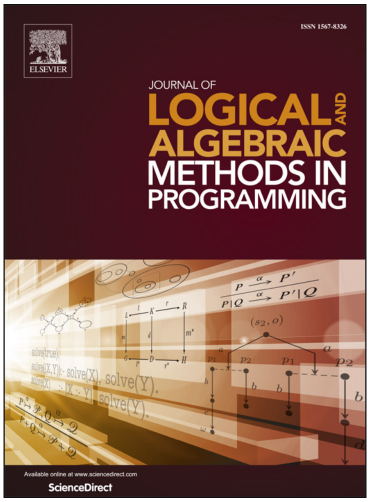

Input-Output Conformance Testing for Software Product Lines

Harsh Beohar, Mohammad Reza Mousavi

PII: S2352-2208(16)30117-1
DOI: http://dx.doi.org/10.1016/j.jlamp.2016.09.007
Reference: JLAMP 143

To appear in: *Journal of Logical and Algebraic Methods in Programming*

Received date: 1 April 2014
Revised date: 23 September 2016
Accepted date: 24 September 2016

Please cite this article in press as: M.R. Mousavi, Input-Output Conformance Testing for Software Product Lines, *J. Log. Algebraic Methods Program.* (2016), http://dx.doi.org/10.1016/j.jlamp.2016.09.007

Highlights

- Extending the theory of input output conformance testing to support input output featured transition systems as foundational behavioral models for software product lines.
- Defining notions of refinement on models and test-suites and showing their correspondence.
- Defining notion of residual and spinal test suites to factor out test executions and save on test effort for software product lines.
- Analyzing the relative testing power of residual and spinal test suites and their completeness.
- Analyzing an example regarding the Ceiling Speed Monitoring Function from the European Train Control System.

Input-Output Conformance Testing for Software Product Lines [*]

Harsh Beohar[a], Mohammad Reza Mousavi[a,1]

[a] *Centre for Research on Embedded Systems,*
School of Information Technology,
Halmstad University, Sweden

Abstract

We extend the theory of input-output conformance (IOCO) testing to accommodate behavioral models of software product lines (SPLs). We present the notions of residual and spinal testing. These notions allow for structuring the test process for SPLs by taking variability into account and extracting separate test suites for common and specific features of an SPL. The introduced notions of residual and spinal test suites allow for focusing on the newly introduced behaviour and avoiding unnecessary re-test of the old one. Residual test suites are very conservative in that they require retesting the old behaviour that can reach to new behavior. However, spinal test suites more aggressively prune the old tests and only focus on those test sequences that are necessary in reaching the new behaviour. We show that residual testing is complete but does not usually lead to much reduction in the test-suite. In contrast, spinal testing is not necessarily complete but does reduce the test-suite. We give sufficient conditions on the implementation to guarantee completeness of spinal testing. Finally, we specify and analyze an example regarding the Ceiling Speed Monitoring Function from the European Train Control System.

Keywords: Model based testing, Input-output conformance testing, Software product lines, Input-output featured transition systems

1. Introduction

1.1. Motivation

Software product lines (SPLs) have been proposed as a response to the ever-increasing demand for mass production and mass customization of software. Since their introduction, SPLs have gained popularity and have been increasingly used in the practice of software development. Briefly, an SPL consists of a variety of computer systems (products) that are built upon a common base (platform). The products share several core features, but also differ from each other in some features, commonly referred to as variability points.

Testing such SPLs is known to be very challenging due to the large spectrum of variability and the complexity of products. There have been several attempts to provide a structured discipline for testing SPLs. However, it appears from the recent surveys [3–7] that several fundamental approaches to model-based testing are not yet fully adapted to and adopted in this domain (also see Section 1.2 for a brief overview of the related work).

The theory of input-output conformance (IOCO) testing [8] is one such fundamental approach that uses labeled transition systems for model-based testing. The *testing hypothesis* of this approach is that the behaviour of the implementation under test can be viewed as an (unknown) input-output labeled transition system that is input-enabled, i.e., can accept any input action. We are not aware of any prior work in adapting the theory of IOCO to cater for variability in SPLs. The present paper addresses this gap by extending IOCO to the setting of SPLs.

[*] This article integrates and extends the authors' ACM SAC SVT 2014 and MBT 2014 papers [1, 2]. An extended abstract was presented at the 25th Nordic Workshop on Programming Theory, NWPT 2013, in Tallinn.

Email addresses: harsh.beohar@hh.se (Harsh Beohar), m.r.mosuavi@hh.se (Mohammad Reza Mousavi)

[1] The work of Mohammad Reza Mousavi has been partially supported by the Swedish Research Council (Vetenskapsradet) award number: 621-2014-5057 (Effective Model-Based Testing of Concurrent Systems) and the Swedish Knowledge Foundation (Stiftelsen for Kunskaps- och Kompetensutveckling) in the context of the AUTO-CAAS HöG project (number: 20140312).

To this end, we propose input-output featured transition systems (IOFTSs) as simple yet expressive behavioural models of SPLs and adapt the traditional IOCO theory to allow for using IOFTSs (instead of plain input-output transition system models) as test models for model-based testing. Our approach preserves the testing hypothesis of IOCO; although we include more information in our test models to capture the structure of SPLs, the interaction with the system under test only goes via plain input and output actions and the internal structure of the product is not revealed during the test execution. We define the test suite and the test cases that are generated from an IOFTS, which can be used for checking conformance. Furthermore, we define two notions of refinement, one at the level of IOFTSs and another one at the level of test suites, which allow for focusing on particular sets of features and eventually on a particular product. We show that these two refinements interact nicely, in that they lead to the same set of test cases. The techniques proposed in this paper are rather generic and we believe these techniques can be adapted to other model-based testing theories (such as those proposed in [9–12].)

In addition, we take first step towards an efficient and coordinated test process for applying IOCO to SPLs. To this end, we develop a theoretical framework of residual and spinal test suites. Intuitively, both residual and spinal test suites are IOFTSs (whose underlying graph is tree-like), which allow one to test the common features once and for all, and subsequently, only focus on the specific features when moving from one product configuration to another. However, they differ in their testing power and efficiency: testing power refers to the possibility of rejecting non-conforming implementations (ideally a test suite is complete, i.e., it can reject each and every non-conforming implementation by generating at least one failing test case), and efficiency refers to the size of the test-suite. On one hand, spinal test suites have strictly less testing power than residual test suites; on the other hand, spinal test suites produce more compact test cases when compared to test cases produced by residual test suites. We show that residual test suites are complete, i.e., for each product it is always sufficient to use the residual test suite with respect to the features present in the afore-tested products, whereas spinal test suites are not necessarily complete. Lastly, we also show that spinal test suites are exhaustive, i.e., they reject each and every non-conforming implementation under test, when the implementation satisfies the *orthogonality criterion*. This is a rather mild criterion, which implies that old features are not capable of disabling any enabled behaviour from the new features on their own and without involving any interaction with the new feature's components.

The proposed theory is the first step towards a feature-based analysis [13] of SPLs based on the IOCO theory. For example, once a feature (combination) selection criterion is fixed, one can use the spinal testing method to focus test those features (feature combinations) in a selection of concrete products.

1.2. Related work

Various attempts have been made regarding formal and informal modeling of SPLs, of which [14–18] provide comprehensive surveys. By and large, the literature can be classified into two categories: structural modeling and behavioural modeling techniques.

Structural models specify variability in terms of presence and absence of features (assets, artifacts) in various products and their mutual inter-relations. Behavioural models, however, concern the working of features and their possible interactions, mostly based on some form of finite state machines or labeled transition systems. The main focus in behavioural modeling of SPLs (cf. [19–25]) has been on formal specification of SPLs and adaptation of formal verification (mostly model checking) techniques to this new setting. Our notion of input-output featured transition system is a slight extension of featured transition system [21]. There are few alternatives to FTSs that could be used as behavioural test models for model-based testing [26]. Such models include the extensions of process algebras- [24, 27] and Petri nets with features [28], modal transition systems [23, 29] and higher-level models such as UML state- and sequence-diagrams [30–32].

In this paper, we assume a predefined structure of the SPL in terms of a feature diagram. The structural information in the feature diagram is used to annotate the behavioural model and steer the test process. An alternative approach to specifying and programming SPLs is the delta-oriented approach [11, 12, 33–35], where the SPL is specified in terms of additions to, removals from, or modifications of the core product. Although our work is based on input-output featured transition systems, we envisage that the ideas pursued in this paper can be adapted to other behavioural test models and to other conformance testing theories, such as those on finite state machines [9, 10] and on delta-oriented methods. For example, recently, in [12, 35] related techniques have been explored in the area of delta-oriented SPL models. For higher-level modeling frameworks, our input-output featured transition systems can serve as a semantic domain; this way, our techniques can be applied to higher-level modeling and specification languages (such as UML state diagrams or domain specific languages).

Several testing techniques have been adapted to SPLs, of which [3–6] provide recent overviews. Hitherto, most fundamental approaches to formal conformance testing [10] have not been adapted sufficiently to the SPL setting. The only exceptions that we are aware of are [11] and [36–38], which, respectively, present LTS- and FSM-based incremental derivation of test suites by applying principles of delta-oriented modeling and incremental testing [33].

This paper integrates and extends the earlier conference and workshop publications of the authors [1, 2]. Namely, we first proposed the extension of IOCO to Featured Transition Systems in [1] and the definition of spinal test suites in [2]. The present paper includes complete proofs and more elaborate explanations, which could not be fitted into the earlier publications due to space limitation. Additionally, the present paper also introduces the notion of residual test suite as an intermediate step towards the notion of spinal test suite and studies its properties. It also improves the earlier definition of spinal test suite to make it a subset of residual test suite. The improved definition is shown to satisfy all its earlier properties. We also apply our theory to an example regarding the Ceiling Speed Monitoring Function from the European Train Control System [39, 40].

1.3. Running Example

To motivate various concepts throughout the paper, we use the following running example. Consider an informal description of a cruise controller, present in contemporary cars. The purpose of a cruise controller is to automatically maintain the speed of the car as specified by the driver.

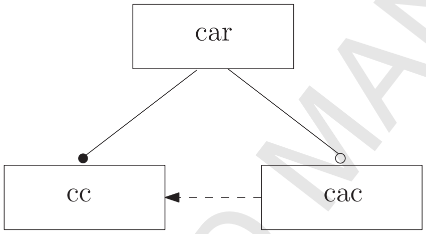

Figure 1: Feature diagram of our running example: a cruise controller (cc) with an optional collision controller (cac)

The feature diagram of this example is depicted in Figure 1. We denote the basic feature of a cruise controller by cc. This feature is mandatory for a car, which is reflected by the filled circle at the end of the relation between car and cc. Cruise controllers also have an optional feature, called collision avoidance controller (cac), whose task is to react to any obstacle detected ahead of the car within a danger zone. In case the collision avoidance feature is included in a cruise controller and an obstacle is detected, the engine power is regulated using an emergency control algorithm. The fact the cac is optional is depicted by the empty circle at the end of the relation between car and cac. In the feature diagram, we also see that cac can only be included in products that also include cc; this is depicted by the dashed arrow from the former feature to the latter.

1.4. Organization

In Section 2, we define the notion of input-output featured transition systems as our basic modeling framework. In Section 3, a notion of refinement is proposed that allows for projecting the behaviour of an SPL into the behaviour of a product or a product sub-line. In Section 4, we define the notions of test suite and test case. In Section 5, a notion of refinement is given on test suites, which allows for deriving more specific test suites from the more generic ones. In the same section, we show that

- the above-mentioned notions of refinement (i.e., on models and test suites) are consistent in that they lead to the same set of test cases, and
- the intensional and extensional notions of conformance testing coincide, i.e., non-conformance can always be established by means of running test-cases.

We then turn our attention to efficient testing techniques for SPLs. In Section 6, we define the notion of residual test suites and show that although residual test suites are complete, they do not result in much saving in test effort. In Section 7, we define spinal test suites which are not necessarily complete, but result in more compact test cases. In the same section, we show that the incompleteness of spinal test suites can be remedied by imposing mild conditions on the implementation under test. In Section 8, we specify the Ceiling Speed Monitoring Function and illustrate the different aspects of our proposed techniques using this example. In Section 9, we conclude the paper and outline the direction of our ongoing research.

2. Input-Output Featured Transition Systems

Feature diagrams [41, 42] have been used to model variability constraints in SPLs using a graphical notation. A feature diagram represents all valid products of an SPL in terms of features that are arranged hierarchically. (Note that the hierarchical structure of features does not imply that the specified products are also arranged hierarchically: the products are rather the result of interpreting the relations among the different features and hence, may not be related in any hierarchical form.) Usually, feature diagrams are represented by a directed acyclic graph, of which each node is a feature. There are different kinds of edges between a parent node (feature) and its children (sub-features), namely, the ones representing the mandatory sub-features, and the others representing the optional sub-features. Furthermore, a feature diagram can specify three additional type of constraints on features:

1. Alternative relationship, i.e., the designated sub-features can never be simultaneously present in any product.
2. Exclude relationship, i.e., different features (possibly at different levels of the hierarchy) can never be simultaneously present in any product.
3. Require relationship, i.e., if a feature is present in a product, the related feature should also be present in the same product.

Alternative and exclude relationship are conceptually similar; their difference is in that the alternative relationship is among sub-features of a single feature, while exclude can be between any two arbitrary features.

A feature diagram only specifies the structural aspects of variability in an SPL. To formally analyze the behaviour of an SPL, we follow the approach of [21] in annotating the transitions of a labeled transition system with logical constraints on the presence or absence of features. The features used in such logical constraints are assumed to be already specified in a feature diagram. We slightly extend the featured transition system of [21] to cater for the distinction between input and output actions. This is a necessary ingredient for extending the theories of testing, and particularly IOCO, to this setting.

Let $\mathbb{B} = \{\top, \bot\}$ be the set of Boolean constants and let $\mathbb{B}(F)$ be the set of all propositional formulae generated by using the usual propositional logic connectives (e.g., negation, disjunction, conjunction, and implication) and by interpreting the elements of the set F of *features* as propositional variables. For instance, in our running example, the formula $\mathsf{cc} \land \neg\mathsf{cac}$ asserts the presence of cruise controller and the absence of collision avoidance controller.

Definition 1. An *input-output featured transition system* (IOFTS) is a 6-tuple $\mathcal{F} = (S, s_0, A_\tau, F, T, \Lambda)$, where

1. S is a set of *states*.
2. $s_0 \in S$ is the *initial state*.
3. $A_\tau = A_\mathcal{I} \uplus A_\mathcal{O} \uplus \{\tau\}$ is a set of *actions*, where $A_\mathcal{I}$ and $A_\mathcal{O}$ are disjoint sets of *input* and *output* actions, respectively, and τ is the silent (internal) action.
4. F is a set of *features*.
5. $T \subseteq S \times A_\tau \times \mathbb{B}(F) \times S$ is the *transition relation* satisfying the following condition (for every $s_1, s_2 \in S, a \in A_\tau, \varphi, \varphi' \in \mathbb{B}(F)$):
$$(s_1, a, \varphi, s_2) \in T \land (s_1, a, \varphi', s_2) \in T \Rightarrow \varphi = \varphi'.$$
Informally, this condition states that for any two transitions with the common source, target, and an action label, a unique feature constraint is annotated. In practice, one can ensure this condition by the following normalisation procedure: for each $s, s' \in S$ and $a \in A_\tau$, replace all $(s, a, \varphi_i, s') \in T$ (for each $i \in I$) by $(s, a, \bigvee_{i \in I} \varphi_i, s')$. We require this condition to stay close with the original formulation of featured transition system as proposed in [21].
6. $\Lambda \subseteq \{\lambda : F \to \mathbb{B}\}$ is a non-empty set of *product configurations*.

Notation. We let φ, φ' range over the set $\mathbb{B}(F)$ of feature constraints and reserve the symbols $s, s', s_1, s_1', \cdots$ to denote the states of an IOFTS. In addition, we write $s \xrightarrow{a}_{\varphi} s'$ to denote an element $(s, a, \varphi, s') \in T$. Graphically, we denote the initial state of an IOFTS by an incoming arrow with no source state and we refer to an IOFTS by its initial state.

Note that IOFTS are not necessarily finite, e.g., they may be the result of unfolding a symbolic specification with infinite data types. Our theory can deal with such infinite specifications; for practical applications, however, one needs to tame the complexity, e.g., by considering finite fault models or finitely feasible model coverage metrics.

Definition 2. Let $\twoheadrightarrow \subseteq S \times A^* \times S$ denote the *reachability* relation of an IOFTS $\mathcal{F} = (S, s_0, A_\tau, F, T, \Lambda)$, inductively defined in the following way:

$$\frac{}{s \overset{\varepsilon}{\twoheadrightarrow}_{\top} s} \qquad \frac{s \overset{\sigma}{\twoheadrightarrow}_{\varphi} s' \quad s' \xrightarrow{\tau}_{\varphi'} s'' \quad \lambda \in \Lambda \quad \lambda \models \varphi \wedge \varphi'}{s \overset{\sigma}{\twoheadrightarrow}_{\varphi \wedge \varphi'} s''} \qquad \frac{s \overset{\sigma}{\twoheadrightarrow}_{\varphi} s' \quad s' \xrightarrow{a}_{\varphi'} s'' \quad a \neq \tau \quad \lambda \in \Lambda \quad \lambda \models \varphi \wedge \varphi'}{s \overset{\sigma a}{\twoheadrightarrow}_{\varphi \wedge \varphi'} s''},$$

where $\lambda \models \varphi$ denotes that valuation λ of features satisfies feature constraint φ. The set of *reachable* states from a state $s \in S$ by a trace $\sigma \in A^*$ is denoted by $\text{Reach}(s, \sigma) = \{s' \mid \exists_{\varphi}\ s \overset{\sigma}{\twoheadrightarrow}_{\varphi} s'\}$. Furthermore, we fix $\text{Reach}(s) = \{s' \mid \exists_{\sigma,\varphi}\ s \overset{\sigma}{\twoheadrightarrow}_{\varphi} s'\}$ and, for brevity, we write $s \overset{a}{\twoheadrightarrow} s'$ if and only if $\exists_{\varphi}\ s \overset{a}{\twoheadrightarrow}_{\varphi} s'$.

We say an IOFTS $\mathcal{F}$ is *deterministic* if and only if the set $\text{Reach}(s_0, \sigma)$ (for any $\sigma \in A^*$) is singleton. Furthermore, an IOFTS $\mathcal{F}$ is *B-enabled* (for $B \subseteq A_\tau$) if and only if for every reachable state $s \in \text{Reach}(s_0)$ and an action $a \in B$, we find a feature constraint $\varphi \in \mathbb{B}(F)$, a product configuration $\lambda \in \Lambda$, and a state $s' \in S$ such that $s \overset{a}{\twoheadrightarrow}_{\varphi} s'$ and $\lambda \models \varphi$. Lastly, we say an IOFTS $\mathcal{F}$ is *input-enabled*, whenever $\mathcal{F}$ is $A_{\mathcal{I}}$-enabled.

Our notion of $\text{Reach}(s, \sigma)$ is similar to the corresponding notion of $\text{after}(s, \sigma)$ in the theory of IOCO [8]; the only difference is that in our notion, the states should be reachable through paths whose constraints are satisfied by some valid product.

Example 1. Recall our running example of a cruise controller described in Section 1.3. Consider the IOFTS of the cruise controller, drawn in Figure 2, where inputs and outputs are prefixed with symbols ? and !, respectively, and the feature constraints are attached to the action labels by the symbol /. Please note that the feature constraints refer to the features present in the feature diagram depicted in Figure 1, namely, cc stands for the cruise controller feature and cac stands for the collision avoidance controller feature. (Note that ? and ! are *not part* of the action names and are left out when the type of the action is irrelevant or clear from the context.) The regulate action, indicated by rgl, regulates the engine power of the car when the cruise controller is activated. Furthermore, when cac is included in a product, some additional behaviour may emerge. Namely, while the cruise controller is on, if an object is detected within a danger zone, then the cruise controller regulates the engine power in a safe manner denoted by srgl. When the sensor signals a normal state, the cruise controller returns to the normal regulation regime. (For a realistic case study of a cruise controller and its formal model, we refer to [43].)

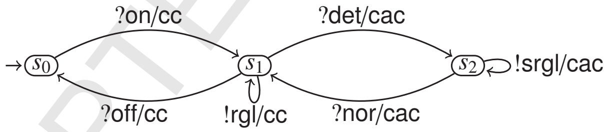

Figure 2: IOFTS of the cruise controller.

The set of product configurations for this IOFTS includes two products: one including only car and cc, and the other including car, cc, and cac.

3. Refinement of Models

In [21], a family of operators, parameterised by product configuration, were introduced to project an FTS into a labeled transition system describing the behaviour of a specific product. In this paper, we generalise this approach by defining a family of product derivation operators (parameterised by feature constraints), which project the behaviour of an IOFTS into another IOFTS representing a selection of products (a product sub-line).

Definition 3. Given a feature constraint $\varphi \in \mathbb{B}(F)$ and an IOFTS $\mathcal{F} = (S, s_0, A_\tau, F, T, \Lambda)$, the projection of $\mathcal{F}$ into φ, denoted by $\Delta_\varphi(\mathcal{F})$, induces an IOFTS $(\Delta_\varphi(S), \Delta_\varphi(s_0), A_{\tau\delta}, F, T', \Lambda')$, where

1. $\Delta_\varphi(S) = \{\Delta_\varphi(s) \mid s \in S\}$ is the set of states (N.B. in the set comprehension $\Delta_\varphi(s)$ is just a new *name* for the state and does not have any semantical connotation),
2. $\Delta_\varphi(s_0)$ is the initial state,
3. $A_{\tau\delta} = A_\tau \cup \{\delta\}$ is the set of actions, where δ is the special action label modeling quiescence [8],
4. T' is the smallest relation satisfying:

$$\frac{s \overset{a}{\rightarrow}_{\varphi'} s' \quad \exists_\lambda\, (\lambda \in \Lambda \wedge \lambda \models (\varphi \wedge \varphi'))}{\Delta_\varphi(s) \overset{a}{\rightarrow}_{\varphi \wedge \varphi'} \Delta_\varphi(s')} \quad (1) \qquad \frac{\bar{\Lambda} = \{\lambda \in \Lambda \mid \lambda \models \varphi \wedge \mathbf{Q}(s, \lambda)\} \quad \bar{\Lambda} \neq \emptyset}{\Delta_\varphi(s) \overset{\delta}{\rightarrow}_{\varphi \wedge (\bigvee_{\lambda \in \bar{\Lambda}} \lambda)} \Delta_\varphi(s)} \quad (2)$$

   where the predicate $\mathbf{Q}(s, \lambda)$ holds if and only if $\forall_{s', a, \varphi'}\, (s \overset{a}{\rightarrow}_{\varphi'} s' \wedge a \in A_O \cup \{\tau\}) \Rightarrow \lambda \not\models \varphi'$.
5. $\Lambda' = \{\lambda \in \Lambda \mid \lambda \models \varphi\}$ is the set of product configurations.

Intuitively, rule (1) describes the behaviour of those valid products that satisfy the feature constraint φ in addition to the original annotation of the transition emanating from s. Rule (2) models quiescence (the absence of outputs and internal actions) from the state $\Delta_\varphi(s)$. Namely, it specifies that the projection with respect to φ is quiescent, when there exists a valid product λ that satisfies φ and is quiescent, i.e., it cannot perform any output or internal transition. Quiescence at state s for a feature constraint λ is formalised using the predicate $\mathbf{Q}(s, \lambda)$, which states that from state s there is no output or silent transition with a constraint satisfied by λ. In the conclusion of the rule, a δ self-loop is specified and its constraint holds when φ holds and the feature constraint of at least one quiescent valid product holds.

The ability to observe quiescence is crucial in defining the input-output conformance relation between a specification and an implementation. In the original IOCO theory, quiescence is used reject those implementations that fail to produce *any* output when they in fact should produce some. In the SPL setting the issue of detecting quiescence becomes more intricate; namely, at a high level of abstraction, the SPL specification is more allowing (for producing different outputs) and hence admits less quiescent states. As the SPL specification is refined towards concrete products, the domain of outputs in states becomes more and more restricted and hence, more quiescent states appear. The way quiescence is defined in rule (2) is essential in the top-down testing methodology prescribed by the refinement relation: one can start with a more generic test suite and move on to more specific test suites using the refinement operator and the test results using the more generic test suite remain sound with respect to the more specific test suite (cf. Section 4 for quiescence in test suites).

Example 2. Consider the feature constraint $\varphi = \mathsf{cc} \wedge \neg\mathsf{cac}$. The IOFTS generated by projecting the IOFTS of cruise controller (in Figure 2) using feature constraint φ is depicted in Figure 3. As mentioned before, this represents the product that has the basic cruise controller functionality but does not contain collision avoidance controller.

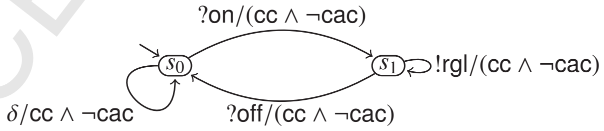

Figure 3: Cruise controller after projecting with feature constraint $\mathsf{cc} \wedge \neg\mathsf{cac}$.

In the sequel, we use the phrase "a feature specification $\Delta_\varphi(s)$" to mean an IOFTS $(\mathrm{Reach}(\Delta_\varphi(s)), \Delta_\varphi(s), A_{\tau\delta}, F, T, \Lambda)$, where $\mathrm{Reach}(\Delta_\varphi(s))$ is the reachable set of states in products satisfying φ given in Definition 2. We interpret the original IOFTS of Definition 1 as $\Delta_\top(s_0)$; this has the implicit advantage of always including quiescence in appropriate states.

We end this section by the following proposition which relates the traces in the refined specification to those of the original (more generic) specification. This proposition has been formulated in a slightly different context earlier in [44]. As a corollary, it follows that the set of traces of a refined feature specification is a subset of the traces of the more generic specification.

Proposition 1. *For any $\sigma \in A_\delta{}^*$ we have, if $\Delta_{\varphi\wedge\varphi'}(s) \overset{\sigma}{\twoheadrightarrow} \Delta_{\varphi\wedge\varphi'}(s')$ then $\Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow} \Delta_\varphi(s')$.*

Proof. By induction on the depth of the derivation leading to $\Delta_{\varphi\wedge\varphi'}(s) \overset{\sigma}{\twoheadrightarrow} \Delta_{\varphi\wedge\varphi'}(s')$ (see Definition 2). The induction basis, i.e., when $\sigma = \varepsilon$ and derivation is due the left-most axiom, holds trivially. For the induction steps, it suffices to show that $\Delta_\varphi(s'') \xrightarrow{a} \Delta_\varphi(s')$ whenever $\Delta_{\varphi\wedge\varphi'}(s'') \xrightarrow{a} \Delta_{\varphi\wedge\varphi'}(s')$ for some $a \in A_{\tau\delta}$. (Once we prove this claim, using the induction hypothesis, the proven claim, and the last deduction rule used in the derivation of the $\overset{\sigma}{\twoheadrightarrow}$, we obtain $\Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow} \Delta_\varphi(s')$.) We distinguish the following cases based on the type of action.

1. Let $a \in A_\tau$. It follows from $\Delta_{\varphi\wedge\varphi'}(s'') \xrightarrow{a} \Delta_{\varphi\wedge\varphi'}(s')$ that there exists a φ'' such that $\Delta_{\varphi\wedge\varphi'}(s'') \xrightarrow{a}_{\varphi\wedge\varphi'\wedge\varphi''} \Delta_{\varphi\wedge\varphi'}(s')$. From the latter statement and rule (1) in Definition 3, we obtain:

$$\begin{aligned}
&\Delta_{\varphi\wedge\varphi'}(s'') \xrightarrow{a}_{\varphi\wedge\varphi'\wedge\varphi''} \Delta_{\varphi\wedge\varphi'}(s') \\
&\Rightarrow s'' \xrightarrow{a}_{\varphi''} s' \wedge \exists_{\lambda\in\Lambda}\ \lambda \models \varphi \wedge \varphi' \\
&\Rightarrow s'' \xrightarrow{a}_{\varphi''} s' \wedge \exists_{\lambda\in\Lambda}\ \lambda \models \varphi \\
&\Rightarrow \Delta_\varphi(s'') \xrightarrow{a}_{\varphi\wedge\varphi''} \Delta_\varphi(s')\ .
\end{aligned}$$

2. Let $a = \delta$. Then, using rule (2) in Definition 3, we obtain

$$\begin{aligned}
&\bar{\Lambda} = \{\lambda \in \Lambda \mid \lambda \models \varphi \wedge \varphi' \wedge \mathbf{Q}(s'', \lambda)\} \wedge \bar{\Lambda} \neq \emptyset \\
&\Rightarrow \bar{\Lambda}' = \{\lambda \in \Lambda \mid \lambda \models \varphi \wedge \mathbf{Q}(s'', \lambda)\} \wedge \bar{\Lambda}' \neq \emptyset \\
&\Rightarrow \Delta_\varphi(s'') \xrightarrow{\delta}_{\varphi\wedge(\vee_{\lambda\in\bar{\Lambda}'})} \Delta_\varphi(s')\ .
\end{aligned}$$

□

4. Test suite and test cases

The IOCO testing theory [8] formalises model-based testing in terms of a conformance relation between a model and a system under test (SUT). This relation can be checked by constantly providing the SUT with inputs that are deemed relevant by the model (expressed as an IOTS: input-output labeled transition system) and observing outputs from the SUT and comparing them with the possible outputs prescribed by the model. The IOCO theory is based on the *testing assumption* that the behaviour of the system under test can be expressed by an input-enabled IOTS, which is unknown to the tester. In addition to the above-sketched *extensional* definition of IOCO, there is an equivalent *intensional* definition, which relies on comparing the traces of the underlying IOTSs.

In what follows, we first extend the intensional notion of conformance between any two feature specifications (Definition 5). To be consistent with the theory of IOCO, we base our theory on the same testing assumption as IOCO. In particular, our testing hypothesis requires that no product under test refuses any input. Then, using the concept of test suite (Definition 6), we give an extensional definition of the class of test cases for a given specification $\Delta_\varphi(s)$.

To formally define both the intensional and the extensional notion of IOCO, we need the notion of *suspension traces* [8] in an IOFTS. Informally, a suspension trace is a trace that may also contain quiescence. For example, in the IOFTS of Example 2, δ ?on !rgl is a suspension trace starting from the initial state s_0.

Definition 4. The set of *suspension traces* of a feature specification $\Delta_\varphi(s)$ is defined as:

$$\mathrm{Straces}(\Delta_\varphi(s)) = \left\{\sigma \in A_\delta{}^* \mid \mathrm{Reach}(\Delta_\varphi(s), \sigma) \neq \emptyset\right\}.$$

Intuitively, the IOCO relation asserts that the experiments derived from a feature specification (i.e., the suspension traces of a feature specification) and executed on the implementation under test, result in outputs among those that are prescribed by the feature specification.

Definition 5. An implementation modeled as a feature specification $\Delta_\varphi(i)$ is *input-output conforming* to a specification modeled as a feature specification $\Delta_{\varphi'}(s)$, denoted by $\Delta_\varphi(i) \sqsubseteq_{\mathbf{ioco}} \Delta_{\varphi'}(s)$, if and only if

$$\mathrm{out}(\mathrm{Reach}(\Delta_{\varphi'}(s), \sigma)) \subseteq \mathrm{out}(\mathrm{Reach}(\Delta_\varphi(i), \sigma)),$$

for every suspension trace $\sigma \in \mathrm{Straces}(\Delta_{\varphi'}(s))$, where $\mathrm{out}(X)$ denotes the set of output enabled from the states in the set X, i.e., $\mathrm{out}(X) = \{a \in A_O \cup \{\delta\} \mid \exists_{s \in X, s'}\ s \xrightarrow{a} s'\}$.

Next, with the help of the following theorem, we establish a formal link between the refinement of feature constraints and the IOCO relation (cf. Corollary 1).

Theorem 1. *Let Λ be the set of valid products of a feature specification $\Delta_\varphi(s)$. Then, the following statements hold.*

1. *If $\Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow}_{\bar{\varphi}} \Delta_\varphi(s')$ (for some $\sigma, \bar{\varphi}, s'$) then $\exists_{\lambda \in \Lambda}\ \lambda \models \bar{\varphi}$.*
2. *Let φ' be a feature constraint such that $\forall_\lambda\ \lambda \models \varphi \implies \lambda \models \varphi'$. If $\Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow}_{\bar{\varphi}} \Delta_\varphi(s')$ and $\lambda \models \bar{\varphi}$ (for $\lambda \in \Lambda$), then*
$$\exists_{\hat{\varphi}}\ \Delta_{\varphi'}(s) \overset{\sigma}{\twoheadrightarrow}_{\hat{\varphi}} \Delta_{\varphi'}(s') \ \wedge\ \lambda \models \hat{\varphi}.$$
3. *Let φ' be a feature constraint such that $\forall_\lambda\ \lambda \models \varphi \implies \lambda \models \varphi'$. Then, Straces$(\Delta_\varphi(s)) \subseteq$ Straces$(\Delta_{\varphi'}(s))$.*

Proof. Item (1) follows directly from the induction on φ and the definition of reachability relation. Next, again using induction on σ we prove (2). Let Λ, Λ' be the set of valid products of the feature specifications $\Delta_\varphi(s), \Delta_{\varphi'}(s)$, respectively. Without loss of generality, assume $\sigma = \sigma' a$, for some $\sigma' \in \mathrm{Straces}(\Delta_\varphi(s))$, $a \in A_\delta$ (the case when $\sigma = \varepsilon$ holds vacuously). Then there are states s', s'' such that $\Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow}_{\bar{\varphi}} \Delta_\varphi(s') \xrightarrow{a}_{\bar{\varphi} \wedge \bar{\varphi}'} \Delta_\varphi(s'')$. We distinguish the following two cases:

- Let $a \in A_\tau$. Then $\bar{\varphi}'$ is the feature constraint associated with the triple (s', a, s''). By the definition of reachability relation and the semantics of Δ, we find a product $\lambda \in \Lambda$ such that $\lambda \models \bar{\varphi} \wedge \bar{\varphi}'$. Thus, $\lambda \models \bar{\varphi}'$. Moreover, from the inductive hypotheis we find a feature constraint $\hat{\varphi}$ such that $\Delta_{\varphi'}(s) \overset{\sigma}{\twoheadrightarrow}_{\hat{\varphi}} \Delta_{\varphi'}(s')$ and $\lambda \models \hat{\varphi}$. Thus, we conclude that $\Delta_{\varphi'}(s) \overset{\sigma a}{\twoheadrightarrow}_{\hat{\varphi} \wedge \bar{\varphi}'} \Delta_{\varphi'}(s'')$ and $\lambda \models \hat{\varphi} \wedge \bar{\varphi}'$.
- Let $a = \delta$. Then, by the semantics of Δ we know that $\bar{\varphi}' = \varphi \wedge \bigvee_{\bar{\lambda} \in \bar{\Lambda}} \lambda$ and $\bar{\Lambda} = \{\bar{\lambda} \in \Lambda \mid \bar{\lambda} \models \varphi \wedge \mathbf{Q}(s', \bar{\lambda})\} \neq \emptyset$. Moreover, using the definition of reachability relation we find a product $\lambda \in \Lambda$ such that $\lambda \models \bar{\varphi} \wedge \bar{\varphi}'$. I.e., there is a $\lambda \in \bar{\Lambda}$ such that $\lambda \models \bar{\varphi} \wedge \varphi \wedge \bigvee_{\bar{\lambda} \in \bar{\Lambda}} \bar{\lambda}$. Using the assumption on φ', we have $\lambda \models \varphi'$ (since $\lambda \models \varphi$). Consider the set $\bar{\Lambda}' = \{\lambda' \in \Lambda' \mid \lambda' \models \varphi' \wedge \mathbf{Q}(s', \lambda')\}$. Clearly, $\lambda \in \bar{\Lambda}'$ and thus, $\bar{\Lambda}' \neq \emptyset$.

  Moreover, from the induction hypothesis we find a feature constraint $\hat{\varphi}$ such that $\Delta_{\varphi'}(s) \overset{\sigma}{\twoheadrightarrow}_{\hat{\varphi}} \Delta_{\varphi'}(s')$ and $\lambda \models \hat{\varphi}$. Let $\hat{\varphi}' = \varphi' \wedge \bigvee_{\lambda' \in \bar{\Lambda}'} \lambda'$. Then, we find $\Delta_{\varphi'}(s) \overset{\sigma\delta}{\twoheadrightarrow}_{\hat{\varphi} \wedge \hat{\varphi}'} \Delta_{\varphi'}(s')$ and $\lambda \models \hat{\varphi} \wedge \hat{\varphi}'$.

(3) directly follows from (1) and (2), i.e., symbolically, $(1) \wedge (2) \implies (3)$. □

Corollary 1. *Let φ, φ' be two feature constraints such that $\forall_\lambda\ \lambda \models \varphi \Rightarrow \lambda \models \varphi'$. If $\Delta_{\varphi''}(s') \sqsubseteq_{\mathbf{ioco}} \Delta_{\varphi'}(s) \wedge \Delta_\varphi(s) \sqsubseteq_{\mathbf{ioco}} \Delta_\varphi(s)$, for some φ'', s', then $\Delta_{\varphi'}(s') \sqsubseteq_{\mathbf{ioco}} \Delta_\varphi(s)$.*

Proof. Let $\sigma \in \mathrm{Straces}(\Delta_\varphi(s))$. Then, we need to show that $\mathrm{out}(\mathrm{Reach}(\Delta_{\varphi'}(s'), \sigma)) \subseteq \mathrm{out}(\mathrm{Reach}(\Delta_\varphi(s), \sigma))$.

From Theorem 1, we know that $\sigma \in \mathrm{Straces}(\Delta_{\varphi'}(s))$. Thus, we have

$$\mathrm{out}(\mathrm{Reach}(\Delta_{\varphi''}(s'), \sigma)) \subseteq \mathrm{out}(\mathrm{Reach}(\Delta_{\varphi'}(s), \sigma)) \subseteq \mathrm{out}(\mathrm{Reach}(\Delta_\varphi(s), \sigma)).$$ □

Next we give an operational definition (in the sense of [45]) of test suites, which allows for generating a test suite for a product line and refining it into test suites for more specific sub-lines (and eventually generating test cases for a specific product).

Definition 6. The *test suite* for an IOFTS $(\mathrm{Reach}(\Delta_\varphi(s)), \Delta_\varphi(s), A_{\tau\delta}, F, T, \Lambda)$, denoted by $\mathcal{T}(s, \varphi)$, is the IOFTS $(\mathbf{X} \cup \{\mathbf{pass}, \mathbf{fail}\}, (\mathbf{X}_0, \varepsilon), A_\delta, F, T', \Lambda)$, where

1. $\mathbf{X} = \left\{\left(\{s' \mid \Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow} \Delta_\varphi(s')\}, \sigma\right) \mid \sigma \in \text{Straces}(\Delta_\varphi(s))\right\}$ is the set of intermediate states and $\{\mathbf{pass}, \mathbf{fail}\}$ is the set of *verdict states*,
2. $(\mathbf{X}_0, \varepsilon)$ is the initial state of the test suite, where $\mathbf{X}_0 = \{s' \mid \Delta_\varphi(s) \overset{\varepsilon}{\twoheadrightarrow} \Delta_\varphi(s')\}$,
3. $A_\delta = A \uplus \{\delta\}$ is the set of actions, and
4. the transition relation T' is defined as the smallest relation satisfying the following rules.

$$\frac{a \in A_\delta \quad (X,\sigma),(Y,\sigma a) \in \mathbf{X}}{(X,\sigma) \xrightarrow{a}_\varphi (Y,\sigma a)} \quad (3) \qquad \frac{a \in A_O \cup \{\delta\} \quad (X,\sigma) \xrightarrow{a}_\varphi (Y,\sigma')}{(X,\sigma) \xrightarrow{a}_\varphi \mathbf{pass}} \quad (4)$$

$$\frac{a \in A_O \cup \{\delta\} \quad \neg\left((X,\sigma) \xrightarrow{a}_\varphi \mathbf{pass}\right)}{(X,\sigma) \xrightarrow{a}_\varphi \mathbf{fail}} \quad (5) \qquad \frac{a \in A_O \cup \{\delta\}}{\mathbf{pass} \xrightarrow{a}_\varphi \mathbf{pass} \quad \mathbf{fail} \xrightarrow{a}_\varphi \mathbf{fail}} \quad (6)$$

Intuitively, the test suite for a feature specification is an IOFTS (possibly with an infinite number of states) which compactly represents all possible test cases that can be generated. Rule (3) states that if X and Y are nonempty sets of reachable states from s (under feature restriction φ) with the suspension traces σ and σa, respectively, then there exists a transition of the form $(X,\sigma) \xrightarrow{a}_\varphi (Y,\sigma a)$ in the test suite.

Rules (4) and (5) model, respectively, the successful and the unsuccessful observation of outputs and quiescence. Note that input actions are not included in rules (4) and (5) because the implementation is assumed to be input-enabled [8]; hence, they are only covered in rule (3). Rule (6) states that the verdict states contain self-loop for every output action and quiescence. Our notion of test-suite bears some resemblance to the notion of suspension automaton in [8]; the former is an extension of the latter with verdicts and with feature constraints.

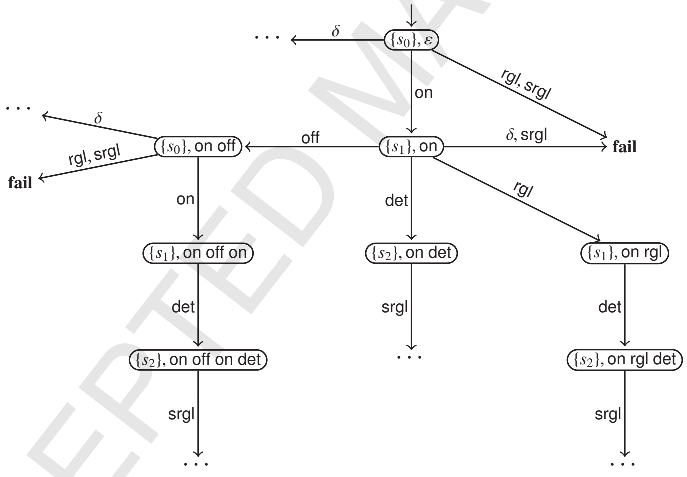

Figure 4: The test suite of the cruise controller example. Note that, for the sake of readablity, only two failed verdict states are drawn.

Example 3. The test suite for the IOFTS of Example 2 is (partially) depicted in Figure 4.

The following properties are immediate from the rules given in Definition 6.

Lemma 1. *If* $(X,\sigma) \overset{\sigma'}{\twoheadrightarrow} (Y,\sigma'')$ *then* $\sigma'' = \sigma\sigma'$.

Proof. By induction on σ'. The base case, when $\sigma' = \varepsilon$, holds trivially. For the induction step, let $\sigma' = \sigma'_1 a$ and $(X,\sigma) \overset{\sigma'_1}{\twoheadrightarrow} (Z,\sigma_1) \xrightarrow{a} (Y,\sigma_2)$ for some $\sigma_1, \sigma'_1, \sigma_2 \in {A_\delta}^*, a \in A_\delta$. Then by the induction hypothesis we have $\sigma_1 = \sigma\sigma'_1$. Moreover, from rule 3 we obtain $\sigma_2 = \sigma_1 a$ which further implies that $\sigma_2 = \sigma_1 a = \sigma(\sigma' a) = \sigma\sigma'$, which was to be shown. □

Lemma 2. *Let* $(\mathbf{X}_0, \varepsilon)$ *be the initial state of the test suite generated from a feature specification* $\Delta_\varphi(s)$*. If* $(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} (X, \sigma)$ *then* $\forall_{s'}\ \Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow} \Delta_\varphi(s') \iff s' \in X$.

Proof. Direct from the construction of intermediate states in Definition 6(1). □

Lemma 3. *Let* $(\mathbf{X}_0, \varepsilon)$ *be the initial state of the test suite generated from a feature specification* $\Delta_\varphi(s)$*. If* $\Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow} \Delta_\varphi(s')$ *for some* s' *then* $\exists_X\ (\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} (X, \sigma) \wedge s' \in X$.

Proof. Direct from the construction of intermediate states in Definition 6(1) because the set $X = (\{s' \mid \Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow} \Delta_\varphi(s')\}, \sigma)$, whenever $\sigma \in \mathrm{Straces}(\Delta_\varphi(s))$. □

Lemma 4. *If* $(X, \sigma) \overset{\sigma'}{\twoheadrightarrow} (Y, \sigma\sigma')$ *and* $(X, \sigma) \overset{\sigma'}{\twoheadrightarrow} (Z, \sigma\sigma')$ *then* $Y = Z$.

Proof. By induction on σ'. The base case, when $\sigma' = \varepsilon$, holds trivially because $X = Y = Z$. For the inductive case, let $\sigma' = \sigma'' a$ (for $\sigma'' \in {A_\delta}^*, a \in A_\delta$), $(X, \sigma) \overset{\sigma'}{\twoheadrightarrow} (Y, \sigma\sigma')$, and $(X, \sigma) \overset{\sigma'}{\twoheadrightarrow} (Z, \sigma\sigma')$. By the induction hypothesis we have $(X, \sigma) \overset{\sigma''}{\twoheadrightarrow} (Y', \sigma\sigma'')$ and $(X, \sigma) \overset{\sigma''}{\twoheadrightarrow} (Z', \sigma\sigma'')$ with $Y' = Z'$ and $(Y', \sigma\sigma'') \overset{a}{\rightarrow} (Y, \sigma\sigma'' a), (Y', \sigma\sigma'') \overset{a}{\rightarrow} (Z, \sigma\sigma'' a)$. Furthermore, from deduction rule 3 in Definition 6, we obtain $Y = Z$. □

Note that our test suites are inherently infinite structures (if the system allowed for infinite interactions) and hence, to obtain the traditional notion of finite test cases, we need to restrict them to a certain depth. Next, we formalise the intuition that a test case is a finite projection of a test suite, plus the restriction that at each moment of time at most one input can be fed into the system under test (cf. [8]).

Definition 7. Given a test suite $\mathcal{T}(s, \varphi)$ with initial state $(\mathbf{X}_0, \varepsilon)$, the set of *test cases of* $\mathcal{T}$ *up depth n*, denoted by $t_n(\mathbf{X}_0, \varepsilon)$, is an IOFTS whose transition relation is the minimal relation satisfying the following two deduction rules:

$$\frac{(X, \sigma) \overset{a}{\rightarrow}_\varphi (Y, \sigma') \quad |\sigma'| < n}{t_n(X, \sigma) \overset{a}{\rightarrow}_\varphi t_n(Y, \sigma')} \quad (7) \qquad \frac{(X, \sigma) \overset{a}{\rightarrow}_\varphi \mathcal{Y} \quad \mathcal{Y} \in \{\mathbf{pass}, \mathbf{fail}\}}{t_n(X, \sigma) \overset{a}{\rightarrow}_\varphi \mathcal{Y}} \quad (8),$$

and the following restrictions due to Tretmans [8]:

1. For any reachable state $\mathcal{X}$ such that $t_n(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \mathcal{X}$, either $\iota(\mathcal{X}) = \{a\} \cup A_O$ (for some $a \in A_I$) or $\iota(\mathcal{X}) = A_O \cup \{\delta\}$, where $\iota(\mathcal{X}) = \{a \mid \exists_{\mathcal{Y}}\ \mathcal{X} \overset{a}{\rightarrow} \mathcal{Y}\}$.
2. For any reachable state $\mathcal{X}$ such that $t_n(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \mathcal{X}$, if $\mathcal{X} \overset{a}{\rightarrow} \mathbf{pass}$ then $\forall_{\mathcal{Y}}\ \mathcal{X} \overset{a}{\rightarrow} \mathcal{Y} \Rightarrow \mathcal{Y} = \mathbf{pass}$.

A *test case* of depth n for a feature specification $\Delta_\varphi(s)$ is $t_n(\mathbf{X}_0, \varepsilon)$, where $(\mathbf{X}_0, \varepsilon)$ is the initial state of the test suite generated from $\Delta_\varphi(s)$.

Example 4. Recall the feature specification $\Delta_\top(s_0)$ from Figure 2. A test case of depth 1 generated from the test suite of the feature specification $\Delta_\varphi(s_1)$ is shown in Figure 5.

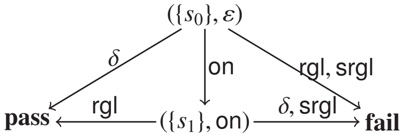

Figure 5: A test case of the cruise controller.

A reader familiar with the original IOCO theory [8] will immediately notice that our definition of a test suite (Definition 6) is nonstandard. In particular, a test suite is defined as a set of test cases (i.e., input-output transition systems with certain restrictions) with a finite number of states in [8]; whereas we represent a test suite by an IOFTS, possibly with an infinite number of states. Nevertheless, we defined a test case to be a finite projection of a test-suite with the additional restriction that at each moment of time at most one input can be fed into the system under test. (Note that inputs to the system are represented as outputs of the test suite / test case and vice versa.) As a result, our test cases are structurally similar to Tretmans' formulation of the test cases, by which we mean that:

- a test case is always deterministic and input enabled (Proposition 2).
- a test case has no cycles except those in the verdict states **pass** and **fail** (Proposition 3).

Another notable difference, which is key to define the concepts of Section 7, is that the states of a test suite (or test case) carry a structure (i.e., denote the sets of reachable states and the trace of actions to reach them), whereas the states of a test case in [8] are abstract and carry no structure.

Proposition 2. *A test case is always deterministic and* $A_O \cup \{\delta\}$*-enabled.*

Proposition 3. *A test case has no cycles except those in the verdict states* **pass** *and* **fail**.

Next, we show that our intensional and extensional notions of testing coincide. To do so, we recall the definition of the *synchronous parallel composition* operator $\rceil|$ that allows for modeling a test run on an implementation (cf. [8]). The synchronous parallel composition operator $\rceil|$ is defined over a test suite and an IOFTS (the implementation under test) as follows. Note that the calligraphic letters $\mathcal{X}, \mathcal{Y}$ in the following rules range over the states of a test suite.

$$\frac{\mathcal{X} \xrightarrow{a} \mathcal{Y} \quad \Delta_\varphi(s) \xrightarrow{a} \Delta_\varphi(s') \quad a \in A}{\mathcal{X}\rceil|\Delta_\varphi(s) \xrightarrow{a}_\top \mathcal{Y}\rceil|\Delta_\varphi(s')} \quad (9) \qquad \frac{\Delta_\varphi(s) \xrightarrow{\tau} \Delta_\varphi(s')}{\mathcal{X}\rceil|\Delta_\varphi(s) \xrightarrow{\tau}_\top \mathcal{X}\rceil|\Delta_\varphi(s')} \quad (10) \qquad \frac{\mathcal{X} \xrightarrow{\delta} \mathcal{Y} \quad \Delta_\varphi(s) \xrightarrow{\delta} \Delta_\varphi(s')}{\mathcal{X}\rceil|\Delta_\varphi(s) \xrightarrow{\delta}_\top \mathcal{Y}\rceil|\Delta_\varphi(s')} \quad (11)$$

By having a notion of running a test suite on a feature specification (representing the behaviour of the implementation under test), we can now define what it means for a feature specification to pass (fail) a test suite. Informally, a test suite is passed by a feature specification if and only if no interaction between the test suite and the feature specification leads to the **fail** verdict state.

Definition 8. Let $(\mathbf{X}_0, \varepsilon)$ be the initial state of the test suite $\mathcal{T}(s, \varphi)$. A feature specification $\Delta_{\varphi'}(s')$ passes the test suite $\mathcal{T}(s, \varphi)$ if and only if

$$\forall_{\sigma \in A_\delta{}^*, s'', \mathcal{X}} \ (\mathbf{X}_0, \varepsilon)\rceil|\Delta_{\varphi'}(s') \overset{\sigma}{\twoheadrightarrow} \mathcal{X}\rceil|\Delta_{\varphi'}(s'') \Rightarrow \mathcal{X} \neq \mathbf{fail}$$

Next we prove that the intensional and the extensional characterization of the $\sqsubseteq_{\mathbf{ioco}}$ relation coincide, i.e., $\sqsubseteq_{\mathbf{ioco}}$ can always be checked by means of the generated test suite.

Theorem 2. *A feature specification* $\Delta_{\varphi'}(s')$ *passes the test suite* $\mathcal{T}(s, \varphi)$ *if and only if* $\Delta_{\varphi'}(s') \sqsubseteq_{\mathbf{ioco}} \Delta_\varphi(s)$.

Proof. ($\Leftarrow$) Suppose the feature specification $\Delta_{\varphi'}(s')$ passes the test suite $\mathcal{T}(s, \varphi)$ whose initial state is $(\mathbf{X}_0, \varepsilon)$ and $\Delta_\varphi(s) \not\sqsubseteq_{\mathbf{ioco}} \Delta_{\varphi'}(s')$. Then, for some suspension trace $\sigma \in \mathrm{Straces}(\Delta_\varphi(s))$ and $a \in A_O \cup \{\delta\}$ we have

$$a \in \mathrm{out}(\mathrm{Reach}(\Delta_{\varphi'}(s'), \sigma)) \quad \text{and} \quad a \notin \mathrm{out}(\mathrm{Reach}(\Delta_\varphi(s), \sigma)).$$

Thus, $\exists_{s''}$ $(\mathbf{X}_0, \varepsilon)\rceil|\Delta_{\varphi'}(s') \overset{\sigma a}{\twoheadrightarrow} \mathbf{fail}\rceil|\Delta_{\varphi'}(s'')$. But, $\Delta_{\varphi'}(s')$ passes the test suite; hence, a contradiction follows.

($\Rightarrow$) Suppose $\Delta_{\varphi'}(s') \sqsubseteq_{\mathbf{ioco}} \Delta_\varphi(s)$. Then we prove by contradiction that the feature specification $\Delta_{\varphi'}(s')$ passes the test suite $\mathcal{T}(s, \varphi)$ whose initial state is $(\mathbf{X}_0, \varepsilon)$. Without loss of generality, let $(\mathbf{X}_0, \varepsilon)\rceil|\Delta_{\varphi'}(s') \overset{\sigma}{\twoheadrightarrow} (X, \sigma)\rceil|\Delta_{\varphi'}(s'_1) \xrightarrow{a} \mathbf{fail}\rceil|\Delta_{\varphi'}(s'_2)$, for some $X, \sigma, s'_1, s'_2, a \in A_O \cup \{\delta\}$. From Lemma 2 we have $\sigma \in \mathrm{Straces}(\Delta_\varphi(s))$. Furthermore, from the semantics of a test suite and by the definition of reachability relation we have $a \notin \mathrm{out}(\mathrm{Reach}(\Delta_\varphi(s), \sigma))$ and $a \in \mathrm{out}(\mathrm{Reach}(\Delta_{\varphi'}(s'), \sigma))$, respectively. Thus, $\Delta_{\varphi'}(s') \not\sqsubseteq_{\mathbf{ioco}} \Delta_\varphi(s)$, which leads to contradiction. □

We end this section by giving an application of the above theorem.

Example 5. Recall the feature specification $\Delta_\varphi(s)$ of the cruise controller from Example 2. Consider a faulty implementation of the cruise controller as shown in Figure 6, where all the transitions are labelled with feature constraint $\top$. Note that this implementation is faulty because $\delta \in \mathrm{out}(\mathrm{Reach}(t), \mathsf{on})$ whereas $\delta \notin \mathrm{out}(\mathrm{Reach}(\Delta_\varphi(s)), \mathsf{on})$. Then, Theorem 2 suggests that such an information can be inferred by interacting the faulty implementation with the test suite of the feature specification. In particular, when we compose the faulty implementation in Figure 6 in parallel with the test suite depicted in Figure 4, the trace $(\{s_0\}, \varepsilon)\rceil|t \xrightarrow{\mathsf{on}} (\{s_1\}, \mathsf{on})\rceil|t' \xrightarrow{\delta} \mathbf{fail}\rceil|t'$ leads to **fail** verdict state.

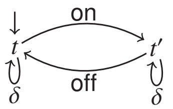

Figure 6: A faulty implementation of the cruise controller.

5. Refinement of test suites

In this section, we define the notion of refinement on test suites, to project them into more specific product sublines and eventually into products. As the main result of this section, we show that the two notions of refinements (the one on IOFTS as models defined in Section 2 and the other defined in this section) are consistent. More precisely, we show that restricting a test suite of the feature specification $\Delta_\varphi(s)$ by a feature constraint φ' is isomorphic to the test suite of the feature specification $\Delta_{\varphi\wedge\varphi'}(s)$.

Definition 9. Two states $\mathcal{X}$ and $\mathcal{Y}$ are *isomorphic*, denoted $\mathcal{X} \cong \mathcal{Y}$, if there exists a bijection $f : \text{Reach}(\mathcal{X}) \rightarrow \text{Reach}(\mathcal{Y})$ such that f preserves the transition structure, i.e.,

$$\forall_{\mathcal{X}_1,\mathcal{X}_2 \in \text{Reach}(\mathcal{X}),a} \;\; \mathcal{X}_1 \xrightarrow{a} \mathcal{X}_2 \;\Leftrightarrow\; f(\mathcal{X}_1) \xrightarrow{a} f(\mathcal{X}_2).$$

Next, we introduce the projection operator $\Delta^{\mathbf{t}}_{\varphi}$ that restricts the behaviour of the test suite of the feature specification $\Delta_\varphi(s)$ by φ'.

Definition 10. Let $(\mathbf{X} \cup \{\mathbf{pass}, \mathbf{fail}\}, (\mathbf{X}_0, \varepsilon), A_\delta, F, T, \Lambda)$ be the test suite generated from a feature specification $\Delta_\varphi(s)$. For a feature constraint φ', the *test-projection operator* $\Delta^{\mathbf{t}}_{\varphi'}(\_)$ induces an IOFTS

$$(\{\Delta^{\mathbf{t}}_{\varphi'}(x) \mid x \in \mathbf{X}\} \cup \{\mathbf{pass}, \mathbf{fail}\}, \Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon), A_\delta, F, T', \Lambda'),$$

where the transition relation T' is defined as the smallest relation satisfying the following rules.

$$\frac{\begin{array}{c}(X,\sigma) \xrightarrow{a}_{\varphi} (Y,\sigma') \\ \exists_\lambda\, (\lambda \in \Lambda \wedge \lambda \models \varphi')\end{array}}{\Delta^{\mathbf{t}}_{\varphi'}(X,\sigma) \xrightarrow{a} \Delta^{\mathbf{t}}_{\varphi'}(Y,\sigma')} \quad (12) \qquad\qquad \frac{\begin{array}{c}a \in A_O \cup \{\delta\} \\ \Delta^{\mathbf{t}}_{\varphi'}(X,\sigma) \xrightarrow{a}_{\varphi} \Delta^{\mathbf{t}}_{\varphi'}(Y,\sigma')\end{array}}{\Delta^{\mathbf{t}}_{\varphi'}(X,\sigma) \xrightarrow{a} \mathbf{pass}} \quad (13)$$

$$\frac{\begin{array}{c}a \in A_O \cup \{\delta\} \\ \neg\left(\Delta^{\mathbf{t}}_{\varphi'}(X,\sigma) \xrightarrow{a}_{\varphi} \mathbf{pass}\right)\end{array}}{\Delta^{\mathbf{t}}_{\varphi'}(X,\sigma) \xrightarrow{a} \mathbf{fail}} \quad (14) \qquad\qquad \frac{a \in A_O \cup \{\delta\}}{\begin{array}{c}\mathbf{pass} \xrightarrow{a}_{\varphi} \mathbf{pass} \\ \mathbf{fail} \xrightarrow{a}_{\varphi} \mathbf{fail}\end{array}} \quad (15)$$

The component Λ' is defined as $\Lambda' = \{\lambda \in \Lambda \mid \lambda \models \varphi'\}$.

Note that, similar to Definition 3, the notation $\Delta^{\mathbf{t}}_{\varphi'}(x)$ in the set comprehension in Definition 10, is used to give a new name to the states of the refined test suite and does not have any semantical connotation.

Intuitively, rule (12) states that if an a-transition can be executed in the test suite for the specification $\Delta_\varphi(s)$ (i.e., $(X,\sigma) \xrightarrow{a} (Y,\sigma a)$) and there exists a product configuration in the test suite that satisfies φ' then the a-transition can be executed in the restricted test suite. Rules (13) and (14) model the successful and the unsuccessful observations of outputs and quiescence, respectively.

The remainder of this section is devoted to proving the main result (see Figure 7) of this section which states that restricting a test suite leads to an isomorphic test suite by restricting a feature specification.

Theorem 3. *Let* $(\mathbf{X}_0, \varepsilon)$ *and* $(\mathbf{X}'_0, \varepsilon)$ *be the initial states of the test suites* $\mathcal{T}(s,\varphi)$ *and* $\mathcal{T}(s, \varphi \wedge \varphi')$*, respectively. Then,* $\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon) \cong (\mathbf{X}'_0, \varepsilon)$.

$$\begin{tikzcd}
\Delta_{\varphi}(s) \arrow[r, "\text{test suite generation}"] \arrow[d, "\Delta_{\varphi'}(\_)"'] & (\mathbf{X}_0, \varepsilon) \arrow[d, "\Delta^{\mathbf{t}}_{\varphi'}(\_)"] \\
\Delta_{\varphi \wedge \varphi'}(s) \arrow[r, "\text{test suite generation}"'] & (\mathbf{X}'_0, \varepsilon) \cong \Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon)
\end{tikzcd}$$

Figure 7: An illustration of Theorem 3

The main idea is to construct a bijection between the reachable states of the two test suites such that it preserves the transition structure. Consider the following definition of a mapping $f : \text{Reach}(\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon)) \to \text{Reach}(\mathbf{X}'_0, \varepsilon)$ as a candidate for the isomorphism:

$$\begin{aligned} f(\Delta^{\mathbf{t}}_{\varphi'}(X, \sigma)) &= (Y, \sigma) \quad \text{if} \quad \Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \Delta^{\mathbf{t}}_{\varphi'}(X, \sigma) \wedge (\mathbf{X}'_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} (Y, \sigma); \\ f(\mathbf{pass}) &= \mathbf{pass}; \\ f(\mathbf{fail}) &= \mathbf{fail}. \end{aligned} \tag{1}$$

In the following, through a series of lemmas, we prove some properties on the restriction of test suite of the specification $\Delta_{\varphi}(s)$ under φ' that ensures the above mapping is a bijection.

Lemma 5. *The mapping f defined in* (1) *is a function.*

Proof. Direct from Lemma 4. □

Lemma 6, which is similar to Lemma 4, states that a unique state is always reachable for every trace in the restricted test suite. This lemma is required to show that the function f is indeed injective.

Lemma 6. *Let $(\mathbf{X}_0, \varepsilon)$ be the initial state of the test suites $\mathcal{T}(s, \varphi)$. Then,*

$$\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \Delta^{\mathbf{t}}_{\varphi'}(Y, \sigma) \wedge \Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \Delta^{\mathbf{t}}_{\varphi'}(Z, \sigma) \;\Rightarrow\; Y = Z.$$

Proof. Direct from Lemma 4. □

Lemma 7 states that any reachable state in the test suite of the specification $\Delta_{\varphi \wedge \varphi'}(s)$ is a subset of a reachable state in the restricted test suite (see Figure 8 for an illustration, where the subset relationship is indicated by a partition). Lemma 7 together with Lemma 4 ensure that the function f defined in (1) is indeed surjective.

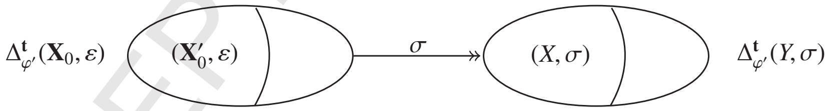
$\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon)$
$(\mathbf{X}'_0, \varepsilon)$
σ
(X, σ)
$\Delta^{\mathbf{t}}_{\varphi'}(Y, \sigma)$

Figure 8: An illustration of Lemma 7, where $\mathbf{X}_0$ and $\mathbf{X}'_0$ are the initial states of the test suites generated from $\Delta_{\varphi}(s)$ and $\Delta_{\varphi \wedge \varphi'}(s)$, respectively.

Lemma 7. *Let $(\mathbf{X}_0, \varepsilon)$ and $(\mathbf{X}'_0, \varepsilon)$ be the initial states of the test suites $\mathcal{T}(s, \varphi)$ and $\mathcal{T}(s, \varphi \wedge \varphi')$, respectively. If $(\mathbf{X}'_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} (X, \sigma)$ then $\exists_Y\, \Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \Delta^{\mathbf{t}}_{\varphi'}(Y, \sigma) \wedge X \subseteq Y$.*

Proof. We prove this lemma by induction on σ. We identify the following cases:

1. Let $\sigma = \varepsilon$. We need to show that $\mathbf{X}'_0 \subseteq \mathbf{X}_0$.

$$\begin{aligned} & s' \in \mathbf{X}'_0 && \text{(Assumption)} \\ & \Rightarrow \Delta_{\varphi \wedge \varphi'}(s) \overset{\varepsilon}{\twoheadrightarrow} \Delta_{\varphi \wedge \varphi'}(s') && \text{(Lemma 2)} \\ & \Rightarrow \Delta_{\varphi}(s) \overset{\varepsilon}{\twoheadrightarrow} \Delta_{\varphi}(s') && \text{(Proposition 1)} \\ & \Rightarrow s' \in \mathbf{X}_0 && \text{(Lemma 2)}\ . \end{aligned}$$

2. Let $\sigma \neq \varepsilon$. Suppose $(\mathbf{X}'_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} (X, \sigma) \xrightarrow{a} (X', \sigma a)$. By the induction hypothesis we have

$$\exists_Y\, \Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \Delta^{\mathbf{t}}_{\varphi'}(Y, \sigma) \wedge X \subseteq Y.$$

Furthermore, by construction of sets X, X' we have

$$\begin{array}{ll} \exists_{s_1 \in X, s_2 \in X'} \Delta_{\varphi \wedge \varphi'}(s_1) \xrightarrow{a} \Delta_{\varphi \wedge \varphi'}(s_2) & \\ \Rightarrow s_1 \in Y \wedge \Delta_{\varphi}(s_1) \xrightarrow{a} \Delta_{\varphi}(s_2) & (X \subseteq Y \text{ and Proposition 1}) \\ \Rightarrow \exists_{Y'}\ (Y, \sigma') \xrightarrow{a} (Y', \sigma' a) \wedge s_2 \in Y' & (\text{Lemma 3}) \\ \Rightarrow \Delta^{\mathbf{t}}_{\varphi'}(Y, \sigma') \xrightarrow{a} \Delta^{\mathbf{t}}_{\varphi'}(Y', \sigma' a) & (12). \end{array}$$

Next, we need to show that $X' \subseteq Y'$. Let $s'_2 \in X'$, for some $s'_2 \in S$. Then there is a transition $\Delta_{\varphi \wedge \varphi'}(s_1) \xrightarrow{a} \Delta_{\varphi \wedge \varphi'}(s'_2)$, for some $s_1 \in X$. And from Proposition 1 we get $\Delta_{\varphi}(s_1) \xrightarrow{a} \Delta_{\varphi}(s'_2)$. But, we know that $X \subseteq Y$ and from Lemma 2 we get $s'_2 \in Y'$; hence, $X' \subseteq Y'$. □

Lastly, the following lemma states that a trace in the test suite $\mathcal{T}(s, \varphi)$ when restricted under φ' is a suspension trace of the specification $\Delta_{\varphi \wedge \varphi'}(s)$.

Lemma 8. *Let $(\mathbf{X}_0, \varepsilon)$ be the initial state of the test suite $\mathcal{T}(s, \varphi)$. If $\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \Delta^{\mathbf{t}}_{\varphi'}(X, \sigma)$ then $\sigma \in Straces(\Delta_{\varphi \wedge \varphi'}(s))$.*

Proof. Suppose $\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \Delta^{\mathbf{t}}_{\varphi'}(X, \sigma)$. Then by construction of X we have $\exists_{s' \in X}\ \Delta_{\varphi \wedge \varphi'}(s) \overset{\sigma}{\twoheadrightarrow} \Delta_{\varphi \wedge \varphi'}(s')$. Thus, $\sigma \in \text{Straces}(\Delta_{\varphi \wedge \varphi'}(s))$. □

Proof of Theorem 3. Recall the mapping f from (1). Clearly, Lemmas 4, 6, and 7 ensure that f is a bijection from $\text{Reach}(\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon))$ to $\text{Reach}(\mathbf{X}'_0, \varepsilon)$. Thus, it remains to be shown that f preserves the transition structure. Let $\mathcal{X} \xrightarrow{a} \mathcal{Y}$, for some $\mathcal{X}, \mathcal{Y} \in \text{Reach}(\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon))$. The case when $\mathcal{X}$ is either **pass** or **fail** is trivial. Hence, the interesting case is when $\mathcal{X} = \Delta^{\mathbf{t}}_{\varphi'}(X, \sigma)$. We further distinguish the following cases:

1. Let $\mathcal{Y} = \Delta^{\mathbf{t}}_{\varphi'}(Y, \sigma')$. Then, from Lemma 8 we know that $\sigma' \in \text{Straces}(\Delta_{\varphi \wedge \varphi'}(s))$; thus, there exists Y' such that $(\mathbf{X}'_0, \varepsilon) \overset{\sigma'}{\twoheadrightarrow} (Y', \sigma')$. Hence, $f(\mathcal{Y}) = (Y', \sigma')$. For the converse, suppose $f(\mathcal{X}) \xrightarrow{a} (Y', \sigma')$, for some $(Y', \sigma') \in \text{Reach}(\mathbf{X}_0, \varepsilon)$. Using Lemmas 6 and 7 we have $f(\mathcal{Y}) = (Y', \sigma')$, for some $\mathcal{Y} \in \text{Reach}(\mathbf{X}_0, \varepsilon)$.
2. Let $\mathcal{Y} = \mathbf{pass}$. Then,

$$\begin{array}{ll} \mathcal{X} \xrightarrow{a} \mathbf{pass} & \\ \Leftrightarrow \exists_{Y, \sigma'} \mathcal{X} \xrightarrow{a} \Delta^{\mathbf{t}}_{\varphi'}(Y, \sigma') & (\text{rule (13)}) \\ \Leftrightarrow f(\mathcal{X}) \xrightarrow{a} f(\Delta^{\mathbf{t}}_{\varphi'}(Y, \sigma')) & (\text{Case 1}) \\ \Leftrightarrow f(\mathcal{X}) \xrightarrow{a} \mathbf{pass} & (\text{rule (4)}). \end{array}$$

3. Let $\mathcal{Y} = \mathbf{fail}$. Suppose otherwise $f(\mathcal{X}) \xrightarrow{a} \mathbf{pass}$. Then, from rule (4) we know that there exist Y', σ' such that $f(\mathcal{X}) \xrightarrow{a} (Y', \sigma')$. And by Lemma 7 we have $\exists_Y\, \mathcal{X} \xrightarrow{a} (Y, \sigma)$. But, $\mathcal{X} \xrightarrow{a} \mathbf{fail}$; hence, a contradiction follows.
   For the converse, suppose $\mathcal{X} \xrightarrow{a} \mathbf{pass}$ and $f(\mathcal{X}) \xrightarrow{a} \mathbf{fail}$. Then, from rule (13) we know that there exist Y, σ' such that $\mathcal{X} \xrightarrow{a} \Delta^{\mathbf{t}}_{\varphi'}(Y, \sigma')$. And from Case 1 we know that $f(\mathcal{X}) \xrightarrow{a} f(Y, \sigma')$, which again leads to a contradiction because $f(\mathcal{X}) \xrightarrow{a} \mathbf{fail}$. □

Corollary 2. *Let $(\mathbf{X}_0, \varepsilon)$ be the initial state of the test suite $\mathcal{T}(s, \varphi)$. If $(\mathbf{X}_0, \varepsilon)\rceil|\Delta_{\varphi''}(s') \overset{\sigma}{\twoheadrightarrow} \mathbf{fail}\rceil|\Delta_{\varphi''}(s')$ then, for every φ', we have*

$$\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon)\rceil|\Delta_{\varphi''}(s') \overset{\sigma}{\twoheadrightarrow} \mathbf{fail}\rceil|\Delta_{\varphi''}(s').$$

Proof. The result follows directly from the fact that $\Delta^{\mathbf{t}}_{\varphi'}(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \mathbf{fail}$, whenever $(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} \mathbf{fail}$. □

6. Residual test suites

As mentioned in the introduction, one of the challenges in testing a software product line is to minimise the test effort. The idea pursued in this section and the next one is to organise the test process of a product line incrementally. This is achieved by reusing the test results of an already tested product to test a product with similar features, thereby dispensing with the test cases targeted at the common features. To this end, we introduce the notion of *residual test suite*, which prunes away the behaviour of a specified set of features from an *abstract* test suite $\mathcal{T}(s, \varphi)$ with respect to a *concrete* test suite $\mathcal{T}(s, \lambda)$ of the already tested product λ. We begin with the definition of the predicate $\mathbf{new}_\lambda(\sigma, a)$ asserts whether there is an a-transition after the suspension trace σ that is "new" with respect to the tested product λ. Formally,

$$\mathbf{new}_\lambda(\sigma, a) \Leftrightarrow \sigma \in \text{Straces}(\Delta_\lambda(s)) \wedge \exists_{s',s''}\ \Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow} \Delta_\varphi(s') \xrightarrow{a}_{\varphi'} \Delta_\varphi(s'') \wedge \lambda \not\models \varphi'.$$
$$\mathbf{new}_\lambda(\sigma) \Leftrightarrow \exists_{a \in A_\delta}\ \mathbf{new}_\lambda(\sigma, a).$$

As an example, consider the product λ which enables the basic feature of cruise controller and disables the optional feature of collision avoidance controller, i.e., $\lambda(\mathsf{cc}) = \top$ and $\lambda(\mathsf{cac}) = \bot$. Then, the predicate $\mathbf{new}_\lambda(\mathsf{on}, \mathsf{det})$ holds for the feature specification given in Figure 2, because from the state the event det is enabled, whose feature constraint is not satisfied by λ. In other words, after the suspension trace on of the feature specification, some new behaviour can emerge with respect to the product specified by λ.

Now we are ready to formally define a residual test suite.

Definition 11. Let $(\mathbf{X} \cup \{\mathbf{pass}, \mathbf{fail}\}, (\mathbf{X}_0, \varepsilon), A_\delta, F, T, \Lambda)$ be the test suite generated from a feature specification $\Delta_\varphi(s)$ and let λ be a product such that $\lambda \models \varphi$. Then a *residual test suite* with respect to λ, denoted by $\mathcal{R}_\lambda(s, \varphi)$, is an IOFTS $(\mathbf{X}' \cup \{\mathbf{pass}, \mathbf{fail}\}, (\mathbf{X}_0, \varepsilon), A_\delta, F, T', \Lambda')$, where

1. The set of non-verdict states $\mathbf{X}'$ is defined as the smallest set satisfying the following conditions:
   (a) If $(X, \sigma) \in \mathbf{X} \wedge \mathbf{new}_\lambda(\sigma)$ then $(X, \sigma) \in \mathbf{X}'$.
   (b) If $(X, \sigma) \in \mathbf{X}'$ and $(Y, \sigma') \in \text{Reach}(X, \sigma)$ then $(Y, \sigma') \in \mathbf{X}'$.
   (c) If $(Y, \sigma') \in \mathbf{X}'$ and $(Y, \sigma') \in \text{Reach}(X, \sigma)$ then $(X, \sigma) \in \mathbf{X}'$.
2. The set of transition relations T' is defined as
$$T' = \{(\mathcal{X}, a, \mathcal{Y}) \in T \mid \mathcal{X}, \mathcal{Y} \in \mathbf{X}\}.$$
3. The set of product configurations $\Lambda' = \Lambda \setminus \{\lambda\}$.

Intuitively, condition 1(a) asserts that if a state of the given test suite has new behaviour with respect to the product λ then this state is also a state of a residual test suite. Condition 1(b) asserts that all states that are reachable from a state with new behaviour (w.r.t. λ) are also the states of a residual test suite. Lastly, condition 1(c) asserts that if a state ($(X, \sigma) \in \mathbf{X}$) of the given test suite leads to a state that has new behaviour (w.r.t. λ) then the state (X, σ) is also a state of a residual test suite. (Note that due to tree structure of test-suites, the backward path from any new state to the initial state is unique.) Next, we define the notion of residual test case, which exploits a residual test suite in order to test the new features.

Definition 12. A *residual test case* of $\mathcal{R}_\lambda(s, \varphi)$ is any finite projection of a residual test-suite satisfying the following conditions:

1. from each state, there is at most one outgoing input transition,
2. all leaves are labeled either **pass** or **fail**, and
3. from every non-leave state, there is a state (X, σ) reachable such that $\mathbf{new}_\lambda(\sigma)$ holds.

Unfortunately, with the notion of residual test suite there is little gain in discarding the 'common' transitions. For instance, the residual test suite does not allow to prune any transition from the original test suite (Figure 4) of the cruise controller specification. However, using an example, we explain when residual test suites actually removes some transitions from a given test suite. Consider the feature specification $\Delta_\top(s_1)$ drawn in Figure 9 and two products

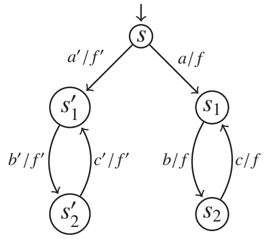

Figure 9: An example illustrating when residual test suites prunes away transitions from the original test suite.

λ, λ' defined as $\lambda(f) = \top, \lambda(f') = \bot$ and $\lambda'(f) = \bot, \lambda'(f') = \top$. Now while constructing the residual test suite $\mathcal{R}'_\lambda(s, \top)$ we note that all of the paths labelled with $a', a'b'c', a'b'c'b', \cdots$ will be pruned from the original test suite $\mathcal{T}(s, \top)$.

Thus, in hindsight, a residual test suite $\mathcal{R}_\lambda(s, \varphi)$ prunes only those paths in the test suite $\mathcal{T}(s, \varphi)$ that do not lead to any new behaviour with respect to an already tested product λ. Therefore, in the next section, we explore a notion of spinal test suite in which it is possible to prune more behaviour than a residual test suite. We end this section by showing that our notion of residual testing is complete (Theorem 4), i.e., a concrete test suite of a tested product λ together with a residual test suite $\mathcal{R}_\lambda(s, \varphi)$ has the same testing power as the test suite $\mathcal{T}(s, \varphi)$.

Theorem 4. *If a feature specification $\Delta_{\varphi'}(s')$ passes the test suites $\mathcal{T}(s, \lambda)$ and $\mathcal{R}_\lambda(s, \varphi)$ with $\lambda \models \varphi$, then $\Delta_{\varphi'}(s')$ passes the test suite $\mathcal{T}(s, \varphi)$.*

Proof. We will prove this theorem by contradiction. Let $\Delta_{\varphi'}(s')$ pass the test suites $\mathcal{T}(s, \lambda)$ and $\mathcal{R}_\lambda(s, \varphi)$, whose initial states are $(\mathbf{X}'_0, \varepsilon)$ and $(\mathbf{X}_0, \varepsilon)$, respectively. Suppose $\Delta_{\varphi'}(s')$ fails in passing the test suite $\mathcal{T}(s, \varphi)$. Then, there exists the following sequences of transitions $(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow}$ **fail** and $\Delta_{\varphi'}(s') \overset{\sigma}{\twoheadrightarrow} \Delta_{\varphi'}(s'')$ (for some σ, s'') in the test suite $\mathcal{T}(s, \varphi)$ and the feature specification $\Delta_{\varphi'}(s')$. (Note that the initial states of the test suite $\mathcal{T}(s, \varphi)$ and $\mathcal{R}_\lambda(s, \varphi)$ are identical by construction.) There are two possibilities:

1. Either $\sigma \in \text{Straces}(\Delta_\lambda(s))$, then there is a path $(\mathbf{X}'_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} (X, \sigma)$, for some X, in the test suite $\mathcal{T}(s, \lambda)$. Since $\lambda \models \varphi$ then there is a path $(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow} (Y, \sigma)$, for some Y, in the test suite $\mathcal{T}(s, \varphi)$. But this contradicts the above-mentioned transition $(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow}$ **fail**.
2. Or $\sigma \notin \text{Straces}(\Delta_\lambda(s))$, then the sequence of transitions $(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow}$ **fail** can be decomposed in the following way: $(\mathbf{X}_0, \sigma) \overset{\sigma'}{\twoheadrightarrow} (X, \sigma') \overset{\sigma''}{\twoheadrightarrow}$ **fail** with $\sigma = \sigma'\sigma''$ and $\mathbf{new}_\lambda(\sigma')$. Thus, $\mathbf{X}_0 \overset{\sigma'\sigma''}{\twoheadrightarrow}$ **fail** is a transition in the residual test suite $\mathcal{R}_\lambda(s, \varphi)$ and the feature specification $\Delta_{\varphi'}(s')$ fails to pass the residual test suite $\mathcal{R}_\lambda(s, \varphi)$; hence, a contradiction follows. □

7. Spinal test suites

In the previous section, we pruned test suites *by allowing only those reachable states in the abstract test suite from which a new behaviour relative to the already tested product emanates*. However, we noticed there that, despite the completeness result (Theorem 4), such a strategy does not result in any considerable saving in the test effort.

For example, consider the test suite depicted in Figure 4 and suppose we have already tested the cruise controller without collision avoidance feature and now are interested in the correct implementation of the collision avoidance feature. By following the aforementioned strategy of pruning, none of the following states $(\{s_1\}, \mathsf{on}), (\{s_1\}, \mathsf{on\ off\ on}), \cdots$ will be removed because event det is enabled from each of these states. On the other hand, since we know that cruise controller without collision avoidance feature was already tested, it is safe to consider the new suspension traces (or testing experiments) from only one state in $\{(\{s_1\}, \mathsf{on}), (\{s_1\}, \mathsf{on\ off\ on}), \cdots\}$.

Definition 13. Let $\mathbf{X}_0$ be the initial state of a test suite $\mathcal{T}(s, \varphi)$. An execution $\mathbf{X}_0 \overset{\sigma}{\twoheadrightarrow} (X, \sigma)$ is a *spine* of an execution $\mathbf{X}_0 \overset{\sigma'}{\twoheadrightarrow} (X, \sigma')$, denoted by $\sigma \dagger \sigma'$, when σ is a sub-trace of σ' (obtained by removing zero or more action from σ')

and no two states visited in the former execution (during the trace σ) have the same X-component; this is formalised by the predicate $\mathbf{bt}(X,\sigma)$, defined below:

$$\forall_{\sigma_1,\sigma_2,\sigma_3,Y,Z}\ (\mathbf{X}_0 \xrightarrow{\sigma_1} (Y,\sigma_1) \xrightarrow{\sigma_2} (Z,\sigma_2) \xrightarrow{\sigma_3} (X,\sigma) \wedge \sigma_2 \neq \varepsilon \wedge \sigma = \sigma_1\sigma_2\sigma_3) \Rightarrow Y \neq Z.$$

Furthermore, we let $\mathbf{bt}(\mathbf{X}_0) = \top$.

Example 6. Recall the feature specification given $\Delta_\varphi(s_0)$ in Example 2, where $\varphi = \mathsf{cc} \wedge \neg\mathsf{cac}$. Since collision avoidance controller is an optional feature, we know that there exists a product configuration λ with $\lambda(\mathsf{cc}) = \top$ and $\lambda(\mathsf{cac}) = \bot$. Then, the execution labelled "on" (in the test suite drawn in Figure 4) is a spine of the execution labelled "on off on" because they both reach to a common X-component $\{s_1\}$ in the test suite and $\mathbf{bt}(\{s_1\}, \mathsf{on}) = \top$.

Definition 14. Let $(\mathbf{X} \cup \{\mathbf{pass}, \mathbf{fail}\}, \mathbf{X}_0, A_\delta, F, T, \Lambda)$ be a test suite $\mathcal{T}(s,\varphi)$ and let λ be a product such that $\lambda \models \varphi$. Then a *spinal test suite* with respect to a product λ, denoted by $\mathcal{S}(\varphi,\lambda)$, is an IOFTS $(\mathbf{X}' \cup \{\mathbf{pass}, \mathbf{fail}\}, \mathbf{X}_0, A_\delta, F, T', \Lambda')$, where

1. The set of non-verdict states $\mathbf{X}'$ is defined as $\mathbf{X}'_1 \cup \mathbf{X}'_2$, where

$$\mathbf{X}'_1 = \{(X,\sigma) \in \mathbf{X} \mid \sigma \in \mathrm{Straces}(\Delta_\lambda(s)) \wedge \mathbf{bt}(X,\sigma)\}$$
$$\mathbf{X}'_2 = \{(Y,\sigma a \sigma') \in \mathbf{X} \mid \mathbf{new}_\lambda(\sigma, a) \wedge \exists_X\ (X,\sigma) \in \mathbf{X}'_1 \wedge (X,\sigma) \xrightarrow{a\sigma'} (Y, \sigma a \sigma')\}.$$

2. The set of transition relations T' is defined as

$$T' = \{(\mathcal{X}, a, \mathcal{Y}) \in T \mid \mathcal{X}, \mathcal{Y} \in \mathbf{X}'\}.$$

3. The set of product configurations $\Lambda' = \Lambda \setminus \{\lambda\}$.

Intuitively, Condition 1 defines $\mathbf{X}'$ to be a set of non-verdict states of the form (X,σ) such that σ is a suspension trace of the already tested product $\Delta_\lambda(s)$ and the predicate $\mathbf{bt}(X,\sigma)$ holds; whereas, $\mathbf{X}''$ is the set of non-verdict states reachable from a state in $\mathbf{X}'$ by a trace that is not a suspension trace of the tested product $\Delta_\lambda(s)$. Condition 2 and 3 are self-explanatory.

As an example, the spinal test suite generated from the test suite in Figure 4 is partially drawn in Figure 10.

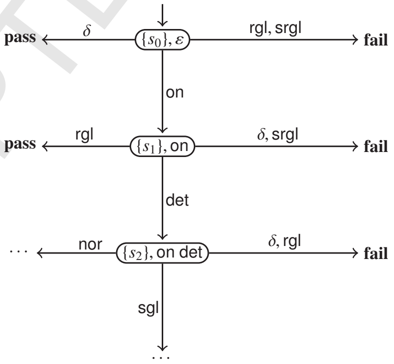

Figure 10: Spinal test suite of the cruise controller

The spinal test suite $\mathcal{S}(\varphi,\lambda)$ contains the spines of those executions from the test suite $\mathcal{T}(s,\varphi)$ that lead to new behaviour w.r.t. to the already-tested product λ. Next, we show that the spinal test suite $\mathcal{S}(\varphi,\lambda)$ is not necessarily exhaustive for an arbitrary implementation under test, i.e., it may have *strictly* less testing power than the test suite $\mathcal{T}(s,\varphi)$. We exemplify this through the following example.

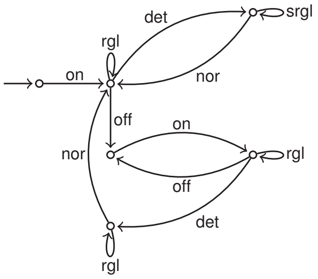

Figure 11: A faulty implementation of the cruise controller with control avoidance.

Example 7. Consider an implementation of a cruise controller with a collision avoidance feature modeled as the IOFTS depicted in Figure 11. Clearly, this implementation is a faulty one as the action ‘rgl’ must be prohibited after detecting an obstacle, i.e., after executing the transition labeled ‘det’.

As soon as we place the test suite (Figure 4) in parallel ($\rceil\!|$) with the above-given implementation, we observe that the following synchronous interactions emerge: on.off.on.det.rgl, which lead to the **fail** verdict state. However, note that the aforementioned fault in the implementation cannot be detected while interacting with the spinal test suite of Figure 10, because there are no transitions labeled with off in the spinal test suite. Thus, a spinal test suite $\mathcal{S}(\varphi, \lambda)$ has strictly less testing power than the test suite $\mathcal{T}(s, \varphi)$.

Next, we explore when a spinal test suite $\mathcal{S}(\varphi, \lambda)$ (where $\lambda \models \varphi$) together with a concrete test suite $\mathcal{T}(s, \lambda)$ have the same testing power as the abstract test suite $\mathcal{T}(s, \varphi)$.

Definition 15. Let $\lambda \models \varphi$. A feature specification $\Delta_{\varphi'}(s')$ is *orthogonal* w.r.t $\Delta_\varphi(s)$ and the product λ iff

$$\forall_{s_1,\sigma',a,\sigma''}\ \left(\mathbf{new}_\lambda(\sigma', a) \wedge \Delta_{\varphi'}(s') \xrightarrow{\sigma' a \sigma''} \Delta_{\varphi'}(s_1)\right) \Rightarrow \exists_{s_2,\sigma}\ \Delta_{\varphi'}(s') \xrightarrow{\sigma a \sigma''} \Delta_{\varphi'}(s_2) \wedge \sigma \dagger \sigma'.$$

Example 8. Recall the feature specification $\Delta_\varphi(s_0)$ and the product λ (which omits the control avoidance feature) from Example 6. Note that the implementation given in Figure 11 is not orthogonal w.r.t the feature specification $\Delta_\varphi(s_0)$ and the product λ because the underlined subsequence in “on off on $\underline{\text{det rgl}}$” cannot be extended with the spine sequence on.

In the remainder, we prove the main result (Theorem 5) of this section that an orthogonal implementation passes the test suite $\mathcal{T}(s, \varphi)$ whenever it passes the concrete test suite $\mathcal{T}(s, \lambda)$ and the spinal test suite $\mathcal{S}(\varphi, \lambda)$.

Lemma 9. *Let* $(\mathbf{X}_0, \varepsilon)$ *be the initial state of a test suite* $\mathcal{T}(s, \varphi)$ *and let* λ *be a product with* $\lambda \models \varphi$. *If* $(\mathbf{X}_0, \varepsilon) \xrightarrow{\sigma' a \sigma''}$ **fail**, $\mathbf{new}_\lambda(\sigma', a)$, *and* $\sigma \dagger \sigma'$ *then* $(\mathbf{X}_0, \varepsilon) \xrightarrow{\sigma a \sigma''}$ **fail**.

Proof sketch. Let us first decompose the sequence of transitions $(\mathbf{X}_0, \varepsilon) \xrightarrow{\sigma' \sigma''}$ **fail** as $(\mathbf{X}_0, \varepsilon) \xrightarrow{\sigma'} (X, \sigma') \xrightarrow{\sigma''}$ **fail**, for some X. Then by definition of a spine execution we get $(\mathbf{X}_0, \varepsilon) \xrightarrow{\sigma} (X, \sigma)$. Next, it is straightforward to show by induction on σ'' that $(X, \sigma) \xrightarrow{a\sigma''}$ **fail**, whenever $(X, \sigma') \xrightarrow{a\sigma''}$ **fail** and $\mathbf{new}_\lambda(\sigma', a)$. □

Theorem 5. *Let* $\Delta_{\varphi'}(s')$ *be orthogonal w.r.t. to* $\Delta_\varphi(s)$ *and* λ. *If* $\Delta_{\varphi'}(s')$ *passes the test suites* $\mathcal{T}(s, \lambda)$ *and* $\mathcal{S}(\varphi, \lambda)$, *then* $\Delta_{\varphi'}(s')$ *passes the test suite* $\mathcal{T}(s, \varphi)$.

Proof. Let $\mathbf{X}_0$ be the initial state of the test suite $\mathcal{T}(s, \varphi)$. We will prove this theorem by contradiction. Let $\Delta_{\varphi'}(s')$ pass the test suites $\mathcal{T}(s, \lambda)$ and $\mathcal{S}(\varphi, \lambda)$. Suppose $\Delta_{\varphi'}(s')$ fails in passing the test suite $\mathcal{T}(s, \varphi)$. Then, there exists the following sequences of transitions $(\mathbf{X}_0, \varepsilon) \xrightarrow{\sigma}$ **fail** and $\Delta_{\varphi'}(s') \xrightarrow{\sigma} \Delta_{\varphi'}(s'_1)$ (for some σ, s'_1) in the test suite $\mathcal{T}(s, \varphi)$ and the feature specification $\Delta_{\varphi'}(s')$. Now there are two possibilities:

1. Either, $\sigma \in \text{Straces}(\Delta_\lambda(s))$. Similar to the corresponding case of Theorem 4.

2. Or, $\sigma \notin \text{Straces}(\Delta_\lambda(s))$. Then, the sequence of transitions $(\mathbf{X}_0, \varepsilon) \overset{\sigma}{\twoheadrightarrow}$ **fail** can be decomposed in the following way: $(\mathbf{X}_0, \varepsilon) \overset{\sigma_1 a \sigma_2}{\twoheadrightarrow}$ **fail** with $\sigma = \sigma_1 a \sigma_2$ and $\mathbf{new}_\lambda(\sigma_1, a)$. Since the feature specification $\Delta_{\varphi'}(s')$ is orthogonal w.r.t. $\Delta_\varphi(s)$ and λ, we have
$$\exists_{s_2', \sigma_1'}\ \Delta_{\varphi'}(s') \overset{\sigma_1' a \sigma_2}{\twoheadrightarrow} \Delta_{\varphi'}(s_2') \wedge \sigma_1' \dagger \sigma_1.$$
Then, by applying Lemma 9 we get the following execution in the spinal test suite: $\mathbf{X}_0 \overset{\sigma_1' a \sigma_2}{\twoheadrightarrow}$ **fail**. Thus, $\Delta_{\varphi'}(s')$ fails to pass the spinal test suite $\mathcal{S}(\varphi, \lambda)$; hence, a contradiction. □

8. The Ceiling Speed Monitoring Function - A Case-Study

In this section, we apply the residual and spinal testing techniques to a part of an actual software system taken from [39, 40]. The goal is to give an empirical estimate in the reduction of testing effort when comparing the spinal test suites with the residual test suites. In addition, we will also show how a model checker can be used to generate (residual/spinal) test suites up to finite depth. To this end, we exploit the model-checker within the mCRL2 toolset. (mCRL2 [46, 47] is a specification language based on ACP-style process algebra that incorporates data-enriched behavioural modelling of computer systems.)

To demonstrate these points, we test the Ceiling Speed Monitoring (CSM) system – a part of the European Train Control System [39, 40] – that ensures the maximal speed allowed abides by the current most restrictive speed profile.

As stated before, the case study is supposed to serve two purposes: firstly, to provide a proof of concept for our test technique and an initial evidence / refutation of its efficiency gain and secondly, to give an idea how our technique can be implemented using an off-the-shelf state-space generation or reachability analysis tool.

8.1. Modeling

A reason for selecting this case-study is the challenge – identified by the authors of [39] – in automated derivation of the test cases that are insensitive to change in a parameter which marks the availability of service brakes. Typical hardware configuration of a train is as follows: a train *must* have an emergency brake feature; however, a train *may* have a service brake feature. The idea is that a train without service brake feature must use emergency brake feature to decrease the speed of a train regardless of the situation, whereas the train with service brake feature must use emergency brake feature only in an emergency situation.

In Figure 12, an IOFTS modelling the CSM system is depicted. This model is derived from the Mealy machine obtained in [39, 40] after employing the equivalence class partitioning technique of [48] on the original SysML model of the CSM system. The translation from Mealy machine to an IOFTS is as follows: every transition of the form $s \xrightarrow{a/b} s'$ is interpreted as $s \xrightarrow{?a}_{\top} \circ \xrightarrow{!b}_{\top} s'$ in the IOFTS. Thus, the states depicted as rounded rectangles in Figure 12 correspond to the states of the Mealy machine of the CSM system, whereas the states depicted as white dots in Figure 12 are the intermediate states which are introduced due to our translation from the Mealy machine of the CSM system. Lastly, the variability in the CSM model is reflected by the presence/absence of service brakes, which is modelled by the proposition s and its negation.

8.2. Generating Test Suites

Without further elaborating the details of the CSM model, in the remainder, we discuss how to generate test suites in the sense of Definition 6. The key observation is that if we forget the verdict states (**pass** and **fail**) from the test suite of feature specification $\Delta_\varphi(s)$, then this mathematical structure is nothing but an unfolding of the determinised version of $\Delta_\varphi(s)$. Although constructing a deterministic transition system from a nondeterministic one is an expensive procedure, automated tools such as mCRL2 are capable of achieving this. Furthermore, the unfolding operation can be easily encoded as the synchronous parallel composition between a transition system and an unbounded queue that only grows in one direction and synchronises on every action (regardless of input and output polarity) of the transition system. Next, we formalise these ideas and present a method to generate a test suite from a feature specification.

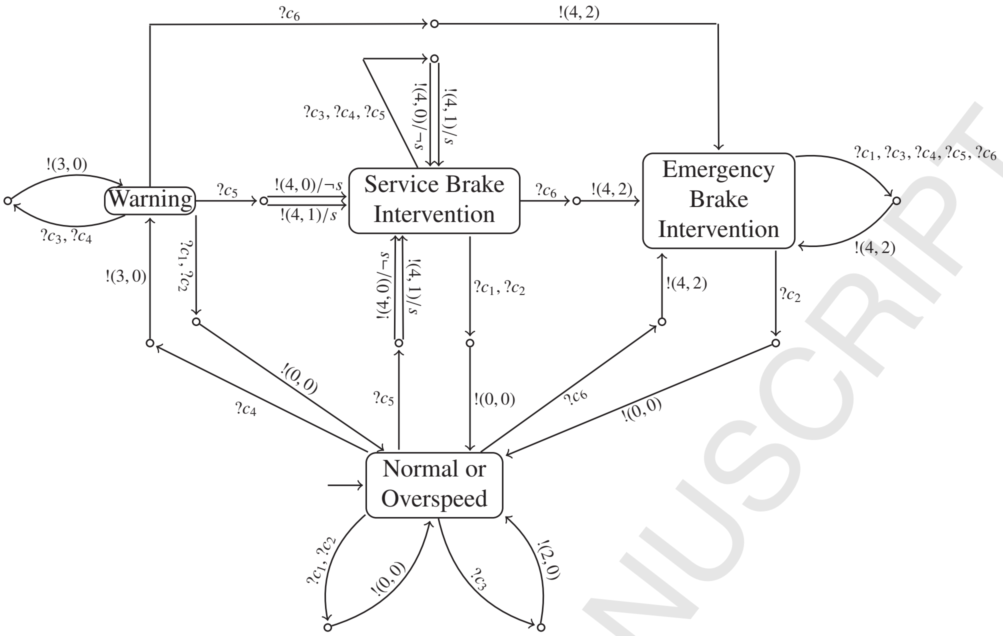

Figure 12: An abstract IOFTS modelling the ceiling speed monitor. For the sake of readability, the quiescent transitions (self-loops) and the true feature constraint ⊤ are not drawn.

Consider a feature specification $\Delta_\varphi(s)$ with F and Λ as the set of features and the set of valid products, respectively. We write $\Delta_\varphi(s)$ **after** $\sigma = \{\Delta_\varphi(s)' \mid \Delta_\varphi(s) \overset{\sigma}{\twoheadrightarrow} \Delta_\varphi(s)'\}$ (for $\sigma \in A_\delta{}^*$) as the set of reachable states via the trace σ. Next, we associate a transition relation between the sets of reachable states:

$$\Delta_\varphi(s)\ \textbf{after}\ \sigma \xrightarrow{a}_\varphi \Delta_\varphi(s)\ \textbf{after}\ \sigma' \iff \sigma' = \sigma a.$$

Then, the obtained structure $(\bigcup_{\sigma \in \mathrm{Straces}(\Delta_\varphi(s))} \Delta_\varphi(s)\ \textbf{after}\ \sigma, \Delta_\varphi(s)\ \textbf{after}\ \varepsilon, A_\delta, F, \rightarrow_\varphi, \Lambda)$ forms a deterministic IOFTS associated with a given feature specification $\Delta_\varphi(s)$. Such structures in the literature are called *suspension automata* (cf. [8] and also Section 4 in this paper). Note that a suspension automaton in general may violate the tree property.

Proposition 4. *The suspension automaton of a feature specification is always deterministic.*

Now in order to handle the verdict states, we add **pass** and **fail** states in the states of a suspension automaton and enrich the transition relations of a suspension automaton with the following conditions akin to rules (4), (5), and (6).

1. If $\sigma, \sigma a \in \mathrm{Straces}(\Delta_\varphi(s))$ and $a \in A_O \cup \{\delta\}$, then add the transition $\Delta_\varphi(s)$ **after** $\sigma \xrightarrow{a}$ **pass**.
2. If $\sigma \in \mathrm{Straces}(\Delta_\varphi(s)), \sigma a \notin \mathrm{Straces}(\Delta_\varphi(s))$, and $a \in A_O \cup \{\delta\}$, then add the transition $\Delta_\varphi(s)$ **after** $\sigma \xrightarrow{a}$ **fail**.
3. Add self-loops for every $a \in A_O \cup \{\delta\}$ at the verdict states, i.e., **pass** $\xrightarrow{a}$ **pass** and **fail** $\xrightarrow{a}$ **fail**.

We call such structures *extended* suspension automata.

Consider the following definition of queue as a form of transition system: $(A_\delta{}^*, \{\sigma \xrightarrow{a} \sigma a \mid \sigma \in A_\delta{}^*, a \in A_\delta\}, \varepsilon)$ with ε as the initial state. Furthermore, consider the following synchronous parallel composition between an extended suspension automaton and the queue defined as the smallest relation satisfying (where X, Y are states in an suspension automaton):

$$\frac{X \xrightarrow{a}_\varphi Y \quad \sigma \xrightarrow{a} \sigma a}{X \mid \sigma \xrightarrow{a}_\varphi Y \mid \sigma a}\ .$$

As a result, we have the following method to generate test suites from a feature specification.

Theorem 6. *The test suite generated by a feature specification is isomorphic to the synchronous parallel composition between the extended suspension automaton and the queue ε. Furthermore, the test suite generated by a feature specification up to depth n is isomorphic to the synchronous parallel composition between the extended suspension automaton and the bounded queue up to length n.*

Proof. Let $(\mathbf{X}_0, \varepsilon)$ be the initial state of the test suite generated by a feature specification $\Delta_\varphi(s)$. Let $\Delta_\varphi(s)$ **after** ε be the initial state of the extended suspension automaton associated with the feature specification $\Delta_\varphi(s)$. Notice that the sets of features and valid products in the two IOFTSs (i.e., the generated test suite and the extended suspension automaton) are identical by construction. Thus, it remains to find a witnessing isomorphism between the two. To prove this, we define a function

$$f : \mathrm{Reach}(\mathbf{X}_0, \varepsilon) \rightarrow \bigcup_{\sigma \in \mathrm{Straces}(\Delta_\varphi(s))} \{\Delta_\varphi(s)\ \textbf{after}\ \sigma \mid \sigma\} \cup \{\textbf{pass}, \textbf{fail}\}$$

as follows: $f(\mathcal{X}) = \mathcal{Y}$ if and only if

- If $\mathcal{X} \in \{\textbf{pass}, \textbf{fail}\}$, then $\mathcal{Y} = \mathcal{X}$.
- If $\mathcal{X} = (X, \sigma)$, then $\mathcal{Y} = \Delta_\varphi(s)$ **after** $\sigma \mid \sigma$.

It is then straightforward to verify that the function f is a witnessing isomorphism between the test suite and the parallel composition of the extended suspension automaton with the empty queue. □

We took the following steps (using the mCRL2 tool-set) to obtain a bounded test case from the IOFTS given in Figure 12.

- Firstly, the IOFTS depicted in Figure 12 was manually translated into a set of guarded linear process equations, where the guards model the feature constraints.
- Secondly, the verdict states and their associated transitions (cf. rules (4), (5), and (6)) are manually added into the mCRL2 specification. Note that the original IOFTS model is deterministic, so the suspension automaton remains isomorphic to the IOFTS drawn in Figure 12.
- Thirdly, a queue process term is defined whose data structure is a (finite) list over the alphabet of the CSM model. Lastly, the queue process term is composed with the mCRL2 specification modelling the CSM model.

8.3. Generating Residual and Spinal Test Suites

Next, we show how to use the mCRL2 model checker to generate residual test suites. Recall that a residual test suite prunes away the behaviour of a specified set of features from an abstract test suite $\mathcal{T}(s, \varphi)$ with respect to a concrete test suite $\mathcal{T}(s, \lambda)$ of the already tested product $\lambda \models \varphi$. The idea is to lift this pruning operation to an extended suspension automaton, instead of applying it directly on the abstract test suite $\mathcal{T}(s, \varphi)$ (cf. Definition 11).

Definition 16. Let $\mathcal{T}(s, \varphi)$ be a test suite generated by a feature specification $\Delta_\varphi(s)$ with F and Λ be the set of features and the set of valid products, respectively. Furthermore, let $\lambda \models \varphi$ be an already tested product and let $\Delta_\varphi(s)$ **after** ε be the extended suspension automaton. Then, a *residual suspension automaton* (w.r.t. λ) is an automaton $(\mathbf{X} \uplus \{\textbf{pass}, \textbf{fail}\}, \Delta_\varphi(s)\ \textbf{after}\ \varepsilon, A_\delta, F, \rightarrow_\varphi, \Lambda \setminus \{\lambda\})$ satisfying:

1. The set of non verdict states $\mathbf{X}$ is defined as the smallest set satisfying:
   (a) If $\textbf{new}_\lambda(\sigma)$ then $\Delta_\varphi(s)$ **after** $\sigma \in \mathbf{X}$.
   (b) If $\Delta_\varphi(s)$ **after** $\sigma \in \mathbf{X}$ and $\sigma \preceq \sigma'$ then $\Delta_\varphi(s)$ **after** $\sigma' \in \mathbf{X}$.
   (c) If $\Delta_\varphi(s)$ **after** $\sigma \in \mathbf{X}$ and $\sigma' \preceq \sigma$ then $\Delta_\varphi(s)$ **after** $\sigma' \in \mathbf{X}$.
2. The transition relation is defined just like in the case of extended suspension automaton.

As a result, we have the following alternative way to generate residual test suites from a feature specification.

Theorem 7. *The residual test suite generated by a feature specification is isomorphic to the synchronous parallel composition between the residual suspension automaton and the queue ε. Furthermore, the residual test suite generated by a feature specification up to depth n is isomorphic to the synchronous parallel composition of the residual suspension automaton and the bounded queue up to length n.*

Proof. Similar to Theorem 6. Function f defined in Theorem 6 is a witnessing isomorphism between the residual test suite and the synchronous parallel composition of a residual suspension automaton with the empty queue. □

We now turn our attention to the generation of spinal test suites using the mCRL2 tool set.

The stark point differentiating spinal test suites from residual test suites is that the former only keeps the spinal executions of the already tested product that lead to new behaviour, rather than keeping all executions leading to new behaviour. In order to encode spinal executions in mCRL2, we introduce a concept of spinal queues, which maintains the lists of actions and processes as its data structure.

Definition 17. Let $\Delta_\varphi(s)$ be a feature specification with the set of features F and the set of valid products Λ. Furthermore, let $\mathbf{X} = \{\Delta_\varphi(s) \textbf{ after } \sigma \mid \sigma \in \text{Straces}(\Delta_\varphi(s))\}$ whose elements are ranged over by the symbols X, Y. Then, a *spinal queue* for a feature specification $\Delta_\varphi(s)$ and an already tested product $\lambda \models \varphi$ is an automaton $(A_\delta{}^* \times \mathbf{X}^* \times \mathbb{B}, (\varepsilon, \Delta_\varphi(s) \textbf{ after } \varepsilon, \bot), A_\delta, F, \rightarrow_\varphi, \Lambda \setminus \{\lambda\})$, whose initial state is $(\varepsilon, \Delta_\varphi(s) \textbf{ after } \varepsilon, \bot)$ and the transition relation $\rightarrow_\varphi$ is defined as the smallest relation satisfying the following rules:

$$\frac{X \xrightarrow{a}_\varphi Y \quad \mathbf{new}_\lambda(\sigma, a)}{(\sigma, \rho X, \bot) \xrightarrow{a}_\varphi (\sigma a, \rho XY, \top)} \qquad \frac{X \xrightarrow{a}_\varphi Y}{(\sigma, \rho X, \top) \xrightarrow{a}_\varphi (\sigma a, \rho XY, \top)} \qquad \frac{X \xrightarrow{a}_\varphi Y \quad \neg\mathbf{new}_\lambda(\sigma) \quad Y \notin \rho}{(\sigma, \rho X, \bot) \xrightarrow{a}_\varphi (\sigma a, \rho XY, \bot)}.$$

Theorem 8. *The spinal test suite generated by a feature specification is isomorphic to the synchronous parallel composition between the extended suspension automaton and the spinal queue. Furthermore, the spinal test suite generated by a feature specification up to depth n is isomorphic to the synchronous parallel composition of the extended suspension automaton and the bounded spinal queue up to length n.*

Proof. Let $(\mathbf{X}_0, \varepsilon)$ be the initial state of the spinal test suite generated by a feature specification $\Delta_\varphi(s)$ w.r.t. an already tested product $\lambda \models \varphi$. Let $\Delta_\varphi(s) \textbf{ after } \varepsilon$ be the initial state of the extended suspension automaton associated with the feature specification $\Delta_\varphi(s)$. Notice that the sets of features and valid products in the two IOFTSs (i.e., the generated spinal test suite and the extended suspension automaton) are identical by construction. Thus, it remains to find a witnessing isomorphism between the two. To prove this, we define a function

$$f : \text{Reach}(\mathbf{X}_0, \varepsilon) \rightarrow \bigcup_{\sigma \in \text{Straces}(\Delta_\varphi(s))} \{\Delta_\varphi(s) \textbf{ after } \sigma \mid (\sigma, \rho, b)\} \cup \{\textbf{pass}, \textbf{fail}\}$$

as follows: $f(\mathcal{X}) = \mathcal{Y}$ if and only if

- If $\mathcal{X} \in \{\textbf{pass}, \textbf{fail}\}$, then $\mathcal{Y} = \mathcal{X}$.
- If $\mathcal{X} = (X, \sigma)$ and $\sigma \in \text{Straces}(\Delta_\lambda(s))$, then $\mathcal{Y} = \Delta_\varphi(s) \textbf{ after } \sigma \mid (\sigma, \rho, \bot)$, where ρ is the list of X-component traversed in the order of the executions σ from the initial state $(\mathbf{X}_0, \varepsilon)$.
- If $\mathcal{X} = (X, \sigma), \sigma \in \text{Straces}(\Delta_\varphi(s))$ and $\sigma \notin \text{Straces}(\Delta_\lambda(s))$, then $\mathcal{Y} = \Delta_\varphi(s) \textbf{ after } \sigma \mid (\sigma, \rho, \top)$, where ρ is the list of X-component traversed in the order of the execution σ from the initial state $(\mathbf{X}_0, \varepsilon)$.

It is straightforward to verify that the function f is a witnessing isomorphism between the test suite and the parallel composition of the extended suspension automaton with the empty queue. □

Test case depth	Test suite type	Number of states	Number of transitions
$n = 2$	Normal / Residual	1804	1839
	Spinal	1535	1565
$n = 3$	Normal / Residual	22805	23128
	Spinal	16396	16649
$n = 4$	Normal / Residual	294594	295673
	Spinal	175167	175870

Table 1: A comparison of different test suites for the CSM model generated by mCRL2 tool-set.

8.4. Results and discussion

We used the above-given models to generate traditional IOCO, residual, spinal test-cases of depths 2 to 4. The mCRL2 toolset was able to efficiently handle such models and generate state spaces in the order of a few seconds to a few minutes on an off-the-shelf ordinary laptop.

In Table 1, we report the state space, i.e., test case, sizes of the given depths for our CSM model. As it can be noted, applying the definitions of residual test suite/suspension automaton on the CSM model does not remove any state or transition from the original model (Figure 12). This is to be expected because the whole specification is a strongly connected component and all paths can potentially lead to new behavior with the emergency break. However, spinal test suites do bring about a reduction in the state space and the reduction increases from ca. 15% in the case of test cases of depth 2 to ca. 40% for test cases of depth 4.

The reduction introduced by spinal test suites seems substantial. We envisage that combining the idea of spinal test suites with firstly, a feature selection and combination criteria, and secondly, a feature interaction detection mechanism (e.g., a syntactic method for checking orthogonality) can lead to a practical testing technique for SPLs.

This is a small scale case study and in order to evaluate the practical applicability of our method, we need to model various larger case studies and combine our method with different feature selection criteria (e.g., pairwise feature selection and maximal feature selection).

9. Conclusions

In this paper, we extended the theory of input-output conformance (IOCO) to use behavioural models of software product lines for conformance testing. In addition, we developed a theoretical framework for generating test suites incrementally based on the features included in different products. To this end, we defined the notions of residual and spinal test suites, which reach to the untested behaviour through a trace of already tested behaviour. In residual testing all such traces are included, which in practical cases, will lead to little or no saving in test effort. However, spinal testing allows for saving test effort by only going through a minimal set of tested traces to cover the untested behaviour. We showed that residual testing is always exhaustive, while spinal testing is only exhaustive under some (rather mild) conditions.

In the future, we would like to study orthogonality at a higher level of abstraction (i.e., in a modeling or programming language) and identify sufficient syntactic conditions for the orthogonality criterion. Also, implementing the notion of spinal test-suite and applying it to practical cases is another item in our future to-do list. To this end, reachability analysis and satisfiability solving can be used to check for new behaviour in the definition of spinal test suites. Moreover, when orthogonality fails, e.g., due to feature interaction, we would like to identify the semantic differences and include them as an input to the test-case generation process.

Adapting the model-based coverage criteria to the SPL setting and incorporating them into the notion of test suite is another item in our agenda in order to make our theory of conformance testing applicable to practical case studies.

The notion of exhaustiveness used in IOCO and adopted in our theory is only of theoretical interest. For real-world applications, one has to come up with a finitely feasible notion of coverage, e.g., fault coverage [49] or model coverage [50, 51] to refine the test case generation- and the test case execution method and tame their complexity.

Acknowledgements

The insightful comments of the anonymous referees of ACM SAC- SVT 2014 and MBT 2014 are gratefully acknowledged. We are also thankful to JLAMP reviewers, whose comments led to further substantial improvement in the results and the presentation.

References

[1] H. Beohar, M. R. Mousavi, Input-output conformance testing based on featured transition systems, in: Proc. of 29th ACM Symposium of Applied Computing: Software Verification and Testing Track (ACM SAC SVT '14), ACM, 2014, pp. 1272-1278.

[2] H. Beohar, M. R. Mousavi, Spinal test suites for software product lines, in: H. Schlingloff, A. K. Petrenko (Eds.), Proc. of 9th Workshop on Model Based Testing (MBT'2014), Vol. 141 of Electron. Proc. in Theor. Comput. Sci., Open Publishing Association, 2014, pp. 44–55.

[3] P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor, E. S. de Almeida, S. R. de Lemos Meira, A systematic mapping study of software product lines testing, Inf. Softw. Technol. 53 (5) (2011) 407–423. doi:10.1016/j.infsof.2010.12.003.

[4] E. Engström, P. Runeson, Software product line testing - a systematic mapping study, Information & Software Technology 53 (1) (2011) 2–13.

[5] S. Oster, A. Wübbeke, G. Engels, A. Schürr, Model-based software product lines testing survey, in: J. Zander, I. Schieferdecker, P. Mosterman (Eds.), Model-based Testing for Embedded Systems, CRC Press, 2011, pp. 339–381.

[6] B. P. Lamancha, M. P. Usaola, M. P. Velthius, Systematic review on software product line testing, in: J. Cordeiro, M. Virvou, B. Shishkov (Eds.), Software and Data Technologies, Vol. 170 of Comm. in Computer and Information Science, Springer, 2013, pp. 58–71.

[7] T. Thüm, S. Apel, C. Kästner, M. Kuhlemann, I. Schaefer, G. Saake, Analysis strategies for software product lines, Technical report FIN-004-2012, School of Computer Science, University of Magdeburg (2012).

[8] J. Tretmans, Model based testing with labelled transition systems, in: R. M. Hierons, J. P. Bowen, M. Harman (Eds.), Formal Methods and Testing, Vol. 4949 of LNCS, Springer, 2008, pp. 1–38.

[9] M. Yannakakis, D. Lee, Testing of finite state systems, in: G. Gottlob, E. Grandjean, K. Seyr (Eds.), Proc. of 12th International Workshop on Computer Science Logic (CSL '98), Vol. 1584 of Lect. Notes in Comput. Sci., Springer, 1999, pp. 29–44.

[10] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner (Eds.), Model-Based Testing of Reactive Systems, Vol. 3472 of Lect. Notes in Comput. Sci., Springer, 2005.

[11] M. Lochau, I. Schaefer, J. Kamischke, S. Lity, Incremental model-based testing of delta-oriented software product lines, in: A. D. Brucker, J. Julliand (Eds.), Proc. of 6th International Conference on Tests and Proofs, Vol. 7305 of Lect. Notes in Comput. Sci., Springer, 2012, pp. 67–82.

[12] M. Varshosaz, H. Beohar, M. R. Mousavi, Delta-oriented FSM-based testing, in: M. Butler, S. Conchon, F. Zaïdi (Eds.), Proc. of 17th International Conference on Formal Engineering Methods (ICFEM 2015), Vol. 9407 of Lect. Notes in Comput. Sci., Springer, 2015, pp. 366–381.

[13] T. Thüm, S. Apel, C. Kästner, I. Schaefer, G. Saake, A classification and survey of analysis strategies for software product lines, ACM Comput. Surv. 47 (1) (2014) Article 6. doi:10.1145/2580950.

[14] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, K. Villela, Software diversity: state of the art and perspectives, Softw. Tools for Technol. Transfer 14 (5) (2012) 477–495. doi:10.1007/s10009-012-0253-y.

[15] A. Classen, Modelling with FTS: a collection of illustrative examples, Tech. Rep. P-CS-TR SPLMC-00000001, University of Namur (2010).

[16] K. Schmid, R. Rabiser, P. Grünbacher, A comparison of decision modeling approaches in product lines, in: Proc. of 5th International Workshop on Variability Modeling of Software-Intensive Systems (VAMOS '11), ACM, 2011, pp. 119–126.

[17] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, A. Wasowski, Cool features and tough decisions: a comparison of variability modeling approaches, in: Proc. of 6th International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS '12), ACM, 2012, pp. 173–182.

[18] M. Sinnema, S. Deelstra, Classifying variability modeling techniques, Information & Software Technology 49 (7) (2007) 717–739. .

[19] P. Asirelli, M. H. ter Beek, S. Gnesi, A. Fantechi, Formal description of variability in product families, in: Proc. of 15th International Software Product Line Conference (SPLC '11), IEEE, 2011, pp. 130–139.

[20] P. Asirelli, M. H. ter Beek, A. Fantechi, S. Gnesi, A model-checking tool for families of services, in: R. Bruni, J. Dingel (Eds.), Proc. of the Joint 13th IFIP WG 6.1 and 30th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Systems (FMOODS'11/FORTE'11), Vol. 6722 of Lect. Notes in Comput. Sci., Springer, 2011, pp. 44–58.

[21] A. Classen, M. Cordy, P. Y. Schobbens, P. Heymans, A. Legay, J. F. Raskin, Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking, IEEE Transactions on Software Engineering 39 (8) (2013) 1069–1089. doi:10.1109/TSE.2012.86.

[22] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, J.-F. Raskin, Model checking lots of systems: efficient verification of temporal properties in software product lines, in: Proc. of 32nd ACM/IEEE International Conference on Software Engineering, Vol. 1 of ICSE '10, ACM, 2010, pp. 335–344.

[23] D. Fischbein, S. Uchitel, V. Braberman, A foundation for behavioural conformance in software product line architectures, in: Proc. on Role of software architecture for testing and analysis, ACM, 2006, pp. 39–48.

[24] A. Gruler, M. Leucker, K. Scheidemann, Modeling and model checking software product lines, in: G. Barthe, F. S. de Boer (Eds.), Proc. of 10th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS '08), Vol. 5051 of Lect. Notes in Comput. Sci., Springer, 2008, pp. 113–131.

[25] K. G. Larsen, U. Nyman, A. Wąsowski, Modal I/O automata for interface and product line theories, in: R. De Nicola (Ed.), Proc. of 16th European Symposium on Programming (ESOP '07), Vol. 4421 of Lect. Notes in Comput. Sci., Springer, 2007, pp. 64–79.

[26] H. Beohar, M. Varshosaz, M. R. Mousavi, Basic behavioral models for software product lines: Expressiveness and testing pre-orders, Science of Computer Programming 123 (2016) 42–60. doi:10.1016/j.scico.2015.06.005.

[27] M. Tribastone, Behavioral relations in a process algebra for variants, in: Proc. of 18th International Software Product Line Conference (SPLC'14), ACM Press, 2014, pp. 82–91.

[28] R. Muschevici, J. Proença, D. Clarke, Modular modelling of software product lines with feature nets, in: G. Barthe, A. Pardo, G. Schneider (Eds.), Proc. of 9th Int. Conf. on Softw. Eng. and Formal Methods (SEFM '11), Vol. 7041 of Lect. Notes in Comput. Sci., Springer, 2011, pp. 318–333.

[29] K. Larsen, B. Thomsen, A modal process logic, in: Proc. of 3rd IEEE Annual Symposium on Logic in Computer Science (LICS '88), IEEE, 1988, pp. 203–210.

[30] S. Weissleder, H. Schlingloff, Automatic model-based test generation from uml state machines, in: J. Zander, I. Schieferdecker, P. J. Mosterman (Eds.), Model-based Testing for Embedded Systems, CRC Press, 2011, pp. 339–381.

[31] H. Gomaa, Designing Software Product Lines with UML, Addison-Wesley, 2005.

[32] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Peach, J. Wüst, J. Zettel, Component-Based Product Line Engineering with UML, Addison-Wesley, 2001.

[33] D. Clarke, M. Helvensteijn, I. Schaefer, Abstract delta modeling, in: E. Visser, J. Järvi (Eds.), Proc. of 9th International Conference on Generative Programming and Component Engineering (GPCE '10), ACM, 2010, pp. 13–22.

[34] M. Lochau, S. Lity, R. Lachmann, I. Schaefer, U. Goltz, Delta-oriented model-based integration testing of large-scale systems, Journal of Systems and Software 91 (2014) 63–84. doi:10.1016/j.jss.2013.11.1096.

[35] S. Lity, T. Morbach, T. Thüm, I. Schaefer, Applying incremental model slicing to product-line regression testing, in: G. M. Kapitsaki, E. Santana de Almeida (Eds.), Proc. of 15th International Conference on Software Reuse: Bridging with Social-Awareness (ICSR '16), Vol. 9679 of Lect. Notes in Comput. Sci., Springer, 2016, pp. 3–19.

[36] K. EI-Fakih, N. Yevtushenko, G. van Bochmann, FSM-based incremental conformance testing methods, IEEE Transactions on Software Engineering 30 (7) (2004) 425–436. doi:10.1109/TSE.2004.31.

[37] Z. Pap, M. Subramaniam, G. Kovács, G. Á. Németh, A bounded incremental test generation algorithm for finite state machines, in: A. Petrenko, M. Veanes, J. Tretmans, W. Grieskamp (Eds.), Proc. of 19th IFIP TC6/WG6.1 International Conference on Testing of Software and Communicating Systems and 7th International Workshop on Formal Approaches to Testing Software (TestCom/FATES '07), Vol. 4581 of Lect. Notes in Comput. Sci., Springer, 2007, pp. 244–259.

[38] A. da Silva Simão, A. Petrenko, Fault coverage-driven incremental test generation, Computer Journal 53 (9) (2010) 1508–1522. doi:10.1093/comjnl/bxp073.

[39] C. Braunstein, J. Peleska, U. Schulze, F. Hübner, W.-L. Huang, A. E. Haxthausen, L. Vu Hong, A SysML test model and test suite for the ETCS ceiling speed monitor, Work Package 4 OETCS/WP4/CSM–01/00, University of Bremen (2014).

[40] C. Braunstein, A. E. Haxthausen, W.-L. Huang, F. Hübner, J. Peleska, U. Schulze, L. Vu Hong, Complete model-based equivalence class testing for the etcs ceiling speed monitor, in: S. Merz, J. Pang (Eds.), Proc. of 12th Int. Conf. on Formal Methods and Software Engineering (SEFM'14), Vol. 8829 of Lect. Notes in Comput. Sci., Springer, 2014, pp. 380–395.

[41] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990).

[42] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, Feature diagrams: A survey and a formal semantics, in: Proc. of the 14th IEEE International Conference on Requirements Engineering (RA '06), IEEE, 2006, pp. 136–145.

[43] M. A. de Langen, Vehicle function correctness, Masters thesis, Eindhoven University of Technology (2013).
URL `http://alexandria.tue.nl/extra1/afstversl/wsk-i/langen2013.pdf`

[44] M. Lochau, J. Kamischke, Parameterized preorder relations for model-based testing of software product lines, in: T. Margaria, B. Steffen (Eds.), Proc. of 5th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change (ISoLA '12), Part I, Vol. 7609 of Lect. Notes in Comput. Sci., Springer, 2012, pp. 223–237.

[45] G. D. Plotkin, A Structural Approach to Operational Semantics, Tech. Rep. DAIMI FN-19, University of Aarhus (1981).

[46] J. F. Groote, M. R. Mousavi, Modeling and Analysis of Communicating Systems, MIT press, 2014.

[47] S. Cranen, J. F. Groote, J. J. A. Keiren, F. P. M. Stappers, E. P. de Vink, W. Wesselink, T. A. C. Willemse, An overview of the mCRL2 Toolset and its recent advances, in: N. Piterman, S. A. Smolka (Eds.), Proc. of 19th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '13), Vol. 7795 of Lect. Notes in Comput. Sci., Springer, 2013, pp. 199–213.

[48] W.-L. Huang, J. Peleska, Exhaustive model-based equivalence class testing, in: H. Yenigün, C. Yilmaz, A. Ulrich (Eds.), Proc. of 25th IFIP WG 6.1 Int. Conf. on Testing Software and Systems, Vol. 8254 of Lect. Notes in Comput. Sci., Springer, 2013, pp. 49–64.

[49] A. da Silva Simão, A. Petrenko, Generating complete and finite test suite for ioco: Is it possible?, in: H. Schlingloff, A. K. Petrenko (Eds.), Proc. of 9th Workshop on Model-Based Testing (MBT '14), Vol. 141 of Electron. Proc. in Theor. Comput. Sci., Open Publishing Association, 2014, pp. 56–70.

[50] X. Devroey, G. Perrouin, A. Legay, M. Cordy, P. Schobbens, P. Heymans, Coverage criteria for behavioural testing of software product lines, in: T. Margaria, B. Steffen (Eds.), Proc. of 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change (ISoLA '14), Vol. 8802 of Lect. Notes in Comput. Sci., Springer, 2014, pp. 336–350.

[51] M. Volpato, J. Tretmans, Towards quality of model-based testing in the ioco framework, in: Proc. of 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation (JAMAICA '13), ACM, 2013, pp. 41–46.