Validated Test Models for Software Product Lines: Featured Finite State Machines

Vanderson Hafemann Fragal[1] *, Adenilso Simao[1], and Mohammad Reza Mousavi[2] **

[1] Institute of Math. and Computer Sciences - ICMC, University of São Paulo, Brazil
[2] Centre for Research on Embedded Systems - CERES, Halmstad University, Sweden

Abstract. Variants of the finite state machine (FSM) model have been extensively used to describe the behaviour of reactive systems. In particular, several model-based testing techniques have been developed to support test case generation and test case executions from FSMs. Most such techniques require several validation properties to hold for the underlying test models. In this paper, we propose an extension of the FSM test model for software product lines (SPLs), named featured finite state machine (FFSM). As the first step towards using FFSMs as test models, we define feature-oriented variants of basic test model validation criteria. We show how the high-level validation properties coincide with the necessary properties on the product FSMs. Moreover, we provide a mechanised tool prototype for checking the feature-oriented properties using satisfiability modulo theory (SMT) solver tools. We investigate the applicability of our approach by applying it to both randomly generated FFSMs as well as those from a realistic case study (the Body Comfort System). The results of our study show that for random FFSMs over 16 independent non-mandatory features, our technique provides substantial efficiency gains for the set of proposed validity checks.

Keywords: Formal Modelling, Model Validation, Software Product Line, Finite State Machine.

1 Introduction

Motivation. Different forms of finite state machines (FSMs) have been extensively used as the fundamental semantic model for various behavioural specification languages and design trajectories. In particular, several test case generation techniques have been developed for hardware and software testing based on FSMs; an overview of these techniques can be found in [19, 8, 17]. All FSM-based testing techniques require the underlying test models to satisfy some basic validation criteria such as connectedness and minimality.

* The work of V. Hafemann has been partially supported by the Science Without Borders project number 201694/2015-8.

** The work of M. R. Mousavi has been partially supported by the Swedish Research Council award number: 621-2014-5057 and the Swedish Knowledge Foundation project number 20140312.

Software Product Lines (SPLs) [20] are used for systematic reuse of artefacts and are effective in mass production and customisation of software. However, testing large SPLs demand substantial effort, and effective reuse is a challenge. Model-Based Testing (MBT) approaches need to be adapted to the SPL domain (see [27] for a survey of existing approaches).

There are a few recent attempts [24, 34] to extend the FSM-based testing techniques to SPLs, mostly using the delta-oriented approach to SPL modelling. We are not aware of any prior work that addresses the basic test model validation criteria for SPLs at the family-wide level. The present paper aims at bridging this gap. To this end, we first propose a product-line extension of FSMs, named Featured Finite State Machine (FFSM). An FFSM unifies the test models of the valid product configurations in a family into a single model. Our aim is to extend FSM-based test case generation techniques [28, 32] to generate test suites for groups of SPL products. As the first step to this end, we define feature-oriented family-based validation criteria that coincide with the necessary conditions of such test case generation techniques at the product level.

Our family-based validation criteria are implemented in a tool using Java and the Z3 [26] tool. A case study from the automotive domain concerning the Body Comfort System [21] was performed to show the applicability of our criteria and tool. Our research question is: *How large does an FFSM have to be in order to save time in the validation of the FFSM instead of its valid product FSMs?* To this end, we performed an empirical study on randomly generated FFSMs with various parameters. The results indicate that for random FFSMs with over 10 independent non-mandatory features, we have substantial efficiency gains for the set of proposed validity checks.

Contributions. The main contributions of this paper are summarised below:

1. Proposing family-based validation criteria for FSM-based test models and proving them to coincide with their product-based counterparts, and
2. Implementing efficient family-based validation techniques and investigating their applicability by applying them to a large set of examples.

Also as a carrier for these contributions, we propose a feature-oriented extension of FSMs.

Organisation. The remainder of this paper is organised as follows. Section 2 presents some preliminary notions and concepts regarding SPL testing and FSMs. Section 3 describes the FFSM formalism, the proposed validation properties, and the associated theoretical results. Section 4 provides an overview of the implementation used for property checking in Java and Z3. Section 5 illustrates the experimental study and the analysis of results. Section 6 provides an overview of the related works and a comparison among the relevant approaches in the literature. Section 7 concludes the paper and presents the directions of our future work.

2 Background

This section recapitulates the basic concepts and definitions of SPLs and FSMs that we are going to use through the rest of the paper.

2.1 Software Product Lines

A `feature` is an atomic unit used to differentiate the products of an SPL. Let F be the set of features. A `product` p is defined by a set of features $p \subseteq F$. The feature structure can be represented by a `feature diagram` [30]. In a feature diagram, some notational conventions are used to represent `commonalities` and `variabilities` of an SPL (e.g., mandatory, optional, and alternative features). To illustrate the concepts throughout the paper, we use the following SPL.

Example 1. The Arcade Game Maker (AGM) [31] can produce arcade games with different game rules. The objective of the player in any game is to get more points. Figure 1 shows the feature diagram of AGM. There are three alternative features for the game rule (`Brickles`, `Pong` and `Bowling`) and one optional feature (`Save`) to save the game.

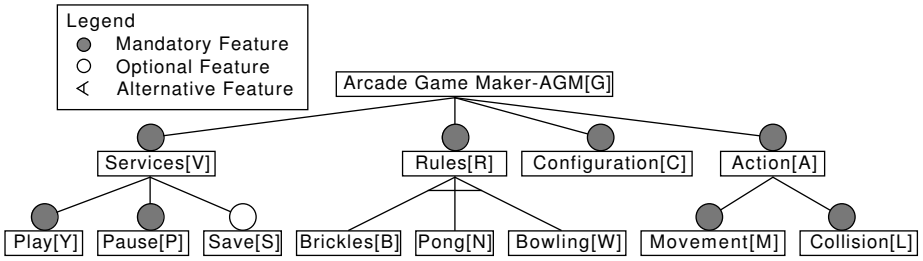

Fig. 1. AGM Feature Model (adapted from [31]).

In general, not all combinations of features are valid. Dependencies and constraints on feature combinations reduce the power set $\mathcal{P}(F)$ of all potential feature combinations to a subset of `valid products` $P \subseteq \mathcal{P}(F)$ [2, 13]. A `feature constraint` is a propositional formula generated by interpreting the elements of the set F as propositional variables. We denote by $B(F)$ the set of all feature constraints.

A `product configuration` ρ of a product $p \in P$ is the feature constraint that uses all features of F, where all features in p are true, i.e., $\rho = (\bigwedge_{f \in p} f) \wedge (\bigwedge_{f \notin p} \neg f)$. We denote by Λ the set of all valid product configurations. Given a feature constraint $\chi \in B(F)$, a product configuration $\rho \in \Lambda$ `satisfies` χ (denoted by $\rho \vDash \chi$), if the assertion $\rho \wedge \chi$ is not false.

Consider a feature model FM, let F be a set of features extracted from FM. Given $F = \{Y, S\}$, we know from the FM that feature Play[Y] is mandatory

and Save[S] is optional; an example feature constraint involving both features is $(Y \wedge \neg S) \in B(F)$, which specifies the products in which Y is included and S is excluded.

2.2 Finite State Machine

The classic Finite State Machine (FSM) formalism is often used due to its simplicity and rigour for systems such as communication protocols and reactive systems [8]. In this study, we use the following definition of FSM.

Definition 1. *An FSM M is a 5-tuple (S, s_0, I, O, T), where S is a finite set of* `states` *with the* `initial state` s_0*, I is a finite set of* `inputs`*, O is a finite set of* `outputs`*, and T is a set of* `transitions` $t = (s, x, o, s') \in T$*, where $s \in S$ is the source state, $x \in I$ is the input label, $o \in O$ is the output label, and $s' \in S$ is the target state.*

Given an input sequence $\alpha = (x_1, ..., x_k), x_i \in I, 1 \leq i \leq k$, a `path` *from state s_1 to s_{k+1} exists when there are transitions $t_i = (s_i, x_i, o_i, s_{i+1}) \in T$, for each $1 \leq i \leq k$. A path υ is a 5-tuple $(s_1, \alpha, \tau, \beta, s_{k+1})$, where*

1. $s_1 \in S$ is the source state where the path begins,

2. $\alpha \in I^*$ is the defined input sequence,

3. $\tau \in T^*$ is the transition sequence, i.e., $\tau = (t_1, ..., t_k)$,

4. $\beta \in O^*$ is the output result, i.e., $\beta = (o_1, ..., o_k)$

5. $s_{k+1} \in S$ is the target state where the path ends.

Notation $\Omega(s)$ is used to denote all paths that start on state $s \in S$. Ω_M is used to denote $\Omega(s_0)$.

Test case generation methods such as the *Harmonised State Identification* (HSI) [28] method, and the *Fault Coverage-Driven Incremental* (P) [32] method require FSMs with some of the semantic properties defined below.

Definition 2. *The following validation properties are defined for FSMs:*

***1. Deterministic**: if two transitions leave a state with a common input, then both transitions reach the same state, i.e., $\forall_{(s,x,o,s'),(s,x,o',s'') \in T} \bullet s' = s''$;*

***2. Complete** (required only by some algorithms): every state has one transition for each input, i.e., $\forall_{s \in S, x \in I} \bullet \exists_{o \in O, s' \in S} \bullet (s, x, o, s') \in T$;*

***3. Initially Connected**: there is a path to every state from the initial state, i.e., $\forall_{s \in S} \bullet \exists_{\alpha \in I^*, \tau \in T^*, \beta \in O^*} \bullet (s_0, \alpha, \tau, \beta, s) \in \Omega_M$;*

***4. Minimal**: all pairs of states are distinguishable, i.e., $\forall_{s_a, s_b \in S} \bullet \exists_{(s_a, \alpha, \tau_a, \beta_a, s'_a) \in \Omega(s_a), (s_b, \alpha, \tau_b, \beta_b, s'_b) \in \Omega(s_b)} \bullet \beta_a \neq \beta_b$.*

Example 2. There are six possible products that can be derived from the AGM FM. The FSM M_1 of the first configuration is presented in Figure 2. This test model is an abstracted version of the design model where observable events are represented by inputs and the correspondent outputs. The inputs are in-game commands, while the outputs 0 and 1 are abstract captured responses. We selected the `Pong[N]` rule and discarded the `Save[S]` option represented by $\rho_1 = (G \wedge V \wedge R \wedge C \wedge A \wedge M \wedge L \wedge Y \wedge P \wedge N \wedge \neg B \wedge \neg W \wedge \neg S) \in \Lambda$. It is straightforward to check that M_1 is a deterministic, complete, initially connected and minimal FSM.

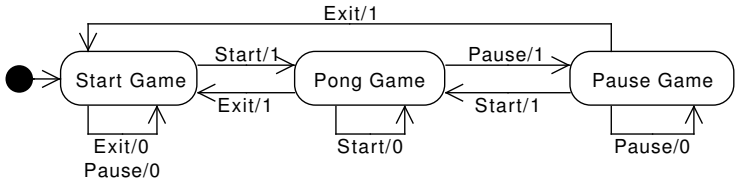

Fig. 2. FSM of the first product configuration of AGM.

3 Featured Finite State Machines

A Featured Finite State Machine (FFSM) is an extension of a Finite State Machine (FSM) by annotating states and transitions with feature constraints.

This Section presents the basic definitions for FFSMs, followed by the notion of product derivation, and the high-level validation properties required for test case generators.

3.1 Basic Definitions

The simplified syntax (with conditions) of an FFSM is defined as follows.

Definition 3. *An FFSM is a 7-tuple* $(F, \Lambda, C, c_0, Y, O, \Gamma)$*, where*

1. F *is a finite set of features,*

2. Λ *is the set of* product configurations*,*

3. $C \subseteq S \times B(F)$ *is a finite set of* conditional states*, where* S *is a finite set of state labels,* $B(F)$ *is the set of all feature constraints, and* C *satisfies the following condition:*

$$\forall_{(s,\varphi)\in C} \bullet \exists_{\rho\in\Lambda} \bullet \rho \vDash \varphi$$

4. $c_0 = (s_0, true) \in C$ *is the* initial conditional state*,*

5. $Y \subseteq I \times B(F)$ *is a finite set of* conditional inputs*, where* I *is the set of input labels,*

6. O *is a finite set of* outputs*,*

7. $\Gamma \subseteq C \times Y \times O \times C$ *is the set of conditional transitions satisfying the following condition:*

$$\forall_{((s,\varphi),(x,\varphi''),o,(s',\varphi'))\in\Gamma} \bullet \exists_{\rho\in\Lambda} \bullet \rho \vDash (\varphi \wedge \varphi' \wedge \varphi'')$$

The components of FFSM are self-explanatory; the above-given two conditions ensure that every conditional state and every transition is present in at least one valid product of the SPL. A conditional transition from conditional state c to c' with conditional input y and output o is represented by quadruple $t = (c, y, o, c')$, or alternatively by $c \xrightarrow[o]{y} c'$.

Example 3. Figure 3 shows the FFSM for the AGM SPL. The notation of a conditional state in the model is $s(\varphi) \equiv (s, \varphi) \in C$, the transition line by $x(\varphi)/o \equiv \overset{(x,\varphi)}{\underset{o}{\rightarrow}} \in Y \times O$, and the operators of feature constraints are denoted by

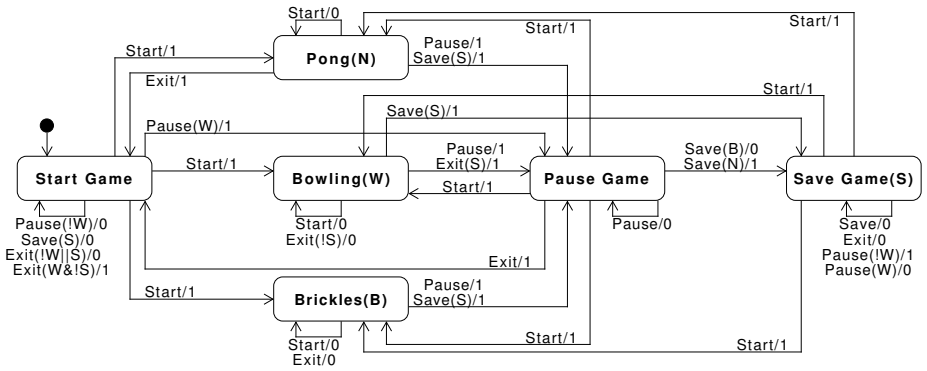

Fig. 3. FFSM for the AGM SPL.

& (and), || (or), and ! (not). Omitted feature conditions mean that the condition is true, i.e., for states $s \equiv (s, true) \in C$, and transitions $\underset{o}{\overset{x}{\rightarrow}} \equiv \underset{o}{\overset{(x,true)}{\rightarrow}}$.

Next, we define auxiliary definitions on FFSMs that are used to describe transfer sequences; they are subsequently used in expressing the FFSM validation properties.

Definition 4. *Given a conditional input sequence $\delta = (y_1, ..., y_k), y_i \in Y, 1 \leq i \leq k$, a* `conditional path` *from conditional state c_1 to c_{k+1} exists when there are conditional transitions $t_i = (c_i, y_i, o_i, c_{i+1}) \in \Gamma$, for each $1 \leq i \leq k$. A conditional path σ is a 6-tuple $(c_1, \delta, \nu, \gamma, \omega, c_{k+1})$, where*

1. $c_1 \in C$ is the conditional state where the path begins,

2. $\delta \in Y^*$ is the conditional input sequence,

3. $\nu \in \Gamma^*$ is the conditional transition sequence, i.e., $\nu = (t_1, ..., t_k)$,

4. $\gamma \in O^*$ is the output result, i.e., $\gamma = (o_1, ..., o_k)$

5. $\omega \in B(F)$ is the resulting path condition, i.e., $\omega = (\varphi_1, ...\varphi_{k+1}) \wedge (\theta_1, ..., \theta_k), y_i = (x_i, \theta_i), c_i = (s_i, \varphi_i)$

6. $c_{k+1} \in C$ is the conditional state where the path ends.

Notation $\Theta(c)$ is used to denote the set of all conditional paths that start at conditional state $c \in C$. Θ_{FF} is used to denote $\Theta(c_0)$.

We also define a valid transfer sequence that is used to transfer the machine from one conditional state to another.

Definition 5. *Given two conditional states $c, c' \in C$, a conditional input sequence $\delta \in Y^*$ is a valid* `transfer sequence` *if there are at least one path $(c, \delta, \nu, \gamma, \omega, c') \in \Theta(c)$ and one product that satisfies the path condition, i.e., $\exists_{\rho \in \Lambda} \bullet \rho \vDash \omega$.*

Example 4. Consider the FFSM of Figure 3. Note that a transfer sequence $\delta = (Start, Pause)$ of a conditional path $(StartGame, \delta, (StartGame \overset{Start}{\underset{1}{\rightarrow}}$

$Brickles(B), Brickles(B) \overset{Pause}{\underset{1}{\rightarrow}} PauseGame), (1,1), (B), PauseGame) \in \Theta_{FF}$ has a transfer condition $\omega = (B)$ and only two products can satisfy ω, namely, $\rho_5 = (G \wedge V \wedge R \wedge C \wedge A \wedge M \wedge L \wedge Y \wedge P \wedge B \wedge \neg N \wedge \neg W \wedge \neg S) \in \Lambda$ and $\rho_6 = (G \wedge V \wedge R \wedge C \wedge A \wedge M \wedge L \wedge Y \wedge P \wedge B \wedge S \wedge \neg N \wedge \neg W) \in \Lambda$. Thus, ω is not satisfied by valid product $\rho_1 = (G \wedge V \wedge R \wedge C \wedge A \wedge M \wedge L \wedge Y \wedge P \wedge N \wedge \neg B \wedge \neg W \wedge \neg S)$.

3.2 Product Derivation

We define a product derivation operator, reminiscent of the operator in [4, 6], that is parameterised by feature constraints. Given a feature constraint, the product derivation operator reduces an FFSM into an FSM representing the selection of products.

Definition 6. *Given a feature constraint $\phi \in B(F)$ and an FFSM $FF = (F, \Lambda, C, c_0, Y, O, \Gamma)$, if exactly one product $\rho \in \Lambda$ satisfies ϕ,* i.e.,$\exists!_{\rho \in \Lambda} \bullet \rho \vDash \phi$*, then the product derivation operator Δ_ϕ induces an FSM $\Delta_\phi(FF) = (S, s_0, I, O, T)$, where:*

1. $S = \{s | (s, \varphi) \in C \wedge \rho \vDash (\varphi \wedge \phi)\}$ *is the set of states;*
2. $s_0 = s$, $c_0 = (s, \varphi) \in C$ *is the initial state;*
3. $T = \{(s, x, o, s') | (s, \varphi) \overset{(x, \varphi'')}{\underset{o}{\rightarrow}} (s', \varphi') \in \Gamma \wedge \rho \vDash (\varphi \wedge \varphi' \wedge \varphi'' \wedge \phi)\}$ *is the set of transitions.*

The set of all valid products of FF is the set of all induced FSMs. Figure 3 shows the FFSM generated for the AGM SPL that can induce six products. Using the feature constraint $\phi = N \wedge \neg S$ the FFSM is projected into the FSM presented in Figure 2.

3.3 Validation Properties

To adopt FFSMs as test models, first, we need to validate the product-line-based specification with properties used for FSMs. Next, we define the high-level counterparts of the four basic properties, namely, determinism, completeness, initially connected-ness, and minimality, and show that they coincide with the aforementioned properties for their valid FSM products.[1]

Definition 7. *An FFSM FF is* `deterministic` *if for all conditional states when exists two enabled conditional transitions with the same input for a product ρ, then both transitions lead to the same state, i.e.,* $\forall_{(s,\varphi) \overset{(x,\varphi')}{\underset{o}{\rightarrow}} (s', \varphi_a), (s,\varphi) \overset{(x,\varphi'')}{\underset{o}{\rightarrow}} (s'', \varphi_b) \in \Gamma} \bullet \forall_{\rho \in \Lambda} \bullet \rho \nvDash (\varphi \wedge \varphi' \wedge \varphi'' \wedge \varphi_a \wedge \varphi_b) \vee s' = s''$.

Next, we state and prove that an FFSM is deterministic when all its valid product FSMs are deterministic.

[1] Due to space limitation proof sketches are provided below; detailed proofs of correctness for these properties is available at `http://ceres.hh.se/mediawiki/Vanderson_Hafemann`

Theorem 1. *An FFSM FF is deterministic if and only if all derived product FSMs $\Delta_\phi(FF)$ are deterministic.*

Proof. We break the bi-implication in the thesis into two implications and prove each by contradiction. For the implication from left to right, assume that FFSM FF is deterministic, but there is a derived FSM $\Delta_\phi(FF)$ for a product ρ which is non-deterministic; we obtain a contradiction. Let FFSM $FF = (F, \Lambda, C, c_0, Y, O, \Gamma)$ be deterministic and a derived FSM $\Delta_\phi(FF) = (S, s_0, I, O, T)$ be non-deterministic for a product $\rho \in \Lambda$ on state $s \in S$. As $\Delta_\phi(FF)$ is non-deterministic, then by the negation of Definition 2 item 1 there is an input $x \in I$ such that two transitions $(s, x, o, s'), (s, x, o, s'') \in T$ reach different states $s' \neq s''$. By Definition 6 item 3 if $\Delta_\phi(FF)$ has two transitions (s, x, o, s') and (s, x, o, s''), then both were induced from conditional transitions $(s, \varphi) \overset{(x,\varphi')}{\underset{o}{\rightarrow}} (s', \varphi_a), (s, \varphi) \overset{(x,\varphi'')}{\underset{o}{\rightarrow}} (s'', \varphi_b) \in \Gamma$ of FF and $\rho \vDash (\varphi \wedge \varphi' \wedge \varphi_a \wedge \varphi_b)$. However, FF is deterministic and by Definition 7 the condition $\rho \nvDash (\varphi' \wedge \varphi'' \wedge \varphi_a \wedge \varphi_b) \vee s' = s''$ holds for all pairs of conditional transitions, which is a contradiction as there is a pair of conditional transitions that the negation of the condition $\rho \vDash (\varphi \wedge \varphi' \wedge \varphi_a \wedge \varphi_b) \wedge (s' \neq s'')$ also holds.

Likewise, for the implication right to left, assume that $\Delta_\phi(FF)$ is deterministic for ρ, but FF is non-deterministic; we obtain a contradiction. Let $FF = (F, \Lambda, C, c_0, Y, O, \Gamma)$ be non-deterministic on conditional state $(s, \varphi) \in C$, $\rho \models \varphi$, and $\Delta_\phi(FF) = (S, s_0, I, O, T)$ is deterministic for ρ. As FF is non-deterministic, then by the negation of Definition 7 there is an input $x \in I$ such that two conditional transitions $(s, \varphi) \overset{(x,\varphi')}{\underset{o}{\rightarrow}} (s', \varphi_a), (s, \varphi) \overset{(x,\varphi'')}{\underset{o}{\rightarrow}} (s'', \varphi_b) \in \Gamma$ are satisfied by $\rho \vDash (\varphi \wedge \varphi' \wedge \varphi_a \wedge \varphi_b)$ and reach different states $s' \neq s''$. As $\rho \vDash \phi$ and by Definition 6 item 3 each transition of FF that satisfies ϕ is induced to $\Delta_\phi(FF)$, thus $(s, x, o, s'), (s, x, o, s'') \in T$. However, $\Delta_\phi(FF)$ is deterministic and by Definition 2 item 1 the condition $s' = s''$ is true for all pairs of transitions $(s, x, o, s'), (s, x, o, s'') \in T$, which is a contradiction as there is a pair of transitions $(s, x, o, s'), (s, x, o, s'') \in T$ such $(s' \neq s'')$. □

Definition 8. *An FFSM FF is* `complete` *if for all conditional states in a product there is an outgoing valid transition for each and every input, i.e.,* $\forall_{(s,\varphi)\in C} \bullet \forall_{\rho\in\Lambda} \bullet \forall_{x\in I} \bullet\ \rho \nvDash \varphi \vee \exists_{(s,\varphi)\overset{(x,\varphi'')}{\underset{o}{\rightarrow}}(s',\varphi')\in\Gamma} \bullet \rho \models \varphi' \wedge \varphi''$.

Next, we state and prove that an FFSM is complete when all its valid product FSMs are complete.

Theorem 2. *An FFSM is complete if and only if all derived product FSMs are complete.*

Proof. We break the bi-implication in the thesis into two implications and prove each by contradiction. For the implication left to right, assume that FFSM FF is complete, but there is a derived FSM $\Delta_\phi(FF)$ for a product ρ which is non-complete; we obtain a contradiction. Let FFSM $FF = (F, \Lambda, C, c_0, Y, O, \Gamma)$ be complete and a derived FSM $\Delta_\phi(FF) = (S, s_0, I, O, T)$ be non-complete for a

product $\rho \in \Lambda$ on state $s \in S$ for input $x \in I$. As $\Delta_\phi(FF)$ is non-complete, then, by the negation of Definition 2 item 2 there is no transition $(s, x, o, s') \in T$ on s with input x. By Definition 8 if FF is complete, then for all products $\rho \in \Lambda$ that satisfies a conditional state $(s, \varphi) \in C \wedge \rho \models \varphi$ and for all inputs $x \in I$ there are conditional transitions $(s, \varphi) \overset{(x,\varphi'')}{\underset{o}{\rightarrow}} (s', \varphi') \in \Gamma$ such $\rho \models \varphi' \wedge \varphi''$. However, by Definition 6 item 3 every conditional transition $(s, \varphi) \overset{(x,\varphi'')}{\underset{o}{\rightarrow}} (s', \varphi') \in \Gamma$ in FF that satisfies $\rho \vDash \phi$ induces a transition $(s, x, o, s') \in T$ in $\Delta_\phi(FF)$, which is a contradiction as $\Delta_\phi(FF)$ does not have a transition $(s, x, o, s') \in T$ on state s for input x.

Likewise, for the implication right to left, assume that $\Delta_\phi(FF)$ is complete for ρ, but FF is non-complete; we obtain a contradiction. Let $FF = (F, \Lambda, C, c_0, Y, O, \Gamma)$ be non-complete on conditional state $(s, \varphi) \in C$ for input $x \in I$, $\rho \models \varphi$, and $\Delta_\phi(FF) = (S, s_0, I, O, T)$ is complete for ρ. As FF is non-complete, then by the negation of Definition 8 on conditional state $(s, \varphi) \in C$ there is no conditional transition $(s, \varphi) \overset{(x,\varphi'')}{\underset{o}{\rightarrow}} (s', \varphi') \in \Gamma$ with input $x \in I$ for FF, or it exists but is not satisfied $\rho \not\models \varphi' \wedge \varphi''$. By Definition 6 item 3 if a conditional transition $(s, \varphi) \overset{(x,\varphi'')}{\underset{o}{\rightarrow}} (s', \varphi')$ does not exist in FF, or it exists but $\rho \not\models \varphi' \wedge \varphi''$, then there is no transition $(s, x, o, s') \in T$ induced in $\Delta_\phi(FF)$. However, $\Delta_\phi(FF)$ is complete and by Definition 2 item 2 for all states $s \in S$ and for all inputs $x \in I$ there are transitions $(s, x, o, s') \in T$, which is a contradiction as there is no transition $(s, x, o, s') \in T$ in $\Delta_\phi(FF)$ for state s and input x. □

Definition 9. *An FFSM FF is* `initially connected` *if there exist transfer sequences from the initial conditional state to every conditional state for every satisfiable product, i.e.,* $\forall_{c=(s,\varphi)\in C} \bullet \forall_{\rho\in\Lambda} \bullet \rho \models \varphi \implies \exists_{(c_0,\delta,\nu,\gamma,\omega,c)\in\Theta_{FF}} \bullet \rho \vDash \omega$.

Next, we state and prove that an FFSM is initially connected when all its valid product FSMs are initially connected.

Theorem 3. *An FFSM is initially connected if and only if all derived product FSMs are initially connected.*

Proof. We break the bi-implication in the thesis into two implications and prove each by contradiction. For the implication left to right, assume that FFSM FF is initially connected, but there is a derived FSM $\Delta_\phi(FF)$ for a product ρ which is non-initially connected; we obtain a contradiction. Let FFSM $FF = (F, \Lambda, C, c_0, Y, O, \Gamma)$ be initially connected and a derived FSM $\Delta_\phi(FF) = (S, s_0, I, O, T)$ be non-initially connected for a product $\rho \in \Lambda$ on state $s_k \in S$. As $\Delta_\phi(FF)$ is non-initially connected, then, by the negation of Definition 2 item 3 there is no path $\upsilon \in \Omega_{\Delta_\phi(FF)}$ to s_k from the initial state s_0. By Definition 9 if FF is initially connected, then there is a path $\sigma_k \in \Theta_{FF}$ to every conditional state $(s_k, \varphi_k) \in C$ from the initial conditional state c_0, and ρ satisfies the path condition ω. However, by Definition 5 every conditional transition $(s_i, \varphi_i) \overset{(x_i,\varphi_i')}{\underset{o}{\rightarrow}} (s_{i+1}, \varphi_{i+1}) \in \Gamma$, $0 \leq i \leq k$ forms a path to reach (s_k, φ_k) which is satisfied by ρ. As $\rho \vDash \phi$, and by Definition 6 item 3 every conditional transition

$(s_i, \varphi_i) \overset{(x_i,\varphi_i')}{\underset{o}{\rightarrow}} (s_{i+1}, \varphi_{i+1}) \in \Gamma$ is induced to $(s_i, x_i, o, s_{i+1}) \in T$ that forms a path to reach s_k, which is a contradiction as there is no path for $\upsilon \in \Omega_{\Delta_\phi(FF)}$ to reach state s_k.

Likewise, for the implication right to left, assume that $\Delta_\phi(FF)$ is initially connected for ρ, but FF is non-initially connected; we obtain a contradiction. Let $FF = (F, \Lambda, C, c_0, Y, O, \Gamma)$ be non-initially connected on conditional state $(s, \varphi) \in C$, $\rho \models \varphi$, and $\Delta_\phi(FF) = (S, s_0, I, O, T)$ is initially connected for ρ. As FF is non-initially connected, then by the negation of Definition 9 there is no path $\sigma \in \Theta_{FF}$ to reach (s_k, φ_k) from the initial conditional state c_0. By Definition 2 item 3 if $\Delta_\phi(FF)$ is initially connected, then there is a path $\upsilon \in \Omega_{\Delta_\phi(FF)}$ to reach every state $s \in S$ from the initial state s_0. As $\rho \vDash \phi$, and by Definition 6 item 3 every transition $(s_i, x_i, o, s_{i+1}) \in T$ was induced from a conditional transition $(s_i, \varphi_i) \overset{(x_i,\varphi_i')}{\underset{o}{\rightarrow}} (s_{i+1}, \varphi_{i+1}) \in \Gamma$ and $\rho \vDash \varphi_i \wedge \varphi_i' \wedge \varphi_{i+1}$ that forms a path to reach (s_k, φ_k), which is a contradiction as there is no path $\sigma \in \Theta_{FF}$ to reach (s_k, φ_k). □

Definition 10. *An FFSM FF is* `minimal` *if for all pairs of conditional states of all satisfiable products there are common valid transfer sequences that distinguish both conditional states, i.e.,* $\forall_{c_a=(s_a,\varphi_a),c_b=(s_b,\varphi_b)\in C} \bullet \forall_{\rho\in\Lambda} \bullet \rho \vDash \varphi_a \wedge \varphi_b \Rightarrow \exists_{(c_a,\delta,\nu_a,\gamma_a,\omega_a,c_a')\in\Theta(c_a),(c_b,\delta,\nu_b,\gamma_b,\omega_b,c_b')\in\Theta(c_b)} \bullet \gamma_a \neq \gamma_b \wedge \rho \vDash (\omega_a \wedge \omega_b)$.

Next, we state and prove that an FFSM is minimal when all its valid product FSMs are minimal.

Theorem 4. *An FFSM is minimal if and only if all derived product FSMs are minimal.*

Proof. We break the bi-implication in the thesis into two implications and prove each by contradiction. For the implication left to right, assume that FFSM FF is minimal, but there is a derived FSM $\Delta_\phi(FF)$ for a product ρ which is non-minimal; we obtain a contradiction. Let FFSM $FF = (F, \Lambda, C, c_0, Y, O, \Gamma)$ be minimal and a derived FSM $\Delta_\phi(FF) = (S, s_0, I, O, T)$ be non-minimal for a product $\rho \in \Lambda$ on states $s_a, s_b \in S$. As $\Delta_\phi(FF)$ is non-minimal, then, by the negation of Definition 2 item 4 there is no common input sequence $\alpha \in I^*$ of two paths $\upsilon_a \in \Omega(s_a), \upsilon_b \in \Omega(s_b)$ that distinguish states s_a and s_b. By Definition 10 if FF is minimal, then for every pair of conditional states $c_a = (s_{a_0}, \varphi_{a_0}), c_b = (s_{b_0}, \varphi_{b_0}) \in C$ and for all products $\rho \in \Lambda$ that satisfy the condition $\varphi_{a_0} \wedge \varphi_{b_0}$ there are two paths with a common a distinguishing sequence $\delta \in Y^*$ and ρ also satisfies both path conditions $\omega_a \wedge \omega_b$. However, by Definition 5 every pair of conditional transitions $(s_{a_i}, \varphi_{a_i}) \overset{(x_i,\varphi_i')}{\underset{o}{\rightarrow}} (s_{a_{i+1}}, \varphi_{a_{i+1}}), (s_{b_i}, \varphi_{b_i}) \overset{(x_i,\varphi_i'')}{\underset{o'}{\rightarrow}} (s_{b_{i+1}}, \varphi_{b_{i+1}}) \in \Gamma$, $0 \leq i \leq k$ of the distinguishing sequence δ is satisfied by ρ. As $\rho \vDash \phi$, and by Definition 6 item 3 every pair of conditional transitions $(s_{a_i}, \varphi_{a_i}) \overset{(x_i,\varphi_i')}{\underset{o}{\rightarrow}} (s_{a_{i+1}}, \varphi_{a_{i+1}}), (s_{b_i}, \varphi_{b_i}) \overset{(x_i,\varphi_i'')}{\underset{o'}{\rightarrow}} (s_{b_{i+1}}, \varphi_{b_{i+1}}) \in \Gamma$ is induced to $(s_{a_i}, x_i, o, s_{a_{i+1}}), (s_{b_i}, x_i, o', s_{b_{i+1}}) \in T$ in $\Delta_\phi(FF)$ that distinguishes s_a and s_b, which is a contradiction as there is no distinguishing sequence $\alpha \in I^*$ for states s_a and s_b.

Likewise, for the implication right to left, assume that $\Delta_\phi(FF)$ is minimal for ρ, but FF is non-minimal; we obtain a contradiction. Let $FF = (F, \Lambda, C, c_0, Y, O, \Gamma)$ be non-minimal on conditional state $c_a = (s_{a_0}, \varphi_{a_0}), c_b = (s_{b_0}, \varphi_{b_0}) \in C$, $\rho \models \varphi$, and $\Delta_\phi(FF) = (S, s_0, I, O, T)$ is minimal for ρ. As FF is non-minimal, then by the negation of Definition 10 there is no common input sequence $\delta \in Y^*$ that distinguish conditional states c_a and c_b. By Definition 2 item 4 if $\Delta_\phi(FF)$ is minimal, then there are two paths with a common a distinguishing sequence $\alpha \in I^*$ for every pair of states s_a and s_b. As $\rho \vDash \phi$, and by Definition 6 item 3 every pair of transitions $(s_{a_i}, x_i, o, s_{a_{i+1}}), (s_{b_i}, x_i, o', s_{b_{i+1}}) \in T$ were induced from $(s_{a_i}, \varphi_{a_i}) \overset{(x_i, \varphi_i')}{\underset{o}{\rightarrow}} (s_{a_{i+1}}, \varphi_{a_{i+1}}), (s_{b_i}, \varphi_{b_i}) \overset{(x_i, \varphi_i'')}{\underset{o'}{\rightarrow}} (s_{b_{i+1}}, \varphi_{b_{i+1}}) \in \Gamma$ and ρ satisfies both conditional paths that distinguishes c_a and c_b, which is a contradiction as there is no distinguishing sequence $\delta \in Y^*$ for c_a and c_b. □

4 Implementation

It is well-known that feature models can be translated into propositional formulas; see, e.g., [2, 11]. This translation enables mechanising the analysis of feature-based specifications using existing logic-based tools, such as SAT solvers. In our approach the Z3 tool [26] was used to check propositional formulas for FFSM properties.

We implemented a tool in Java to parse and process FFSMs in an adapted version of KISS format [15] and subsequently generate assertions in the SMT format that correspond to the initial syntactical checks on the FFSM definition (Definition 3) and the semantic FFSM validation properties in Section 3.3.

To check the initial FFSM conditions on the FFSM of Figure 3, we: (i) transform the feature model of Figure 1 into a propositional formula; and (ii) generate assertions to check feature constraints of conditional states and transitions. Subsequently, we check the validity conditions on the generated propositional formulae. The validation process is progressive, starting with validating conditional states and transitions, and proceeding with checking determinism, completeness, initially connected and then minimality.

Example 5. Figure 4 presents parts of the generated SMT files to check validity of conditional states and completeness, where: (a) all features are declared as Boolean variables: (b) the root mandatory and also the feature model propositional formula are asserted; (c) conditional states `Brickles(B)` and `Save(S)` are verified; and (d) a completeness check on the conditional state `Save(S)` for input `Pause` is verified (see Figure 3). To check conditional states we combine and execute parts (a), (b), and (c), while to check completeness (a), (b), and (d) are combined and executed. In the end, for every $(check - sat)$ command we have an answer that we connect back to Java.

In Z3, `push` and `pop` commands are used to temporarily set the context (e.g., with assertions), and once a verification goal is discharged the context can be reset. The $(check - sat)$ command is used to evaluate the assertions which returns (sat or $unsat$). If a conditional state check yields $unsat$, then there is no product that will ever have this state and hence, the FFSM is invalid.

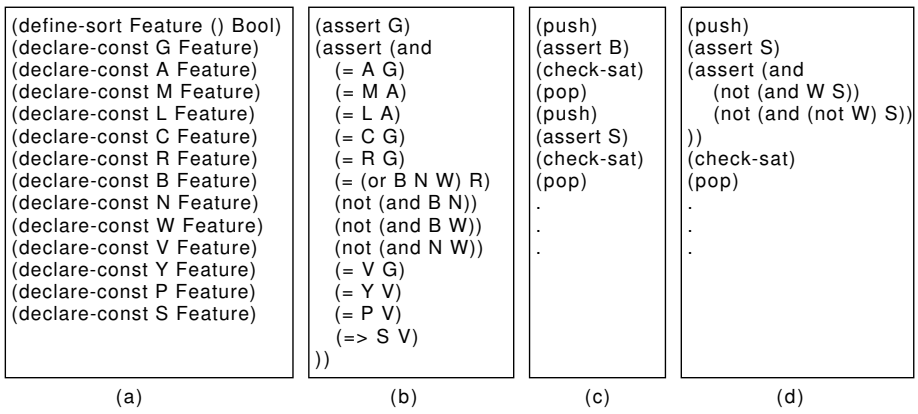

Fig. 4. SMT file generated to check some conditional states and part of the completeness property.

5 Experimental Study

To evaluate the applicability and the efficiency of our approach, we conducted an experiment to evaluate and compare the time required to check properties of FFSMs with the Product by Product (PbP) approach. Our research question is: *How large does an FFSM have to be in order to save time in validation of the FFSM instead of its valid product FSMs?* In the future, we plan to use the same setup (extended with more case studies), to evaluate the test case generation methods on FFSMs.

5.1 Experimental Setup

The setup of our experiment consists of generating random FFSMs varying the number of conditional states from 8 to 70. Every FFSM uses different types of feature models and the arrangement of the features (structure) defines the number of configurations. Initially, we manually inspected a large sample of generated FFSMs, their underlying FSMs and their validation times.

We also modeled the Body Comfort System (BCS) that is used on the VW Golf SPL [21] to reduce the threats to validity and contrast the results from randomly generated (F)FSMs with their real-world counterparts. The original BCS system has 19 non-mandatory features and can have 11616 configurations. In order to manage its complexity, we picked a subset of the feature model (without unresolved dependencies) with 13 non-mandatory features and 8 independent features at the leafs of the feature model. (We plan to introduce hierarchy into our models and treat the complete example in the future.)

Figure 5 shows the feature model of a selected part of features for the BCS. Figure 6 shows the FFSM for a small part of this specification featuring the `Alarm System` (AS) with an optional `Interior Monitoring` (IM) function; and (ii) the `Central Locking System` (CLS) with an optional `Automatic Locking`

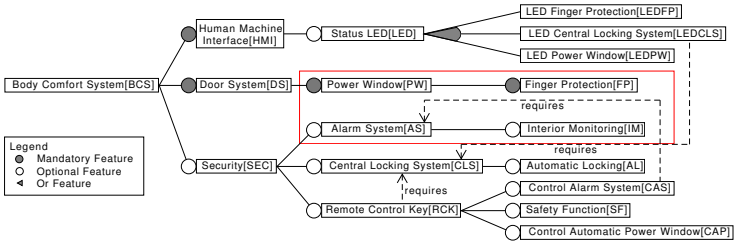

Fig. 5. Reduced feature model of BCS.

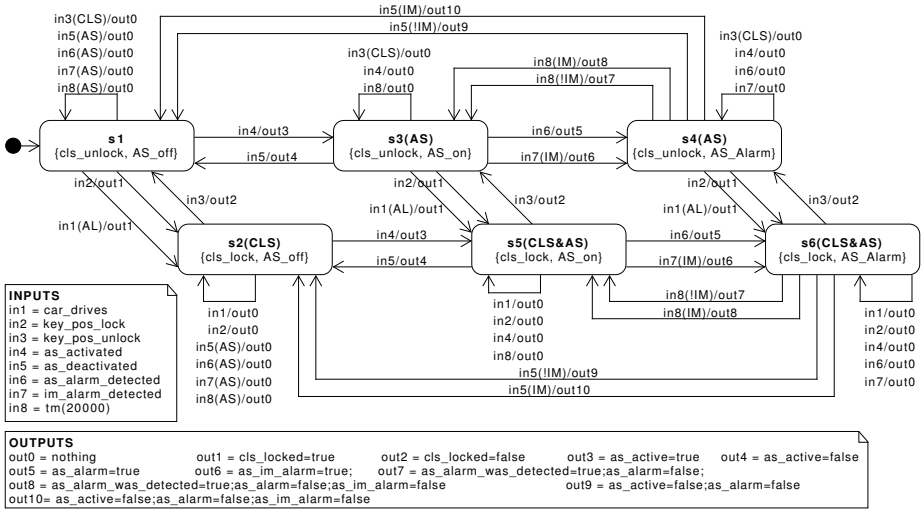

Fig. 6. FFSM for AS and CLS.

(AL) function. This FFSM turns out to be deterministic, complete, initially connected and minimal.

The implementation of our experiments is explained in Section 4. We also implemented a random generator for feature models and (F)FSMs. We designed the random generator to map features to conditional states of the FFSMs. The number of conditional states is hence designed to be proportional to the number of features. We used FeatureIDE [33] to visualise and inspect the feature models and gain insight into the complexity with respect to their structure. The running environment used Windows 7 (64 bit) on an Intel processor i5-5300U at 2.30GHz. [2]

[2] The experiment package for Eclipse IDE can be found in `http://ceres.hh.se/mediawiki/Vanderson_Hafemann`

5.2 Analysis and Threats to Validity

The collected data after running our experiments is visualised in Figure 7. The total validation time is calculated in milliseconds, and we stopped our experiments around 2 million milliseconds (approximately 30 minutes) for 68 non-mandatory features when checking FFSMs.

As an immediate observation, we noticed that the number of non-mandatory features is the dominant factor in the complexity of validation time in both approaches. This was an early observation that was verified by inspecting several data points and resulted in the way we visualised the data in Figure 7.

Additionally, the FSM-based analysis (the product-by-product approach) is very sensitive to the structure of the feature model: the number of independent optional features (optional features appearing in different branches of the feature model) play a significant role in the number of products and hence, the validation time. Thus, we classified the FSM-based data regarding the relative number of independent non-mandatory features in Figure 7; the worst-case time is where all the non-mandatory features are independent; the average case is where half of the non-mandatory features are independent, and the best case is where all non-mandatory features are dependent (form a line in the feature model).

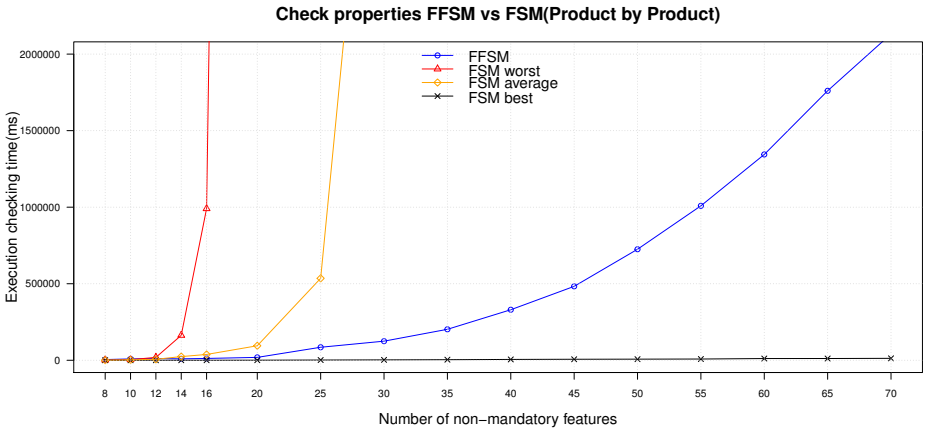

Fig. 7. Execution time for each case per number of non-mandatory features.

To summarise, we conclude that for random SPLs with more than 16 independent non-mandatory features, the FSM-based approach fails to perform within a reasonable amount of time (e.g., ca. 30 minutes), while the FFSM-based approach scales well (regardless of the feature model structure) for up to 70 non-mandatory features.

Regarding our BCS case study, we have obtained similar results regarding the difference between the FFSM and the FSM-based approaches. Namely, for the resulting FFSM with 13 non-mandatory features (of which 8 are independent) and 50 conditional states, the validation takes approx. 500 seconds (˜8 minutes) while

for its 384 configurations (FSMs) we have approx. 700 seconds (˜11 minutes). We expect the scalability of the FFSM-based approach to improve if more structural aspects, e.g., hierarchy, are taken into account. We plan to investigate this further in the near future.

Our experiment has been limited mostly to random feature models and (F)FSMs with a given mapping from feature models to FFSMs. We have only included one realistic case-study for comparison. Both of these issues (the actual structure of feature models and of FFSMs) are threats to the validity of our results for real-world cases. We plan to mitigate this threat by analyzing a number of realistic case studies as a benchmark for our future research. Regarding feature models, as our current results suggest, the FFSM-based approach is not very sensitive to the structure of the feature model and hence, our results are not likely to change much for realistic feature models. Regarding realistic FFSMs, it is common that the flat (i.e., non-hierarchical) FFSMs are the result of the composition of parallel features and hence, their number of states grows exponentially with the number of independent non-mandatory features. Hence, for realistic FFSMs, using the hierarchical structure in the validation process is necessary for sustaining scalability.

6 Related Work

There have been several proposals for the behavioural modelling of SPLs in the literature; we refer to [7, 12, 29] for recent surveys and Thüm et al.'s recent survey [33] for a classification of different SPL analysis techniques. A number of behavioural models proposed in the literature, e.g., these in [22, 1, 10] are based on Finite State Machines or Labeled Transition Systems. They are mainly used to provide the formal specification of SPLs and their formal verification using model checking.

In Feature-Annotated State Machines, some approaches [11, 16] (e.g. the 150% test model) propose a pruning-based approach to UML modelling of SPLs, separating variability from the base models using mapping models. Similar approaches [22, 25] use Statecharts to model reusable components and in their approaches, the instances can also be derived syntactically by pruning. Recent approaches [18, 9] encode feature annotations into transition guards to project model elements. In [18], the authors use model slicing to generate tests for parts of the model to reduce complexity. In Featured Transition Systems [9], model fragments are annotated with presence conditions, i.e., Boolean expressions that define to which products a fragment belongs. However, in none of these approaches, the authors deal with semantic issues in FSMs/LTSs, such as the validation properties considered in our approach, and only verify the syntactical correctness of possible valid products. Moreover, there is a sizable literature focusing on product-based analysis techniques such as syntactic consistency, type checking and model-checking of SPLs [1, 3, 33].

Our proposed test model and validation criteria can be classified as a family-based and feature-oriented specification and analysis method. To our knowledge, however, there only a few pieces of research that extend test models, test case

generation and test case execution to the family-based level; examples of such work include earlier delta-oriented techniques such as [24, 23, 34] and feature-oriented approaches [4, 5, 14]. However, the approach proposed in [4, 6] exploits a non-deterministic test case generation algorithm (with no fault model or finite test suite) and hence, validation of test models is not an issue in their approach. Thus, we are not aware of any prior study one extending the FSM-based test-model validation techniques to the family-based setting.

7 Conclusion

In this paper, we presented the Featured Finite State Machine (FFSM) model as a behavioural test model for software product lines (SPLs). Validation properties were specified for adopting FFSMs as input models for test case generation algorithms and we showed that they coincide with their corresponding properties for the product FSM models. Moreover, a framework for validation of test models using Java and Z3 was implemented.

We conducted an experimental study comparing the validation time of FFSM properties with the accumulated time of validating all FSM product models (both using randomly generated models and a case study for the Body Comfort System). We found that checking collective FFSM models can save significant amount of time for SPLs that have 16 or more independent non-mandatory features.

As future work, we plan to use FFSMs to extend FSM-based test case generation methods to SPLs. Moreover, we plan to extend the FFSM model to Hierarchical FFSMs (using concepts from Statecharts and UML State Machines) to handle the state explosion problem identified in the case study and apply validation (and test case generation) on hierarchical models. Also, in addition to the validation issues, other aspects of the FFSM model can be explored such as applicability, maintainability and the relation between semantic properties such as determinism and minimality.

References

1. Asirelli, P., ter Beek, M.H., Gnesi, S., Fantechi, A.: Formal Description of Variability in Product Families. In: Proceedings of the 15th International Software Product Line Conference (SPLC). pp. 130–139. IEEE (2011)
2. Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: Proceedings of the 9th International Software Product Line Conference (SPLC). pp. 7–20. IEEE (2005)
3. Benduhn, F., Thüm, T., Lochau, M., Leich, T., Saake, G.: A survey on modeling techniques for formal behavioral verification of software product lines. In: Proceedings of the 9th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2015). p. 80. ACM (2015), http://dl.acm.org/citation.cfm?id=2701319
4. Beohar, H., Mousavi, M.R.: Input-output conformance testing based on featured transition systems. In: Proceedings of the Symposium on Applied Computing (SAC 2014). pp. 1272–1278. ACM (2014), http://dl.acm.org/citation.cfm?id=2554850

5. Beohar, H., Mousavi, M.R.: Spinal test suites for software product lines. In: Proceedings of the 9th Workshop on Model-Based Testing (MBT 2014). EPTCS, vol. 141, pp. 44–55 (2014), http://dx.doi.org/10.4204/EPTCS.141
6. Beohar, H., Mousavi, M.: Spinal Test Suites for Software Product Lines. In: Proc. of MBT 2014. EPTCS, vol. 141, pp. 44–55 (2014)
7. Beohar, H., Varshosaz, M., Mousavi, M.R.: Basic behavioral models for software product lines: Expressiveness and testing pre-orders. Science of Computer Programming 123, 42–60 (2016), http://dx.doi.org/10.1016/j.scico.2015.06.005
8. Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., Pretschner, A.: Model-Based Testing of Reactive Systems, Advanced Lectures, Lecture Notes in Computer Science, vol. 3472. Springer-Verlag (2005)
9. Classen, A., Cordy, M., Schobbens, P.Y., Heymans, P., Legay, A., Raskin, J.F.: Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. IEEE Transactions on Software Engineering 39(8), 1069–1089 (2013)
10. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A.: Symbolic model checking of software product lines. In: Proceeding of the 33rd international conference on Software engineering (ICSE). p. 321. ACM Press (2011)
11. Czarnecki, K., Antkiewicz, M.: Mapping Features to Models: A Template Approach Based on Superimposed Variants. In: Proceedings of the 4th international conference on Generative Programming and Component Engineering (GPCE). pp. 422–437. Springer (2005)
12. Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wasowski, A.: Cool features and tough decisions. In: Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS). pp. 173–182. ACM Press (2012)
13. Czarnecki, K., Wasowski, A.: Feature diagrams and logics: There and back again. In: Proc. of SPLC 2007. pp. 23–34. IEEE (2007)
14. Devroey, X., Perrouin, G., Papadakis, M., Legay, A., Schobbens, P., Heymans, P.: Featured model-based mutation analysis. In: Proceedings of the 38th International Conference on Software Engineering (ICSE 2016). pp. 655–666. ACM (2016), http://doi.acm.org/10.1145/2884781
15. Edwards, S.A.: Languages for Digital Embedded Systems. Springer (2000)
16. Grönninger, H., Krahn, H., Pinkernell, C., Rumpe, B.: Modeling Variants of Automotive Systems using Views. In: Tagungsband Modellierungs-Workshop MBEFF: Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen. p. 14. TU Braunschweig (2008)
17. Hierons, R.M., Bogdanov, K., Bowen, J.P., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., et al.: Using formal specifications to support testing. ACM Computing Surveys (CSUR) 41(2), 9 (2009)
18. Kamischke, J., Lochau, M., Baller, H.: Conditioned model slicing of feature-annotated state machines. In: Proceedings of the 4th International Workshop on Feature-Oriented Software Development (FODS). pp. 9–16. ACM (2012)
19. Lee, D., Yannakakis, M.: Principles and Methods of Testing Finite State Machines - A Survey. Proceedings of the IEEE 84(8), 1090–1123 (aug 1996)
20. Linden, F., Schmif, K., Rommes, E.: Software Product Lines in Action. Springer (2007)
21. Lity, S., Lachmann, R., Lochau, M., Schaefer, I.: Delta-oriented Software Product Line Test Models - The Body Comfort System Case Study. Tech. rep. (2013)
22. Liu, J., Dehlinger, J., Lutz, R.: Safety analysis of software product lines using state-based modeling. Journal of Systems and Software 80(11), 1879–1892 (2007)

23. Lochau, M., Lity, S., Lachmann, R., Schaefer, I., Goltz, U.: Delta-oriented model-based integration testing of large-scale systems. Journal of Systems and Software 91, 63–84 (2014), http://dx.doi.org/10.1016/j.jss.2013.11.1096
24. Lochau, M., Schaefer, I., Kamischke, J., Lity, S.: Incremental model-based testing of delta-oriented software product lines. In: Proceedings of the 6th International Conference on Tests and Proofs (TAP 2012). Lecture Notes in Computer Science, vol. 7305, pp. 67–82. Springer (2012), http://dx.doi.org/10.1007/978-3-642-30473-6
25. Luna, C., Gonzalez, A.: Behavior Specification of Product Lines via Feature Models and UML Statecharts with Variabilities. In: Chilean Computer Science Society (SCCC). pp. 9–16. IEEE (2008)
26. de Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 337–340. Springer (2008)
27. Oster, S., Wubbeke, A., Engels, G., Schurr, A.: A Survey of Model-Based Software Product Lines Testing. In: Model-Based Testing for Embedded Systems, pp. 338–381. CRC Press (2012)
28. Petrenko, A., Bochmann, G.v., Luo, G.: Selecting test sequences for partially-specified nondeterministic finite state machines. In: International workshop on Protocol test systems (IWPTS). pp. 95–110. Chapman & Hall (1995)
29. Schaefer, I., Rabiser, R., Clarke, D., Bettini, L., Benavides, D., Botterweck, G., Pathak, A., Trujillo, S., Villela, K.: Software diversity: state of the art and perspectives. International Journal on Software Tools for Technology Transfer 14(5), 477–495 (2012)
30. Schobbens, P.Y., Heymans, P., Trigaux, J.C.: Feature Diagrams: A Survey and a Formal Semantics. In: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE). pp. 139–148. IEEE (2006)
31. SEI: A framework for software product line practice (2011), http://www.sei.cmu.edu/productlines/tools/framework/
32. Simao, A., Petrenko, A.: Fault Coverage-Driven Incremental Test Generation. The Computer Journal 53(9), 1508–1522 (2010)
33. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming 79, 70–85 (2014)
34. Varshosaz, M., Beohar, H., Mousavi, M.R.: Delta-oriented FSM-based testing. In: Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM 2015). Lecture Notes in Computer Science, vol. 9407, pp. 366–381. Springer (2015), http://dx.doi.org/10.1007/978-3-319-25423-4