Family-Based Model Checking of SPL based on mCRL2

Extended Abstract

Ziad Ben Snaiba
Eindhoven University of Technology
z.b.snaiba@student.tue.nl

Erik P. de Vink*
Eindhoven University of Technology
Centrum Wiskunde Informatica
evink@win.tue.nl

Tim A.C. Willemse
Eindhoven University of Technology
timw@win.tue.nl

ABSTRACT

We discuss how the general-purpose model checker mCRL2 can be used for family-based verification of behavioral properties of software product lines. This is achieved by exploiting a feature-oriented extension of the modal $\mu$-calculus for the specification of SPL properties, and for its model checking by encoding it back into the logic of mCRL2. Using the example of the well-known minepump SPL an illustration of the possibilities of the approach is given.

CCS CONCEPTS

• Theory of computation → Logic and verification; Verification by model checking. • Software and its engineering → Software product lines;

KEYWORDS

Software Product Lines, Family-based model checking, mCRL2

ACM Reference format:

1 INTRODUCTION

Research into formal methods and analysis tools for the specification and verification of models of SPL seeks to develop methods and techniques to substantiate and guarantee correct behaviour, cf. [5, 17]. When it comes to the validation of behavioral properties, model checking, an established approach for software systems in general [7], is a viable option for the SPL setting. Regarding the actual use of model checkers, previous work has gone into two directions: (i) the development of SPL-dedicated model checkers adapting non-SPL algorithms at the implementation level to the SPL case, e.g. [10, 12], (ii) the application of general-purpose model checkers by capturing SPL specifications at the modelling level, e.g. [2, 8], as advocated in [1]. This paper falls in the latter category.

Inspired by [8, 9] and building on [3], we extended the modal $\mu$-calculus $\mu L$, the logic used for property specification in the mCRL2 toolset, to incorporate feature expressions to capture product families, resulting in the logic $\mu L_F$ [4]. The modalities of $\mu L$ allowing formulas $\langle a \rangle \varphi$ and $[a] \varphi$ are replaced in $\mu L_F$ to specify formulas $\langle a \rangle_{\gamma} \varphi_1$ and $[a]_{\gamma} \varphi_1$. Aiming at a family-based analysis, the semantics of the may-construct of $\mu L_F$, i.e. of $\langle a \rangle_{\gamma} \varphi_1$, requires the family of products under consideration to execute as a whole. A transition witnessing the possibility to execute the action $a$ should be eligible according to the feature expression $\gamma$ for the complete family.

As a consequence, the logical connection between families and their constituent products is weakened: If $\mu L_F$-formula $\varphi_1$ is proven to hold for a family of products $P$, then the $\mu L$ ‘projection’ of the formula with respect to a product $p$ in $P$ holds as well. However, the converse does not hold in general. If $\varphi_1$ doesn’t hold for $P$ we cannot conclude that it doesn’t hold for all products $p$ in the family $P$ either. The key to the family-based model checking problem is to split $P$ into two disjoint subfamilies $P_0$ and $P_0$ such that products in $P_0$ satisfy $\varphi_1$ and products in $P_0$ satisfy $\lnot \varphi_1$ (the latter to be proven by a work-around). In order to actually execute the above family-based strategy we need to translate the model checking problem for $\mu L_F$ to the setting of the mCRL2 tool suite. Based on a validity-preserving encoding from $\mu L_F$ back into $\mu L$, we have been able to do so [18].

The rest of the paper is organized as follows. After a general outline of the mCRL2 toolset in Section 2 and a discussion of the feature-oriented $\mu$-modal calculus in Section 3, system-level modelling of SPL and specification of behavioral properties are discussed in Section 4, more detailed regarding the use of mCRL2 compared to the reference paper [18]. Section 5 reports on a case study for applying mCRL2 to family-based model checking for the case of the well-known SPL benchmark [11] of the minepump. Section 6 wraps up with the outlook for the proposed approach.

2 THE MCRL2 TOOLS

The mCRL2 toolset is an industrial-strength model checking system designed to reason about distributed and concurrent systems. Its development, building on its predecessor $\mu$CRL, goes back to 2002 and is actively maintained. mCRL2’s modelling language, like its predecessor’s, aims to not restrict the expressiveness of the user. For example, besides several built-in data types, the user is free to add her/his own ADTs. Currently, the mCRL2 toolset consists of over 60 tools that together facilitate modelling, visualization, simulation, minimization, and model checking of complex systems. See the tool’s website1 for download and the overview [13] for a discussion of casestudies in several application areas.

The typical workflow for using mCRL2 for verification purposes is to start from a system modeled with mCRL2’s process language and a property specified in a variant of Kozen’s modal $\mu$-calculus

1http://www.mcrl2.org
extended with data. First, with the tool ncr1221ps, the process model is translated into a so-called linear process, the internal representation format of the toolset. Given this representation various transformations, e.g., lps2bool to clean up vacuous enumerations, can be applied for analysis, reduction or visualization purposes. In particular, given a property for verification, a so-called parametrized Boolean equation system or pbes can be generated by calling lps2pbes. A solution for the pbes, which can be automatically found by the Gaussian elimination tool pbes2bool or parity-game based techniques like the tool pbespsolve (for better performance), directly answers the question whether the modeled system satisfies the specified property.

Appealing of nCRL2’s process language and specification logic is the ability to include data. For SPL, this allows to deal with variability and attributes at the modelling level. In particular, a feature transition system or FTS [11], which can represent an SPL in a compact way, can be captured in nCRL2 as a process, moving from one state to another, while taking transitions that are labeled with an action carrying a feature expression as a parameter. A feature expression (a Boolean expression over features) determines for which products the transition is allowed, viz. for those products that satisfy the feature expression, and for which products it is not. Also, attributes can be taken into account, for example as parameters of a process, which can be communicated to other processes or can be checked against to decide from which transitions to choose [3].

The modal logic of nCRL2, referred to as $\mu L$, fully respects the fine-grained description of behavior of the modeled processes. In particular, as we shall see, feature expressions can be incorporated in the logic. Attributes, possibly dynamic, can be reasoned about like in first-order predicate calculus since the logic provides universal and existential quantification over sorts. Finally, $\mu L$ offers regular expressions over (possibly parametrized) actions as an abbreviation mechanism for a compact description of sets of paths. However, the expressiveness of the modal $\mu$-calculus with data comes at a price. One easily can write down formulas of which the semantics is subtle and difficult to assess.

3 FEATURE-ORIENTED MODAL LOGICS

Using constructs of the modal $\mu$-calculus with data, like quantification over data types, feature expressions can be directly expressed in $\mu L$. However, to more conveniently exploit nCRL2 for SPL verification, the logic $\mu L_f$ was proposed in [4] with the aim to hide the burden of the treatment of features in $\mu L$ for property specification from the user. Inspired by the feature extensions of LTL and CTL in [8, 9], the logic $\mu L_f$ was augmented with feature expressions. The resulting logic $\mu L_f$ includes ternary constructs $(a)\gamma \phi_f$ and $[a]\phi_f$, involving an action $a$, a feature expression $\gamma$, and a $\mu L$-formula $\phi_f$, generalizing the standard constructs $(a)\psi$ and $[a]\psi$, involving an action $a$ and a $\mu L$-formula $\phi$.

A formula $\phi_f$ of $\mu L_f$, expressing a property at the family-level, is to be interpreted over an FTS $F$. For an individual product $p$ (as set of features) we can consider the projection $\text{proj}(p, \phi_f)$ and interpret it over the LTS $F|p$, intuitively the ‘view’ on $F$ from the perspective of the product $p$. The FTS $F$ and the LTS $F|p$ share the same states, but a transition $s \xrightarrow{a|L} t \in F$ will be represented by a transition $s \xrightarrow{a|L} t \in F|p$ only if the product $p$ satisfies the feature expression $\gamma$. No transition is included in $F|p$ if $p$ doesn’t satisfy $\gamma$.

As to consistency, the logic $\mu L_f$ comes with a semantics such that a $\mu L_f$ formula $\psi_f$ holds in a state $s$ for the one-product family $[p]$ with respect to the FTS $F$, exactly when the $\mu L$ formula $\text{proj}(p, \psi_f)$ holds in $s$ with respect to the LTS $F|p$.

However, in order to take advantage of the commonality within a product family $P$, the semantics of formulas $(a)\gamma \phi_f$ is special. We have that the family $P$ satisfies the formula $(a)\gamma \phi_f$ in a state $s$ iff there exists a transition $s \xrightarrow{a|L} t$, in $F$ such that $\psi_f$ holds in $t$ as usual, but moreover, the transition is required to be possible for all products in $P$ at the same time, i.e. each $p$ in $P$ satisfies $\gamma$. Thus, the transition witnessing $(a)\gamma \phi_f$ should be applicable simultaneously, for all products of the family under verification. So, $(a)\gamma \phi_f$ fails to hold for family $P$ in state $s$, if, for example, for some of the products in $P$ a transition to a state $t_1$ satisfying $\psi_f$ is possible (but not for all), while for other products in $P$ another transition to a state $t_2$ satisfying $\psi_f$ is possible (but again, not for all). The semantics of formulas $(a)\gamma \phi_f$ is standard: it holds in a state $s$ for a family $P$ if, for each transition $s \xrightarrow{a|L} t$ in $F$ that can be taken by any non-empty subfamily of products of $P$ (all or some), we must have that $\psi_f$ is satisfied.

A consequence of all this, in particular our choice of giving meaning to $(a)\gamma \phi_f$, is that satisfaction propagates from the family-level to the product-level for negation-free formulas, i.e.,

$$s, P \models F \phi_f \implies \forall p \in P : s, p \models F \text{proj}(p, \phi_f)$$

for $\phi_f \in \mu L_f$ negation-free, but, in general, not the other way around [4]. In particular, if $s, P \not\models F \phi_f$ we cannot conclude $s, p \not\models F \text{proj}(p, \phi_f)$ for all products $p$ in the family $P$. It is for this reason, that for family-based model checking a ‘double-rail’ approach needs to be applied.

Given a negation-free formula $\phi_f$ of $\mu L_f$ we consider its so-called complement $\phi_f^\bot$, which is obtained by so-called flipping, i.e. changing may-modalities into must-modalities and vice versa, and similar for all other constructs. As counterpart of the implication of (1), we have for $\phi_f^\bot$ negation-free,

$$s, P \models F \phi_f^\bot \implies \forall p \in P : s, p \not\models F \text{proj}(p, \phi_f)$$

Thus, refuting $\phi_f$ for each product $p$ in the family $P$ is achieved if $\phi_f^\bot$ can be proven for the family $P$ as a whole. Therefore, when facing the model checking problem of $\phi_f$ for a family of products $P$ we consider two derived problems, viz. $s, P_0 \models F \phi_f$ and $s, P_0 \models F \phi_f^\bot$ for two disjoint subfamilies $P_0$ and $P_0$ of $P$. For, if we can establish this, we are guaranteed formula $\phi_f$ holds in state $s$ for the products in $P_0$ by implication (1), while they do not hold for products in $P_0$ by implication (2). In the next section we discuss how to find an appropriate split-up of a family $P$ into $P_0$ and $P_0$.

Closing this section, a final remark regarding $\mu L_f$ and the proposals of fLTL [9] and fCTL [8]. In $\mu L_f$ we have feature expressions as part of the modalities themselves. In a way the modalities are made relative to the products that satisfy a specific feature expression. So, contrary to fLTL and fCTL that add only a prefix-quantification construct $[\gamma]\psi$ for $\psi$ in LTL and CTL, respectively, that cannot be nested, product-restrictions in $\mu L_f$ are much more fine-grained. Besides this, it is noted that LTL and CTL are incomparable and strict sub-logics of the modal $\mu$-calculus [6]. The observations carry
over directly to the feature-based extensions, making $\mu L_4$ a stronger feature logic than the other proposals.

4 PUTTING MCRL2 TO WORK

In order to exploit the mCRL2 toolset for family-based model checking of $\mu L_4$-formulas over FTS we have implemented the following approach. We start from an FTS $F$ modelling the SPL in mCRL2. The FTS is represented by a single process $F(s:\Nat)$ with a parameter $s$ representing the states of the FTS. As we see, one can use the built-in type $\Nat$ of natural numbers to identify the states. Next, the formula $\phi_f$ in $\mu L_4$ to be verified needs to be encoded into $\mu L$, the specification formalism of mCRL2. For this we follow the translation scheme $\mathcal{tr}$, as given in [18], that takes a feature-expression $\chi$ specifying a family of products and the formula $\phi_f$ to produce a formula in $\mu L$. Typical clauses of this recursive scheme include

$$\mathcal{tr}(\chi, \langle a \rangle \phi_f) = (\chi \Rightarrow y) \land \exists v. (a\langle v \rangle)((\chi \Rightarrow v) \land \mathcal{tr}(\chi, \mu X. \phi_f) = \mu X(v \Rightarrow \chi). \mathcal{tr}(v, \phi_f)$$

for the case of the may-modality and the least-fixed point construction. Based on a grammar for $\mu L_4$ we have used ANTLR [16] with appropriate walkers to implement the recursive scheme $\mathcal{tr}$.

The next and most challenging step, however, is the split-up into positive and negative sub-families. Currently this is done manually guided. From the state $s$ under consideration we collect the conjunctions of feature expressions decorating the transitions of all maximal paths in the FTS. Note, although these paths may be infinite in the presence of cycles, the conjunctions are not; there are finitely many features, reduction modulo propositional equivalence yields a finite number of states, to be it potentially large.

For efficiency, the product families involved are coded as BDDs over features rather than Boolean combinations. The usual operations one desires for BDD manipulation are straightforwardly and efficiently implemented using mCRL2’s support for ADTs and associated rewritings. The general formula pattern we use is

forall $P: FExpr.val(P \in \{BO01, \ldots, BO0m\} \Rightarrow \mathcal{tr}(P, \phi f)$ \&
forall $P: FExpr.val(P \in \{BO01, \ldots, BO0m\} \Rightarrow \mathcal{tr}(P, \phi f_c)$

where $\phi f$ and $\phi f_c$ are the formula $\phi_f$ itself and its complement $\neg\phi_f$, respectively. Assuming the mCRL2 model of the FTS is provided by the file fts.mcr12, the property to be verified by the file prop.mcf, the run script we use is the following.

```bash
#!/bin/bash
echo 'Linearising'
echo 'mcr122lps:' \> wallclock.txt
  ( time mcr122lps -nf -t timings.txt fts.mcr12 |
    lpssume1m > fts.lps ; )
  2>> wallclock.txt
echo 'lps2pbes:' \> wallclock.txt
echo 'Converting to PBEs'
  ( time lps2pbes -t timings.txt -f prop.mcf fts.lps |
    pbesconstelm -t timings.txt |
    pbesparelm -t timings.txt > fts.pbes ; )
  2>> wallclock.txt
echo 'Solving PBEs'
echo 'pbespgsolve:' \> wallclock.txt
  ( time pbespgsolve -t timings.txt -s recursive fts.pbes ; )
  2>> wallclock.txt
```

following the general workflow described earlier and invoking various optimization tools, like lpssume1m for sum elimination from a linear process, available in the toolset.

5 CASE STUDY

To give an impression of what can be handled with the approach sketched in the previous section, we discuss the minepump family model, an SPL benchmark proposed in [11] based on [15].

The minepump family model describes an SPL of a configurable software system controlling a pump for mine drainage. It is used to pump water out of a mine shaft, but it should not do this in the presence of a dangerously high level of methane gas. Therefore, the pump communicates with a number of sensors measuring the water and methane levels. For the SPL we consider 7 independent optional features [9] for a total of $2^7 = 128$ product variants, with features $Ct$, $Cp$, $Ma$, $Mq$, $Ll$, $Ll$, and $Lh$. The minepump model in fPROM is distributed with ProVeracles.²

A small number of results is collected in Table 1. (See [18] for more.) The properties selected come from three classes of formulas: (i) plain LTL formulas, formula $\phi_1$ and $\phi_2$; (ii) plain CTL formulas, formula $\phi_3$ and $\phi_4$; (iii) feature-enriched formulas, formula $\phi_5$ and $\phi_6$.³

Absence of deadlock, property $\phi_1$, expresses that for each path there exists an extension of the path. Here, the regular expression $true$ represents any choice of action; the feature expression $T$ represents the family of all products. For $\phi_2$ the subfamily $P_0$ is the complete set of products and $P_0$ is empty. More precisely, $P_0$ consists of 12 sub-families given by the feature expressions $Ct \land Cp \land Ma \land Mq$, where $X = X$ or $\neg X$, yielding eight families, and four further families yielded by the product sets given by $\neg Ct \land Cp \land Ma$. These families are disjoint and contain 8 and 16 products, respectively. Thus, for all 128 products deadlock does not occur. Formula $\phi_2$ has been borrowed from [9] and is specific for the minepump model. It is the $\mu$-calculus equivalent of the LTL-formula $\neg(\bigcirc C(\neg pumpOn \land methane))$. The subformula $\mu X. [R] X$ for $R = \neg(pumpStop + methaneLower)$ expresses that there is no infinite repetition of actions different from $pumpStop$ and $methaneLower$. Thus, for the subformula $\mu X. [R] X$ to hold, $pumpStop$ or $methaneLower$ must occur after a finite number of steps. The split-up is based on the feature expressions $\neg(Ct \land Lh)$ for $P_0$ and $\neg(Mq \land Ct \land Lh)$ for $P_0$. The approach of [9] focusses on LTL properties rather than on the wider class of $\mu$-calculus properties.

Formula $\phi_3$ being a so-called reset property is a proper CTL formula, not expressible in LTL. The next property, formula $\phi_4$ states that it is excluded that the pump will start at a low water level, if after detection of this the water level didn’t increase to a normal or high level. Formulas $\phi_3$ and $\phi_4$ use the same split-up of $P_0$ in 12 subfamilies and having $P_0$ empty, as is the case for absence of deadlock, formula $\phi_1$. Since no may-modality is present in $\phi_4$, the gain in the runtime of the product-based model checking (column ‘one’) vs. the runtime of the family-based model checking (column ‘all’) is to be attributed to the reduction of overhead.

²https://projects.info.unamur.be/fs/provelines/
³Experiments were run on a Macbook Pro OS X 10.11.3 with a 2.8 Ghz Intel® Core™ 5 CPU, (2 cores) (1 processor, 2 cores) with 16 Gb of RAM (1600 Mhz DDR3, 256Kb L2 Cache per core, 4Mb L3 Cache), and software revision 14493 from the SVN for mCRL2.
The final two properties, formulas $\varphi_5$ and $\varphi_6$ are feature-enriched formulas. The same split-up as before has been used for their verification. In general we see, when going from product-based to family-based verification speed-ups of a factor 5 to 20, with the must-formula $\varphi_1$ being an exception. A more extensive discussion of the case study of applying family-based model checking to the minepump using the mCRL2 toolset is reported in [18].

### 6 CONCLUSION

The mCRL2 toolset can be used profitably for model checking of systems with variability. With an SPL given as FTS and properties specified in the feature-oriented modal logic $\mu L_f$ this can be done semi-automatically. At present, the partitioning of the complete set of products requires human interaction. However, in line with recent work with LTL model checking for SPL with SPIN [14], we are currently exploring how to automate this splitting or reuse of successful splittings earlier collected. So far, initial case studies reveal a substantial decrease in runtimes when moving from product-based to family-based analysis. Further investigations need to be conducted to understand the trade-offs between explicit user-guided family-based approaches and general system-decided distribution of instances of individual products.

### REFERENCES


[18] M. Huisman and J. Rubin (Eds.). LNCS 7795, 199–213. DOI: http://dx.doi.org/10.1007/978-3-642-36742-7_15