{FO}(\xi)(\theta) = \Big(lfp(\lambda \pi{:}2^{\mathcal{P}} \to 2^S.\lambda Q.[\![\varphi]\!]_{FO}(\xi[\pi/\tilde{X}])(\theta[Q/v])\,)\Big)(\theta(\gamma))$$

For existential quantification, the data environment $\theta[Q/v]$ assigns a family of products Q to the data variable v; the set of states that satisfy $\exists v.\varphi$ is then the set of states satisfying φ for any possible assignment to data variable v.

For the diamond modality, a state s is included in its semantics, if in the parametrized LTS, state s admits a transition with parametrized action $a(\gamma)$ to a state t such that the set of products $\theta(v)$ is exactly the set of products Q_γ associated with the feature expression γ of the transition, while the target state t satisfies φ. Note that the set of products Q_γ can be established independently from the environment θ since γ is closed, i.e. variable-free.

The least fixpoint construction is more involved compared to the corresponding construct of μL_f because of the parametrization. Here the semantics of the least fixpoint is taken for the functional that fixes both the recursion variable $\tilde{X}$ and the data variable v, with π and Q, respectively. Next, application to the value of the initializing feature expression γ yields a set of states.

With respect to a parametrized LTS L, we put $s \models_L \varphi$, for $s \in S$ and $\varphi \in \mu L_{\text{FO}}$ closed, if $s \in [\![\varphi]\!]_{FO}(\xi_0)(\theta_0)$ for some $\xi_0 \in XEnv$ and $\theta_0 \in VEnv$.

3.2 Translating the Family-Based Interpretation of μL_f to μL_{FO}

To model check a μL_f-formula against an FTS, we effectively verify its corresponding μL_{FO}-formula against the parametrized LTS that is obtained as follows.

Definition 8. *Let $F = (S, \theta, s_*)$ be an FTS over $\mathcal{A}$ and $\mathcal{F}$. Take $\mathcal{A}[\mathcal{F}] = \{ a(\gamma) \mid a \in \mathcal{A}, \gamma \in \mathbb{B}[\mathcal{F}] \}$. Define the parametrized LTS $L(F)$ for F by $L(F) = (S, \to, s_*)$ where $\to$ is defined by $s \xrightarrow{a(\gamma)} t$ iff $\theta(s, a, t) = \gamma$ and $\gamma \not\equiv \bot$.* □

Thus, we use the parameter of an action as placeholder for the feature expression that guards a transition, writing $s \xrightarrow{a(\gamma)} t$.

Next, we define a translation tr that yields for a set of products P, represented by a closed feature expression γ_{P} of sort FExp, and a μL_f-formula φ_f, a μL_{FO}-formula $tr(\gamma_{\text{P}}, \varphi_f)$. We provide an explanation of this transformation, guided by the family-based semantics of μL_f, afterwards (cf. Definition 6).

Definition 9. *The translation function* $tr : \mathsf{FExp} \times \mu L_f \to \mu L_{FO}$ *is given by*

$$tr(\gamma, \bot) = \bot \qquad tr(\gamma, \top) = \top$$
$$tr(\gamma, \neg\varphi_f) = \neg tr(\gamma, \varphi_f) \qquad tr(\gamma, X) = \tilde{X}(\gamma)$$
$$tr(\gamma, \varphi_f \vee \psi_f) = tr(\gamma, \varphi_f) \vee tr(\gamma, \psi_f) \qquad tr(\gamma, \mu X.\varphi_f) = \mu\tilde{X}(v{:=}\gamma).tr(v, \varphi_f)$$
$$tr(\gamma, \varphi_f \wedge \psi_f) = tr(\gamma, \varphi_f) \wedge tr(\gamma, \psi_f) \qquad tr(\gamma, \nu X.\varphi_f) = \nu\tilde{X}(v{:=}\gamma).tr(v, \varphi_f)$$
$$tr(\gamma, \langle a|\chi\rangle\varphi_f) = (\gamma \Rightarrow \chi) \wedge \exists v.\langle a(v)\rangle((\gamma \Rightarrow v) \wedge tr(\gamma{\wedge}\chi{\wedge}v, \varphi_f))$$
$$tr(\gamma, [a|\chi]\varphi_f) = \forall v.[a(v)]((\gamma{\wedge}\chi{\wedge}v \Rightarrow \bot) \vee tr(\gamma{\wedge}\chi{\wedge}v, \varphi_f)) \qquad \square$$

Logical constants and propositional connectives are translated as expected. The feature expression γ in our translation symbolically represents the set of products that collectively can reach a given state in our parametrized LTS. Note that this expression is 'updated' only in our translation of the modal operators and passed on otherwise. For the $\langle\cdot|\cdot\rangle$-operator, the existence of a feature expression β in Definition 6 with an $a|\beta$-transition is captured by the existentially quantified data variable v: a state s in a parametrized LTS satisfies $\exists v.\langle a(v)\rangle((\gamma \Rightarrow v) \wedge tr(\gamma{\wedge}\chi{\wedge}v, \varphi_f))$ only when a transition from s exists labeled with a parametrized action $a(\beta)$ such that for v matching β, also $\gamma \Rightarrow v$ and $tr(\gamma{\wedge}\chi{\wedge}v, \varphi_f)$ hold. Likewise, for the $[\cdot|\cdot]$-operator, the universal quantification over feature expressions guarding transitions is captured by a universally quantified data variable v that is passed as a parameter to the action a. The formula $(\gamma{\wedge}\chi{\wedge}v \Rightarrow \bot)$ expresses that the corresponding product families are disjoint.

We utilize the data variables associated to recursion variables in $tr(\gamma, X)$ to pass the feature expression γ to the recursion variable $\tilde{X}$. A similar mechanism applies to the fixpoint constructions. Thus, we assign γ to the data variable v associated with $\tilde{X}$, signified by the bindings $\mu\tilde{X}(v{:=}\gamma)$ and $\nu\tilde{X}(v{:=}\gamma)$, and use the data variable in the translation of the remaining subformula, i.e. $tr(v, \varphi_f)$.

Next, we state the correctness of the translation.

Theorem 2. *Let F be an FTS and let $\mathcal{P}$ be a set of products. For each μL_f-formula φ_f, state $s \in S$ and product family $P \subseteq \mathcal{P}$, it holds that*

$$s, P \models_F \varphi_f \iff s \models_{L(F)} tr(\gamma_P, \varphi_f)$$

Proof. (Sketch) The proof relies on the claim that, for state s, product family P, data environment θ, and feature expression γ such that $\theta(\gamma) = P$, we have

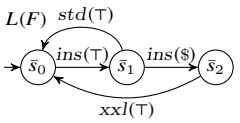
$$(s, P) \in [\![\varphi_f]\!]_F(\zeta) \iff s \in [\![tr(\gamma, \varphi_f)]\!]_{FO}(\xi)(\theta)$$

for environments $\zeta \in sPEnv$, and $\xi \in XEnv$ such that $(s, P) \in \zeta(X)$ iff $s \in \xi(\tilde{X})(P)$ for all $X \in \mathcal{X}$. The claim is shown by structural induction, exploiting iteration for the fixpoint constructions. From the claim the theorem follows directly. $\square$

As a consequence of the above theorem we can model check a μL_f-formula over an FTS by model checking the corresponding μL_{FO}-formula over the corresponding parametrized LTS.

Example 3. From Examples 1 and 2, recall FTS F of family P and μL_f-formula ψ_f stating that for products with $\in$, action *std* occurs infinitely often on all infinite runs over $\{ins, xxl, std\}$. Take the corresponding parametrized LTS $L(F)$. Clearly,

ψ_f holds in state s_0 for all products without both features $\in$ and \$. mCRL2 can verify this, as deciding $s_0, P' \models_F \psi_f$ for the family $P' = \{\varnothing, \{\in\}, \{\$\}\}$ translates to model checking $\bar{s}_0 \models_{L(F)} tr(\gamma_{\mathrm{P}}, \psi_f)$, where $tr(\gamma_{\mathrm{P}}, \psi_f)$ is the μL_{FO}-formula



$$\begin{aligned}\nu\tilde{X}(v_x{:=}\gamma_{\mathrm{P}}).\mu\tilde{Y}(v_y{:=}v_x).\,(\;&\forall v.[ins(v)]\big((v_y \wedge \in \wedge\, v \Rightarrow \bot) \vee \tilde{Y}(v_y \wedge \in \wedge\, v)\big) \wedge\\ &\forall v.[xxl(v)]\big((v_y \wedge \in \wedge\, v \Rightarrow \bot) \vee \tilde{Y}(v_y \wedge \in \wedge\, v)\big) \wedge\\ &\forall v.[std(v)]\big((v_y \wedge \in \wedge\, v \Rightarrow \bot) \vee \tilde{X}(v_y \wedge \in \wedge\, v)\big)\;)\end{aligned}$$

Note the passing of γ_{P} via the respective assignments $v_x{:=}\gamma_{\mathrm{P}}$ and $v_y{:=}v_x$. □

4 Family-Based Partitioning for μL_f

With Theorem 2 in place we are in a position where family-based model-checking a system can be performed using a standard μ-calculus model checker. The final issue we face is to find, given a formula $\varphi_f \in \mu L_f$ and a family of products P, the subfamily of P whose products satisfy φ_f, as well as the subfamily whose products do not satisfy φ_f. Thus, given a negation-free formula φ_f and a family of products P, we are interested in computing a partitioning $(P_\oplus, P_\ominus)$ of P such that

$$\forall p \in P_\oplus \colon s_*, p \models_{F|p} pr(\varphi_f, p) \quad \text{and} \quad \forall p \in P_\ominus \colon s_*, p \not\models_{F|p} pr(\varphi_f, p) \tag{1}$$

Rather than establishing this product-by-product, we are after a procedure that decides Property (1) in a family-based manner.

The previous section provides a sound decision procedure for $s_*, P \models_F \varphi_f$. If the procedure returns **true** for the family P, we are done: Theorem 1 guarantees that the property holds for all products of P, i.e. $s_* \models_{F|p} pr(\varphi_f, p)$ for all $p \in P$. If, on the other hand, the decision procedure for $s_*, P \models_F \varphi_f$ returns **false** and P is not a singleton family, we cannot draw a conclusion for any of the products. However, in view of Lemma 1 below, we can run the decision procedure to decide $s_*, P \models_F \varphi_f^c$, where φ_f^c is the *complement* of φ_f. Formally, for negation-free μL_f-formula φ_f, the formula φ_f^c is defined inductively by

$$\begin{array}{llll} \bot^c = \top & (\varphi_f \vee \psi_f)^c = \varphi_f^c \wedge \psi_f^c & (\mu X.\varphi_f)^c = \nu X.\varphi_f^c \\ \top^c = \bot & (\varphi_f \wedge \psi_f)^c = \varphi_f^c \vee \psi_f^c & (\nu X.\varphi_f)^c = \mu X.\varphi_f^c \\ X^c = X & (\langle a|\chi\rangle\varphi_f)^c = [a|\chi]\varphi_f^c & ([a|\chi]\varphi_f)^c = \langle a|\chi\rangle\varphi_f^c \end{array}$$

We have the following result.

Lemma 1. *For each negation-free formula φ_f and set of products P, it holds that $s_*, P \models_F \varphi_f^c$ implies $s_*, p \not\models_{F|p} pr(\varphi_f, p)$ for all $p \in P$.*

Proof. Let $\varphi_f \in \mu L_f$ be closed and negation-free, and let P be a family of products. For closed and negation-free $\psi_f \in \mu L_f$, state s, and product p,

$$s \models_{F|p} pr(\psi_f^c, p) \iff s \models_{F|p} \neg pr(\psi_f, p) \tag{2}$$

a fact readily proven by induction on ψ_f. Assume $s_*, P \models_F \varphi_f^c$. Observe, if φ_f is negation-free then so is φ_f^c. Hence, by Theorem 1, $s_* \models_{F|p} pr(\varphi_f^c, p)$ for every $p \in P$. By Equivalence (2) we find $s_* \not\models_{F|p} pr(\varphi_f, p)$ for all $p \in P$. □

Algorithm 1 Family-Based Partitioning

```
1: function FBP(P, φ_f)
2:     if s_*, P ⊨_F φ_f then return (P, ∅)
3:     else
4:         if s_*, P ⊨_F φ_f^c then return (∅, P)
5:         else partition P into (P_1, P_2)
6:             (P_1^+, P_1^-) ← FBP(P_1, φ_f)
7:             (P_2^+, P_2^-) ← FBP(P_2, φ_f)
8:             return (P_1^+ ∪ P_2^+, P_1^- ∪ P_2^-)
9:         end if
10:    end if
11: end function
```

On the lemma we base the straightforward partition procedure $\text{FBP}(P, \varphi_f)$ of Algorithm 1 for computing $(P_\oplus, P_\ominus)$ for a product family P such that each product in $P_\oplus$ satisfies the μL_f-formula φ_f, while each product in $P_\ominus$ fails φ_f.

Theorem 3. *For closed and negation-free φ_f, procedure $\text{FBP}(P, \varphi_f)$ terminates and returns a partitioning $(P_\oplus, P_\ominus)$ of P satisfying Property* (1).

Proof. Observe that the algorithm can be called at most $2^{|P|}$ times as each call is performed on a strictly smaller subset of P. Therefore, the algorithm terminates iff the procedure for deciding $s_*, P \models_F \varphi_f$ terminates. The correctness of the resulting partitioning $(P_\oplus, P_\ominus)$ follows by a straightforward induction, using Theorem 1 and Lemma 1. □

Example 4. Applying the algorithm to the FTS of Example 1 and the formula ψ_f of Example 2, running $\text{FBP}(\top, \psi_f)$, we find $s_0, \top \not\models_F \psi_f$ and $s_0, \top \not\models_F \psi_f^c$. Splitting the family in sets $\in \wedge\ \$$ and $\neg(\in \wedge\ \$)$ and recursively running $\text{FBP}(\in \wedge\ \$, \psi_f)$, returns the partition $(\bot, \in \wedge\ \$)$, since we have $s_0, \in \wedge\ \$ \models_F \psi_f^c$, and subsequently running $\text{FBP}(\neg(\in \wedge\ \$), \psi_f)$ returns $(\neg(\in \wedge\ \$), \bot)$ since $s_0, \neg(\in \wedge\ \$) \models_F \psi_f$. Therefore $\text{FBP}(\top, \psi_f)$ returns the partition $(\neg(\in \wedge\ \$), \in \wedge\ \$)$.

Clearly, repeatedly splitting families into subfamilies may lead to an exponential blow-up, in the worst case ultimately yielding a product-based analysis. Examples can be synthesized achieving this. However, in the SPL setting, an obvious strategy to partition a family P is to split along a feature f, i.e. in the algorithm set $P_1 = \{\, p \in P \mid \mathsf{f} \in p \,\}$ and $P_2 = \{\, p \in P \mid \mathsf{f} \notin p \,\}$. In general, the order of subsequent features f will influence the number of split-ups needed. Fortunately, candidate features for splitting along may be distilled from the structure of the system and from specific domain knowledge. The experiments reported in the next section confirm this. As we will see, with an appropriate decomposition a low number of splittings will do.

5 Case Study

In this section, we report on our experiments to use the `mCRL2` toolset to perform product-based and family-based model checking of an SPL model of the minepump from [39], making use of the logics and translations discussed above.

The SPL minepump model was first introduced in [4] as a reformulation of the configurable software system controlling a pump for mine drainage. The purpose of the minepump is to pump water out of a mine shaft, for which a controller operates a pump that may not start nor continue running in the presence of a dangerously high level of methane gas. Therefore, it communicates with a number of sensors measuring the water and methane levels. Here, we consider the model as used in [13] that consists of 7 independent optional features for a total of $2^7 = 128$ variants. These features concern command types, methane detection, and water levels, abbreviated as Ct, Cp, Ma, Mq, $L\ell$, Ln, and Lh.

The minepump model of [13] is distributed with the `ProVeLines` SPL toolset [18] (`http://projects.info.unamur.be/fts/provelines/`). We first manually translated the `fPROMELA` model to a parametrized LTS encoded in `mCRL2`.[3] For our model checking we considered twelve properties expressed in μL_f. The first six are μ-calculus versions of LTL properties of [13] (four of which are analyzed also in [8]). The others are CTL-like properties. Following the approach described in this paper, the formulas were translated into μL_{FO} and model checked over the `mCRL2` model representing a parametrized LTS. The properties, results, and runtimes are summarized in Table 1. All our experiments were run on a standard Macbook Pro using revision 14493 of the mCRL2 toolset.

Family-Based Model Checking. For each of the twelve properties, we provide its intuitive meaning, its specification in μL_f, and the result of model checking the property (indicating also the number of products for which the result holds). This concerns the first three columns of Table 1.[4,5] In the remaining columns, we report the runtimes (in seconds) needed to verify the properties with `mCRL2`, both product-based (*one-by-one*, abbreviated to 'one') and family-based (*all-in-one*, abbreviated to 'all'). We report the internal time as measured by the tools. We immediately notice that family-based model checking with `mCRL2` compares rather favorably to product-based model checking.

Next we discuss the verification of the properties listed in Table 1. Absence of deadlock turns out to be one of the more involved formulas to check family-wise for the case of the minepump. This is because in search for the truth value of the formula, all reachable states need to be visited. The μL-formula $[\texttt{true}^*]\langle\texttt{true}\rangle\top$, translates to the μL_f-formula $[\texttt{true}^*|\top]\langle\texttt{true}|\top\rangle\top$. The main complication arises from the fact that for each non-empty set of products P that can reach a state s in the FTS, the family-based semantics of $\langle\texttt{true}|\top\rangle\top$ requires that there is a transition from s shared among all P. A partitioning of the set of all products that is too coarse leads to a trace indicating a violation of the μL_f-formula. Next, the trace can be analyzed with the `mCRL2` toolset to identify a suitable decomposition into subfamilies.

[3] The `mCRL2` code is distributed with the `mCRL2` toolset (svn revision 14493).

[4] For a compact presentation of formulas in Table 1 we allow regular expressions in the modalities as syntactic sugar, as done in [22, 37].

[5] Standard μ-calculus formulas in μL can be seen as μL_f-formulas by adjoining the feature expression $\top$ to every modality, i.e. replacing each 'diamond' modality $\langle a\rangle$ by $\langle a|\top\rangle$ and each 'box' modality $[a]$ by $[a|\top]$.

Table 1. Minepump properties and results (**true**/**false**) and runtimes (in seconds) of both product-based (*one-by-one*) and family-based (*all-in-one*) verification with `mCRL2`

Φ	property in μL_f	result	one	all
φ_1	*Absence of deadlock* $[\,\mathtt{true}^*\,]\,\langle\,\mathtt{true}\,\rangle\,\top$	128/0	10.02	2.07
φ_2	*The controller cannot infinitely often receive water level readings* $\mu X\,.\,[\,(\,\neg \mathit{levelMsg}\,)^*\,.\,\mathit{levelMsg}\,]\,X$	0/128	10.18	0.16
φ_3	*The controller cannot fairly receive each of the three message types* $\mu X\,.\,(\,[\,\mathtt{true}^*.\,\mathit{commandMsg}\,]\,X \vee [\,\mathtt{true}^*.\,\mathit{alarmMsg}\,]\,X \vee [\,\mathtt{true}^*.\,\mathit{levelMsg}\,]\,X\,)$	0/128	24.33	0.25
φ_4	*The pump cannot be switched on infinitely often* $(\,\mu X\,.\,\nu Y\,.\,(\,[\,\mathit{pumpStart}\,.\,(\,\neg \mathit{pumpStop}\,)^*.\,\mathit{pumpStop}\,]\,X \wedge [\,\neg \mathit{pumpStart}\,]\,Y\,)$ $\quad \wedge\,(\,[\,\mathtt{true}^*.\,\mathit{pumpStart}\,]\,\mu Z\,.\,[\,\neg \mathit{pumpStop}\,]\,Z\,)$	96/32	21.09	0.89
φ_5	*The system cannot be in a situation in which the pump runs indefinitely in the presence of methane* $[\,\mathtt{true}^*\,]\,(\,([\,\mathit{pumpStart}\,.\,(\,\neg \mathit{pumpStop}\,)^*.\,\mathit{methaneRise}\,]\,\mu X.\,[R]X\,)$ $\quad \wedge\,([\,\mathit{methaneRise}\,.\,(\,\neg \mathit{methaneLower}\,)^*.\,\mathit{pumpStart}\,]\,\mu X.\,[R]X\,)\,)$ for $R = \neg(\,\mathit{pumpStop} + \mathit{methaneLower}\,)$	96/32	17.26	0.86
φ_6	*Assuming fairness (φ_3), the system cannot be in a situation in which the pump runs indefinitely in the presence of methane (φ_5)* $[\,\mathtt{true}^*\,]\,(\,([\,\mathit{pumpStart}\,.\,(\,\neg \mathit{pumpStop}\,)^*.\,\mathit{methaneRise}\,]\,\Psi\,)$ $\quad \wedge\,([\,\mathit{methaneRise}\,.\,(\,\neg \mathit{methaneLower}\,)^*.\,\mathit{pumpStart}\,]\,\Psi\,)\,)$ for $\Psi = \mu X\,.\,(\,[\,R^*.\,\mathit{commandMsg}\,]\,X \vee [\,R^*.\,\mathit{alarmMsg}\,]\,X \vee [\,R^*.\,\mathit{levelMsg}\,]\,X\,)$ and $R = \neg(\,\mathit{pumpStop} + \mathit{methaneLower}\,)$	112/16	27.32	3.67
φ_7	*The controller can always eventually receive/read a message, i.e. it can return to its initial state from any state* $[\,\mathtt{true}^*\,]\,\langle\,\mathtt{true}^*.\,\mathit{receiveMsg}\,\rangle\,\top$	128/0	18.36	2.40
φ_8	*Invariantly the pump is not started when the low water level signal fires* $[\,\mathtt{true}^*\,.\,\mathit{lowLevel}\,.\,(\neg(\mathit{normalLevel} + \mathit{highLevel}))^*\,.\,\mathit{pumpStart}.\,]\,\bot$	128/0	5.67	3.05
φ_9	*Invariantly, when the level of methane rises, it inevitably decreases* $[\,\mathtt{true}^*.\,\mathit{methaneRise}\,]\;\mu X.\,[\,\neg \mathit{methaneLower}\,]\,X \wedge \langle\,\mathtt{true}\,\rangle\,\top$	0/128	20.47	0.21
φ_{10}	*Products with feature Ct can switch on the pump* $\langle\,\mathtt{true}^*.\,\mathit{pumpStart} \mid \mathit{Ct}\,\rangle\,\mathtt{true}$	32/96	6.49	0.31
φ_{11}	*Products with feature Ct can always switch on the pump* $[\,\mathtt{true}^* \mid \mathit{Ct}\,]\,\langle\,\mathtt{true}^*.\,\mathit{pumpStart} \mid \mathit{Ct}\,\rangle\,\mathtt{true}$	28/100	21.11	2.32
φ_{12}	*Products with features Ct, Ma, and Lh can start the pump upon a high water level, but products without feature Lh cannot* $[\mathtt{true}^* \mid \top]\,((\,[\mathit{highLevel} \mid \mathit{Ct} \wedge \mathit{Ma} \wedge \mathit{Lh}]\,\langle\mathtt{true}^*.\mathit{pumpStart} \mid \top\rangle\top) \wedge [\mathit{pumpStart} \mid \neg \mathit{Lh}]\,\bot)$	128/0	13.35	3.36

For the minepump we identified 12 subfamilies, whose sets of trajectories are pairwise independent (i.e. for any two distinct subfamilies there exists a complete path possible for all products in one family, but not for all products in the other, and vice versa). These are the product sets characterized by the feature expressions $Ct \wedge \tilde{Cp} \wedge \tilde{Ma} \wedge \tilde{Mq}$, where $\tilde{\mathsf{f}} = \mathsf{f}, \neg\mathsf{f}$, yielding eight families, and four further families yielded by the product sets given by $\neg Ct \wedge \tilde{Cp} \wedge \tilde{Ma}$. As we shall see below, the combinations of features mentioned turn up in the analysis of other properties as well, which shows that the analysis of deadlock freedom (property φ_1) is a fruitful investigation.

Since no specific feature setting is involved in performing *levelMsg* infinitely often (for a stable and acceptable water level), property φ_2 can be refuted for the complete family of products at once by proving its complement. Also property φ_3, seemingly more complex, can be refuted via its complement, requiring a decomposition in subfamilies given by the four Boolean combinations of Cp and Ma.

The properties discussed so far cover general system aspects: absence of deadlock, future execution of an action, and fairness between subsystems. In contrast,

property φ_4 is specific to the minepump model. The property, modeled as a fluent [41], states that every computation involving *pumpStart* has, after a while, a finite number of alternations of starting and subsequent stopping the pump (fluent $\mathit{pumpStart}.(\neg \mathit{pumpStop})^*.\mathit{pumpStop}$), after which it is never again started, and after starting the pump (fluent $\mathtt{true}^*.\mathit{pumpStart}$) it is inevitably switched off. This property does not hold for all eligible products. However, a decomposition into a subfamily of 96 products given by $\neg(\mathit{Ct} \wedge \mathit{Lh})$, i.e. products missing *Ct* or *Lh*, and in two subfamilies $\mathit{Ct} \wedge \mathit{Mq} \wedge \mathit{Lh}$ and $\mathit{Ct} \wedge \neg \mathit{Mq} \wedge \mathit{Lh}$, of 32 products in total, does the job. The products in the first family satisfy φ_4, whereas products in the second and third family do not.

More involved system properties are φ_5 and φ_6, mixing starting and stopping of the pump with the rising and dropping of the methane level. Property φ_5 considers the rising of methane after the pump started but did not stop (fluent $\mathit{pumpStart}.(\neg \mathit{pumpStop})^*.\mathit{methaneRise}$) and, symmetrically, starting the pump after the methane level rose (fluent $\mathit{methaneRise}.(\neg \mathit{methaneLower})^*.\mathit{pumpStart}$). Formula φ_6 is a refinement of formula φ_5, restricting it to fair computations. For property φ_5, family-based model checking is achieved using the same decomposition of the product space, with the same outcome, as for property φ_4. For property φ_6, because of the fairness requirement, the number of satisfying products increases from 96 to 112. This can be checked for all 112 products at once. To identify the violating products, we consider φ_6's complement φ_6^c which is proven to hold for the family $\mathit{Ct} \wedge \neg \mathit{Ma} \wedge \mathit{Lh}$ of 16 products as a whole.

An important liveness property for circuit design, the so-called reset property *AG EF reset*, is expressible in CTL, but not in LTL (cf., e.g., [42]). For our case study, φ_7 is such a reset property. It states that from any state the controller can, possibly after a finite number of steps, receive a message. Thus, it can always return to the initial state. The μL-formula $[\mathtt{true}^*]\langle \mathtt{true}^*.\mathit{receiveMsg}\rangle \top$ can be verified using the same split-up in subfamilies that was used before for absence of deadlock (φ_1). A typical safety property is property φ_8, expressing that the pump is not started as long as water levels are low. It holds for all products, which can be verified for all product families at once. The third CTL-type property φ_9 states that when the level of methane rises, it inevitably drops again. It holds for no products. Refuting φ_9 can also be done for all product families at once.

Finally, we have verified feature-rich μL_f-formulas. Properties φ_{10} and φ_{11} focus on the family of products featuring *Ct* by means of the modalities $[\mathtt{true}^* \mid \mathit{Ct}]$ and $\langle \mathtt{true}^*.\, \mathit{pumpStart} \mid \mathit{Ct}\rangle$. However, by definition of pr, for products without feature *Ct*, property φ_{10} translates into $\bot$ of μL. Since a formula $[R^*|\chi]\varphi$ is to be read as $\nu X.[R|\chi]X \wedge \varphi$, we have that property φ_{11}, for products without *Ct*, coincides with $\nu X.[\mathtt{true}|\mathit{Ct}] \wedge \bot$. Apparently, comparing φ_{10} and φ_{11}, four more products with *Ct* (viz. those without any *Cp*, $L\ell$, or *Ma*) fail to meet the stronger φ_{11}. Finally, property φ_{12} holds for all products. Note that the first conjunct $[\, \mathit{highLevel} \mid \mathit{Ct} \wedge \mathit{Ma} \wedge \mathit{Lh}\,]\,\langle \mathtt{true}^*.\, \mathit{pumpStart} \mid \top\rangle \top$ is trivially true for products without any *Ct*, *Ma*, or *Lh* due to the box modality, while the second conjunct $[\, \mathit{pumpStart} \mid \neg \mathit{Lh}\,]\, \bot$ holds trivially for products that include *Lh*. Model checking this property requires a decomposition into two subfamilies, viz. the set of products with the feature *Mq* and the set of products without.

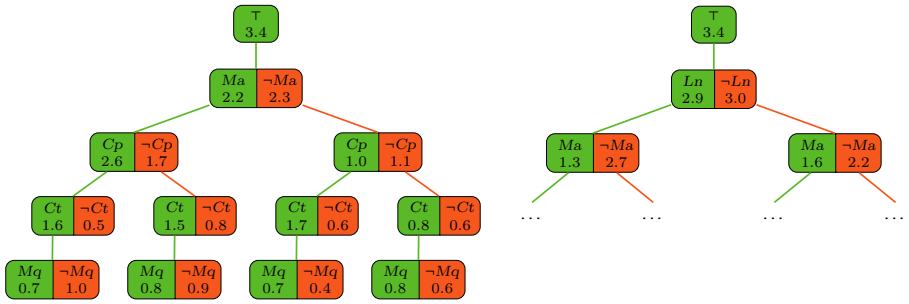


Fig. 1. Execution of Algorithm 1 for deadlock freedom (property φ_1) and the initial product family $\top$, using an optimal partitioning strategy (depicted on the left) vs. using an 'unproductive' refinement strategy, splitting Ln and $\neg Ln$ and following the optimal strategy afterwards (excerpt depicted on the right). The characterized family described at each node is the conjunction of the features along the path from the root to that node. Total computation time for the optimal strategy: 27.9 seconds; total computation time for the 12 leaves (i.e. all Mq, $\neg Mq$, and $\neg Ct$ nodes): 8.4 seconds. Total computation time for partitioning using the 'unproductive' strategy: 45.0 seconds.

Family-Based Partitioning. The results from the case study underline that family-based model checking has the potential to outperform product-based model checking. Next, we explore the requirements for a successful implementation of family-based partitioning using off-the-shelf technology.

Figure 1 (left) shows the runtimes (in seconds) associated with the model-checking problems of lines 2 and 4 of Algorithm 1 for deadlock freedom (property φ_1). The total time needed to run this algorithm, given the refinement strategy indicated in Fig. 1, is 27.9 seconds. Observe that checking all leaves takes 8.4 seconds.[6] We see similar figures for other properties we verified.

We draw two further conclusions from our experiments. First, as expected, refining a family of products with respect to non-relevant features can have a negative effect on runtime. For instance, partitioning with respect to a single non-essential feature Ln at an early stage, cf. Fig. 1(right), while following an optimal splitting otherwise, increases the runtime to 45 seconds; i.e. an additional 60%. Second, as illustrated by Fig. 1 (left), even for 'productive' refinements, model checking a property for a large family of products can consume a disproportionate amount of time. For instance, the three top nodes together account for almost 8 seconds, a quarter of the time spent on all model-checking problems combined.

We conclude that the performance of SPL verification using a general-purpose model checker for family-based partitioning crucially depends on the initial partitioning of products and the 'quality' of the refinements of families of products in the algorithm. This suggests that one must invest in: (i) determining heuristics for finding a good initial partitioning of a family of products, (ii) extracting information from the failed model-checking problems that facilitates an

[6] The additional overhead of approximately 6 seconds compared to the 2.07 seconds we reported in Table 1 is due to the fact that there we could inspect the model *once* for all possible families, whereas here we must inspect the model once *per* family.

informed/productive split-up of the family of products in line 5 of Algorithm 1. In particular for the μ-calculus, the second challenge may be difficult, since easily-interpretable feedback from its model checkers is generally missing so far.

6 Concluding Remarks and Future Work

We have showed how the feature μ-calculus μL_f can be embedded in μL_{FO}, a logic accepted by toolsets such as `mCRL2`. Through this embedding, we obtain a family-based model-checking procedure for verifying μ-calculus properties of SPLs, and similar systems with variability, using off-the-shelf verifiers. Moreover, as our experiments indicate, the resulting family-based model-checking approach trumps the product-based model-checking approach of [19].

The efficiency of *computing* a partitioning of a product family from which we can read which products satisfy which formula, strongly depends on the adopted strategy for splitting product families and may constitute a bottleneck in practice. We leave it for future research to find heuristics to guide this splitting. One possibility may be to deduce an effective strategy from the lattice of product families that can be obtained by exploring the FTS model and keeping track of (the largest) product families that are capable of reaching states. This lattice may even allow for determining a proper partitioning *a priori*. Another potentially promising direction is to split product families using information that is obtained from counterexamples. Indeed, in our product-based and family-based model-checking experiments we used counterexamples to find suitable subfamilies of products by splitting with respect to feature expressions on transitions that led to the violations. We must note, however, that this was largely a manual activity which required a fair share of tool experience. More generally, we note that constructing and interpreting counterexamples for the modal μ-calculus is notoriously difficult, as such counterexamples are not necessarily linear.

Finally, we believe that for particular properties specific insight regarding the model under study is required to quickly identify a successful split-up. We liken this to the approach taken in [8], where the theory of Galois connections is used to establish suitable abstractions of the minepump model prior to model checking with `SPIN`; we quote "Given sufficient knowledge of the system and the property, we can easily tailor an abstraction for analyzing the system more effectively". It is indeed common in SPL engineering to assume substantial understanding of the SPL under scrutiny, in particular of its commonalities and variability as documented in variability models like feature diagrams.

Acknowledgements. Maurice ter Beek was supported by the EU FP7 project QUANTICOL (600708). The authors are grateful to Franco Mazzanti for his help with the minepump model. Finally, we thank the anonymous referees for their suggestions, which helped improve the presentation of this paper.

References

1. T. Thüm *et al.* A classification and survey of analysis strategies for software product lines. *ACM Comput. Surv.*, 47(1), 2014.

2. A. Gruler, M. Leucker, and K. Scheidemann. Modeling and model checking software product lines. In *FMOODS*, volume 5051 of *LNCS*, pages 113–131, 2008.
3. K. Lauenroth, K. Pohl, and S. Töhning. Model checking of domain artifacts in product line engineering. In *ASE*, pages 269–280. IEEE, 2009.
4. A. Classen *et al.* Model checking lots of systems: Efficient verification of temporal properties in software product lines. In *ICSE*, pages 335–344. ACM, 2010.
5. F. Damiani and I. Schaefer. Family-based analysis of type safety of delta-oriented software product lines. In *ISoLA*, volume 7609 of *LNCS*, pages 193–207, 2012.
6. T. Thüm, I. Schaefer, M. Hentschel, and S. Apel. Family-based deductive verification of software product lines. In *GPCE*, pages 11–20. ACM, 2012.
7. M. H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. Using FMC for family-based analysis of software product lines. In *SPLC*, pages 432–439. ACM, 2015.
8. A. S. Dimovski, A. S. Al-Sibahi, C. Brabrand, and A. Wąsowski. Family-based model checking without a family-based model checker. In *SPIN*, volume 9232 of *LNCS*, pages 282–299, 2015.
9. E. M. Clarke, E. A. Emerson, and J. Sifakis. Model checking: Algorithmic verification and debugging. *C. ACM*, 52(11):74–84, 2009.
10. A. Gruler, M. Leucker, and K. Scheidemann. Calculating and modeling common parts of software product lines. In *SPLC*, pages 203–212. IEEE, 2008.
11. M. H. ter Beek, A. Lluch Lafuente, and M. Petrocchi. Combining declarative and procedural views in the specification and analysis of product families. In *SPLC*, volume 2, pages 10–17. ACM, 2013.
12. M. Lochau, S. Mennicke, H. Baller, and L. Ribbeck. Incremental model checking delta-oriented software product lines. *J. Log. Algebr. Meth. Program.*, 85(1):245–267, 2016.
13. A. Classen *et al.* Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. *IEEE Trans. Softw. Eng.*, 39(8):1069–1089, 2013.
14. A. Classen *et al.* Formal semantics, modular specification, and symbolic verification of product-line behaviour. *Sci. Comput. Program.*, 80(B):416–439, 2014.
15. M. H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints. *J. Log. Algebr. Meth. Program.*, 85(2):287–315, 2016.
16. M. H. ter Beek, F. Mazzanti, and A. Sulova. VMC: A tool for product variability analysis. In *FM*, volume 7436 of *LNCS*, pages 450–454, 2012.
17. A. Classen *et al.* Model checking software product lines with SNIP. *Int. J. Softw. Tools Technol. Transf.*, 14(5):589–612, 2012.
18. A. Cordy *et al.* ProVeLines: A product line of verifiers for software product lines. In *SPLC*, volume 2, pages 141–146. ACM, 2013.
19. M. H. ter Beek and E. P. de Vink. Using mCRL2 for the analysis of software product lines. In *FormaliSE*, pages 31–37. IEEE, 2014.
20. M. H. ter Beek, E. P. de Vink, and T. A. C. Willemse. Towards a feature mu-calculus targeting SPL verification. In *FMSPLE*, volume 206 of *EPTCS*, pages 61–75, 2016.
21. S. Cranen *et al.* An overview of the mCRL2 toolset and its recent advances. In *TACAS*, volume 7795 of *LNCS*, pages 199–213, 2013.
22. J. F. Groote and M. R. Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.
23. C. H. P. Kim *et al.* SPLat: Lightweight dynamic analysis for reducing combinatorics in testing configurable systems. In *ESEC/FSE*, pages 257–267. ACM, 2013.

24. J. Bürdek *et al.* Facilitating reuse in multi-goal test-suite generation for software product lines. In *FASE*, volume 9033 of *LNCS*, pages 84–99, 2015.
25. S. Lity et al. Applying incremental model slicing to product-line regression testing. In *ICSR*, volume 9679 of *LNCS*, pages 3–19, 2016.
26. H. Beohar, M. Varshosaz, and M. R. Mousavi. Basic behavioral models for software product lines: Expressiveness and testing pre-orders. *Sci. Comput. Program.*, 123:42–60, 2016.
27. C. Ghezzi and A. Sharifloo. Model-based verification of quantitative non-functional properties for software product lines. *Inform. Softw. Technol.*, 55(3):508–524, 2013.
28. M. Varshosaz and R. Khosravi. Discrete time Markov chain families: Modeling and verification of probabilistic software product lines. In *SPLC*, volume 2, pages 34–41. ACM, 2013.
29. G. N. Rodrigues *et al.* Modeling and verification for probabilistic properties in software product lines. In *HASE*, pages 173–180. IEEE, 2015.
30. C. Dubslaff, C. Baier, and S. Klüppelholz. Probabilistic model checking for feature-oriented systems. In *AOSD*, volume 8989 of *LNCS*, pages 180–220, 2015.
31. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591, 2011.
32. P. Chrszon, C. Dubslaff, S. Klüppelholz, and C. Baier. Family-based modeling and analysis for probabilistic systems – featuring PROFEAT. In *FASE*, volume 9633 of *LNCS*, pages 287–304, 2016.
33. M. H. ter Beek, A. Legay, A. Lluch Lafuente, and A. Vandin. Statistical analysis of probabilistic models of software product lines with quantitative constraints. In *SPLC*, pages 11–15. ACM, 2015.
34. M. H. ter Beek, A. Legay, A. Lluch Lafuente, and A. Vandin. Statistical model checking for product lines. In *ISoLA*, volume 9952 of *LNCS*, pages 114–133, 2016.
35. J. C. Bradfield and C. Stirling. Modal logics and μ-calculi: An introduction. In *Handbook of Process Algebra*, chapter 4, pages 293–330. Elsevier, 2001.
36. J. F. Groote and R. Mateescu. Verification of temporal properties of processes in a setting with data. In *AMAST*, volume 1548 of *LNCS*, pages 74–90, 1999.
37. J. F. Groote and T. A. C. Willemse. Model-checking processes with data. *Sci. Comput. Program.*, 56(3):251–273, 2005.
38. H. Zantema and J. C. van de Pol. A rewriting approach to binary decision diagrams. *J. Log. Algebr. Program.*, 49(1-2):61–86, 2001.
39. J. Kramer, J. Magee, M. Sloman, and A. Lister. CONIC: an integrated approach to distributed computer control systems. *IEE Proc. E*, 130(1):1–10, 1983.
40. M. H. ter Beek, E. P. de Vink, and T. A. C. Willemse, 2016. `https://www.dropbox.com/s/ufw2szfm7uha1md/experiments.zip?dl=0`.
41. D. Giannakopoulou and J. Magee. Fluent model checking for event-based systems. In *ESEC/FSE*, pages 257–266. ACM, 2003.
42. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.