Reducing the Concretization Effort in FSM-Based Testing of Software Product Lines

Vanderson Hafemann Fragal, Adenilso Simao
Institute of Math. and
Computer Sciences (ICMC)
University of São Paulo - Brazil
{vhfragal, adenilso}@icmc.usp.br

André Takeshi Endo
Federal Technological University
of Paraná - Brazil
andreendo@utfpr.edu.br

Mohammad Reza Mousavi
Centre for Research on
Embedded Systems (CERES)
Halmstad University, Sweden
m.r.mousavi@hh.se

***Abstract*—To test a Software Product Line (SPL), the test artifacts and the techniques must be extended to support variability. In general, when new SPL products are developed, more tests are generated to cover new or modified features. A dominant source of extra effort for such tests is the concretization of newly generated tests. Thus, minimizing the amount of new non-concretized tests required to perform conformance testing on new products reduces the overall test effort. In this paper, we propose a test reuse strategy for conformance testing of SPL products that aims at reducing test effort. We use incremental test generation methods based on finite state machines (FSMs) to maximize test reuse. We combine these methods with a selection algorithm used to identify non-redundant concretized tests. We illustrate our strategy using examples and a case study with an embedded mobile SPL. The results indicate that our strategy can save up to 36% of test effort in comparison to current test reuse strategies for the same fault detection capability.**

I. Introduction

Software Product Lines (SPLs) address variability by capturing various software features in an organized structure. The delta-oriented approach [16] is a software product line engineering approach, where the SPL is designed in terms of a core module and a set of delta modules. The core module is a set of features for a basic product and the delta modules add, remove, or modify features from the core module to design new products. Changing the specification of a product or deploying a similar product may require substantial effort for conformance testing. We study this problem in the context of Model-Based Testing (MBT), where models are used to steer the test process effort with the goal of making it more structured and more efficient.

One important step in the MBT test process is concretization of abstract test cases [24]. Generated abstract tests are augmented with concrete implementation-specific data making them executable in the system under test. Checking the conformance of an SPL product requires extra effort for each new test that needs to be concretized. According to case studies [24], [11], the cost to manually concretize a test case is several times (around 200) greater than the cost of executing the same concretized test. To tackle this problem, adapters can be developed to automate the concretization process. However, the adapters often need to be modified for new versions/products. For example, systems that constantly evolve (e.g., graphical user interface systems) cannot afford to update adapters of each new version of the system, which often takes more time than manually testing the system [6].

In this paper, we propose a test reuse strategy named Incremental Regression-based Testing for Software Product Lines (IRT-SPL) that aims at reducing the test effort of newly designed SPL products by reducing the number of new tests that need to be concretized for conformance testing. To this end, we maximize the reuse of tests by processing concretized tests of all old products and incrementing some of them to obtain a small set of tests to concretize. We use finite state machines as test models, which are fundamental semantic models for reactive systems [2].

The contributions of this paper are: (i) the test reuse strategy, (ii) the test case selection algorithm, and (iii) their practical evaluation against other common approaches using an SPL case study. Figure 1 presents an overview of contributions. We improve the reuse of tests by incrementing existing concretized tests from all old products for the new behavior. In this figure, Product 5 is the one that is recently developed and hence, new test cases need to be generated and concretized for its test suite. To this end, we defined a new test case selection and reuse strategy that takes the set of all generated test cases into account and using this information, steers the test case generation algorithm to generate a minimal increment on the readily concretized sequences. An experimental evaluation of our proposed strategy was conducted using a case study for the embedded Mobile Media SPL [10]. The obtained results show that our strategy can save up to 36% of test effort for 24 products with typical concretization effort estimated from the literature.

The remainder of this paper is organized as follows. Section 2 presents some preliminary notions and concepts regarding SPLs and FSMs. Section 3 presents our test reuse strategy and the selection algorithm. Section 4 provides the results of our experimental study for the embedded Mobile Media SPL. Section 5 discusses the related work, and finally, Section 6 concludes the paper and presents the directions for future work.

II. Background

In this section, we present basic definitions regarding SPLs, FSMs and some of their test properties, and finally, the

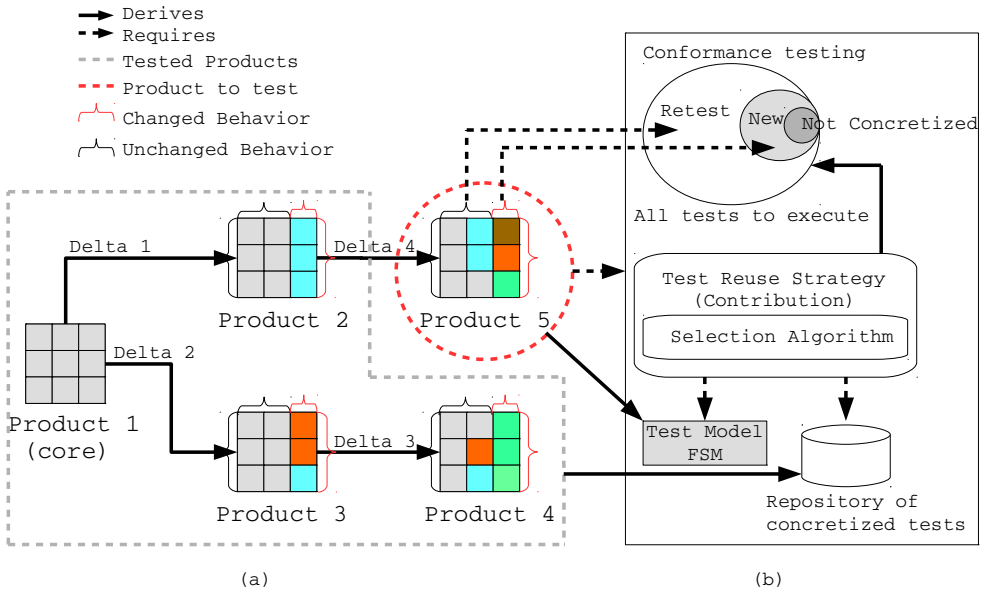


Fig. 1. (a) Derivation of SPL products; and (b) Overview of our contributions.

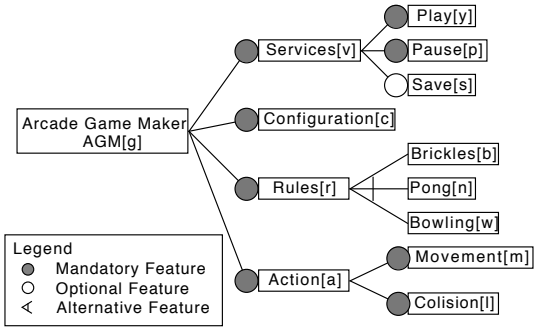


Fig. 2. AGM Feature Model (adapted from [21]).

concretization effort analysis.

A. Software Product Lines

A *feature* is an atomic unit used to differentiate the products of an SPL. Let F be the set of features. The feature structure can be represented by a *feature diagram* [20]. In a feature diagram, some notational conventions are used to represent *commonalities* and *variabilities* of an SPL (e.g., mandatory, optional, and alternative features). A *product configuration* p is defined by a set of features $p \subseteq F$ that the conjunction of literals expressed as boolean variables satisfies all feature constraints. To illustrate the concepts throughout the paper, we use the following SPL.

Example 1. The Arcade Game Maker (AGM) [21] can produce arcade games with different game rules. Figure 2 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.

B. Finite State Machines

FSMs, as defined below, have been traditionally used for modeling and model-based testing different types of reactive hardware [14] and software systems [13], [12], [26].

Definition 1. FSM Model. An FSM M is a 7-tuple $(S, s_0, I, O, D, \delta, \lambda)$, 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, $D \subseteq S \times I$ is a specification domain, $\delta : D \to S$ is a transition function, and $\lambda : D \to O$ is an output function.

A tuple $(s, x) \in D$ determines uniquely a *defined transition* of M. A sequence $\alpha = x_1, ..., x_k, \alpha \in I^*$ is a *defined input sequence* at state $s \in S$, if there exist states $s_1, ..., s_{k+1} \in S$, where $s = s_1$ such that $(s_i, x_i) \in D$ and $\delta(s_i, x_i) = s_{i+1}$, for all $1 \leq i \leq k$. Notation $\Omega(s)$ is used to denote all defined input sequences for state $s \in S$ and Ω_M denotes $\Omega(s_0)$. We extend the transition function from input symbols to defined input sequences, including the *empty sequence* ε, assuming $\delta(s, \varepsilon) = s$ for $s \in S$. Given a set $C \subseteq \Omega(s) \cap \Omega(s')$, states s and s' are C*–equivalent* if $\lambda(s, \gamma) = \lambda(s', \gamma)$ for all $\gamma \in C$. Otherwise, if there exists a sequence $\gamma \in C$ such that $\lambda(s, \gamma) \neq \lambda(s', \gamma)$, then s and s' are C*–distinguishable*.

According to Definition 1, M is deterministic. If $D = S \times I$, then M is a *complete* FSM; otherwise, it is a *partial* FSM. An FSM M is said to be *initially connected*, if for each state $s \in S$, there exists an input sequence $\alpha \in \Omega_M$, such that $\delta(s_0, \alpha) = s$, α is called a *transfer sequence* for state s. Unreachable states need to be removed to make the FSM valid for test case generation. An FSM M is *minimal* (or *reduced*), if every pair of distinct states $s, s' \in S$ can be distinguished by a set $C \subseteq \Omega(s) \cap \Omega(s')$.

Example 2. There are six possible product configurations that

can be derived from the AGM FM. The FSM M_3 of the third configuration is shown in Figure 3. 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. It is straightforward to check that M_3 is a deterministic, complete, initially connected and minimal FSM.

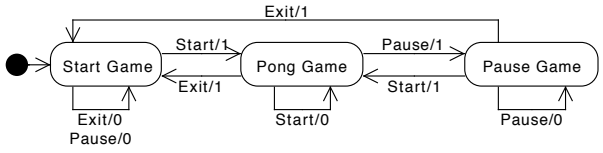


Fig. 3. FSM of the third product configuration of AGM.

C. Test Properties

In this paper, we adapt the P method, a test generation method based on fault domain for FSMs [22]. We use the notion of test suite completeness with respect to a given fault domain and sufficiency conditions based on convergence and divergence properties introduced in [22].

Given sequences $\alpha, \beta, \gamma \in I^*$, a sequence α is *prefix* of a sequence β, denoted by $\alpha \leq \beta$, if $\beta = \alpha\gamma$, for some sequence γ, and γ is a *suffix* of β. A sequence α is *proper prefix* of $\beta, \alpha < \beta$, if $\beta = \alpha\omega$ for some $\omega \neq \varepsilon$. We denote by $pref(\beta)$ the set of prefixes of β, i.e. $pref(\beta) = \{\alpha | \alpha \leq \beta\}$. For a set of sequences A, $pref(A)$ is the union of $pref(\beta)$ for all $\beta \in A$. If $A = pref(A)$, then we say that A is *prefix-closed*. Moreover, we say that a sequence $\alpha \in A$ is *maximal* in A if there is no sequence $\beta \in A$ such that α is a proper prefix of β.

Definition 2. [22] Test case and test suite. A defined input sequence of FSM M is called a *test case* of M. A *test suite* of M is a finite prefix-closed set of test cases of M.

A set $C \subseteq \Omega_M$ is a *state cover* for an FSM M if, for each state $s \in S$, there exist sequences $\alpha \in C$ such that $\delta(s_0, \alpha) = s$. The set $C \subseteq \Omega_M$ *covers* a transition (s, x) when there exist sequences $\alpha \in C$ such that $\delta(s_0, \alpha) = s$ and $\alpha x \in C$. The set C is a *transition cover* (for M) if it covers every defined transition of M. A set of sequences is *initialized* if it contains the empty sequence.

Throughout this paper, we assume that $M = (S, s_0, I, O, D, \delta, \lambda)$ and $N = (Q, q_0, I, O', D', \Delta, \Lambda)$ are a specification and an implementation, respectively. Moreover, n is the number of states of M. We denote by $\Im$ the set of all deterministic FSMs (different from M) with the same input alphabet as M for which all sequences in Ω_M are defined, i.e. for each $N \in \Im$ it holds that $\Omega_M \subseteq \Omega_N$. The set $\Im$ is called a *fault domain* for M and $\Im_n$ is the set of FSMs from $\Im$ with n states.

The distinguishability of FSMs is defined as the corresponding relation of their initial states. Thus, test cases are assumed to be applied in the initial state. Given a test suite T, k FSMs are *T-equivalent* if for every test case of T all k FSMs return the same output sequence. Let $N \in \Im$ and $\Im(T)$ be the subset from $\Im$ such that N and M are T-equivalent, and $\Im_n(T) = \Im_n \cap \Im(T)$ be the set of FSMs from $\Im$ in which are T-equivalent to M and have at most n states.

The main results from [22] establish the *full fault coverage criterion* based on convergence and divergence properties. Convergent test cases reach the same state of M, while divergent test cases reach different states in M. A test suite T is n-complete for an FSM M with n states if T contains a specific convergence-preserving transition cover set for M. Two convergent test cases of T satisfy the convergence-preserving property if they are convergent in all FSMs of the set $\Im_n(T)$. When a test suite T is n-complete for an FSM M (satisfies the full fault coverage criterion), then, by executing T we are capable of detecting all faults in any FSM implementation $N \in \Im_n(T)$.

There exist several methods to generate n-complete test suites [5], [17], [22]. For example, the P method [22] uses two input parameters: a deterministic, initially connected, and minimal FSM M; and an initial test suite T. The initial set T can be empty, and new test cases are added/incremented (if necessary) until an n-complete test suite for M is produced. Therefore, the P method checks if all implementations $N \in \Im_n$ can be distinguished from M using T, and decides if more sequences need to be added to T. Experimental evaluation indicates that the P method often results in smaller n-complete test suites compared with other methods [8].

The reuse of test cases is important to save test effort in several domains that develop similar systems. In this paper, we demonstrate how this can be exploited in the testing of SPLs.

D. Concretization Effort

Given a new product configuration to test, first, we check the changed behavior to ensure that they behave as intended by concretizing and executing a set of new test cases. Then, to ensure that the unchanged behavior was not affected by modifications we execute a set of test cases to retest such behavior.

Definition 3. Test case and test suite size. The size of a test case $\alpha \in I^*$ denoted by $|\alpha|$ is calculated by the number of inputs that it contains, i.e., $|\alpha| = k, \alpha = x_1, ..., x_k$. Similarly, $|T|$ is the size of a test suite T calculated by the sum of all test cases plus the reset operation for each maximal test case, i.e., $|T| = \sum(|\alpha| + 1), \alpha \in T, \nexists_{\beta \in T} \bullet \beta = \alpha\gamma \wedge \gamma \neq \varepsilon$.

Example 3. Given a test suite $T = \{a, b, c\,;\, a, c\}$ the size of T is $|T| = (|a, b, c| + 1) + (|a, c| + 1) = 7$.

Definition 4. Test effort. The *effort* to test a new product configuration is the sum of test cases that have to be executed plus those that have to be concretized times a value x (manual

concretization value over execution), i.e., $effort = (concrete * x) + execution$.

Case studies [24], [11] show that the value x is around 200. Execution cost is calculated based on the number of test cases that have to be executed for both changed and unchanged behavior. Concretization cost is calculated from new test cases.

Definition 5. Execution and concretization costs. Given a prefix-closed test suite T, a set of new test cases $NT \subseteq T$ and a set of concretized test cases $D = T \backslash \{NT\}$, the *execution cost* is equivalent to the size of $|T|$, i.e., $execution = |T|$. The *concretization cost* is calculated from new test cases $\alpha \in NT$ that have to be concretized. If a proper prefix β of a new test case $\alpha = \beta\gamma, \gamma \neq \varepsilon$ was already concretized before, i.e., $\beta \in D$, then we reuse β and the concretization cost is the sum of the size of all suffixes γ, i.e., $concrete = \sum |\gamma|$.

Example 4. Given a test suite $T = pref\{a, b, c\,;\, a, c\,;\, a, d\}$ and a subset of new test cases $NT = \{a, d\}$, the concretization cost is $concrete = |d| = 1$.

III. Testing Products Incrementally

In this section, we present our test reuse strategy and the selection algorithm.

A. Test Reuse Strategy

The Incremental Regression-based Testing for Software Product Lines (IRT-SPL) strategy was inspired by earlier approaches in this domain [22], [25], [3], [15] and developed to improve the reuse of tests cases to reduce concretization effort. We use incremental test generation methods (to increment concretized tests) for full fault coverage criterion explained in Section II-C. Figure 4 (a) presents the main steps of IRT-SPL.

Given a new product configuration $p \in F$ that require testing, to check its conformance we design the test model as an FSM M, obtain all defined test cases $D \subseteq \Omega_M$ that were concretized in old products, and execute the following sequence of steps and conditions:

Step 1: Process all defined test cases of D to find divergent, convergent, and convergence-preserving test cases [22]. Also, initialize the set of new test cases $NT = \varnothing$.

Condition 1: Is set D n-complete for M? When the answer is false, move to Step 2; otherwise, copy D to set T and move to Step 3.

Step 2: Increment test cases using a test generation method. Incremental test generation methods use a cost calculation that decides which new test case gives a small increment based on test cases of D. Thus, we used the P method [22] for this step. New test cases are incremented from D and put in NT, resulting in an n-complete test suite $T = D \cup NT$.

Step 3: Obtain test cases using our selection algorithm. Execute the selection algorithm using M and T as parameters, obtain the n-complete test suite $S \subseteq T$ and return $R = S - NT$ as the set of selected concretized test cases and NT as the set of non-concretized test cases.

In general, incremental test case generation algorithms check available test cases that can be incremented. Sometimes two equivalent test cases (with the same size for a given test criteria) can be used to increment and generate a new test case. Selecting one of such tests might result in a larger/smaller test suite at the end of the process. Thus, we pick the first test case (greedy) as generating a minimal test suite (even by incrementing test cases) can lead to an exponential number of situations.

B. Selection Algorithm

The selection algorithm proposed was developed on the last step of our IRT-SPL strategy. Given an FSM M, an n-complete test suite T for M, and an initialized convergence-preserving transition cover set $O \subseteq T$, we select non-redundant test cases of O to obtain the n-complete test suite $S \subseteq T$. Set O is the one that controls model coverage regarding the full fault coverage which might contain redundant tests. Each test in O requires a set of tests cases in T to maintain the set property considering M. Thus, we select a subset $G \subseteq O$ resulting set $S \subseteq T$.

The main steps are:

Step 1: Identify all redundant test cases. All test cases of O that cover each transition of M are identified. Also, the resulting test suite S and set $G \subseteq O$ are initialized.

Condition 1: Is set S n-complete for M? The set $G \subseteq O$ must be a transition cover set with a convergence-preserving property [22]. When the answer is true, return S; otherwise, move to Step 2.

Step 2: Select one transition t of M. Only transitions not covered by G are selected. Select a test case $a \in O \backslash G$ that covers t. Sometimes there are several redundant test cases in O that cover t. Then, select the test case that gives the smallest increment of test cases for S. For equivalent test cases (which lead to the same size) we just select the first in the line (greedy).

Step 3: Given a test case a from Step 2, identify the set of related test cases $E \subseteq T$ required to cover t. Then, a is included in G and E in S.

Theorem 1. Given an n-complete test suite T for an FSM M, the selection algorithm terminates with an n-complete test suite $S \subseteq T$.

An extended version of this paper with the proof and detailed analysis on the selection algorithm can be found elsewhere.

Example 5. Assume that two products were already tested for *Brickles*, with and without the *Save* option. Figure 5 shows four test case sets generated by IRT-SPL for the third product presented in Example 2: (a) defined test cases D for M_3 that were already concretized before; (b) n-complete test suite T for M_3 generated by P method by incrementing D; (c) a selected n-complete test suite $S \subseteq T$ for M_3 generated by our selection algorithm; and (d) test case set R for retest unchanged behavior. Test cases were simplified for readability and each input corresponds to: (i) SG - Start; (ii) PS - Pause; (iii) EX - Exit; and (iv) SV - Save.

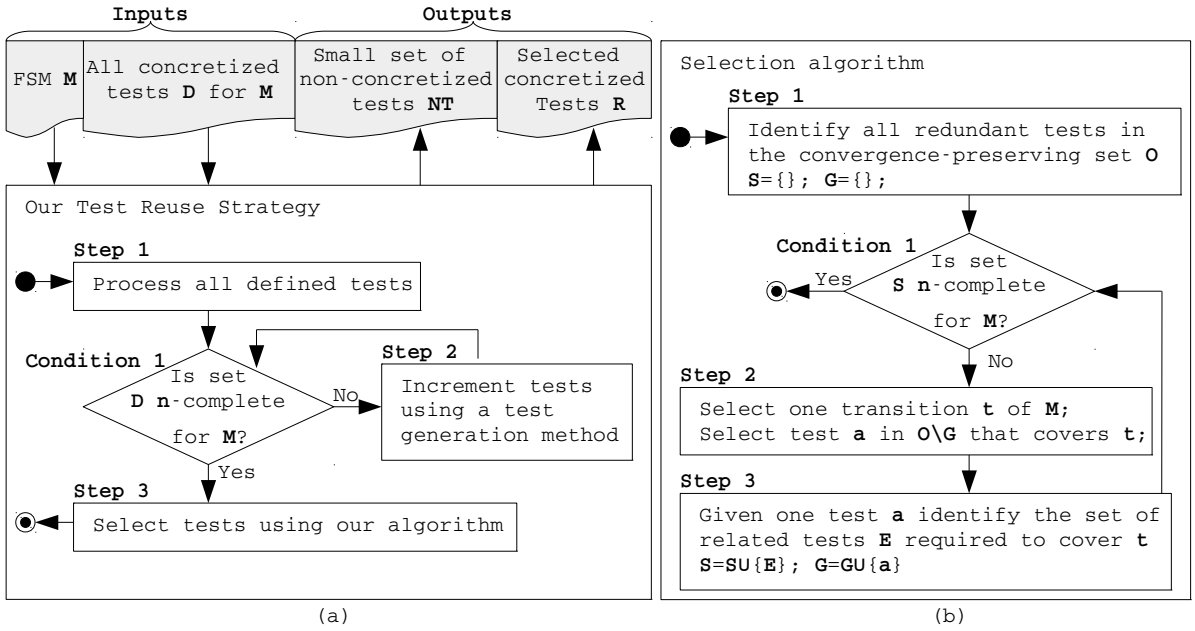


Fig. 4. (a) IRT-SPL test reuse strategy, and (b) selection algorithm.

1 PS,EX	1 PS,EX
2 PS,PS,EX	2 PS,PS,EX
3 EX,PS,EX	3 EX,PS,EX
4 SV,PS,EX	4 SV,PS,EX
5 SG,PS,PS,EX	5 SG,PS,PS,EX
6 SG,EX,PS,EX	6 SG,EX,PS,EX
7 SG,SV,PS,EX	7 SG,SV,PS,EX
8 SG,SG,PS,EX	8 SG,SG,PS,EX
9 SG,PS,EX,PS,EX	9 SG,PS,EX,PS,EX
10 SG,PS,SG,PS,EX	10 SG,PS,SG,PS,EX
11 SG,PS,SV,EX,PS,EX,SG,PS,EX	11 SG,PS,SV,EX,PS,EX,SG,PS,EX
12 SG,SV,SV,PS,EX,PS,EX,SV,PS,EX	12 SG,SV,SV,PS,EX,PS,EX,SV,PS,EX
	13 SG,PS,PS,**PS**
(a)	(b)
1 PS,EX	1 PS,EX
2 EX,PS,EX	2 EX,PS,EX
3 SG,SG,PS	3 SG,SG,PS
4 SG,PS,PS,EX	4 SG,PS,PS,EX
5 SG,EX,PS,EX	5 SG,EX,PS,EX
6 SG,PS,SG,PS	6 SG,PS,SG,PS
7 SG,PS,EX,PS,EX	7 SG,PS,EX,PS,EX
8 SG,PS,PS,**PS**	
(c)	(d)

Fig. 5. Test case sets: (a) defined test cases D for M_3; (b) n-complete test suite T for M_3; (c) selected n-complete test suite S for M_3; and (d) test case set R for retest unchanged behavior.

Note that the difference between (a) and (b) is the addition of Line 13 on (b) and only the last inputs are in bold. Since all four test case sets are prefix-closed, every prefix is also a test case to be counted. Notice that the prefix SG, PS, PS of Line 13 (b) is already present on Line 5 as a prefix that can be reused. On (c), the algorithm to select concretized test cases is executed using as input M_3 and (b), such that it keeps new test cases and reduce the test cases set of (a) as some of them are redundant to cover the unchanged behavior. Thus, the set of new test cases is composed by Line 8 (c) (i.e., SG,PS,PS,PS) and the test case set for the retest is (d).

The effort required to check the conformance of p_3 using M_3 and IRT-SPL is calculated by $\mathit{effort} = (\mathit{concrete} * x) + \mathit{execution}$. Assuming that $x = 10$, we have the set of new test cases $NT = \{SG, PS, PS, PS\}$, the reduced test suite S is Figure 5 (c), $\mathit{execution} = |S| = 37$, $\mathit{concrete} = |\{PS\}| = 1$, resulting in $\mathit{effort} = (1 * 10) + 37 = 47$.

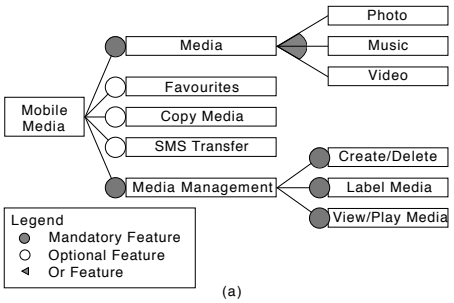
IV. Experimental Study

To evaluate the applicability of IRT-SPL and the reuse efficiency of our selection algorithm, we conducted a study to compare the effort required to test new products to other test case reuse strategies. Our research question is: *How much test effort can be saved using IRT-SPL to test a set of new SPL products compared to existing test case reuse strategies?*

A. Experimental Setup

The setup of our experiment consists of designing several SPL products in different orders and comparing the total effort required to test all products. We compare the effort of IRT-SPL to other reuse strategies found in the literature. A survey on some approaches [7], [19], [4], [25], [3], [15] showed two generic reuse strategies, which are described as follows:

1) The first reuse strategy TSPL was named after the FSM-TSPL approach from Capellari et al. [3]. They only reuse test cases of the last product for conformance testing of a new product where they increment test cases.
2) The second reuse strategy DIATP was named after the Delta-oriented Incremental Architecture-based Testing Process found in the approach of Lochau et al. [15].



(a)

Products	MP	MM	MV	F	CM	SMS	Products	MP	MM	MV	F	CM	SMS
1	X						13	X			X	X	
2		X					14		X		X	X	
3			X				15			X	X	X	
4	X			X			16	X				X	X
5	X				X		17		X			X	X
6	X					X	18			X		X	X
7		X		X			19	X			X		X
8		X			X		20		X		X		X
9		X				X	21			X	X		X
10			X	X			22	X			X	X	X
11			X		X		23		X		X	X	X
12			X			X	24			X	X	X	X

(b)

Fig. 6. (a) Mobile Media feature model; and (b) derived products from Mobile Media SPL.

> They reuse test cases of all previous products selecting test cases to verify the unchanged behavior of the new product, but they generate new test cases for the changed behavior without the increment of test cases.

To obtain a fair comparison on the effort of our strategy, TSPL, and DIATP we setup similar environments also using the P method for TSPL and DIATP.

The embedded camera Mobile Media SPL [10] was used to compare all three reuse strategies. The Mobile Media SPL contains several features, such as photo manipulation, music, and videos on mobile devices which results in total 56 product configurations where we selected 24 relevant product configurations. Figure 6 (a) presents the feature diagram with three Or features (*Photo(MP), Music(MM),* and *Video(MV)*) and three optional (*Favourites(F), Copy Media(CM),* and *SMS Transfer(SMS)*) used to characterize all possible configurations of the SPL. Figure 6 (b) presents 24 configurations of Mobile Media used to design corresponding products. Note that each product of the order 1 to 24 increases the number of features compared to the previous products.

For each product, an FSM was modeled with varying number of states from three to six, and with fixed eight inputs and two outputs. All FSMs share some properties: complete, deterministic, reduced, and initially connected.

B. Analysis of Results and Discussion

The collected data from the experiments is shown in Figure 7. The variable x is the manual concretization value (from literature $x = 200$) from the effort formula $effort = (concrete * x) + execution$ presented in Section II-D. On (a) and (b) we have the derivation of products with an increasing number of features when $x = 10$ and $x = 100$, respectively. We pick values 10 and 100 to check how it scales for low and high values of x. On (c) and (d) we have the derivation of products with decreasing number of features when $x = 10$ and $x = 100$, respectively. Finally, on (e) and (f) we have the derivation of products with a random number of features when $x = 10$ and $x = 100$, respectively.

We noticed that the total effort required to test the Mobile Media SPL vary according to the value x. First, noticed that for our case study the effort saving percentage when $x = 100$ and $x = 200$ is approximately the same and when x is below 10 our approach does not provide significant saving. Also, the number of newly designed products should be considered as for few products there is no significant difference of effort bellow 4 product configurations. Moreover, only a few design product random orders were considered and relatively small feature models. These are threats to validity of this study.

Our approach assumes new products to be similar to old ones. In the worst case scenario, the new product to be tested is completely different from any product developed before. However, in the SPL approach, this is unlikely as commonalities are propagated throughout the entire family.

In summary, we conclude that our approach provides effort savings starting from 5% when we have more than 4 product configurations and the x value is over 10 and goes up to 36% when we have 24 product configurations and the x value is over 100 compared to other test reuse strategies.

The random order the time to execute every strategy depends on the complexity of the P method. Traditional test case generation methods for the full fault coverage that use FSMs as test models (e.g. W [5], HSI [17]) are not incremental. Thus, they cannot increment test cases for new specifications based on old test cases to improve reuse. However, generated new test cases can be compared to old test cases for reuse resulting in a weaker reuse strategy as some of these new test cases may be equivalent to existing old test cases.

Our experiment has been limited mostly to flat FSMs with few states and few new products. One of the issues regarding the P method is the increasing time required to generate test cases based on the number of states times inputs plus the size of the test input set to start with. Both of these issues (few subjects and time) are threats to the validity of our results for real-world cases. We plan to mitigate these threats by analyzing some realistic case studies as a benchmark for our future research. Realistic FSMs use hierarchy to sustain scalability. Thus, an extension of FSM-based test generation methods for full fault coverage is required. We plan to investigate this further in the near future.

V. Related Work

Much recent research has been devoted to developing efficient testing techniques for SPLs by exploiting variability in a systematic manner; we refer to [9], [18], [23] for recent surveys of the field.

There are several incremental test approaches [7], [19], [25], [3], [1], [28], [27] devoted to generating, reusing, and optimizing test suites for SPLs. El-Fakih et al. [7] adapted FSM-based test generation methods for conformance testing that allow

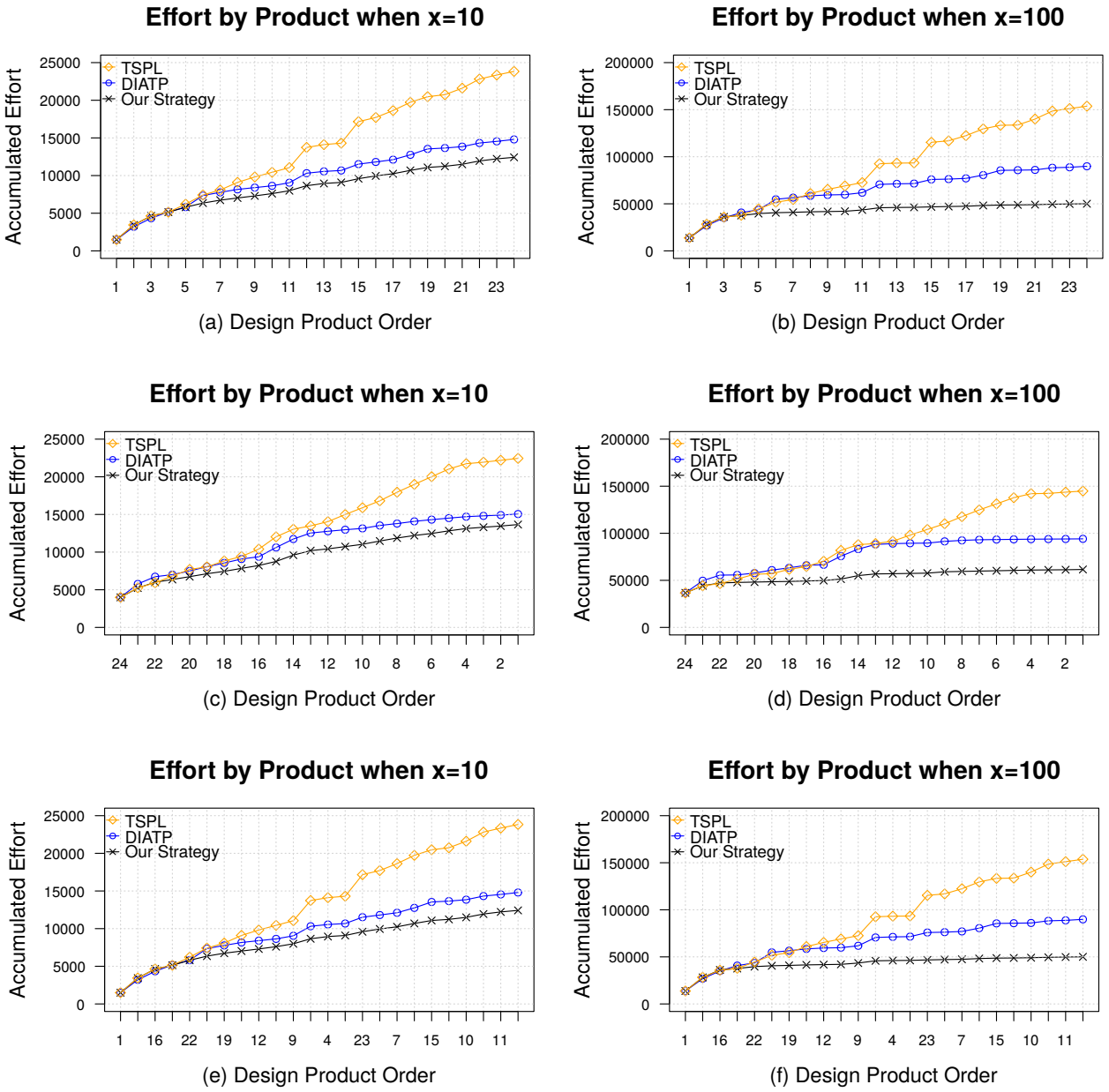


Fig. 7. Accumulated effort per designed product when concretization cost is x times the cost of execution: (a) the increment of features for $x = 10$; (b) the increment of features for $x = 100$; (c) the decrement of features for $x = 10$; (d) the decrement of features for $x = 100$; (e) random features for $x = 10$; and (f) random features for $x = 100$.

the generation of test cases only for the modified parts of an evolving specification. Pap et al. [19] extended their work and designed a bounded incremental algorithm that maintains two sets based on the HSI method [17]. They utilize existing test cases of the previous version of the system to generate test cases for the modified version. Similarly, Capellari et al. [3] explored the FSM-based Testing of SPLs (FSM-TSPL) testing strategy where the P method is used to design new test cases based on the last product derived. Uzuncaova et al. [25] also developed an incremental test generation approach that uses SAT-based analysis to develop tests suites for every product of an SPL, while Baller and Lochau [1] focused on test suite optimization. Moreover, recent delta-oriented approaches [16], [15], [26] developed regression-based SPL approaches to design and reuse test artifacts.

In contrast to current approaches, our work introduces a test reuse strategy focused on reducing test effort of a set of new SPL products. We analyze concretized test cases of

derived products to generate a small set of new test cases to concretize for conformance testing. To our knowledge, there is no proposal that reuse test cases from all previous products to reduce concretization effort for new SPL products using incremental test case generation methods.

VI. Conclusion

This paper proposed a test reuse strategy named Incremental Regression-based Testing for Software Product Lines (IRT-SPL) that aims at reducing test effort on checking conformance of several SPL products. Test cases of previously designed products can be efficiently reused for a newly designed product using incremental test case generation methods to reduce the number of required test cases for concretization. We assume that manual concretization of test cases (as seen in some case studies [24], [11]) is several times (around 200) more expensive than executing the same test case. Thus, the effort required to test a new product is directly related to concretization costs.

Finite State Machines were used to represent the abstract behavior of the products as test models. To maximize reuse of test cases, all test cases that were concretized before are analyzed and some of them are selected to retest the unchanged behavior of the new product under test. Thus, our strategy also contains a selection algorithm to perform the selection of non-redundant concretized test cases.

To illustrate our strategy, we used examples and a case study of an embedded Mobile Media SPL [10]. The results indicate that our approach can save 5% up to 36% test effort for 24 selected products when the manual concretization cost is 10 up to 100 times, respectively, more expensive than execution compared to current test reuse strategies for the same fault detection capability. We found out that the effort saving percentage stabilizes after 100 times for small specifications due to the small number of tests cases for retest.

The problem and the approach described above are inspired by a similar problem and approach in regression testing. Regression testing concerns testing software evolution in time (in versions) while SPL testing is about testing software evolution in space (in features) [9]. Hence, we believe our approach will also be applicable to regression testing. As future work, we plan to investigate test models with hierarchy and adapt test generation methods for such models to handle scalability problems. Also, we intend to investigate more studies regarding formal representations of SPLs to perform incremental reuse of test artifacts.

Acknowledgment: 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 supported by grants from the Swedish Knowledge Foundation (KKS), Swedish Research Council (VR), and the ELLIIT Strategic Research Environment.

References

[1] H. Baller and M. Lochau. Towards incremental test suite optimization for software product lines. In *Proc. of FOSD'14*, pp. 30–36. ACM, 2014.

[2] M. Broy, B. Jonsson, J. P. Katoen, M. Leucker, and A. Pretschner. *Model-Based Testing of Reactive Systems, Advanced Lectures*, Springer, 2005.

[3] M. L. Capellari, I. M. S. Gimenes, A. Simao, and A. T. Endo. Towards Incremental FSM-based Testing of Software Product Lines. In *Proc. of SBQS'12*, pp. 9–23, 2012.

[4] Y. Chen, R. L. Probert, and H. Ural. Regression test suite reduction based on SDL models of system requirements. *J. Software Maintenance and Evolution: Research and Practice*, 21(6):379–405, 2009.

[5] T. S. Chow. Testing Software Design Modeled by Finite-State Machines. *IEEE TSE*, SE-4(3):178–187, 1978.

[6] R. Dev, A. Jääskeläinen, and K. M. *Advances in Computers*, Elsevier, 2012.

[7] K. EI-Fakih, N. Yevtushenko, and G. Bochmann. FSM-based incremental conformance testing methods. *IEEE TSE*, 30(7):425–436, 2004.

[8] A. T. Endo and A. Simao. Evaluating test suite characteristics, cost, and effectiveness of FSM-based testing methods. *Information and Software Technology*, 55(6):1045–1062, 2013.

[9] E. Engström. *Exploring Regression Testing and Software Product Line Testing -Research and State of Practice*. PhD thesis, Lund U., 2010.

[10] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. Ferrari, S. Khan, F. Filho, and F. Dantas. Evolving software product lines with aspects. *Proc. of ICSE'08*, pp. 261–270, 2008.

[11] F. M. Graham D. *Experiences of Test Automation: Case Studies of Software Test Automation*, Addison-Wesley , 2012.

[12] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating Finite State Machines from Abstract State Machines. In *Proc. of ISSTA'02*, pp. 112–122. ACM, 2002.

[13] H. S. Hong, Y. R. Kwon, and S. D. Cha. Testing of object-oriented programs based on finite state machines. In *Proc. of APSEC'95*, pp. 234–241. IEEE, 1995.

[14] D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines - A Survey. *Proc. of the IEEE*, 84(8):1090–1123, 1996.

[15] M. Lochau, S. Lity, R. Lachmann, I. Schaefer, and U. Goltz. Delta-oriented model-based integration testing of large-scale systems. *J. Systems and Software*, 91:63–84, 2014.

[16] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity. Incremental Model-Based Testing of Delta-Oriented Software Product Lines. In *Proc. of TAP'12*, pp. 67–82, 2012.

[17] G. Luo, A. Petrenko, R. Petrenko, and G. V. Bochmann. Selecting Test Sequences For Partially-Specified Nondeterministic Finite State Machines. In *Proc. of Protocol Test Sys.*, pp. 91–106, 1994.

[18] S. Oster, A. Wubbeke, G. Engels, and A. Schurr. A Survey of Model-Based Software Product Lines Testing. In *Model-Based Testing for Embedded Systems*, pp. 338–381. CRC, 2012.

[19] Z. Pap, M. Subramaniam, G. Kovács, and G. Á. Németh. A Bounded Incremental Test Generation Algorithm for Finite State Machines. In *Proc. of TESTCOM'07*, pp. 244–259. Springer, 2007.

[20] P. Y. Schobbens, P. Heymans, and J. C. Trigaux. Feature Diagrams: A Survey and a Formal Semantics. In *Proc. of RE'06*, pp. 139–148. IEEE, 2006.

[21] SEI. A framework for software product line practice, 2011.

[22] A. Simao and A. Petrenko. Fault Coverage-Driven Incremental Test Generation. *Computer J.*, 53(9):1508–1522, 2010.

[23] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake. A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Comp. Surv.*, 47(6):1–45, 2014.

[24] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*, Morgan Kaufmann, 2006.

[25] E. Uzuncaova, S. Khurshid, and D. Batory. Incremental Test Generation for Software Product Lines. *IEEE TSE*, 36(3):309–322, 2010.

[26] M. Varshosaz, H. Beohar, and M. Mousavi. Delta-oriented FSM-based testing. In *Proc. of ICFEM'15*, pp. 366-381. Springer, 2015.

[27] S. Wang, S. Ali, A. Gotlieb, and M. Liaaen. A systematic test case selection methodology for product lines: results and insights from an industrial case study. *Empirical Software Engineering*, 21(4):1586-1622, 2016.

[28] S. Weißleder and H. Lackner. Top-down and bottom-up approach for model-based testing of product lines. In *Proc. MBT'13*, pp. 82-94, 2013.