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 μ-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:
Ziad Ben Snaiba, Erik P. de Vink, and Tim A.C. Willemse. 2017. Family-Based Model Checking of SPL based on mCRL2. In *Proceedings of 21st International Conference on Software Product Line, Seville, Spain, 25–29 September, 2017 (SPLC 2017),* 4 pages.
https://doi.org/10.475/123_4

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 specifics 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 μ-calculus μL, the logic used for property specification in the mCRL2 toolset, to incorporate feature expressions to capture product families, resulting in the logic μL_{f} [4]. The modalities of μL allowing formulas $\langle a\rangle\varphi$ and $[a]\varphi$ are replaced in μL_{f} to specify formulas $\langle a|\gamma\rangle\varphi_{\mathrm{f}}$ and $[a|\gamma]\varphi_{\mathrm{f}}$. Aiming at a family-based analysis, the semantics of the may-construct of μL_{f}, i.e. of $\langle a|\gamma\rangle\varphi_{\mathrm{f}}$, 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 γ for the complete family.

As a consequence, the logical connection between families and their constituent products is weakened: If μL_{f}-formula φ_{f} is proven to hold for a family of products P, then the μL 'projection' of the formula with respect to a product p in P holds as well. However, the converse does *not* hold in general. If φ_{f} 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 $P_{\oplus}$ and $P_{\ominus}$ such that products in $P_{\oplus}$ satisfy φ_{f} and products in $P_{\ominus}$ satisfy $\neg\varphi_{\mathrm{f}}$ (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 μL_{f} to the setting of the mCRL2 tool suite. Based on a validity-preserving encoding from μL_{f} back into μL, 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 μ-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 TOOLSET

The mCRL2 toolset is an industrial-strength model checking system designed to reason about distributed and concurrent systems. Its development, building on its predecessor *μCRL*, 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 website[1] 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 μ-calculus

*Corresponding author.

SPLC 2017, 25–29 September, 2017, Seville, Spain

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

[1] http://www.mcrl2.org

extended with data. First, with the tool `mcrl22lps`, 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. `lpssumelm` 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 `pbespgsolve` (for better performance), directly answers the question whether the modeled system satisfies the specified property.

Appealing of `mCRL2`'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 `mCRL2` 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 `mCRL2`, referred to as μL, 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, μL 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 μ-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 μ-calculus with data, like quantification over data types, feature expressions can be directly expressed in μL. However, to more conveniently exploit `mCRL2` for SPL verification, the logic μL_{f} was proposed in [4] with the aim to hide the burden of the treatment of features in μL for property specification from the user. Inspired by the feature extensions of LTL and CTL in [8, 9], the logic μL was augmented with feature expressions. The resulting logic μL_{f} includes ternary constructs $\langle a|\gamma\rangle\varphi_{\mathrm{f}}$ and $[a|\gamma]\varphi_{\mathrm{f}}$, involving an action a, a feature expression γ, and a μL_{f}-formula φ_{f}, generalizing the standard constructs $\langle a\rangle\varphi$ and $[a]\varphi$, involving an action a and a μL-formula φ.

A formula φ_{f} of μL_{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 $proj(p, \varphi_{\mathrm{f}})$ 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 $s \xrightarrow{a|\gamma} t$ in F will be represented by a transition $s \xrightarrow{a} t$ in $F|p$ only if the product p satisfies the feature expression γ. No transition is included in $F|p$ if p doesn't satisfy γ. As to consistency, the logic μL_{f} comes with a semantics such that a μL_{f} formula φ_{f} holds in a state s for the one-product family $\{p\}$ with respect to the FTS F, exactly when the μL formula $proj(p, \varphi_{\mathrm{f}})$ 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 $\langle a|\gamma\rangle\varphi_{\mathrm{f}}$ is special. We have that the family P satisfies the formula $\langle a|\gamma\rangle\varphi_{\mathrm{f}}$ in a state s iff there exists a transition $s \xrightarrow{a|\gamma} t$ in F such that φ_{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 γ. Thus, the transition witnessing $\langle a|\gamma\rangle\varphi_{\mathrm{f}}$ should be applicable simultaneously, for all products of the family under verification. So, $\langle a|\gamma\rangle\varphi_{\mathrm{f}}$ 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 φ_{f} is possible (but not for all), while for other products in P another transtion to a state t_2 satisfying φ_{f} is possible (but again, not for all). The semantics of formulas $[a|\gamma]\varphi_{\mathrm{f}}$ is standard: it holds in a state s for a family P if, for each transition $s \xrightarrow{a|\gamma} t$ in F that can be taken by any non-empty subfamily of products of P (all or some), we must have that φ_{f} is satisfied.

A consequence of all this, in particular our choice of giving meaning to $\langle a|\gamma\rangle\varphi_{\mathrm{f}}$, is that satisfaction propagates from the family-level to the product-level for negation-free formulas, i.e.,

$$s, P \models_F \varphi_{\mathrm{f}} \implies \forall p \in P\colon s, p \models_{F|p} proj(p, \varphi_{\mathrm{f}}) \tag{1}$$

for $\varphi_{\mathrm{f}} \in \mu L_{\mathrm{f}}$ negation-free, but, in general, not the other way around [4]. In particular, if $s, P \not\models_F \varphi_{\mathrm{f}}$ we cannot conclude $s, p \not\models_{F|p} proj(p, \varphi_{\mathrm{f}})$ 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 φ_{f} of μL_{f} we consider its so-called complement φ_{f}^c, 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 φ_{f} negation-free,

$$s, P \models_F \varphi_{\mathrm{f}}^c \implies \forall p \in P\colon s, p \not\models_{F|p} proj(p, \varphi_{\mathrm{f}}) \tag{2}$$

Thus, refuting φ_{f} for each product p in the family P is achieved if φ_{f}^c can be proven for the family P as a whole. Therefore, when facing the model checking problem of φ_{f} for a family of products P we consider two derived problems, viz. $s, P_{\oplus} \models_F \varphi_{\mathrm{f}}$ and $s, P_{\ominus} \models_F \varphi_{\mathrm{f}}^c$ for two disjoint subfamilies $P_{\oplus}$ and $P_{\ominus}$ of P [**?**]. For, if we can establish this, we are guaranteed formula φ_{f} holds in state s for the products in $P_{\oplus}$ by implication (1), while they do not hold for products in $P_{\ominus}$ by implication (2). In the next section we discuss how to find an appropriate split-up of a family P into $P_{\oplus}$ and $P_{\ominus}$.

Closing this section, a final remark regarding μL_{f} and the proposals of fLTL [9] and fCTL [8]. In μL_{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 $[\gamma]\varphi$ for φ in LTL and CTL, respectively, that *cannot* be nested, product-restrictions in μL_{f} are much more fine-grained. Besides this, it is noted that LTL and CTL are incomparable and strict sub-logics of the modal μ-calculus [6]. The observations carry

over directly to the feature-based extensions, making μL_f 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 μL_f-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 φ_f in μL_f to be verified needs to be encoded into μL, the specification formalism of mCRL2. For this we follow the translation scheme tr, as given in [18], that takes a feature-expression χ specifying a family of products and the formula φ_f to produce a formula in μL. Typical clauses of this recursive scheme include

$$tr(\chi, \langle a|\gamma\rangle\varphi_f) = (\chi \Rightarrow \gamma) \wedge \exists v.\langle a(v)\rangle((\chi \Rightarrow v) \wedge tr(\chi \wedge \gamma \wedge v, \varphi_f))$$
$$tr(\chi, \mu X.\varphi_f) = \mu\tilde{X}(v{:=}\chi).tr(v, \varphi_f)$$

for the case of the may-modality and the least-fixed point construction. Based on a grammar for μL_f we have used ANTLR [16] with appropriate walkers to implement the recursive scheme 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; as there are finitely many features, reduction modulo propositional equivalence yields a finite number only, 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 rewriters. The general formula pattern we use is

```
forall P:FExpr.val(P in [BDD1,...,BDDn] => tr(P,phif)
  &&
forall P:FExpr.val(P in [BDD1',...,BDDm'] => tr(P,phif_c)
```

where `phif` and `phif_c` are the formula φ_f itself and its complement φ_f^c, respectively. Assuming the mCRL2 model of the FTS is provided by the file `fts.mcrl2`, the property to be verified by the file `prop.mcf`, the run script we use is the following.

```
#!/bin/bash
echo 'Linearising'
echo 'mcrl22lps:' >> wallclock.txt
  { time mcrl22lps -nf -t timings.txt fts.mcrl2
    | lpssumelm > 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 `lpssumelm` 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*, *Ln*, and *Lh*. The minepump model in `fPROM` is distributed with `ProVeLines`.[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 φ_1 and φ_2; (ii) plain CTL formulas, formula φ_3 and φ_4; (iii) feature-enriched formulas, formula φ_5 and φ_6.[3]

Absence of deadlock, property φ_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 $\top$ represents the family of all products. For φ_1 the subfamily $P_\oplus$ is the complete set of products and $P_\ominus$ is empty. More precisely, $P_\oplus$ consists of 12 sub-families given by the feature expressions $Ct \wedge \tilde{Cp} \wedge \tilde{Ma} \wedge \tilde{Mq}$, where $\tilde{X} = X$ or $\neg X$, yielding eight families, and four further families yielded by the product sets given by $\neg Ct \wedge \tilde{Cp} \wedge \tilde{Ma}$. These families are disjoint and contain 8 and 16 products, respectively. Thus, for all 128 products deadlock does not occur. Formula φ_2 has been borrowed from [9] and is specific for the minepump model. It is the μ-calculus equivalent of the LTL-formula $\neg(\Diamond\Box(pumpOn \wedge methane))$. 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 $\neg(Ct \wedge Lh)$ for $P_\oplus$ and $\tilde{Mq} \wedge Ct \wedge Lh$ for $P_\ominus$. The approach of [9] focusses on LTL properties rather than on the wider class of μ-calculus properties.

Formula φ_3 being a so-called reset property is a proper CTL formula, not expressible in LTL. The next property, formula φ_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 φ_3 and φ_4 use the same split-up of $P_\oplus$ in 12 subfamilies and having $P_\ominus$ empty, as is the case for absence of deadlock, formula φ_1. Since no may-modality is present in φ_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.

[2] https://projects.info.unamur.be/fts/provelines/

[3] Experiments were run on a Macbook Pro OS X 10.11.3 with a 2.8 Ghz Intel® Core™ i7 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.

	property in μL_f	true/false	one	all
φ_1	*Absence of deadlock.* $[\,\mathtt{true}^*\,|\top\,]\,\langle\,\mathtt{true}|\top\,\rangle\,\top$	128/0	10.02	2.07
φ_2	*It cannot happen that the pump runs indefinitely in the presence of methane.* $[\,\mathtt{true}^*|\top\,]\;(\,[\,pumpStart\,.\,(\,\neg pumpStop\,)^*.\,methaneRise\,|\top\,]\,\mu X.\,[R]X\,)$ $\wedge\,(\,[\,methaneRise\,.\,(\,\neg methaneLower\,)^*.\,pumpStart\,|\top\,]\,\mu X.\,[R]X\,)$ for $R = \neg(\,pumpStop + methaneLower\,)$	96/32	17.26	0.86
φ_3	*The controller can always eventually receive/read a message,* *i.e. it can get back to the initial state from any state.* $[\,\mathtt{true}^*|\top\,]\,\langle\,\mathtt{true}^*.\,receiveMsg\,|\top\,\rangle\,\top$	128/0	18.36	2.40
φ_4	*Invariantly the pump is not started when the low water level signal fires.* $[\,\mathtt{true}^*.\,lowLevel\,.(\neg(normalLevel + highLevel))^*\,.\,pumpStart\,|\top\,]\,\bot$	128/0	5.67	3.05
φ_5	*Products with feature Ct can switch on the pump.* $\langle\,\mathtt{true}^*.\,pumpStart\,|\,Ct\,\rangle\,\top$	32/96	6.49	0.31
φ_6	*Products with features Ct, Ma, and Lh can start the pump upon a high water level,* *but products without feature Lh cannot.* $[\,\mathtt{true}^*|\top\,]\,(([\,highLevel|\,Ct \wedge Ma \wedge Lh\,]\langle\,\mathtt{true}^*.pumpStart|\top\,\rangle\top) \wedge [\,pumpStart|\neg Lh\,]\bot)$	128/0	13.35	3.36

Table 1: Properties, results, and run times (in sec.) of product-based (*one-by-one*) and family-based (*all-in-one*) verification

The final two properties, formulas φ_5 and φ_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 φ_4 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 μL_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 casestudies 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

[1] J.M. Atlee, S. Beidu, N.A. Day, F. Faghih, and P. Shaker. 2013. Recommendations for improving the usability of formal methods for product lines. In *FormaliSE 2013, San Francisco*. IEEE Computer Society, 43–49. DOI: http://dx.doi.org/10.1109/FormaliSE.2013.6612276

[2] M.H. ter Beek, A. Fantechi, S. Gnesi, and M. Mazzanti. 2015. Using FMC for Family-Based Analysis of Software Product Lines. In *Proc. SPLC*, D. Schmidt (Ed.). ACM, 432–439. DOI: http://dx.doi.org/10.1145/2791060.2791118

[3] M.H. ter Beek and E.P. de Vink. 2014. Using mCRL2 for the analysis of software product lines. In *FormaliSE workshop at ICSE'14*, S. Gnesi and N. Plat (Eds.). IEEE, 31–37. DOI: http://dx.doi.org/10.1145/2593489.2593493

[4] M.H. ter Beek, E.P. de Vink, and T.A.C. Willemse. 2016. Towards a feature mu-calculus targeting SPL verification. In *Proc. FMSPLE 2016*, J. Rubin and T. Thuem (Eds.). EPTCS, 15pp. DOI: http://dx.doi.org/10.4204/EPTCS.206.6

[5] P. Borba, M.B. Cohen, A. Legay, and A. Wasowski. 2013. Analysis, Test and Verification in The Presence of Variability (Dagstuhl Seminar 13091). *Dagstuhl Reports* 3, 2 (2013), 144–170. DOI: http://dx.doi.org/10.4230/DagRep.3.2.144

[6] J. Bradfield and C. Stirling. 2006. Modal mu-calculi. In *The Handbook of Modal Logic*, P. Blackburn, J. van Benthem, and F. Wolter (Eds.). Elsevier, 721–756.

[7] E.M. Clarke, E.A. Emerson, and J. Sifakis. 2009. Model checking: algorithmic verification and debugging. *Commun. ACM* 52, 11 (2009), 74–84. DOI: http://dx.doi.org/10.1145/1592761.1592781

[8] A. Classen, M. Cordy, P. Heymans, A. Legay, and P.-Y. Schobbens. 2014. Formal Semantics, modular specification, and symbolic verification of product-line behaviour. *Science of Computer Programming* 80, B (2014), 416–439. DOI: http://dx.doi.org/10.1145/2499777.2499781

[9] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin. 2013. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Trans. Software Eng.* 39, 8 (2013), 1069–1089. DOI: http://dx.doi.org/10.1109/TSE.2012.86

[10] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay. 2011. Symbolic model checking of software product lines. In *Proc. ICSE 2011, Honolulu*. IEEE, 321–330. DOI: http://dx.doi.org/10.1145/1985793.1985838

[11] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. 2010. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *Proc. ICSE 2010*. 335–344. DOI: http://dx.doi.org/10.1145/1806799.1806850

[12] M. Cordy, A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay. 2013. ProVeLines: a product line of verifiers for software product lines. In *Proc. SPLC 2013, Tokyo*. ACM, 141–146. DOI: http://dx.doi.org/10.1145/2499777.2499781

[13] S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, W. Wesselink, and T.A.C. Willemse. 2013. An Overview of the mCRL2 Toolset and Its Recent Advances. In *Proc. TACAS 2013*, N. Piterman and S.A. Smolka (Eds.). LNCS 7795, 199–213. DOI: http://dx.doi.org/10.1007/978-3-642-36742-7_15

[14] A.S. Dimovski and A. Wasowski. 2017. Variability-Specific Abstraction Refinement for Family-Based Model Checking. In *Proc. FASE 2017*, M. Huisman and J. Rubin (Eds.). LNCS 10202, 406–423. DOI: http://dx.doi.org/10.1007/978-3-662-54494-5_24

[15] J. Kramer, J. Magee, M. Sloman, and A. Lister. 1983. CONIC: an integrated approach to distributed computer control systems. *IEE Proc. E* 130, 1 (1983), 1–10. DOI: http://dx.doi.org/10.1049/ip-e.1983.0001

[16] T. Parr. 2012. *The definitive ANTLR 4 reference*. The Pragmatic Programmer.

[17] I. Schaefer and R. Hähnle. 2011. Formal Methods in Software Product Line Engineering. *IEEE Computer* 44, 2 (2011), 82–85. DOI: http://dx.doi.org/10.1109/MC.2011.47

[18] Beek M.H. ter, E.P. de Vink Vink, and T.A.C. Willemse. 2017. Family-Based Model Checking with mCRL2. In *Proc. FASE 2017*, M. Huisman and J. Rubin (Eds.). LNCS 10202, 387–405. DOI: http://dx.doi.org/10.1007/978-3-662-54494-5_23