Design and Validation of Variability in Product Lines

Patrizia Asirelli
ISTI–CNR, Pisa, Italy
asirelli@isti.cnr.it

Maurice H. ter Beek
ISTI–CNR, Pisa, Italy
terbeek@isti.cnr.it

Alessandro Fantechi
Università di Firenze, Italy
fantechi@dsi.unifi.it

Stefania Gnesi
ISTI–CNR, Pisa, Italy
gnesi@isti.cnr.it

Franco Mazzanti
ISTI–CNR, Pisa, Italy
mazzanti@isti.cnr.it

ABSTRACT

We propose an emerging solution technique, pushing the application of model-checking techniques to the design and validation of variability in a product line (PL), mainly aimed at those industrial domains where model-based development is adopted for the development of safety-critical systems.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation, Verification

Keywords

Product Lines, Variability, Temporal logic, Model checking

1. INTRODUCTION

A common problem addressed by many companies working in domains regulated by standards, such as safety-critical domains like avionics and railways, and that slows down the application of PL-based technologies in these domains, is the high cost of validation and certification efforts. In such domains, every product has to pass a costly certification stage, even in case it belongs to an established family of products for which certification efforts have already been performed.

A significant advancement in certification was the introduction of model-checking techniques, which allow properties (e.g., safety properties) to be proved over (a model of) the system to be certified. Model checking in itself obviously does not resolve the repeated certification problem in a PL: satisfaction of a property by a product in general does not imply the satisfaction of the same property by a different product. A possible way out is to investigate how to validate, by proper model-checking techniques, that possible variability that is present in a product family does not violate any desired property. This would allow one to formally prove that whenever certain (safety) properties hold for a product family, they continue to hold for every product generated according to the variability rules of that family.

For this purpose, we launched a research effort to, on the one hand, investigate the most promising existing modelling structures that allow (behavioural) variability to be described and product derivation to be defined, and, on the other hand, develop a proper temporal logic that can be interpreted over such structures and which can express interesting properties over families and products. We recently proposed a temporal logic for modelling variability in product families [2, 3] by taking advantage of the way in which deontic logic formalises concepts like violation, obligation, permission, and prohibition. In [4, 5], we extended this logic to capture also the behaviour of product families, using modal transition systems as the underlying semantic model, and we developed efficient algorithms to derive valid products of families as well as to verify properties over products and families, based on existing model checkers.

We refer to [2, 3] and in particular [4] for detailed references to the concepts, models, logics, etc., used throughout this paper, while related work is briefly discussed in Sect. 6.

2. TECHNICALITIES: MTS

A Modal Transition System (MTS) is a Labelled Transition System (LTS) which distinguishes *may* and *must* transitions, which can thus model optional or mandatory features of a PL in a natural way. Technically, it is a directed edge-labelled graph whose nodes represent states and whose edges model transitions (labelled with the actions executed as the result of a state change). Formally, all must transitions are also may transitions. A sequence of state changes is a path.

Figure 1 is an example of an MTS: its dashed edges are (possible) *may* transitions that are not must transitions, while its solid edges are (mandatory) *must* transitions. Unfortunately, MTSs cannot fully characterize the constraints regarding *alternative* features nor those regarding the *requires* and *excludes* inter-feature relations [4].

To model product families as MTSs one thus needs a 'translation' from features to actions (not necessarily a one-to-one mapping) and the introduction of a behavioural relation (temporal ordering) among them. A family's products are then considered to differ w.r.t. the actions they are able to perform in any given state of the MTS. This means that the MTS of a product family has to accommodate all the possibilities desired for each derivable product, predicating on the choices that make a product belong to that family.

PLEASE'11, May 22-23, 2011, Waikiki, Honolulu, HI, USA

3. RUNNING EXAMPLE

We consider a family of section controllers from the domain of railway interlocking systems. An *interlocking* is the safety-critical system that controls the movement of trains in a station and between adjacent stations. It monitors the status of the objects in the railway yard (e.g., points, switches, track circuits) and allows or denies the safe routing of trains.

The controller we consider approaches the interlocking problem from a *geographical* and distributed point of view [6] as it associates a controller to each prominent geographical entity of a railway line. In particular, it allows or denies the access of a train to a section of a line, thus guaranteeing a minimal distance between consecutive trains. This kind of controller may be subject to different configurations due to the different track layouts that occur in practice: to be generic, we consider a family of section controllers. In order to keep it simple for the sake of the presentation, we consider only two variability options. Moreover, to reduce the complexity for the sake of the presentation, some simplifications are applied to obtain the model that we will present shortly. However, the model is not far from the actual implementation techniques of this class of systems [8].

3.1 Family of Section Controllers

A section controller takes as input an access request to the section (from a train or an operator), and can sense the occupancy of the section by means of a 'track circuit' sensor. The permission to access the section is given by a protection signal. These features are mandatory for any control element of the family. A level crossing (LC) may be present on the section (*optional* feature) and it may come in two *alternative* variants: with or without barriers. A blinking (red) light for the road vehicles is a further *optional* feature which, adopting the typical terminology of feature diagrams, is a feature that *requires* the presence of the LC feature.

The (behavioural) requirements for section controllers are:

- The section is unidirectional (traversed from left to right) with at its entrance a protection signal that only grants permission upon requests (by incoming trains);
- The permission is shown by a green light of a protection signal, and otherwise the signal shows a red light;
- The permission is granted only if the (left) section is free and the next section (on the right) is free as well;
- The section controller can ask the section controller on the right if that section is free (analogously, it can be asked itself from the left section controller if it is free);
- The actual occupancy/freeness of the section is sensed by a track circuit sensor;
- It is supposed (simplifying assumption) that when a train leaves the section, it does so because it has been given the permission to enter the right section (i.e., no backward movement is allowed) and as a result of this event, the track circuit switches from occupied to free;
- A LC may be present with or without barriers;
- In case of an unprotected LC, a mandatory red light for the road vehicles is switched on from when the train approaches until when the train frees the track circuit;

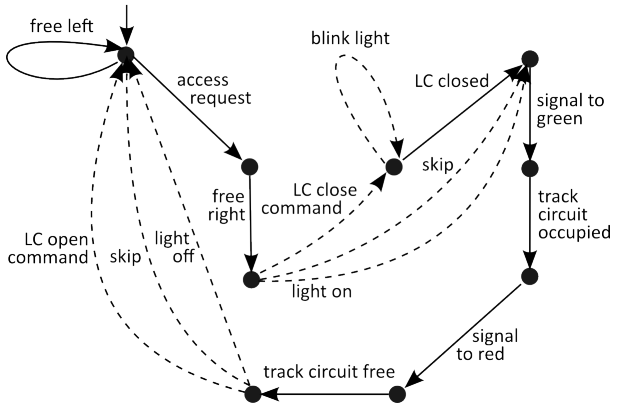

Figure 1: MTS of a family of section controllers.

- In case of a LC with barriers, the controller, when it is asked for an access permission, first issues the closing command to the barriers and then checks whether the LC is actually closed by means of a proper sensor. When the train leaves the section (freeing the track circuit) a command 'open' is sent to the LC which, optionally, is provided with a blinking red light for road vehicles, which blinks while the barriers are closing;

The MTS in Fig. 1 models the family of section controllers. Note that the *free_right* and *free_left* actions have a complementary role in synchronizations between adjacent section controllers: the *free_right* action has to synchronize with the *free_left* action of the section to its right, etc.

4. TECHNICALITIES: vACTL

vACTL is a variability and action-based branching-time temporal logic interpreted over MTSs. It contains the box and diamond modal operators from Hennessy–Milner logic, both with and without a *deontic* interpretation, existential and universal state operators (quantifying over paths) from CTL, and an action-based Until operator, again with and without a deontic interpretation.

Technically, vACTL consists of state formulae (denoted by ϕ), path formulae (denoted by π), and action formulae (boolean compositions of actions, denoted by φ) defined over a set of atomic actions $\{a, b, \ldots\}$. Besides the standard operators of propositional logic (i.e., negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), and implication ($\Longrightarrow$)) it allows the following operators:

- $\langle a \rangle \, \phi$: a next state exists, reachable by a *may* transition that executes action a, in which ϕ holds;
- $[a] \, \phi$: in all next states, reachable by a *may* transition that executes action a, ϕ holds;
- $\langle a \rangle^{\Box} \, \phi$: a next state exists, reachable by a *must* transition that executes a, in which ϕ holds;
- $[a]^{\Box} \, \phi$: in all next states, reachable by a *must* transition that executes a, ϕ holds;
- $E \, \pi$: there exists a path on which π holds;
- $A \, \pi$: on all possible paths, π holds;

- $\phi\ \{\varphi\}\, U\, \{\varphi'\}\ \phi'$: in a state of a path reached by an action satisfying φ', ϕ' holds, whereas ϕ holds in all preceding states and all actions executed meanwhile along the path satisfy φ
- $\phi\ \{\varphi\}\, U^{\Box}\, \{\varphi'\}\ \phi'$: the only difference with the previous operator is that the path entails only *must* transitions

Further operators can now be derived in the usual way. $F\,\phi$ abbreviates $(true\ U\ \phi)$: there exists a future state in which ϕ holds. $AG\,\phi$ abbreviates $\neg EF\,\neg\phi$: in every state on every path, ϕ holds. $F\{\varphi\}\ true$ abbreviates $true\ \{true\}\, U\, \{\varphi\}\ true$: there exists a future state reached by an action satisfying φ. $F^{\Box}\ \phi$, $AG^{\Box}\ \phi$, and $F^{\Box}\ \{\varphi\}\ \phi$ are derived likewise.

vACTL formulae can express properties over product families of the following nature (see below):

- It is allowed that in some products of the family, the action a is never executed (or a feature a never occurs);
- All products of the family should be such that execution of action c is always preceded by that of action b.

vACTL formulae/patterns for expressing such properties are:

- An a is never executed: $\neg\, EF^{\Box}\ \{a\}\ true$;
- Execution of a c is always preceded by that of a b (i.e., it is impossible that a c is not preceded by a b): $\neg\, E\ [\, true\ \{\neg b\}\, U\, \{c\}\ \ true\,]$.

At the same time, the typical constraints that are used to characterize the products allowed by a product family or by one of its subfamilies can be described using the following vACTL templates:

Template ALT Features F1 and F2 are *alternative*:

$$(EF^{\Box}\ \{F1\}\ true \vee EF^{\Box}\ \{F2\}\ true) \wedge \neg\,(EF\ \{F1\}\ true \wedge EF\ \{F2\}\ true)$$

Template EXC Feature F1 *excludes* feature F2:

$$((EF\ \{F1\}\ true) \implies (AG\ \neg\,\langle F2\rangle\ true)) \wedge ((EF\ \{F2\}\ true) \implies (AG\ \neg\,\langle F1\rangle\ true))$$

Template REQ Feature F1 *requires* feature F2:

$$(EF\ \{F1\}\ true) \implies (EF^{\Box}\ \{F2\}\ true)$$

Hence, vACTL is able to complement the behavioural description of an MTS by expressing constraints over the general model of a product family, modelling in this way the constraints that an MTS cannot express.

We appropriately adapted the use of the on-the-fly model checker FMC [22] to enable efficient verification of vACTL formulae. The advantage of an on-the-fly approach is that, depending on the formula, only a fragment of the overall state space might need to be generated and analysed in order to produce the correct result. The basic idea underlying FMC is that, given a state of an MTS, the validity of an vACTL formula in that state is evaluated by analysing the transitions allowed from that state and the validity of a certain subformula in only some of the next reachable states, all this recursively. FMC's on-the-fly model-checking algorithm is of linear complexity.

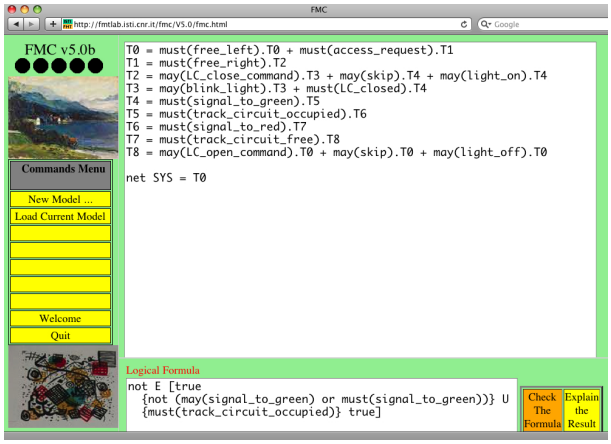

Figure 2: FMC screenshot of section controller.

5. OUR PROPOSAL

We now propose our solution technique for the design and validation of variability in a PL, hoping for feedback on this technique's applicability to current industrial problems.

The idea is to start from a given feature diagram (or simply from a set of requirements) of a product family, as developed by a PL engineer. With the help of proper guidelines, and possibly using high-level formalisms that in the end boil down to MTSs as the underlying semantic models, a PL engineer then produces an MTS model of this product family, together with a set of vACTL formulae. The latter can straightforwardly be obtained by filling in a number of template formulae (so-called temporal logic patterns), like those presented in the previous section.

At this point, FMC can be used to automatically validate the resulting set of properties over the MTS.

The screenshot in Fig. 2 shows the family of section controllers specified in the CCS-like input language of FMC as well as the property "A train can enter the section only if it has been granted permission", which fits the second vACTL formula/pattern above:

$$\neg E\,[true\ \{\neg signal\_to\_green\}\, U\, \{track\_circuit\_occupied\}\ true]$$

This formula can be translated as follows in FMC's syntax:

$$not\ E\ [\, true\ \{not\ (may(signal\_to\_green)\ or\ must(signal\_to\_green))\}\, U\ \{must(track\_circuit\_occupied)\}\ true\,]$$

This is a property that should obviously hold for this family.

Upon verifying this property, FMC produces the result shown in the screenshot in Fig 3: the formula is true over the MTS depicted in Fig. 1, from which we conclude that the property holds for the family of section controllers.

A property that clearly does not hold for the family of section controllers, but that should hold for a subfamily (of products), namely those with a LC, is the property "A train can enter the section only if the LC barriers are closed". Also this property 'fits' the second vACTL formula/pattern:

$$\neg E\ [true\ \{\neg LC\_closed\}\, U^{\Box}\{signal\_to\_green\}\ true] \qquad (1)$$

Figure 3: FMC screenshot of verification result.

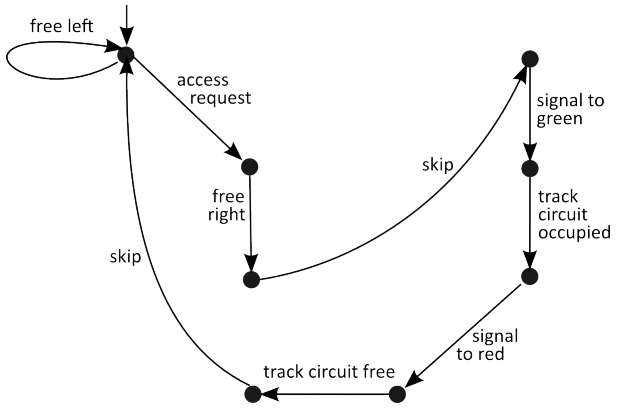

Figure 4: Product of family of section controllers.

Note that, compared to the previous formula, we now use our deontic interpretation of the Until operator to express that we want to limit verification of Formula 1 to subfamilies and products in which the *optional* LC feature is present. This formula can be translated as follows in FMC's syntax:[1]

$$not\ E\ [\ true\ \{must(*)\ and\ not\ must(LC\_closed)\}\ U\ \{must(signal\_to\_green)\}\ true\ ]$$

Upon verifying this formula, FMC returns true, which confirms that it is possible that *signal_to_green* is not preceded by *LC_closed* (as is in fact the case in section controllers without a LC), i.e. it is confirmed that the property "A train can enter the section only if the LC barriers are closed" does not hold for the entire family of section controllers. Below we will discuss its validity for subfamilies and products.

We recall that MTSs are not sufficient to describe all types of variability in product families: while they can model both optional and mandatory features, MTSs alone cannot model the constraints in feature diagrams regarding *alternative* features nor those regarding the *requires* and *excludes* inter-feature relations. In Sect. 4, we showed that vACTL can, and to help PL engineers we formalised these types of constraints as the vACTL templates ALT, EXC, and REQ.

If we consider the template REQ and instantiate F1 as *blink_light* and F2 as *LC_close_command*, then we have formalised the requirement that the blinking light feature *requires* the presence of the LC feature (cf. Sect. 3.1):

$$(EF\langle blink\_light\rangle\ true) \Longrightarrow (EF^{\Box}\langle LC\_close\_command\rangle^{\Box}\ true)$$

This formula can be translated as follows in FMC's syntax:

$$(EF\ \langle may(blink\_light)\ or\ must(blink\_light)\rangle\ true)\ implies\ (E\ [\ true\ \{must(*)\}\ U\ \{must(LC\_close\_command)\}\ true\ ])$$

This property of course does not hold for the family of section controllers, while it trivially holds for subfamilies or products in which the blinking light is absent. Furthermore, if we consider a product of the family of section controllers which is obtained (cf. Fig. 1) by turning the may transition labelled with *LC close command* as well as the one labelled with *blink light* into must transitions, then the property can be shown to hold over this particular product by FMC.

The latter is an example of a *requires* relation between features. Such a relation does not imply an ordering of the features: a controller blinking a red light *after* the level crossing has been closed cannot be excluded as a product of the family of section controllers on the basis of this relation. Such orderings must be imposed by the behavioural description of a product (family) as provided by an LTS (MTS).

As anticipated, a PL engineer can also use FMC to check whether or not a certain product or a subfamily is part of a family. A subfamily (product) is obtained from a family by preserving at least all must transitions and turning some of the may transitions that are not must transitions into must transitions (and removing all of the remaining may transitions). This task is not difficult for a PL engineer, and visual assistance for this task is planned to be included in a future release of FMC. In Sect. 7, we discuss scalability.

For instance, the product obtained by turning both may transitions labelled *skip* into must transitions and removing all others (i.e., a section controller without LC) is depicted in Fig. 4. When asked to verify Formula 1 over this product, FMC returns *false* (as this product has no LC barriers).

When asked to verify Formula 1 over a subfamily or product which does have a LC, FMC correctly returns *true*.

vACTL can be used to guide the derivation of products that satisfy constraints by construction. In [5], we designed an algorithm to automatically derive, from an MTS description of a product family *and* a set of vACTL formulae expressing further constraints for this family, *all* correct products (a set of LTS descriptions of products, each correct w.r.t. all vACTL constraints). The algorithm's complexity is bounded by $O(n \times m \times w)$, with n the number of transitions of the MTS (family), m the number of LTSs (products) returned, and w the maximum width of may transitions with labels referred to in the vACTL formulae (constraints).

6. RELATED WORK

In [1–5, 11, 12, 14, 19, 20, 24], MTSs and variants are used to model and analyse product families. We previously mentioned our recent work [2–5] towards a modelling and verification framework capable of addressing both static and behavioural conformance of products of a product family. In [15], automatic verification is limited to structural conformance of a PL and its products w.r.t. a feature model.

[1]In our formulae, the wildcard $*$ stands for any action.

In general, modelling and verifying static constraints over products of a product family requires separate expressions in a first-order logic [7,13,21], while modelling and verifying behavioural constraints over the products of a product family is typically not addressed in feature modelling. In [11,12], we extended MTSs to model variability constraints regarding alternative features. In [17, 18], an algebraic approach to behavioural modelling and analysis of product families was developed. In [23], (Dynamic) Feature Petri Nets were introduced to model the behaviour of product families with a high degree of variability.

We now discuss the three approaches closest to ours in some more detail.

In [14], the authors present an algorithm for checking conformance of LTSs against MTSs according to a given branching relation, i.e. checking conformance of the behaviour of a product against that of its product family. It is a fixed-point algorithm that starts with the Cartesian product of the states and iteratively eliminates pairs that are invalid according to the given relation. The algorithm is implemented in a tool that allows one to check whether or not a given LTS conforms to a given MTS according to a number of different branching relations.

In [20], variable I/O automata are introduced to model product families, together with a model-checking approach to verify conformance of products w.r.t. a family's variability. This is achieved by using variability information in the model-checking algorithm (while exploring the state space an associated variability model is consulted continuously). Properties (expressed in CTL) are verified by explicit-state model checking, progressing one state at a time.

In [9], an explicit-state model-checking technique to verify properties (expressed in LTL) over Featured Transition Systems (FTSs) is defined. This results in a means to check that whenever a behavioural property is satisfied by an FTS modelling a product family, then it is also satisfied by every product of that family, and whenever a property is violated, then a counterexample is provided as well as the products violating the property. In [10], this approach is improved by using symbolic model checking, examining sets of states at a time, and a feature-oriented version of CTL. In [5], we compare the approach of [9] with ours, while a comparison with the (not yet published) [10] is left for future work.

7. CONCLUSIONS

In this paper, we propose a solution technique for the design and validation of variability in a PL that has emerged from our recent research [2–5, 11, 12, 22]. Note that our proposal uses a unique formal framework, namely a logic (vACTL) with its natural interpretation structure (MTSs). The notion of product derivation is defined inside this framework and logical formulae are used both as constraints and as properties, to be proved for families and products alike.

We are particularly interested in obtaining feedback on this technique's applicability to current industrial problems.

In this context, it is important to underline that MTSs are not intended to be used directly by PL engineers, but rather to be considered as the proper underlying semantic framework: in practice, behavioural descriptions are to be given as high-level formalisms used in model-based development (e.g., statecharts, UML state diagrams) augmented with mechanisms to express variability analogous to those employed by MTSs. Similarly, the use of vACTL is to be facilitated by appropriate template formulae (temporal logic patterns) in order to hide the logic from the PL engineers. Working this out in detail is left for future work.

Since MTSs cannot model constraints regarding *alternative* features nor those regarding inter-feature relations like *requires* and *excludes*, we started to use an Extended MTS (EMTS) as the semantic model underlying vACTL [11, 12]. An EMTS is an MTS in which subsets of the outgoing may transitions in a state can be defined to be such that at least or at most one of the outgoing transitions is required. EMTSs can model *alternative* features in a natural way and we are currently studying a way to model the *requires* and *excludes* inter-feature relations as well.

Finally, another future research direction concerns scalability. Our running example is very small, but current model-checking techniques allow to deal with huge state spaces and there is no reason for which such techniques cannot be applied to MTSs (checking an MTS can be reduced to twice checking an LTS [14]). The actual issues to be addressed are twofold:

1. Address the exponential blow-up of products when a large number of features is considered. Note, however, that MTSs already embed a combinatorial number of products (LTSs) in a single structure;

2. How to express families of complex systems, which are typically described through compositions of transition systems. For instance, an interlocking system controlling a line composed of several sections like that of our running example, could be built by synchronizing these section controllers. Synchronization and parallel composition of MTSs has been addressed in [1, 16], but we still need to gain more insight into coping with vACTL requirements and properties over such compositions.

8. ACKNOWLEDGEMENTS

The research reported in this paper has been partially funded by the Italian projects D-ASAP (MIUR–PRIN 2007) and XXL (CNR–RSTL).

9. REFERENCES

[1] A. Antonik, M. Huth, K.G. Larsen, U. Nyman, and A. Wąsowski. 20 Years of modal and mixed specifications. In *The Concurrency Column*, pages 94–129, number 95 of *Bulletin of the European Association of Theoretical Computer Science*. EATCS, June 2008.

[2] P. Asirelli, M.H. ter Beek, A. Fantechi, and S. Gnesi. Deontic logics for modeling behavioural variability. In D. Benavides, A. Metzger, and U. Eisenecker, editors, *Proceedings 3rd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'09)*, pages 71–76. ICB Research Report 29, Universität Duisburg–Essen, 2009.

[3] P. Asirelli, M.H. ter Beek, A. Fantechi, and S. Gnesi. A deontic logical framework for modelling product families. In D. Benavides, D. Batory, and P. Grünbacher, editors, *Proceedings 4th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'10)*, pages 37–44. ICB Research Report 37, Universität Duisburg–Essen, 2010.

[4] P. Asirelli, M.H. ter Beek, A. Fantechi, and S. Gnesi. A logical framework to deal with variability. In D. Méry and S. Merz, editors, *Proceedings 8th International Conference on Integrated Formal Methods (IFM'10)*, volume 6396 of *Lecture Notes in Computer Science*, pages 43–58. Springer, Berlin, 2010.

[5] P. Asirelli, M.H. ter Beek, A. Fantechi, and S. Gnesi. Formal description of variability in product families. Submitted, 2011.

[6] M. Banci and A. Fantechi. Geographical versus functional modelling by statecharts of interlocking systems. In *Proceedings 9th workshop on Formal Methods for Industrial Critical Systems (FMICS'04)*, volume 133 of *Electronic Notes in Theoretical Computer Science*, pages 3–19. Elsevier, Amsterdam, 2005.

[7] D. Batory. Feature models, grammars and propositional formulas. In J.H. Obbink and K. Pohl, editors, *Proceedings 9th International Conference on Software Product Lines (SPLC'05)*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, Berlin, 2005.

[8] C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, and D. Romano. A formal verification environment for railway signaling system design. In *Selected Papers 1st International Workshop on Formal Methods for Industrial Critical Systems (FMICS'96)*, *Formal Methods in System Design* 12(2):139–161, 1998.

[9] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model checking lots of systems: Efficient verification of temporal properties in software product lines. In *Proceedings 32nd International Conference on Software Engineering (ICSE'10)*, pages 335–344. ACM, New York, 2010.

[10] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay. Symbolic model checking of software product lines. To appear in *Proceedings 33rd International Conference on Software Engineering (ICSE'11)*. ACM, New York, 2011.

[11] A. Fantechi and S. Gnesi. A behavioural model for product families. In *Proceedings 6th joint meeting of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering (ESEC/FSE'07)*, pages 521–524. ACM, New York, 2007.

[12] A. Fantechi and S. Gnesi. Formal modelling for product families engineering. In *Proceedings 12th International Software Product Lines Conference (SPLC'08)*, pages 193–202. IEEE Computer Society, Los Alamitos, 2008.

[13] A. Fantechi, S. Gnesi, G. Lami, and E. Nesti. A methodology for the derivation and verification of use cases for product lines. In R.L. Nord, editor, *Proceedings 3rd International Conference on Software Product Lines (SPLC'04)*, volume 3154 of *Lecture Notes in Computer Science*, pages 255–265. Springer, Berlin, 2004.

[14] D. Fischbein, S. Uchitel, and V.A. Braberman. A foundation for behavioural conformance in software product line architectures. In R.M. Hierons and H. Muccini, editors, *Proceedings ISSTA 2006 Workshop on Role of Software Architecture for Testing and Analysis (ROSATEA'06)*, pages 39–48. ACM, New York, 2006.

[15] R. Gheyi, T. Massoni, and P. Borba. A theory for feature models in Alloy. In *Proceedings First Alloy Workshop*, pages 71–80. ACM, New York, 2006.

[16] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In K.G. Larsen and M. Nielsen, editors, *Proceedings 11th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440. Springer, Berlin, 2001.

[17] A. Gruler, M. Leucker, and K.D. Scheidemann. Modelling and model checking software product lines. In G. Barthe and F.S. de Boer, editors, *Proceedings 10th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'08)*, volume 5051 of *Lecture Notes in Computer Science*, pages 113–131. Springer, Berlin, 2008.

[18] A. Gruler, M. Leucker, and K.D. Scheidemann. Calculating and modelling common parts of software Product Lines. In *Proceedings 12th International Software Product Lines Conference (SPLC'08)*, pages 203–212. IEEE Computer Society, Los Alamitos, 2008.

[19] K.G. Larsen, U. Nyman, and A. Wąsowski. Modal I/O automata for interface and product line Theories. In R. De Nicola, editors, *Proceedings 16th European Symposium on Programming (ESOP'07)*, volume 4421 of *Lecture Notes in Computer Science*, pages 64–79. Springer, Berlin, 2007.

[20] K. Lauenroth, K. Pohl, and S. Töhning. Model checking of domain artifacts in product line engineering. In *Proceedings International Conference on Automated Software Engineering (ASE'09)*, pages 269–280. IEEE Computer Society, Los Alamitos, 2009.

[21] M. Mannion and J. Camara. Theorem proving for product line model verification. In F. van der Linden, editor, *Proceedings 5th International Workshop on Software Product Family Engineering (PFE'03)*, volume 3014 of *Lecture Notes in Computer Science*, pages 211–224. Springer, Berlin, 2004.

[22] F. Mazzanti. FMC v5.0b. `http://fmtlab.isti.cnr.it/fmc`

[23] R. Muschevici, D. Clarke, and J. Proenca. Feature Petri nets. In *Proceedings International Workshop on Formal Methods in Software Product Line Engineering (FMSPLE'10)*. Technical Report, University of Lancaster, 2010.

[24] H. Schmidt and H. Fecher. Comparing disjunctive modal transition systems with an one-selecting variant. *Journal of Logic and Algebraic Programming* 77(1-2):20–39, 2008.