2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)

Featured Event Sequence Graphs for Model-Based Incremental Testing of Software Product Lines

Tugkan Tuglular
Dept. of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
tugkantuglular@iyte.edu.tr

Mutlu Beyazıt
Dept. of Computer Engineering
Yaşar University
İzmir, Turkey
mutlu.beyazit@yasar.edu.tr

Dilek Öztürk
Dept. of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
dilekozturk@iyte.edu.tr

***Abstract*—The goal of software product lines (SPLs) is rapid development of high-quality software products in a specific domain with cost minimization. To assure quality of software products from SPLs, products need to be tested systematically. However, testing every product variant in isolation is generally not feasible for large number of product variants. An approach to deal with this issue is to use incremental testing, where test artifacts that are developed for one product are reused for another product which can be obtained by incrementally adding features to the prior product. We propose a novel model-based test generation approach for products developed using SPL that follows incremental testing paradigm. First, we introduce Featured Event Sequence Graphs (FESGs), an extension of ESGs, that provide necessary definitions and operations to support commonalities and variabilities in SPLs with respect to test models. Then we propose a test generation technique for the product variants of an SPL, which starts from any product. The proposed technique with FESGs avoids redundant test generation for each product from SPL. We compare our technique with in-isolation testing approach by a case study.**

Keywords— software product lines, model-based testing, incremental testing, event sequence graphs

I. Introduction

The software product lines paradigm promises faster development cycles and increased quality by systematically reusing software assets [1]. The paradigm enables a family of related products to be developed by selecting features from a feature diagram. An example feature diagram, Soda Vending Machine (SVM), is given in Fig.1, which is used as a running example in the paper. Using this diagram related products can be developed such as one serving free tea, one serving both tea and soda in EUR, and one serving just soda in USD.

Model-based testing (MBT) has a high potential to utilize reuse opportunities in testing SPLs [2]. There are various MBT techniques proposed for SPLs in the literature as explained in Section VI. However, it has been noted that most of the existing approaches for SPL testing may potentially show at least one of the following two deficiencies [3]:

1. They require one superimposing specification with all possible variants of the product line [4], which becomes intractable for large-scale product lines because of computational overhead [3].
2. Focus is on structural and syntactical variability [5]; behavioral impact of variations is not considered. Thus, systematic propagation of behavioral properties from one variant to other variants is not available [3].

Incremental testing of SPLs, which is first proposed by Uzuncaova et al. [7], copes with those two deficiencies. The idea of incremental testing of SPLs has been used in various studies (see Section VI). All of those studies are based on finite state machines (FSMs) without explicit mapping between features and FSMs. In other words, it is not shown how a single feature is represented by states, transitions, etc. and how states and transitions representing a single feature are connected to an FSM of a product. These representations are important in practical sense for the techniques to be used by industry for traceability reason.

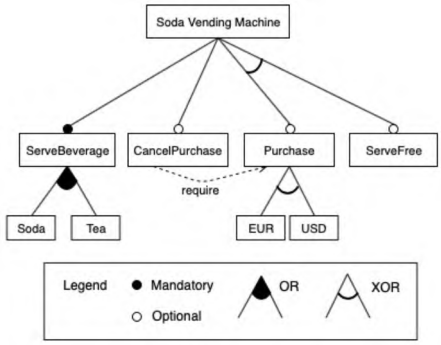

Fig. 1. Example of a SPL feature diagram: soda vending machine (modified from [6])

The novelty of the proposed approach is as follows. We introduce Featured Event Sequence Graphs (FESGs), which are variable test models and used to explicitly capture behavioral variability in SPLs. We model core features, i.e. core of the SPL, and each feature as independent partial or full ESGs, and name them as c-ESG and f-ESG, respectively. The behavior of a given product is then the ESG that results from the combination of core ESG and feature ESGs representing the features of the product.

Initially we have three models for SPL: Feature Model (FM) with product configurations from feature diagram, FESGs for expressing feature-based SPL behavior, and mapping between features and FESGs. We start with a product with core features and selected features, prepare test models for this product and generate test cases for specified coverage criteria. Next, we obtain another valid product with adding features, reuse existing test models as well as test cases through automatic adaptation and composition.

The paper is organized as follows. In Section II, foundations of feature modelling and ESGs are introduced. FESGs for test models is presented in Section III. The incremental SPL testing approach is described in Section IV and validated through a case study in Section V. Related work is given in Section VI and Section VII concludes the paper.

978-1-7281-2607-4/19/$31.00 ©2019 IEEE
DOI 10.1109/COMPSAC.2019.00035

II. FUNDAMENTALS

A. Feature Modeling

Feature diagrams, originally proposed by Kang et al. [8], are used to represent the configuration options and the dependencies of features in software product lines. An example SPL feature diagram is given in Fig. 1. The root represents the SPL and the nodes are features, which can be mandatory or optional. features with OR relationship can exist in a product in different combinations. If there is XOR relationship between two features, only one of them can exist in a product of the SPL. There are also *require* and *exclude* relationships in feature diagrams, where the former denotes a feature that cannot exist without required feature and the latter denotes a feature that omits the excluded feature.

Feature diagrams are generally user-centric. Feature models are formal representations of feature diagrams. In this work, we use the following definitions of feature model and respective product configuration, i.e., a feature selection.

Definition 1. Let B denote the domain of Boolean values by $B = \{false, true\}$. Let F be a finite set of Boolean variables (features). A *feature model (FM) fm:* $(F \rightarrow B) \rightarrow B$ is given as a propositional formula over the set F. [3]

Definition 2. A *product configuration (PC) p:* $F \rightarrow B$ is an assignment of Boolean values to features such that $fm(p) = true$ holds. [3]

Product diagrams are user-centric representations of product configurations, where all decisions about options and selections are made for the product and necessary dependencies are concretized. A product diagram of SVM SPL is given in Fig.2, which shows ServeSodaBeverage, ServeTeaBeverage, CancelPurchase, and EURPurchase features of the product called EUR-Soda-Tea-SVM.

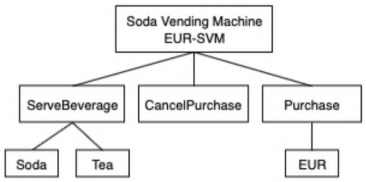

Fig. 2. Product diagram of EUR-Soda-Tea-SVM

B. Test Generation with Event Sequence Graphs

In this work, we use the following definitions of event sequence graphs (ESGs).

Definition 3. An *event sequence graph (ESG)* (V, E) is a directed graph where $V \neq \emptyset$ is a finite set of nodes (vertices) and $E \subseteq V \times V$ is a finite set of *arcs* (*edges*), and $\Xi, \Gamma \subseteq V$ finite sets of distinguished vertices with $\xi \in \Xi, \gamma \in \Gamma$, called entry nodes and exit nodes, respectively. [9]

Example 1. For the ESG given in Fig.3, V= {prompt, payEUR, select, serveSoda, serveTea, cancel, returnMoney}, Ξ={prompt}, Γ={serveSoda, serveTea, returnMoney} and E={(prompt, payEUR), (payEUR, select),(payEUR, cancel), (select, serveSoda),(select, serveTea),(cancel,returnMoney)}. Note that arcs from pseudo vertex [, and to pseudo vertex], are not included in E.

Definition 4. Let (V, E) be an *ESG*. Then a sequence of vertices $< v_0, \ldots, v_k >$ is called an *event sequence* (*ES*) if the sequence is a walk on *ESG*.

Fig. 3. Product ESG of EUR-Soda-Tea-SVM

Pseudo nodes represented by '[' and ']' respectively, are not included in V and also not included in *ESs*. They enable a simpler representation for the algorithms to construct minimal test sets. They are not considered in determining the initial, final vertices, or length of an ES.

Each edge of an ESG represents a legal event pair, or simply, an *event pair* (*EP*). ES $< v_i, v_k >$ of length 2 is an EP.

Example 2. For the ESG given in Fig.3, payEUR-select-serveSoda is an ES of length 3 with initial vertex payEUR and end vertex serveSoda.

Definition 5. An ES $< v_0, \ldots, v_k >$ is called a *complete event sequence* (*CES*), if $v_0 = \xi \in \Xi$ is the entry and $v_k = \gamma$ is the exit. [10]

A complete ES (CES) starts at the entry of the ESG and ends at its exits, i.e., it represents a walk through the ESG. A sequence of n consecutive edges forms an ES of length n+1.

A CES also represents a *test sequence*, i.e. test case, of the ESG realized by the form "(initial) user inputs → (interim) system responses → … → (final) system response" [11].

Example 3. prompt-payEUR-select-serveSoda is a CES of the ESG in Fig.3. CESs represent walks from the entry of the ESG to its exit.

Example 4. For the ESG given in Fig.3, CESs covering all ESs of length 2 (test sequences for length 2) are as follows.

TS1: prompt payEUR, select, serveSoda
TS2: prompt, payEUR, select, serveTea
TS3: prompt, payEUR, cancel, returnMoney

III. FEATURED EVENT SEQUENCE GRAPHS

Information about configurations, which represents a specific product, is not available in feature diagrams. To solve this problem, FMs (Definition 1) and PC (Definition 2) are introduced. Similarly, there is a need for a specific model to associate features to test models, so, when a feature is added to an existing PC, the corresponding test model can be updated accordingly. To fulfill this need, we propose Featured Event Sequence Graphs (FESGs), an extension of ESGs.

Definition 6. A Featured Event Sequence Graph (FESG) is composed of a core ESG (c-ESG) and a set of feature ESGs (f-ESGs) based on PC information. A FESG can be transformed to an ESG but not vice versa.

Definition 7. A core ESG (c-ESG) corresponds to core features, which exist in all product configurations.

Fig.4 shows the c-ESG of the running example. A c-ESG represents the core behavior of SPL. Behavior of selected features represented as f-ESGs are joined to c-ESG to compose the behavior of a specific SPL product. As seen in Fig.4, *prompt* and *select* events are in the core behavior of SVM SPL and exist in every product generated for SVM SPL. The connection information is stored with f-ESGs.

Fig. 4. c-ESG of SVM SPL

Definition 8. A feature ESG (f-ESG) corresponds to a specific feature in the feature model. Different from ordinary ESG nodes and c-ESG nodes, an f-ESG contains nodes associated with variability points, which are called connection events. Connection events are actually events in other c-ESGs or f-ESGs. They are shown as (ESG, Event) pairs.

Fig.5 shows the behavior of Serve Tea Beverage feature of SVM SPL. This f-ESG contains connection point information as seen in the leftmost node and rightmost node, where *serveTea* event can be connected. Fig.6 shows the behavior of EUR Purchase feature of SVM SPL, where *payEUR* event is to be connected to Core_SVM c-ESG and also to Feature_Cancel f-ESG. Those connections are mandatory.

Fig. 5. Serve Tea Beverage feature f-ESG of SVM SPL

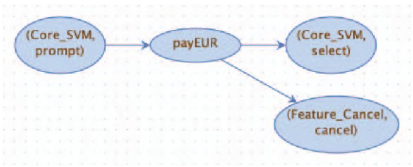

Fig. 6. EUR Purchase feature f-ESG of SVM SPL

In a c-ESG, connection events are not included to exploit reuse. However, each f-ESG holds necessary information about connection events, i.e. variability points. For each leaf in product diagram, there is an f-ESG.

A c-ESG and a set of f-ESGs compose an FESG for the SPL under consideration. An FESG corresponds to the initial test model of SPL, however it is not a superimposed test model. Section IV explains FESG test composition technique.

IV. Model-Based Incremental Testing of Software Product Lines Using FESGs

The proposed model-based incremental testing technique requires three models: (1) Feature Model (FM) with product configurations corresponding to feature diagram, (2) Featured Event Sequence Graphs for expressing SPL behavior of core and features, and (3) mapping between features and FESGs. The first two models are explained in previous sections. The last one, the mapping between features and FESGs. With the selected features for a product, we obtain a product FESG tree. In a product FESG tree, root node stores link to c-ESG and leaf nodes store links to corresponding f-ESGs.

A product FESG tree may contain f-ESGs with events that are not in the product. First, those events must be removed from corresponding f-ESGs. Then the product FESG tree must be converted to a product FESG lattice, where all connection relationships are ordered. The first algorithm of the proposed technique given below achieves this goal. The notation x-ESG means either c-ESG or f-ESG.

Consider another product of SVM SPL, that is USD-Soda-Tea-SVM, which serves Soda and Tea in USD and consumer can cancel the purchase. To exemplify step 2 of Algorithm 1, *EURpay* event is removed from Cancel Purchase feature f-ESG and resulting f-ESG can be seen in Fig.7. Once all events that does not belong to the product are cleared, it is time to build the product FESG lattice.

Algorithm.1: Construction of product FESG lattice

1. for each f-ESG
2. if it contains events, which are not in the product configuration, remove them
3. for each f-ESG (f)
4. if it has connection point(s) to c-ESG (c), build **f → c**
5. if it has connection point(s) to other one or more f-ESGs (g, h, etc.), build **f → g , f → h , etc.**
6. loop back

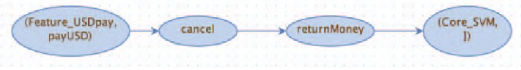

Fig. 7. EURpay removed from Cancel Purchase feature f-ESG

By executing Algorithm 1, we obtain the product FESG lattice for USD-Soda-Tea-SVM as given in Fig.8, where c-ESG is at the top (level 0) and f-ESGs with c-ESG connection points are children of the root (level 1) and so on.

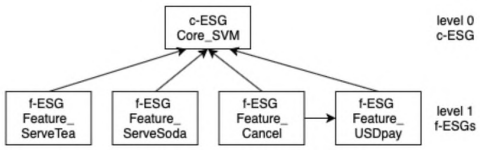

Fig. 8. Product FESG lattice for USD-Soda-Tea-SVM

The partial test sequence(s) to cover ESs of length 2 of each distinct path in the product FESG lattice for product USD-Soda-Tea-SVM are as follows:

PTS1: prompt, ..., select, serveTea
PTS2: prompt, ..., select, serveSoda
PTS3: ..., payUSD, cancel, returnMoney
PTS4: prompt, payUSD, select, ...
PTS5: prompt, payUSD, cancel, returnMoney

As seen above, the first partial test sequence (PTS1) is generated from path **ServeTea → Core**, PTS2 is generated from path **ServeSoda → Core**, PTS3 is generated from path **Cancel → Core,** PTS4 and PTS5 are generated from path **Cancel → USDpay →Core** in Fig.8. It is important to notice that although PTS3 is in PTS5, we do not remove PTS3 since we do not know what kind of combinations it may have with other PTSs.

For the composition of test sequences, i.e. CESs in ESG terminology, we need to combine PTSs in all possible ways. The second algorithm of the proposed technique given below achieves this goal. To improve performance of Algorithm 2, we suggest refactoring design of f-ESGs in such a way that a level n f-ESG stores the connection point information to the same and lower level f-ESGs and if exists to c-ESG.

The *prompt*, *select*, and *payUSD* events are connection points for the partial test sequences of the running example and when they are connected, we obtain the following CESs.

prompt, payUSD, select, serveSoda
prompt, payUSD, select, serveTea
prompt, payUSD, cancel, returnMoney

If we have built the ESG for product USD-Soda-Tea-SVM, we would obtain the ESG shown in Fig.9.

Algorithm.2: Composition of product test sequences

1. find connection points in partial test sequences
2. order them w.r.t their levels in product FESG lattice, c-ESG having the highest order
3. starting from lowest order for each connection point
4. find PTSs from PTS list and classify them as preceding and succeeding sequences with respect to this connection point
5. combine with all preceding sequences and add to PTS list
6. combine with all succeeding sequences and add to PTS list

Fig. 9. Product ESG of USD-Soda-Tea-SVM

The test sequences generated from the ESG in Fig.9 to cover ESs of length 2 would be the same set of CESs.

This simple proof of concept indicates that we can obtain same test sequences from FESG lattice without constructing a model for the complete product. The advantage of using product FESG lattice is that we can reuse partial test sequences for other products that is close as a delta, where delta is a set of paths from leaf to root in product FESG lattice.

Consider another payUSD SVM product that serves soda but not tea. Let's call this product base product as in the literature. If we have built the ESG for this base product USD-Soda-SVM, we would obtain the ESG shown in Fig.10.

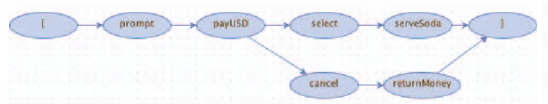

Fig. 10. Product ESG of USD-Soda-SVM

Algorithm.3: Incremental composition of product test sequences

1. find connection points in partial test sequences of the delta
2. order them w.r.t their levels in product FESG lattice, c-ESG having the highest order
3. starting from lowest order for each connection point and using already composed PTSs for the previous product
4. find PTSs from PTS list and classify them as preceding and succeeding sequences with respect to this connection point
5. combine with all preceding sequences and add to PTS list
6. combine with all succeeding sequences and add to PTS list

Assume that PTSs, composed PTSs, and CESs are readily available for this base product. From this base product, we can obtain USD-Soda-Tea-SVM product FESG lattice by adding **ServeTea → Core** path, or delta, to product FESG lattice of base product USD-Soda-SVM. In this case, we would like to obtain CESs for the new product reusing existing PTSs and composed PTSs of the base product. Algorithm 3 achieves this goal. Please note that steps 4-6 are the same with steps 4-6 of Algorithm 2. The main difference is that Algorithm 3 reuses existing PTSs and composed PTSs at each level or order.

The proposed technique in this paper follows regression-based incremental approach. It reuses existing partial test models as well as partial test sequences through automatic adaptation and composition to generate test sequences for the product under consideration.

V. Case Study

As a case study, we work on American type checkers game maker (ATCGM) SPL. To our knowledge, it is the first time this SPL is used in academic research. From American type checkers game maker SPL, three products can be generated: Children checkers, American checkers, and Spanish checkers.

Although American checkers (also British Draughts) and Spanish checkers are well-known games, Children checkers is created by the authors of this paper to show how additional games can be easily generated from SPL. Another reason for selecting ATCGM SPL is to show that if features in feature diagrams are not behavioral, they can be transformed to behavioral features. This is rather the choice of product designers. The proposed approaches in the literature work with both types of feature diagrams.

American checkers is played on a chess board with 12 pieces called pawns. Pawns move one space diagonally and capture by jumping over an opponent's piece. Pawns can only move forward until they reach the last row. After reaching the last row, pawns are promoted to kings and can move forward or backward. Game ends when a player's all pieces are captured. In Spanish checkers, all rules are the same except pawns are promoted to queens. A piece has to capture opponent's piece if it is possible. In our ATCGM SPL, this rule is exempted in children checkers. Moreover, in children checkers, game ends if a pawn reaches to last row.

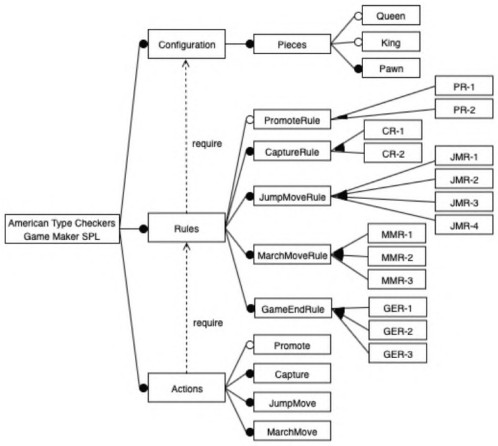

Fig. 11. Feature diagram of American type checkers game maker SPL

Feature diagram of ATCGM SPL is given in Fig.11. Features in the diagram are not necessarily behavioral such as pieces in configuration feature. However, the require property ties them to rules, which are tied to actions and actions are behavioral. The behavioral feature models can be obtained from a transformed product diagram as given in Fig.12.

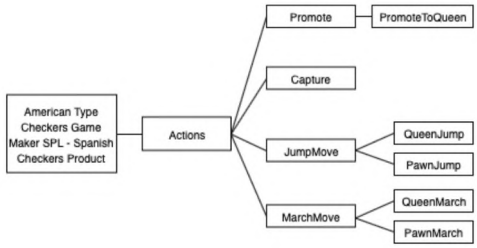

Fig. 12. Transformed product diagram of Spanish checkers

From the transformed product diagram, we obtain product FESG lattice using Algorithm 1. The product FESG lattice for product Spanish checkers is given in Fig.13.

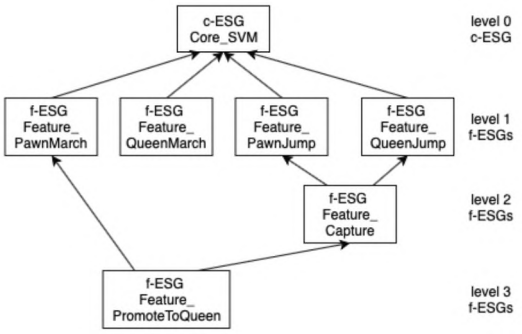

Fig. 13. Product FESG lattice for Spanish checkers

Fig.14 shows the c-ESG of the ATCGM SPL, which appears at the top of product FESG lattice at level 0. The c-ESG represents core behavior that is valid for all products and does not hold any information about the connection points.

Fig. 14. c-ESG of ATCGM SPL

For each behavioral feature seen in the transformed product diagram, there is an f-ESG. Two of those f-ESGs for ATCGM SPL are given in Fig.22 and Fig.23. PromoteToQueen f-ESG in Fig.22 holds connection point information related to Capture f-ESG in Fig.23, whereas Capture f-ESG does not hold connection point information related to PromoteToQueen f-ESG. As explained in Section IV, with these refactoring on f-ESGs we improve performance of Algorithm 2. Table I shows FESG for ATCGM SPL. Four of f-ESGs are refactored and they are typed in bold.

Product ESGs for the Children checkers, American, and Spanish checkers are given in Table II to show difference and at same time complexity of test models for the case study. American checkers ESG is the same in terms of structure except Queens are replaced with Kings. Therefore, they have the same number of nodes and edges as seen in Table II.

TABLE I. ATCGM SPL FESG DETAILS

	Before Refactoring		After Refactoring	
FESG	# of Nodes	# of Edges	# of Nodes	# of Edges
c-ESG	3	2	3	2
KingJump f-ESG	11	24	**10**	**16**
QueenJump f-ESG	11	24	**10**	**16**
PawnJump f-ESG	7	12	**6**	**8**
KingMarch f-ESG	10	24	10	24
QueenMarch f-ESG	10	24	10	24
PawnMarch f-ESG	6	12	6	12
Capture f-ESG	14	13	**12**	**11**
PromoteToKing f-ESG	7	6	7	6
PromoteToQueen f-ESG	7	6	7	6
TOTAL	86	147	81	125

TABLE II. CHECKERS PRODUCT ESGS IN ISOLATION

	Number of Nodes	Number of Edges
Product Children Checkers ESG	14	33
Product American Checkers ESG	31	95
Product Spanish Checkers ESG	31	95
TOTAL	76	223

It is known that SPL-based development approach is better if the software products are similar. Table III shows that our approach agrees with this statement in terms of test artefacts.

TABLE III. MODEL COMPARISON OF SPL AND PRODUCTS IN ISOLATION

	Number of Nodes	Number of Edges
ATCGM SPL refactored FESG	81	125
All Products in isolation (3 ESGs)	76	223

Test sequences for checkers products are obtained using both our proposed technique and existing ESG test generation technique. Our proposed technique is executed on the FESG of each product and results are given with "in SPL" tag, whereas the existing ESG test generation technique is executed on the ESG of each product and results are given with "in Isolation" tag in Table IV for coverage of ESs of length 2. Our proposed technique results in more CESs and greater numbers of events.

TABLE IV. TEST SEQUENCE COMPARISON OF SPL AND PRODUCTS IN ISOLATION

	# of CES	# of Events
Children Checkers in SPL	14	81
Children Checkers in Isolation	1	54
American Checkers in SPL	42	256
American Checkers in Isolation	1	171
Spanish Checkers in SPL	42	256
Spanish Checkers in Isolation	1	171

Time taken to generate CESs using both our proposed technique and existing ESG test generation technique is given in Table V for 10 runs. The runs are performed on a laptop having Intel 0.80 GHz and 4 GB RAM with 64-bit Windows 10 Enterprise operating system.

TABLE V. TEST SEQUENCE GENERATION TIME COMPARISON OF SPL AND PRODUCTS IN ISOLATION

	Min (ms)	Max (ms)	Avg (ms)
Children Checkers in SPL	5	11	6.8
Children Checkers in Isolation	53	61	57.3
American Checkers in SPL	11	21	12.3
American Checkers in Isolation	56	94	78.7
Spanish Checkers in SPL	11	23	12.4
Spanish Checkers in Isolation	57	76	63.8

Results in Table IV indicate that our approach yields test sets, which are ~50% larger than the test sets obtained by generating test cases from complete product models. However, as demonstrated by Table V, composition-based test generation takes ~80% to ~88% less time than full-product-based test generation. These results show that the proposed approach is a good alternative considering the fact that test cases generated from full-product models using an

optimized test generation algorithm tend to be long and few (see Table IV). These properties may cause an increase in the testing costs due to the fact that such test cases are harder to trace and too many events need to be re-executed after each fault correction takes place.

VI. Related Work

One of the first proposed approaches in MBT of SPLs was ScenTED (Scenario based TEst case Derivation) technique, which provides reuse of the core assets and components for the reuse of the test cases [12]. Another important research on MBT of SPLs was performed by Olimpiew [2], where CADeT (Customizable Activity diagrams, Decision tables and Test specifications) method is proposed. This method produces feature-based test cases using UML use case and activity diagrams. It uses the decision tables for modeling variability and generating test cases. Cichos et al. [13] proposed a technique called 150% finite state machines, which employs a superimposed model for the SPL under consideration and includes a coverage-driven SPL test suite generation method. Model-checking is another approach for model-based verification of SPLs. Kishi and Noda [14] proposed to model the design as a finite state machine and to check whether the product actually has the specified features using model checking. There were several proposals for applying model checking to SPLs [15][6][16].

Two research directions for reducing redundancies in testing of SPLs currently exist: regression-based SPL testing and SPL subset selection heuristics [17]. Since our is on regression-based SPL testing, we cover that literature here. Uzuncaova et al. [18] proposed an approach that uses SAT-based analysis to generate test inputs for each product in SPL. Their approach enables incremental refinement of test suites for a particular product variant. Neto et al. [19] proposed an approach that reduces the testing effort through reusing test cases taking advantage of SPL architectures similarities. Lochau et al. [17] suggested a strategy for test artifact reuse between products. Our work follows an extended strategy. The novelty of our strategy is that our delta is simple and that makes our approach flexible in the sense that we do not need to start with a base product and incrementally reach others.

VII. Conclusion

This paper presents a novel compositional and incremental model-based test sequence generation technique for SPLs that reuses existing test artefacts and partial test sequences (PTSs). The proposed approach uses event sequence graphs for features, called FESGs, as test models. An FESG is composed of a c-ESG and a set of f-ESGs. A c-ESG models core features of an SPL, and f-ESGs model selectable features and express variability in SPL. The proposed technique introduces algorithms (1) to form a product FESG lattice, (2) to generate test sequences through composition of PTSs obtained from f-ESGs with the help of product FESG lattice, and (3) to reuse existing sequences and new sequences for incremental testing of another product that is a delta away.

Instantiations of Algorithms 2 and 3 made for Section V ignore connection/variability points in the middle parts of the PTSs. Furthermore, all PTSs generated from the c-ESG are assumed to end with a finish event for efficiency. In the next step, we plan to relax these assumptions and add an explicit variability model to our approach in order to increase utilization of approach in industry.

Acknowledgment

This research is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under the grant 117E884.

References

[1] X. Devroey *et al.*, "A vision for behavioural model-driven validation of software product lines," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, 2012, pp. 208–222.

[2] E. M. Olimpiew, "Model-based testing for software product lines." Ph.D. Dissertation, George Mason, University, 2008.

[3] M. Lochau, S. Mennicke, H. Baller, and L. Ribbeck, "Incremental model checking of delta-oriented software product lines," *Journal of Logical and Algebraic Methods in Programming*, vol. 85, no. 1, pp. 245–267, 2016.

[4] K. Czarnecki and M. Antkiewicz, "Mapping features to models: A template approach based on superimposed variants," in *International conference on generative programming and component engineering*, 2005, pp. 422–437.

[5] S. Apel and D. Hutchins, "A calculus for uniform feature composition," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 32, no. 5, p. 19, 2010.

[6] A. Classen, "Modelling and model checking variability-intensive systems." Ph. D. dissertation, 2011.

[7] E. Uzuncaova, D. Garcia, S. Khurshid, and D. Batory, "Testing software product lines using incremental test generation," in *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, 2008, pp. 249–258.

[8] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[9] F. Belli, "Finite state testing and analysis of graphical user interfaces," in *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*, 2001, pp. 34–43.

[10] F. Belli, C. J. Budnik, and L. White, "Event based modelling, analysis and testing of user interactions: approach and case study," *Software Testing, Verification and Reliability*, vol. 16, no. 1, pp. 3–32, 2006.

[11] F. Belli and C. J. Budnik, "Test minimization for human-computer interaction," *Applied Intelligence*, vol. 26, no. 2, pp. 161–174, 2007.

[12] A. Reuys, E. Kamsties, K. Pohl, and S. Reis, "Model-based system testing of software product families," presented at the International Conference on Advanced Information Systems Engineering, 2005, pp. 519–534.

[13] H. Cichos, S. Oster, M. Lochau, and A. Schürr, "Model-based coverage-driven test suite generation for software product lines," presented at the International Conference on Model Driven Engineering Languages and Systems, 2011, pp. 425–439.

[14] T. Kishi and N. Noda, "Formal verification and software product lines," *Communications of the ACM*, vol. 49, no. 12, pp. 73–77, 2006.

[15] A. Gruler, M. Leucker, and K. Scheidemann, "Modeling and model checking software product lines," presented at the International Conference on Formal Methods for Open Object-Based Distributed Systems, 2008, pp. 113–131.

[16] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay, "Symbolic model checking of software product lines," presented at the Proceedings of the 33rd International Conference on Software Engineering, 2011, pp. 321–330.

[17] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity, "Incremental model-based testing of delta-oriented software product lines," in *International Conference on Tests and Proofs*, 2012, pp. 67–82.

[18] E. Uzuncaova, S. Khurshid, and D. Batory, "Incremental test generation for software product lines," *IEEE transactions on software engineering*, vol. 36, no. 3, pp. 309–322, 2010.

[19] P. A. da M. S. Neto, I. do Carmo Machado, Y. C. Cavalcanti, E. S. de Almeida, V. C. Garcia, and S. R. de Lemos Meira, "A regression testing approach for software product lines architectures," presented at the Software Components, Architectures and Reuse (SBCARS), Fourth Brazilian Symposium on, 2010, pp. 41–50.