Incremental Model-Based Testing of Delta-oriented Software Product Lines

Malte Lochau[1], Ina Schaefer[2], Jochen Kamischke[1], and Sascha Lity[1]

[1] TU Braunschweig, Institute for Programming and Reactive Systems, Germany
{m.lochau,j.kamischke,s.lity}@tu-bs.de

[2] TU Braunschweig, Institute for Software Systems Engineering, Germany
i.schaefer@tu-bs.de

Abstract. Software product line (SPL) engineering provides a promising approach for developing variant-rich software systems. But, testing of every product variant in isolation to ensure its correctness is in general not feasible due to the large number of product variants. Hence, a systematic approach that applies SPL reuse principles also to testing of SPLs in a safe and efficient way is essential. To address this issue, we propose a novel, model-based SPL testing framework that is based on a delta-oriented SPL test model and regression-based test artifact derivations. Test artifacts are incrementally constructed for every product variant by explicitly considering commonality and variability between two consecutive products under test. The resulting SPL testing process is proven to guarantee stable test coverage for every product variant and allows the derivation of redundancy-reduced, yet reliable retesting obligations. We compare our approach with an alternative SPL testing strategy by means of a case study from the automotive domain.

Keywords: Delta-oriented Software Product Lines, Model-based Testing, Regression Testing.

1 Introduction

Diversity is prevalent in modern software systems in order to meet different customer requirements and application contexts [25]. Software product line (SPL) engineering [8] provides a promising approach to develop variant-rich software systems by managed reuse. Since these software systems increasingly control safety- or business-critical applications, it is essential to ensure that they meet their requirements. Recently, there has been considerable progress in applying model checking [24,7,2] and theorem proving [3] to SPLs. However, those techniques are still far from being used in industrial engineering, mainly because of scalability issues, even for single products. Testing is much more established for practical applications in order to ensure that software systems meet their requirements. Testing is indispensable to reveal faults coming from different sources, such as erroneous feature interactions arising from obscured interplays between software and hardware devices [4].

Testing SPLs product by product is, in general, infeasible due to the high number of products to be tested. Recent SPL testing approaches focus on redundancy reduction by considering *representative* product subsets under test. The subset selection is based on (1) combinatorial criteria on feature models [9,20,22], (2) coverage criteria on SPL test models [5], and (3) coverage criteria on feature interactions [16]. But, until now, few attention in SPL research is paid to the problem how to actually conduct an efficient testing process on those subsets that avoids a traditional product by product process, again, contradicting SPL reuse principles. The new ISO 26262 standard for automotive systems even requires comprehensive testing strategies coping with existing system variants [28].

In this paper, we propose a novel approach for incremental model-based testing (MBT) [30] of SPLs based on principles of regression testing [1]. MBT is well-suited for planing reuse potentials in SPL testing [18]. Variable test models are used to explicitly capture behavioral commonality and variability between product variants. On this basis, a concise approach for incrementally assembling and reusing test artifacts for sets of products under test is built.

Our framework comprises state machines as test models extended with delta modeling concepts [6,23] to express variability. When testing a set of products, for each step from a product p to the next product p', an automated adaptation of the test model is performed by applying a regression delta. The regression delta contains the modifications to obtain the test model of product p' from test model of product p. It is computed automatically from the delta modeling structure of the test models. From the regression delta, the test goals for product p', as well as of the set of test cases and retest obligations are derived. Additionally, it can be determined which existing test cases are applicable to product p' and which test results still hold. This framework has two major potentials in SPL testing: (1) test cases can be reused for different product variants while guaranteeing the validity of test cases and the confidential test coverage for every product variant, and (2) test results can be reused according to change impacts between product variants, thus guaranteeing appropriate fault detection efficiency. Our approach is evaluated by means of a case study from the automotive domain w.r.t. previous results obtained from an existing SPL testing approach. It is, to the best of our knowledge, the first SPL MBT framework that captures reuse potentials between different product variants.

The paper is organized as follows. In Sect. 2, foundations of model-based testing are introduced. In Sect. 3, delta modeling for test models is presented. The incremental SPL testing approach is described in Sect. 4 and evaluated in Sect. 5. Sect. 6 discusses related work, and Sect. 7 concludes.

2 Foundations

We briefly introduce the main principles of MBT and regression testing underlying the incremental SPL testing framework developed in the remainder of this paper.

2.1 Model-Based Testing

Model-based testing aims at the automation of black-box testing processes [30]. A *test model* serves as a behavioral specification capturing the functional requirements of a software *product under test* to be verified.

Due to their wide acceptance in industrial control systems engineering, state-machine-like modeling approaches are commonly used as test models. State machine test models define input/output relations by means of sequences of *controllable input* and expected *observable output events.* We focus on basic, i.e., flat basic state machines as test models to keep our illustrations graspable, where the major results are enhanceable to, e.g., UML-like state machine variants providing hierarchy, concurrency, variables etc.

Definition 1. *(State Machine Test Model)*
A state machine test model *is a 4-tuple* $tm = (S, s_0, L, T)$*, where* S *is a finite set of* states*,* $s_0 \in S$ *is the* initial state*,* L *is a set of* transition labels*, and* $T \subseteq S \times L \times S$ *is a* transition relation.

A transition label $l = (\pi_I, \pi_O) \in L = \Pi_I \times \Pi_O$ is a pair of a *controllable* input event $\pi_I \in \Pi_I$ triggering the transition, and an *observable* output event $\pi_O \in \Pi_O$ specifying a system reaction released by the transition, where Π_I and Π_O are disjoint input/output alphabets. We assume state machine test models to be deterministic, and to obey well-formedness properties as usual, i.e., the transition graph has to be *connected* and every state has to be *reachable* from the initial state. By $TM(L)$ we refer to the set of well-formed state machine test models over a label set L.

Test models specify all intended behaviors a product under test is to be verified against by means of *test runs*, i.e., representative executions. Test runs refer to *test cases* derived from a test model $tm \in TM(L)$.

Definition 2. *(State Machine Test Case)*
A test case $tc = (t_0, t_1, \ldots, t_k) \in T^*$ *of a state machine test model* $tm \in TM(L)$ *is a* finite *sequence of* k *transitions of* tm.

Test case tc is *valid* for test model $tm \in TM(L)$, written $valid(tc, tm)$, if its transition sequence corresponds to an alternating sequence $s_0, t_0, s_1, \ldots, s_{k-1}, t_{k-1}, s_k$ of states and transitions conforming tm, i.e., (1) it starts in the initial state s_0, and (2) for all segments $(s_i, t_i, s_{i+1}) \in T$, $0 \leqslant i \leqslant k-1$, holds. For a test case $tc = (t_0, t_1, \ldots, , t_{k-1})$, we define a corresponding *test run*:

$$exec(tm, tc) = (l_0, l_1, \ldots, l_{k-1}) \in L^*$$

to be given as the *trace* traversed by tc in tm, i.e., a sequence of labels l_i of transitions $t_i, 0 \leqslant i \leqslant k-1$. We limit our considerations to deterministic behaviors, i.e., a one-to-one correspondence between test runs and test cases. By $TC(tm)$, we denote the set of test cases, i.e., all valid paths of a test model tm.

In MBT, the behaviors of an implementation of product p are verified for test cases tc to *conform* those specified in its test model tm [29]. By $\approx_{te}$, we denote the *testing equivalence* under consideration in the following, usually some kind of trace equivalence [10]. The equivalence notion applied for the purposes of this paper is discussed in more detail in Sect. 5. According to deterministic behaviors specified in a test model, we assume product variants to also behave deterministically when reasoning about equivalence of test case executions. A product under test *p passes* a test run of a test case $tc \in TC(tm)$, if its observable behavior under the sequence of inputs conforms to the expected output behavior specified in test model *tm*:

$$p \textbf{ passes } tc \;:\Leftrightarrow\; exec(p, tc) \;\approx_{te}\; exec(tm, tc)$$

A test suite $ts \subseteq TC(tm)$ is a collection of test cases, where:

$$p \textbf{ passes } ts \;:\Leftrightarrow\; \forall tc \in ts : p \textbf{ passes } tc$$

A test model *tm* (and thus $TC(tm)$) potentially contains (1) an infinite number of paths, as well as (2) paths of infinite length. For the test conformance to be decidable, test suites $ts \subseteq TC(tm)$ are restricted to those with (1) a finite number of test cases, and (2) each test case to be of finite length. Adequacy criteria for selecting appropriate test suites from test models *tm* usually require structural elements in *tm* to be *traversed* at least once. For state machines, such *coverage criteria* are *all-states*, *all-transitions*, etc. [30]. Formally, a coverage criterion C applied to a test model *tm* selects a finite set of *test goals*:

$$tg = C(tm) = \{g_1, g_2, \ldots, g_n\}$$

for instance, $tg = T$, i.e., the set of all transitions in *tm*. We write $covers(tc, g)$ for a test case $tc \in ts \subseteq TC(tm)$, if the test goal g is traversed in test model *tm* via *tc*. A test suite *ts satisfies* coverage criterion C, if:

$$\forall g \in C(tm) : \exists tc \in ts : covers(tc, g)$$

Summarizing, the set of test artifacts for a product p is given as follows.

Definition 3. *(Product Test Artifacts)*
The collection of test artifacts *for product p is a 4-tuple* $ta_p = (tm_p, tg_p, ts_p, tp_p)$ *consisting of a* test model tm_p*, a finite set* tg_p *of* test goals *in* tm_p *for criterion* C*, a* test suite ts_p*, and a* test plan tp_p*.*

A *test plan* organizes the test suite application by further (de-)selecting, prioritizing, etc. test cases from *ts*. We assume test plans simply to be subsets $tp_p \subseteq ts_p$ containing those test cases actually to be (re-)tested on product p. In case of single product testing, we assume $tp_p = ts_p$.

Example 1. Consider the state machine test model in Fig. 1 consisting of states $S = \{s0, s1, s2\}$ and transitions $T = \{t0, t1, t2, t3\}$. Assuming C is the *all-transitions* coverage criterion, the set of test goals is given as $tg = \{t0, t1, t2, t3\}$. A sample test suite $ts = \{tc1, tc2\}$ that satisfies C consists, e.g., of two test cases $tc1 = (t0, t1)$ and $tc2 = (t0, t2, t3)$, where *tc1* covers *t0* and *t1* and *tc2* covers *t0*, *t2*, and *t3*. A test run of *tc*2 corresponds to the sequence $exec(tc2, tm) = ((\pi_1, \pi_2), (\pi_3, \pi_5), (\pi_3, \pi_2))$.

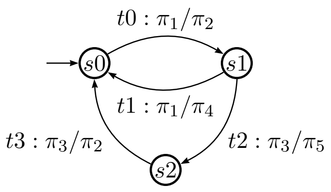

Fig. 1. Sample State Machine Test Model

For the following discussions, we assume the existence of some (black-box) *test case generator* (cf. e.g. [12]) and write $tc = gen(tm, g)$ to generate a test case that covers test goal g on test model tm, and $ts = gen(tm, C)$ for the generation of entire test suites satisfying coverage criterion C on test model tm.

2.2 Regression Testing

The purpose of regression testing is to efficiently verify that changes between different *versions* of a product are as intended [1,11]. For a software product implementation version p evolving to version p' over time, regression testing strategies aim at verifying that (1) the changes are correctly implemented, and (2) the changes do not erroneously influence parts of p reused in p' other than intended. When stepping to the next version p', the test suite evolves accordingly such that $ts' = gen(tm', C)$ is executed on p'. For non-naive regression testing approaches, reuse potentials between p and p' arise, namely (1) the reuse of *test cases* in ts generated from tm in ts' for reducing test generation efforts, and (2) reuse of *test execution results* for test cases in ts applied to p for p', i.e., reducing test execution efforts. For the reuse of test cases $tc \in ts$ in ts', we require:

$$exec(tc, tm) \approx_{te} exec(tc, tm')$$

Thus, tc concerns system reactions equivalently specified in tm and tm', i.e., addressing behaviors similar to p and p'. For the reuse of test execution results obtained from $exec(tc, p)$ for reusable test cases $tc \in ts'$, we further require:

$$exec(tc, tm) \approx_{te} exec(tc, tm') \Rightarrow exec(tc, p) \approx_{te} exec(tc, p')$$

This second reuse problem refers to the well-known retest selection problem of regression testing [1]: select from the set ts_R of reusable test cases a minimum retest subset $ts_{RT} \subseteq ts_R$ such that ts_{RT} is capable to cover all potentially erroneous impacts of changes between the product implementations p and p':

$$exec(ts_{RT}, p') \approx_{te} exec(ts_{RT}, p) \Rightarrow exec(ts_R, p') \approx_{te} exec(ts_R, p)$$

Regression testing approaches categorize test cases into sets of *reusable* $ts_R \subseteq ts$, *obsolete* $ts_O = ts \backslash ts_R$, and *new* $ts_N = ts' \backslash ts_R$ test cases. The set of test cases to be (re-)executed on p' contains the *retest* set $ts_{RT} \subseteq ts_R$, as well as all new test cases in ts_N.

3 Delta-Oriented SPL Test Modeling

In order to apply an incremental MBT approach to SPLs, we need a reusable test model to capture the commonality and variability in a closed form instead of storing each test model variant separately. We base our approach on the concept of delta modeling [6,23], a modular and flexible variability modeling approach that is well suited as basis for regression-based SPL testing by incrementally evolving test artifacts for product variants. In delta modeling, a family of similar products is captured by a designated core product and a set of deltas encapsulating changes to the core model. A delta adds and removes elements from the core product. If there are hierarchically structured elements, a delta operation can be used to change the internal structure of these elements. A product variant is obtained by selecting a subset of the available deltas, determining a suitable ordering and applying the operations of the deltas one by one to the core product.

We apply the principles of delta modeling to state machine test models. A delta over a state machine test model, as defined in Def. 1, can add and remove states and transitions. Changing the label of a transition can be encoded by removing a transition and adding a transition between the same states with a different label. The following definition introduces the syntax of state machine deltas.

Definition 4. *(State Machine Delta)*
A state machine delta *is a set of delta operations $\delta \subseteq Op$, where Op contains (1) for every $s \in \mathcal{S}$, $\{add\ s\}$ and $\{rem\ s\}$, and (2) for every $t \in \mathcal{T}$, $\{add\ t\}$ and $\{rem\ t\}$ for finite sets of possible states $\mathcal{S}$ and transitions $\mathcal{T}$.*

The application of a set of delta operations transforms one state machine into another. A delta is *applicable* to a state machine if the states and transitions to be removed exist and if the states and transitions to be added do not yet exist. A delta is *consistent* if it only adds or removes each state or transition once.

Definition 5. *(State Machine Delta Application)*
The application *of an applicable and consistent delta $\delta \subseteq Op$ to a state machine $tm = (S, s_0, L, T)$ defines a function $apply : TM(L) \times \mathcal{P}(Op) \rightarrow TM(L)$ such that $apply(tm, \delta) = tm' = (S', s_0, L, T')$ where*

- *if $\delta = \varnothing$, $tm' = tm$*
- *if $\delta = \{op\} \cup \delta'$, then $tm' = apply(apply(tm, op), \delta')$*
- *for $\delta = \{add\ s\}$, we have $S' = S \cup \{s\}$ and $T' = T$*
- *for $\delta = \{rem\ s\}$, we have $S' = S \backslash \{s\}$ and $T' = T$*
- *for $\delta = \{add\ t\}$, we have $T' = T \cup \{t\}$ and $S' = S$*
- *for $\delta = \{rem\ t\}$, we have $T' = T \backslash \{t\}$ and $S' = S$*

In order to describe the set of possible test models for an SPL, we connect the deltas to the product variants. Each product test model is defined by a set of deltas to be applied to a given core test model tm_{core} in order to generate the test model of the variant. Instead of specifying sets of deltas for each product test model, the connection can also be made by associating deltas to product features [6]. A suitable ordering of delta application has to be defined such that each delta is applicable to the respective model when it is used. The test model of the product variant is obtained by applying the given deltas in the specified ordering to the core test model tm_{core}. During the generation process, it is possible that an intermediate model is constructed that is not well-formed. However, after applying all deltas, it has to be guaranteed that the resulting test model is well-formed. A more detailed description of the product generation process in delta modeling can be found in [23].

To allow for a flexible delta-oriented SPL test modeling, any potential test model should be usable as core model. This means that a state machine delta has to exist to derive every valid test model variant from that arbitrary core model.

Proposition 1. *(Existence of State Machine Delta)*
For each core state machine test model $tm_{core} \in TM(L)$ and each test model variant $tm \in TM(L)$, there exists a state machine delta $\delta \subseteq Op$ such that $tm = apply(tm_{core}, \delta)$ holds.

Proof: For any potential $tm_{core} = (S, s_0, L, T)$ and test model variant $tm = (S', s_0, L, T')$, we have to show that there exist delta operations $\delta \subseteq Op$ that are sufficient to transform the sets S and T to S' and T', respectively. For each $s \in S'$, three cases arise: (1) for states $s \in S \cap S'$ no delta operation is required, (2) for states $s \in S \backslash S'$, s can be removed from S to build S' via $\{rem\ s\}$, and (3) for states $s \in S' \backslash S$, s can be added to S to built S' via $\{add\ s\}$. For transitions, the same cases hold.

Example 2. Consider Fig. 2. The state machine introduced in Fig. 1 now serves as the *core model*. By applying the delta operations of δ_{tm}, we obtain the left test model variant *tm*. By applying the delta operations of $\delta_{tm'}$, we obtain the right test model variant *tm'*.

4 Delta-Oriented SPL Regression Testing

When applying MBT to SPLs, i.e., a family of similar product variants $P = \{p_1, p_2, \ldots p_n\}$ with explicit commonality and variability, a corresponding collection of *test artifacts* $ta_i = (tm_i, tg_i, ts_i, tp_i)$ is to be provided for every product variant $p_i \in P$. The artifact construction and application of test suites ts_i to implementations of product variants $p_i \in P$ is usually done in some ordering. The result is a chain of product testing campaigns continuously stepping from test artifacts ta of variant p to the next product test artifacts ta' of variant p'. Reuse potentials between ta and ta' arise by incrementally promoting previous test artifacts to subsequent products under test. In contrast to classical regression scenarios, differences between product variants are explicitly specified beforehand in an SPL, e.g., on the basis of a reusable test model.

Based on delta-oriented state machines as reusable SPL test models, we define a model-based SPL regression testing approach that assembles product-specific

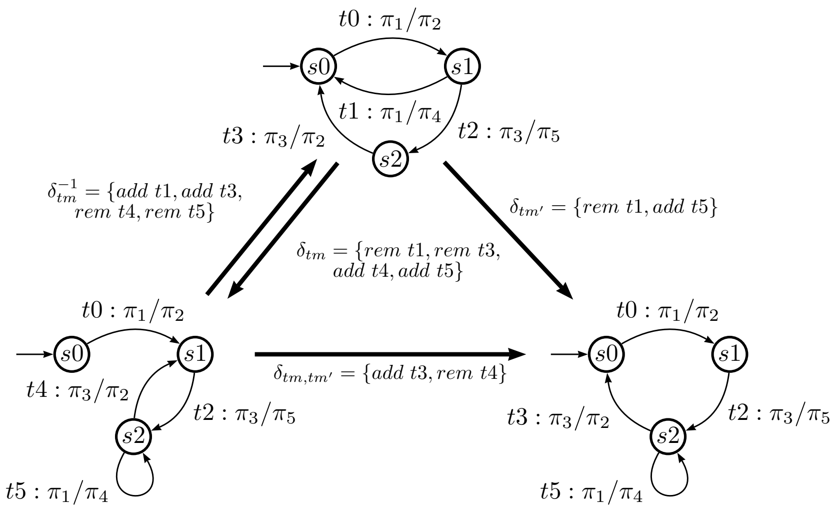
$t0 : \pi_1/\pi_2$
$s0$
$s1$
$s2$
$t1 : \pi_1/\pi_4$
$t3 : \pi_3/\pi_2$
$t2 : \pi_3/\pi_5$
$\delta_{tm}^{-1} = \{add\ t1, add\ t3, rem\ t4, rem\ t5\}$
$\delta_{tm} = \{rem\ t1, rem\ t3, add\ t4, add\ t5\}$
$\delta_{tm'} = \{rem\ t1, add\ t5\}$
$\delta_{tm,tm'} = \{add\ t3, rem\ t4\}$
$t4 : \pi_3/\pi_2$
$t5 : \pi_1/\pi_4$

Fig. 2. Sample Delta-oriented SPL Test Model and Regression Delta Derivation

test artifacts by incrementally reusing test artifacts of previous products. In particular, we incrementally evolve product test artifacts for a sequence $p_1, p_2, \ldots, p_n$ of products under test as follows:

1. Generate an initial collection of product test artifacts ta_1 using MBT techniques for single products as usual and apply the resulting test suite ts_1 to the implementation of p_1.
2. Incrementally evolve ta_i to ta_{i+1}, for $1 \leqslant i < n$, and apply the new (re-)test plan $tp_{i+1} \subseteq ts_{i+1}$ to p_{i+1}.

Although p_1 might be chosen arbitrarily, we suggest to start the incremental SPL testing campaign with the core product p_{core} as it usually comprises most of the commonalities among product variants.

As illustrated in Fig. 3, the incrementation of product test artifacts ta for product p to ta' of a subsequent variant p' decomposes into four levels. For each incrementation from p_i to p_{i+1}, (1) the reuse of product test artifacts from ta_i in ta_{i+1}, as well as (2) the generation of new artifacts required for ta_{i+1} is to be performed. Both steps are to be conducted in a way that ensures the different components of ta_{i+1} to meet the requirements according to their relationships (cf. Sect. 2), namely *validity* of test cases $tc \in ts_{i+1}$ w.r.t. tm_{i+1}, *coverage* of test goals $g \in tg_{i+1}$ for criterion C by test suite ts_{i+1}, and appropriate *(re-)test* selections for test plans $tp_{i+1} \subseteq ts_{i+1}$.

Accordingly, we apply the delta approach to also reason about the incremental changes on test artifacts from ta to ta', where $\delta_{ta,ta'}$ is decomposed into sub deltas for the different components of test artifact collections:

$$\delta_{ta,ta'} = (\delta_{tm,tm'}, \delta_{tg,tg'}, \delta_{ts,ts'}, \delta_{tp,tp'})$$

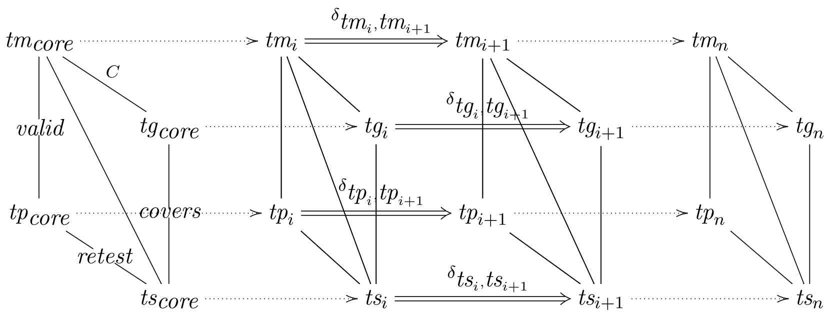

Fig. 3. Incremental Evolution of SPL Test Artifacts

The test model is the central part for deriving any kind of test artifacts in MBT. As a consequence, the sub deltas on the remaining *product test artifacts* are directly deducible from changes on test model specifications.

Test Model Delta. One of the main benefits of a delta-oriented SPL test model is its ability to comprehensively encapsulate the differences of every product variant w.r.t. some core model. However, for the incremental evolution of product test artifacts, we rather have to make explicit the differences of a product variant p' to the previous product p under test. Therefore, we introduce the concept of *regression deltas* to aggregate all changes when evolving from tm to tm'.

Definition 6. *(State Machine Regression Delta)*
A state machine regression delta $\delta_{tm,tm'} \subseteq Op$ *for state machine pair* $(tm, tm') \in TM(L) \times TM(L)$ *is a state machine delta such that* $tm' = apply(tm, \delta_{tm,tm'})$.

The application of a state machine regression delta on a test model yields the subsequent test model variant. As illustrated in Fig. 2, by intuition, a regression delta $\delta_{tm,tm'}$ results from composing the inverted delta δ_{tm}^{-1} of tm and the delta $\delta_{tm'}$ of tm'. The *inverse* $\delta^{-1} \subseteq Op$ of a state machine delta $\delta \subseteq Op$ is built component-wise, i.e., by inverting each delta operation $op \in \delta$ to op^{-1} in δ^{-1} such that $\{add\ e\}^{-1} = \{rem\ e\}$ and $\{rem\ e\}^{-1} = \{add\ e\}$ for $e \in \mathcal{S} \cup \mathcal{T}$.

However, using set union to compose δ_{tm}^{-1} and δ'_{tm} into $\delta_{tm,tm'}$ produces unsound results in case of equal delta operations. For instance, in Fig. 2, $\{add\ t5\} \subseteq \delta_{tm} \cap \delta_{tm'}$ holds, thus set union would yield $\{rem\ t5,\ add\ t5\} \subseteq \delta_{tm,tm'}$, i.e., conflicting operations in the regression delta. Instead, for the correct derivation of regression delta $\delta_{tm,tm'}$ from state machine deltas $\delta_{tm} \subseteq Op$ and $\delta_{tm'} \subseteq Op$, we have to apply an alternative composition operator that takes common delta operations into account. The *symmetric difference* $A \mathbin{\Delta} B = (A \backslash B) \cup (B \backslash A)$ of two sets A and B solely contains those elements being either exclusive to set A or B. In addition, to build the regression delta, we require the first operand of the symmetric difference to be inverted.

Proposition 2. *(State Machine Regression Delta Construction)*
For two state machine deltas $\delta_{tm} \subseteq Op$ *and* $\delta_{tm'} \subseteq Op$, *the regression delta is given as* $\delta_{tm,tm'} = (\delta_{tm} \backslash \delta_{tm'})^{-1} \cup (\delta_{tm'} \backslash \delta_{tm})$.

Proof: For $tm' = (S', s_0, L, T')$ to result from applying $\delta_{tm,tm'} = (\delta_{tm} \backslash \delta_{tm'})^{-1} \cup (\delta_{tm'} \backslash \delta_{tm})$ to $tm = (S, s_0, L, T)$, we have to show the sets S' and T' to be built correctly from S and T. For states $s \in S'$, we have two cases: (1) $s \in S$, and (2) $s \notin S$. For case (1), we have to show that $\{rem\ s\} \nsubseteq \delta_{tm,tm'}$, where we have two further cases: (1a) $s \in S_{core}$, thus $\{add\ s\} \nsubseteq \delta_{tm}$ which implies $\{rem\ s\} \nsubseteq (\delta_{tm} \backslash \delta_{tm'})^{-1}$, and (1b) $s \notin S_{core}$, thus $\{add\ s\} \subseteq \delta_{tm}$, but $\{rem\ s\} \nsubseteq (\delta_{tm} \backslash \delta_{tm'})^{-1}$, because $\{add\ s\} \nsubseteq (\delta_{tm} \backslash \delta_{tm'})$. For case (2), we have to show that $\{add\ s\} \subseteq \delta_{tm,tm'}$, where, again, two further cases arise: (2a) $s \in S_{core}$, thus $\{rem\ s\} \subseteq \delta_{tm}$ which implies $\{add\ s\} \subseteq (\delta_{tm} \backslash \delta_{tm'})^{-1}$, and (2b) $s \notin S_{core}$, thus $\{add\ s\} \nsubseteq (\delta_{tm} \backslash \delta_{tm'})^{-1}$, but $\{add\ s\} \subseteq (\delta_{tm'} \backslash \delta_{tm})$. Symmetric cases arise for ensuring states $s \notin S'$ are either removed if $s \in S$, or not added if $s \notin S$ via $\delta_{tm,tm'}$. Further note, that these cases also hold for the set of transitions. Finally, the existence of a regression delta for arbitrary pairs of state machines follows directly from Prop. 1: as any test model variant is derivable from an arbitrary core model by a set of delta operations, any test model tm can be assumed as core model to derive the test model of tm'.

Example 3. The regression delta between the test model tm and tm' in Fig. 2 results in $\delta_{tm,tm'} = (\delta_{tm} \backslash \delta_{tm'})^{-1} \cup (\delta_{tm'} \backslash \delta_{tm}) = \{add\ t3,\ rem\ t4\}$. As both products share the delta operations concerning $t1$ and $t5$, those transitions are not affected by the regression delta.

Please note, that regression deltas constitute a generalization of state machine deltas, i.e., δ_{tm} can be represented as $\delta_{tm_{core},tm}$.

We now describe the derivation of the deltas concerning the incrementation of the three remaining test artifacts from the state machine regression delta. Those deltas are similar to those for state machines (cf. Sect. 3), but are to be adapted to artifact types considered in the particular components of ta.

Test Goal Delta. The construction of the delta $\delta_{tg,tg'}$ for the incrementation of the set of test goals depends on the coverage criterion C considered. For simple structural criteria such as *all-states* and *all-transitions*, i.e., criteria with $C(tm) \subseteq \mathcal{S} \cup \mathcal{T}$, $\delta_{tm,tm'}$ is directly adaptable to evolve the test goals via the following rules:

– $\forall \{rem\ e\} \subseteq \delta_{tm,tm'} : e \in tg \Rightarrow \{rem\ e\} \subseteq \delta_{tg,tg'}$
– $\forall \{add\ e\} \subseteq \delta_{tm,tm'} : e \in C(tm') \Rightarrow \{add\ e\} \subseteq \delta_{tg,tg'}$

Otherwise, for more complex criteria, e.g., path-oriented criteria like MC/DC coverage [30], a (partial) regeneration of test goals via $C(tm')$ is required, where $\delta_{tm,tm'}$ indicates model parts in tm' potentially affected.

Test Suite Delta. As described in Sect. 2.2, regression testing approaches partition an existing test suite ts of product p into subsets of reusable tests ts_R and obsolete tests ts_O when evolving to product p'. For our incremental SPL testing approach it seems promising not to discard obsolete test cases in the next test suite ts', but rather to collect them for potential reuse for subsequent products under test. Therefore, we partition product test suites $ts = ts_V \cup ts_O$ into sets of *valid* and *obsolete* test cases. When evolving $ts = ts_V \cup ts_O$ to $ts' = ts'_V \cup ts'_O$ via $\delta_{ts,ts'}$, changes in $\delta_{tm,tm'}$ have effects on the incrementation of both sets. Accordingly, we also partition the test suite delta into δ_{ts_V,ts'_V} and δ_{ts_O,ts'_O}.

By $T_{tc} \subseteq \mathcal{T}$, we refer to the subset of transitions from $\mathcal{T}$ such that (1) $tc \in T^*_{tc}$, and (2) T_{tc} is *minimal*. Thus, a test case tc is valid for test model $tm = (S, s_o, L, T)$, if $T_{tc} \subseteq T$, whereas $T_{tc} \nsubseteq T$ holds for obsolete test cases. A test case $tc \in ts_O$ being obsolete for p becomes valid for p' as follows:

$$\forall t \in T_{tc} \backslash T : \exists \{add\ t\} \subseteq \delta_{tm,tm'} \Rightarrow \{add\ tc\} \subseteq \delta_{ts_V,ts'_V} \wedge \{rem\ tc\} \subseteq \delta_{ts_O,ts'_O}$$

i.e., the set of transitions of tc missing in the set T of the test model of p is added to p' via the regression delta. Correspondingly, valid test cases $tc \in ts_V$ become obsolete by the rule:

$$\exists t \in T_{tc} : \{rem\ t\} \subseteq \delta_{tm,tm'} \Rightarrow \{add\ tc\} \subseteq \delta_{ts_O,ts'_O} \wedge \{rem\ tc\} \subseteq \delta_{ts_V,ts'_V}$$

The set of reusable test cases $ts'_R = ts_V \cap ts'_V$ therefore contains those test cases valid for p, as well as for p'. In addition to ts'_R, further test cases may be required in ts to cover all test goals in tg'. A test goal $g \in tg'$ is uncovered by ts'_R if either

- $\{add\ g\} \subseteq \delta_{tg,tg'}$, i.e., the test goal is new in p', or
- $\forall tc \in ts_V : covers(tc, g) \Rightarrow tc \in ts'_O$, i.e., all test cases of p covering g are obsolete for p'.

For covering those test goals, further previously obsolete test cases $tc \in ts_O \cap ts'_V$ with $covers(tc, g)$ may be found and added to ts'_R. Otherwise, a new test case $tc_g = gen(tm', g)$ is required, where $\{add\ tc_g\} \subseteq \delta_{ts_V,ts'_V}$. The set of all new test cases generated for p' thus gives the set ts'_N in terms of regression testing.

Example 4. Consider the test cases $tc1 = (t0, t1)$ and $tc2 = (t0, t2, t3)$ of Example 1 for *all-transition* coverage. When stepping from the core model to tm (cf. Fig. 2), $tc1$ and $tc2$ both become obsolete, thus new test cases, e.g., $tc3 = (t0, t2, t5)$ and $tc4 = (t0, t2, t4)$ are generated. For tm', again, $tc1$ is obsolete, whereas $tc2$ as well as $tc3$ are reusable and cover all test goals.

Test Plan Delta. Test plans $tp \subseteq ts_V$ are used to define which valid test cases from a test suite are actually executed on the product under test, where $tp = ts_N \cup ts_{RT}$. New test cases $tc \in ts_N$ are applied in any case to verify that new, i.e., varying behaviors are correctly implemented. In addition, from the set of reusable test cases ts_R, a retest set $ts_{RT} \subseteq ts_R$ is selected to verify that the changes do not erroneously affect common behaviors covered by ts_R. For the selection of ts_{RT}, different strategies appear in the literature [11], e.g., *retest-all* $ts_{RT} = ts_R$, *retest-non* $ts_{RT} = \emptyset$, and *retest-random*, where some $ts_{RT} \subseteq ts_R$ is chosen. In addition, techniques for change impact analyses such as *program slicing* [13] support the retest selection decision by the following criterion:

$$tc \in ts_{RT} :\Leftrightarrow exec(tc, tm) \approx_{te} exec(tc, tm') \nRightarrow exec(tc, p) \approx_{te} exec(tc, p')$$

Summarizing, the test plan delta $\delta_{tp,tp'}$ is defined by the rules:

- $\forall tc \in tp \backslash ts'_{RT} : \{rem\ tc\} \subseteq \delta_{tp,tp'}$
- $\forall tc \in ts'_{RT} \backslash tp : \{add\ tc\} \subseteq \delta_{tp,tp'}$
- $\forall tc \in ts'_{N} : \{add\ tc\} \subseteq \delta_{tp,tp'}$

For further enhancements, additional information about previous test plans can be used for retest selections, e.g., how often a test case has been already executed (and failed).

Soundness of the Approach. For the soundness of the presented approach, we require the resulting test artifacts to be (1) *valid*, i.e., every test suite solely contains valid test cases, and (2) *complete*, i.e., guaranteeing complete test coverage of every product test model w.r.t. criterion C. Let $ta_1, ta_2, \ldots, ta_n$ be a collection of test artifacts incrementally built for a sequence of products $p_1, p_2, \ldots, p_n$ via deltas on test artifact as defined above.

Theorem 1. *(Validity of Product Test Suites)*
For product test suites ts_i of each ta_i, $1 \leqslant i \leqslant n$, $ts_{V_i} \subseteq TC(tm_i)$ holds.

Proof: By induction over the chain of regression delta applications. For $i = 1$, we assume soundness of the test case generator, i.e., $gen(tm_1, C) \subseteq TC(tm_1)$. For induction steps from i to $i+1$, (1) validity of ts_{V_i} follows from the induction hypothesis, and (2) validity of ts_{V_i} holds as obsolete and reusable test cases from ts_i are confirmed via the regression delta, and new test cases in ts_{i+1} are, again, delivered by the test case generator, i.e., $gen(tm_{i+1}, tg) \in TC(tm_{i+1})$.

Theorem 2. *(Completeness of Product Test Suites)*
For product test suites ts_i and test goals tg_i of each ta_i, $1 \leqslant i \leqslant n$, (1) $tg_i = C(tm_i)$ holds, and (2) ts_i satisfies C.

Proof Idea: Again, by induction over the chain of regression delta applications. For the correct incrementation (1) of test goals for more complex criteria, we rely on a sound implementation of the test goal selection function C, and (2) of test suites, we, again, assume soundness of the test case generator.

Moreover, the approach ensures every test case generated during the incremental testing process to be executed at least once as the set ts_N is always selected for the test plan. Our approach implicitly fulfills the *complete SPL test suite coverage* requirement proposed in [5]. In addition, it supports reasoning about the *reliability* of test plans: the impact on the fault detection efficiency of retesting selections between SPL products in comparison to complete product by product SPL testing is parameterizable via the change impact criterion under consideration.

5 Implementation and Evaluation

We developed a tool chain for the sample implementation of our incremental SPL testing approach. For the delta-oriented state machine SPL test modeling, we developed an Eclipse plug-in incorporating the Eclipse Modeling Framework. The tool supports the configuration of product variants based on a domain feature model and the automated derivation of product test models. Those test models are imported into IBM Rational Rhapsody to apply the add-on ATG for automated test case generation and execution.

To evaluate our approach, we considered an SPL case study from the automotive domain, a simplified Body Comfort System (BCS) including numerous features like automatic power windows, human machine interface, alarm system, etc., comprising $11,616$ valid product variants. We already obtained evaluation results from testing the BCS SPL in previous work for an SPL subset testing approach covering all valid feature pairs [19,16]. This allowed us to compare the results to those of our incremental testing technique w.r.t. gain in efficiency arising from test artifact reuse potentials. The original BCS SPL 150% state machine test model created for the MoSo-PoLiTe approach contains 105 states and 107 transitions comprising 26 input and 33 output events. We remodeled this test model to build a delta-oriented SPL test model including one core model and 40 delta modules.

For our experiments, we considered the *Model Element Coverage* criterion as supported by ATG. For covering every single product variant, an estimated amount of $743,424$ test cases is required including multitudes of redundancies due to similarities among product variants. After applying MoSo-PoLiTe [19] we obtained 17 representative products ($P1 - P17$), thus reducing the number of test cases to $1,093$ for testing this set product by product. To evaluate our incremental approach, we considered the same product subset and further added a core product ($P0$) as the starting point of the incremental SPL testing process. The results of the case study are shown in Fig. 4. Triangles denote the number of test cases generated and applied per product in the MoSo-PoLiTe approach [19]. In contrast, for the incremental SPL testing approach, diamonds denote the number of test cases to be newly generated for a product, and squares denote the number of test cases to be (re-)tested on that product. We focused our experiments on the reuse of test cases, whereat for the reuse of test results, we applied change impact analyses based on test model slicing [13]. Comparing our results to those of MoSo-PoLiTe, a significant reduction of the testing efforts concerning test case generation and execution was achieved, however ensuring the same degree of test model coverage. In particular, the average number of test

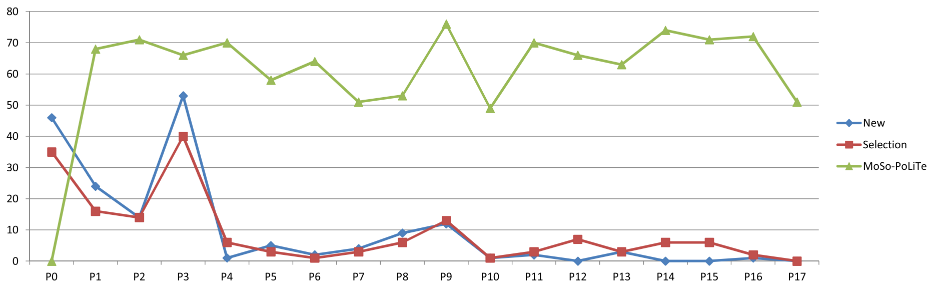

Fig. 4. Evaluation Results for the BCS SPL Case Study

cases generated and executed per product for MoSo-PoLiTe amounts 64, whereas our incremental approach solely requires an average number of 10 new test cases and 9 test cases selected for execution per product. In cases where the number of test case executions exceeds the number of test cases generated, existing test cases are selected for retesting. Most test cases are generated and selected for the first four products. As the number of existing test cases covering commonality between product variants continuously increases, a decreasing number of test cases is generated and executed for the remaining products.

Threats to Validity. The efficiency of the approach depends on the test case generator applied. The quality of the test suite of the initial product under test is particularly crucial for the subsequent iterations. However, this drawback is adherent to model-based testing in general, rather than an inconvenience of our approach. For the reuse of test cases, our current approach uses global repositories $\mathcal{S}$ and $\mathcal{T}$ to identify equality of traces by means of syntactical identity for testing equivalence $\approx_{te}$ and is restricted to deterministic behaviors. This is a rather strict requirement, but weakening this notion to more realistic testing equivalences [10] is far less efficiently decidable. Providing sound criteria for retest selection is, due to the black-box assumption of model-based testing, an open problem as common change impact analysis techniques are usually based on source code investigations [13]. For evaluating the impact of those criteria w.r.t. decreasing fault detection efficiency compared to complete product by product testing, further experiments, e.g., considering mutations, have to be performed.

6 Related Work

Various applications of behavioral models with variabilities to model-based SPL testing were proposed [18]. Cichos et al. propose a coverage-driven SPL test suite generation approach that is based on an annotative 150% test model [5]. Lochau et al. also use an annotative statechart test model for the detection and test coverage of interactions among feature artifacts [15,16]. Weissleder et al. define variabilities in state machines via annotations [32], whereas Szasz et al. add variable parts in Statecharts using composition operators [26].

Two research directions for reducing redundancies in product by product testing of SPLs currently exist: regression-based SPL testing and SPL subset selection heuristics. In [27,11], surveys on regression-based SPL testing approaches are presented mainly concentrating on empirical evaluations of different strategies. A first conceptional approach for regression-based SPL testing was, e.g., proposed by Batory et al. [31]. The authors propose an incremental refinement of test suites for a particular product variant under test w.r.t. the features composed into the product. Neto et al. [17] introduce an SPL testing framework, where regression testing decisions are performed on the basis of architectural similarities between product variants. Subset selection heuristics mainly use combinatorial testing heuristics to select representative products under test, e.g., considering features as combinatorial parameters [14]. For instance, Oster et al. cover pairwise feature combinations [20,21,19], whereas Perrouin et al. consider T-wise combinations [22]. However, no strategies for test artifact reuse between products in those sub sets are mentioned. The notion of SPL test suites introduced in [5] is the closest related to our framework, but no application strategies of those test suites are provided. Furthermore, as our approach incrementally generates test cases on demand rather than symbolically in one pass, it is assumed to obey better scalability properties.

7 Conclusion

In this paper, we presented a novel MBT framework for incrementally deriving test suites for SPL product variants by applying principles of regression testing. As future work, we plan to further optimize the SPL testing process by (1) local minimizations of product test suites as well as global reductions on complete SPL test suites, and (2) delta-oriented, i.e., compositional test suite generation. For reliable fault detection efficiency, further theoretical considerations concerning appropriate test case reuse and retest selection criteria are to be considered.

References

1. Agrawal, H., Horgan, J.R., Krauser, E.W., London, S.A.: Incremental Regression Testing (1993)
2. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: A Model-Checking Tool for Families of Services. In: Bruni, R., Dingel, J. (eds.) FORTE 2011 and FMOODS 2011. LNCS, vol. 6722, pp. 44–58. Springer, Heidelberg (2011)
3. Bruns, D., Klebanov, V., Schaefer, I.: Verification of Software Product Lines with Delta-Oriented Slicing. In: Beckert, B., Marché, C. (eds.) FoVeOOS 2010. LNCS, vol. 6528, pp. 61–75. Springer, Heidelberg (2011)
4. Calder, M., Kolberg, M., Magill, E., Reiff-Marganiec, S.: Feature Interaction: A Critical Review and Considered Forecast. Computer Networks 41(1), 115–141 (2003)
5. Cichos, H., Oster, S., Lochau, M., Schürr, A.: Model-Based Coverage-Driven Test Suite Generation for Software Product Lines. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 425–439. Springer, Heidelberg (2011)
6. Clarke, D., Helvensteijn, M., Schaefer, I.: Abstract Delta Modeling. Mathematical Structures in Computer Science (2011) (to appear)
7. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A., Raskin, J.F.: Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In: ICSE 2010 (2010)
8. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc. (2001)
9. Cohen, M., Dwyer, M., Shi, J.: Interaction Testing of Highly-Configurable Systems in the Presence of Constraints. In: ISSTA, pp. 129–139 (2007)
10. De Nicola, R.: Extensional Equivalence for Transition Systems. Acta Inf. 24, 211–237 (1987)
11. Engström, E., Skoglund, M., Runeson, P.: Empirical Evaluations of Regression Rest Selection Techniques. In: Proc. of ESEM 2008, pp. 22–31 (2008)

12. Fraser, G., Wotawa, F., Ammann, P.: Testing with Model Checkers: A Survey. Software Testing, Verification and Reliability 19(3), 215–261 (2009)
13. Gupta, R., Jean, M., Mary, H., Soffa, L.: An Approach to Regression Testing using Slicing. In: Proceedings of the Conference on Software Maintenance. pp. 299–308. IEEE Computer Society Press (1992)
14. Kim, C.H.P., Batory, D.S., Khurshid, S.: Reducing Combinatorics in Testing Product Lines. In: AOSD 2011, pp. 57–68. ACM (2011)
15. Lochau, M., Goltz, U.: Feature Interaction Aware Test Case Generation for Embedded Control Systems. ENTCS 264, 37–52 (2010)
16. Lochau, M., Oster, S., Goltz, U., Schürr, A.: Model-based Pairwise Testing for Feature Interaction Coverage in Software Product Line Engineering. Software Quality Journal, 1–38 (2011)
17. da Mota Silveira Neto, P.A., do Carmo Machado, I., Cavalcanti, Y.C., de Almeida, E.S., Garcia, V.C., de Lemos Meira, S.R.: A regression testing approach for software product lines architectures. In: SBCARS 2010, pp. 41–50 (2010)
18. Olimpiew, E.M.: Model-Based Testing for Software Product Lines. Ph.D. thesis, George Mason University (2008)
19. Oster, S., Lochau, M., Zink, M., Grechanik, M.: Pairwise Feature-Interaction Testing for SPLs: Potentials and Limitations. In: FOSD 2011 (2011)
20. Oster, S., Markert, F., Ritter, P.: Automated Incremental Pairwise Testing of Software Product Lines. In: Bosch, J., Lee, J. (eds.) SPLC 2010. LNCS, vol. 6287, pp. 196–210. Springer, Heidelberg (2010)
21. Oster, S., Zorcic, I., Markert, F., Lochau, M.: MoSo-PoLiTe - Tool Support for Pairwise and Model-Based Software Product Line Testing. In: VAMOS 2011 (2011)
22. Perrouin, G., Sen, S., Klein, J., Le Traon, B.: Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines. In: ICST 2010, pp. 459–468 (2010)
23. Schaefer, I., Bettini, L., Damiani, F.: Compositional Type-Checking for Delta-oriented Programming. In: AOSD 2011. ACM Press (2011)
24. Schaefer, I., Gurov, D., Soleimanifard, S.: Compositional Algorithmic Verification of Software Product Lines. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 184–203. Springer, Heidelberg (2011)
25. Schaefer, I., Hähnle, R.: Formal Methods in Software Product Line Engineering. IEEE Computer 44(2), 82–85 (2011)
26. Szasz, N., Vilanova, P.: Statecharts and Variabilities. In: VAMOS 2008, pp. 131–140 (2008)
27. Tevanlinna, A., Taina, J., Kauppinen, R.: Product Family Testing: A Survey. ACM SIGSOFT Software Engineering Notes 29, 12–18 (2004)
28. Thiel, S., Hein, A.: Modeling and Using Product Line Variability in Automotive Systems. IEEE Software 19(4), 66–72 (2002)
29. Tretmans, J.: Testing Concurrent Systems: A Formal Approach. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 46–65. Springer, Heidelberg (1999)
30. Utting, M., Legeard, B.: Practical Model-Based Testing. A Tools Approach. M. Kaufmann (2007)
31. Uzuncaova, E., Khurshid, S., Batory, D.S.: Incremental test generation for software product lines. IEEE Trans. Software Eng. 36(3), 309–322 (2010)
32. Weißleder, S., Sokenou, D., Schlingloff, H.: Reusing State Machines for Automatic Test Generation in ProductLines. In: MoTiP 2008 (2008)