Available online at www.sciencedirect.com
ScienceDirect
Science of Computer Programming 00 (2015) 1–21

Basic Behavioral Models for Software Product Lines: Expressiveness and Testing Pre-Orders

Harsh Beohar[1], Mahsa Varshosaz[1], Mohammad Reza Mousavi[1,*]

[a]*Centre for Research on Embedded Systems (CERES), School of IT, Halmstad University, Sweden*

Abstract

In order to provide a rigorous foundation for Software Product Lines (SPLs), several fundamental approaches have been proposed to their formal behavioral modeling. In this paper, we provide a structured overview of those formalisms based on labeled transition systems and compare their expressiveness in terms of the set of products they can specify. Moreover, we define the notion of tests for each of these formalisms and show that our notions of testing precisely capture product derivation, i.e., all valid products will pass the set of test cases of the product line and each invalid product fails at least one test case of the product line.

Keywords:
Software Product Lines, Formal Specification, Behavioral Specification, Labeled Transition Systems, Featured Transition Systems, Modal Transition Systems, Calculus of Communicating Systems (CCS), and Product Line CCS (PL-CCS).

1. Introduction

1.1. Motivation

Software product lines (SPLs) are becoming increasingly popular as efficient means for mass production and mass customization of software. Hence, establishing formal foundations for specification and verification of SPLs can benefit a large community and can have substantial impact. In the last few years, many researchers have spent substantial effort in extending various formalisms and their associated reasoning techniques to the SPL settings, of which [1–5] provide a comprehensive overview.

In this paper, we put some structure to the body of knowledge regarding some of the most fundamental extensions of behavioral models for SPLs, namely, those based on labeled transition systems, such as those proposed or studied in [6–19]. These basic models can serve as semantic models for extensions of higher level models such as domain specific languages (DSLs), or those based on the Unified Modeling Language (UML) state or sequence diagrams. Hence, bringing more structure into the body of knowledge about these fundamental computational models can help the language designers of higher level language to make the right choice when defining the semantics of their language.

*Corresponding author. The work of M.R. Mousavi has been partially supported by the Swedish Research Council (Vetenskapsrådet) award number: 621-2014-5057 (Effective Model-Based Testing of Concurrent Systems) and the Swedish Knowledge Foundation (Stiftelsen för Kunskaps- och Kompetensutveckling) in the context of the AUTO-CAAS HöG project.

Email addresses: harsh.beohar@hh.se (Harsh Beohar), mahsa.varshosaz@hh.se (Mahsa Varshosaz), m.r.mousavi@hh.se (Mohammad Reza Mousavi)

The structure proposed in this paper is twofold: first we compare the expressive power of these fundamental models and second, we explore the extensional and intensional notions of testing equivalence (pre-orders) for each of them. Some of the expressiveness results reported in the present paper are hinted at in the literature, but to our knowledge, have never been formalized and proven before. Regarding the testing equivalences, the extensional notions of testing defined in this paper – for the models proposed in [14, 15, 17] – are novel. They are of course based on and slight extensions of well-known notions of tests for labeled transition systems (e.g., of [20–22] and particularly that of [23]).

1.2. Running Example

In order to illustrate the different approaches, we use the following simple example originally due to Asirelli et al. [7] and further elaborated by Classen [2].

Example 1. *We model a product line for vending machines, which accept one-Euro coins (*1e*) exclusively for the European market and one-Dollar coins (*1d*) exclusively for the American market. A user has a choice of adding* sugar *or* no sugar*, after which she is allowed to choose a beverage among* coffee*,* tea*, or* cappuccino*. Furthermore, the following constraints hold on each product:*

1. *Coffee must be offered by each and every variant of this product line.*
2. *Cappuccino is served only by the European machines and whenever cappuccino is served, a ring-tone must ring.*
3. *Tea is an optional feature for both markets.*

1.3. Contributions

The objects of study in this paper are three popular models of computation that are used in the literature to model SPLs. Namely, we study modal transition systems [24], product line labeled transition systems [17], and featured transition systems [15]. The contributions of this paper are as follows:

- Firstly, we formally show that the class of modal transition systems are strictly less expressive than the class of product line labeled transition systems, which are in-turn strictly less expressive than the class of featured transition systems.
- Secondly, we show how the test expressions of Abramsky [23] can be used to characterize product derivation for each of the above models of software product line.

1.4. Paper Structure

The rest of this paper is organized as follows. In Section 2, we present an overview of the product-line formalisms studied in this paper and recall or define their intuitive notion of derived products. In Section 3, we compare the expressiveness of formalisms by comparing their set of definable products. In Section 4, we define the extensional notions of test for the formalisms and prove that they coincide with their intensional counter-parts. The paper is concluded in Section 5 with a summary of the results and some directions of our ongoing research.

2. Fundamental Behavioral Models of SPLs

2.1. Overview

Conventional formal models such as labeled transition systems (LTSs) can be used to specify the behavior of systems at a high level of abstraction. Namely, LTSs specify how a system execution evolves on an abstract machine in terms of transitions that are labeled with the information that is received / made available through each execution step from / to the outside world. However, in order to formally specify a software product line, one needs specific (semantic) notions to refer to variation points, distinguish different features and refer to their possible interactions. Below, we give an overview of several alternatives proposed as fundamental behavioral models of software product lines.

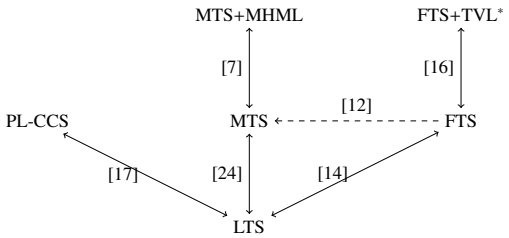


Figure 1. LTS-based behavioral models of SPLs.

As the first alternative, Fischbein et al. [6] argue that modal transition systems (MTSs) [24] are adequate extensions of LTSs to model a software product line: MTSs partition the transitions into may and must transitions and hence, each MTS has an associated set of possible implementations. Subsequently, several researchers [7–9, 12, 13] adopted modal transition system as a formal model to perform rigorous analysis of software product lines. Several pieces of work [7–9] addressed the issue of deriving valid products from a given MTS by model checking against formulae expressed in a deontic logic called Modal-Hennessy-Milner-Logic (MHML). Others [12, 13] developed an interface theory and a testing theory for software product lines.

Classen et al. [14, 15] took a different route by annotating LTSs with features of a feature diagram. Intuitively, a feature diagram specifies, by means of a graphical notation, the set of valid products by specifying constraints on the presence of features. The result of annotating LTSs with features is called a featured transition system (FTS). Furthermore, an LTL-model checking algorithm in the context of featured transition system was given in [15]. Cordy et al. [16] extended the earlier work [14, 15] by incorporating non-boolean features and multi-features in a high-level specification language called TVL*. Also, an algorithm was given to construct an FTS from a behavioral specification written in TVL*.

As another alternative, Gruler et al. [17] extended Milner's CCS [25] into a process calculus called PL-CCS. The extension involves introducing the "binary variant" $\oplus$ operator to represent the alternative features of a product line. Like MTSs, the validity of products is asserted by model checking formulae specified in a multi-valued modal *mu*-calculus, originally due to [26]. The semantics of the logic specifies, for each PL-CCS process, the set of configurations that satisfy the logical formula.

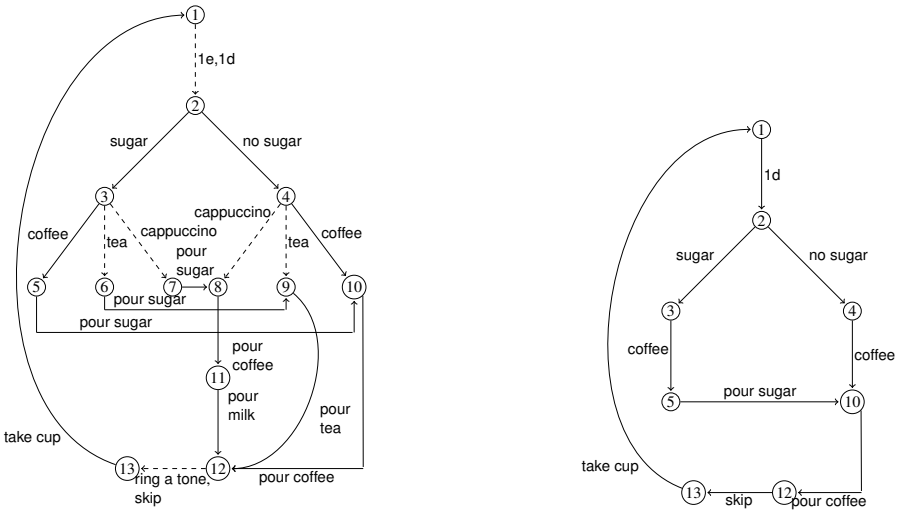
In addition, there are also other alternatives for specifying the behavior of SPLs that fall beyond the scope of the present paper. For example, there are proposals based on re-using existing process algebras with data (see, e.g., [27–29]) or extending Petri Nets with features (see, e.g., [30, 31]). In order to perform a formal comparison of expressiveness, we confine ourselves to models that are based on LTSs (e.g., those models that specify a set of products captured by an LTS) and hence, the other approaches mentioned in this paragraph are not considered any further. Also, there are extensions of higher-level formalisms such as UML that are not considered in this paper, since they either lack formal semantics or their semantics can be expressed in the more fundamental formalisms such as those studied in this paper.

In Fig. 1, we summarize the different extensions of LTSs for the behavioral modeling of SPLs. In this figure, the solid arrows show the possibility of transforming a model from one formalism into another. In [12], the authors gave a semantics for a restrictive notion of FTS in terms of an MTS. One of the two main contributions of this paper is to complete this picture, and hence generalize the result of [12], by presenting (or showing the impossibility of) encodings among PL-CCS, MTSs and FTSs. This is represented by the dashed arrow in Fig. 1. In the remainder of this section, MTSs are surveyed in Section 2.2 and FTSs are reviewed in Section 2.3. In Section 2.4, we survey process-algebraic approaches to SPL specification.

2.2. Modal Transition Systems

2.2.1. Specifying SPLs.

Modal transition systems extend labeled transition systems by distinguishing two different sorts of transitions, namely, *may* and *must* transitions. May transitions, as their name suggests, may (or may not) be present in the implementation behavior, while must transitions are always present. As a sanity condition, it is required that all must transitions also have a corresponding may transition. The following definition formalizes these concepts.



(a) The vending machine product line [7].

(b) A product of Fig. 2(a).

Figure 2. MTS approach to modeling SPLs

Definition 1. *A modal transition system (MTS) [24] is a quadruple* $(\mathbb{P}, A, \rightarrow_{\Diamond}, \rightarrow_{\Box})$, *where* $\mathbb{P}$ *is a set of states or processes,* A *is a set of actions,* $\rightarrow_{\Diamond} \subseteq \mathbb{P} \times A \times \mathbb{P}$ *is the so-called* may *transition relation, and* $\rightarrow_{\Box} \subseteq \rightarrow_{\Diamond}$ *is the so-called* must *transition relation.*

An MTS can only describe the behavior of optional and mandatory features of a product line in terms of may and must transition relations, respectively. An MTS specifies a unique LTS when $\rightarrow_{\Box} = \rightarrow_{\Diamond}$, i.e., when all transitions are must transitions, and vice versa, an LTS can be interpreted as an MTS by interpreting ordinary transitions as must transitions. Throughout the rest of this section, we fix the letters P, P', Q, Q' to denote the states of an MTS, whereas, p, p', q, q' are used to denote the states of an LTS.

Example 2. *Consider the informal description given in Example 1. The MTS shown in Fig. 2(a) (due to [7]), formally specifies this product line. In this MTS, solid arrows denote must transitions and dashed arrows denote may transitions. (For must transitions we dispense with drawing the corresponding may transitions and tacitly assume their presence.)*

2.2.2. Deriving Products

A key notion within the theory of MTS is modal refinement [24], which allows for deriving products (MTSs with fewer may and more must transitions) from product lines, or testing conformance of products to product lines. Informally, if P refines Q then all must transitions of P are simulated by Q, while all may transitions of Q are simulated by P. Another intuition shared by modal refinement relation is that some of the may transitions of P can be either transformed into must transitions of Q or blocked by Q, whenever P refines Q.

Definition 2. *A binary relation* $\mathcal{R} \subseteq \mathbb{P} \times \mathbb{P}$ *is a* modal refinement *[24] relation if and only if the following transfer properties are satisfied.*

1. $\forall_{P,P',Q \in \mathbb{P}, a \in A} \; (P \mathcal{R} Q \wedge P \xrightarrow{a}_{\Box} P') \implies \exists_{Q' \in \mathbb{P}} \; Q \xrightarrow{a}_{\Box} Q' \wedge P' \mathcal{R} Q'$.
2. $\forall_{P,Q,Q' \in \mathbb{P}, a \in A} \; (P \mathcal{R} Q \wedge Q \xrightarrow{a}_{\Diamond} Q') \implies \exists_{P' \in \mathbb{P}} \; P \xrightarrow{a}_{\Diamond} P' \wedge P' \mathcal{R} Q'$.

A modal specification P refines a modal specification Q, denoted $P \preceq Q$, if there exists a modal refinement relation $\mathcal{R}$ such that $P \mathcal{R} Q$. The set of products implementing a modal specification P is denoted as $P = [\![p \mid P \preceq p]\!]$.

Example 3. *Consider the product line MTS specified in Example 2. The LTS shown in Fig. 2(b) specifies a product which serves only coffee and is customized for the American market. It is not hard to see that the LTS shown in Fig. 2(b) refines the MTS of Fig. 2(a) by transforming certain may transitions into must transitions, or prohibiting them.*

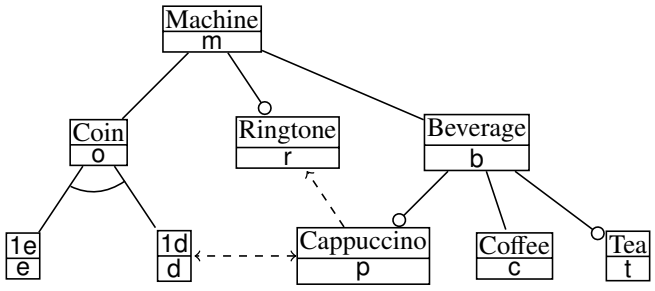


Figure 3. Feature diagram of the vending machine [7].

2.3. Featured Transition Systems

2.3.1. Specifying Structural Aspects.

In [14], the authors pointed out that the derived products from an MTS may be invalid (and counter-intuitive) due to the inherent lack of expressiveness in MTSs for specifying feature constraints. In common practice, feature diagrams [32] have been used to model such constraints using a graphical notation. A feature diagram represents all the products of an SPL in terms of features that are arranged hierarchically. 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 (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 also specifies three additional constraints over features that may span over different levels of abstraction:

1. Alternative relationship, i.e., the designated sub-features can never be simultaneously present in any product.
2. Exclude relationship, i.e., different features at different levels of 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.

For more information and a formal treatment of the syntax and the semantics of feature diagrams, we refer to [32].

Example 4. *Consider the feature diagram shown in Fig. 3, which formalizes the features and feature constraints of Example 1 [7]. In this diagram every machine must consists of the features coin (*o*), and beverage (*b*) and may comprise an optional feature ring-tone (*r*). The coin feature is further decomposed into two alternative features euro (*e*) and dollar (*d*). Furthermore, Fig. 3 also specifies that cappuccino (*p*) requires ring-tone (*r*) denoted by a uni-directional dashed line and cappuccino is absent in the machine that takes dollars represented by bidirectional dashed line.*

A feature diagram only specifies the structural aspects of variability in an SPL. To formally analyze the behavior of an SPL in [15], the transitions of a labeled transition system are annotated 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.

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 interpreting the elements of the set F as propositional variables. For instance, in the context of Example 4, the formula $\mathsf{e} \wedge \neg\mathsf{d}$ asserts the presence of euro coin and the absence of dollar coin payment features. We let ϕ, ϕ' range over the set $\mathbb{B}(F)$.

Definition 3. *A* featured transition system *(FTS) is a quintuple* $(\mathbb{P}, A, F, \rightarrow, \Lambda)$*, where*

1. $\mathbb{P}$ *is the set of* states*,*
2. A *is the set of* actions*,*
3. F *is a set of* features*,*
4. $\rightarrow \subseteq S \times A \times \mathbb{B}(F) \times S$ *is the* transition relation *satisfying the following condition:*

$$\forall_{P,a,P',\phi,\phi'} \left((P, a, \phi, P') \in \rightarrow \ \wedge\ (P, a, \phi', P') \in \rightarrow\ \right) \implies \phi = \phi',$$

5. $\Lambda \subseteq \{\lambda : F \to \mathbb{B}\}$ *is a set of* product configurations.

Just like in the case of MTSs, we reserve the symbols P, P', Q, Q' to denote the states of an FTSs. Furthermore, we write $P \xrightarrow{a}_{\phi} Q'$ to denote an element $(P, a, \phi, Q') \in \to$.

Example 5. *Consider the MTS in Fig. 2(a); we obtain an FTS by discarding the distinction between may and must transitions and instead, annotating every transition with a formula over features given in Fig. 3. In the following table, we give the propositional formula associated with every transition of the vending machine example.*

Transitions	*Features*
$s_1 \xrightarrow{1e} s_2$	e
$s_1 \xrightarrow{1d} s_2$	d
$s_2 \xrightarrow{coffee} s_5$	c
$s_2 \xrightarrow{tea} s_6$	t

Transitions	*Features*
$s_2 \xrightarrow{cappuccino} s_7$	p
$s_{12} \xrightarrow{ring\ a\ tone} s_{13}$	$p \Rightarrow r$
remaining transitions	m

Lastly, the set of product configurations of the vending machine is the following set of 10 products specified by the feature diagram of Example 4 [9]:

$$\begin{aligned}\Lambda = &\{\{m, o, b, c, e\}, \{m, o, b, c, e, r\}, \{m, o, b, c, e, t\}, \{m, o, b, c, e, t, r\},\\ &\{m, o, b, c, e, p, r\}, \{m, o, b, c, d\}, \{m, o, b, c, d, r\}, \{m, o, b, c, d, t\},\\ &\{m, o, b, c, d, t, r\}, \{m, o, b, c, e, p, r, t\}\}.\end{aligned}$$

2.3.2. Deriving Valid Products

In [15] a class of operators $\Delta_\lambda(\_)$, each parameterised by product configurations $\lambda \in \Lambda$, is introduced to project an FTS into various LTSs, thereby obtaining different products from a product line. Roughly, the operation $\Delta_\lambda(\_)$ prunes away those transitions from a product line whose feature constraints are not satisfied by the product configuration λ.

Definition 4. *Given a feature specification P (i.e., a state in an FTS), a set of selected features $\lambda \in \Lambda$ induces a state $\Delta_\lambda(P)$ in an LTS defined by the following operational rule:*

$$\frac{\lambda \models \phi \quad P \xrightarrow{a}_{\phi} Q}{\Delta_\lambda(P) \xrightarrow{a} \Delta_\lambda(Q)}\ .$$

It was argued in [14, 15] that FTSs are better suited to model a software product line than a modal transition system. The crucial difference between the two is that all the may transitions in an MTS are independently optional, while in an FTS, one can make a finer distinction among them by annotating them with more complex boolean formulae pertaining to different types of feature constraints. In other words, the choice among transitions in an FTS depends on the product configuration, whereas the choice among may transitions in an MTS is nondeterministic [15].

Example 6. *Consider the FTS given in Example 5 and the LTS given in Fig. 2(b) with the addition of transition $1 \xrightarrow{1e} 2$. The latter is not a valid product of the former and cannot be derived from Definition 4. Note that there exists no valid set of features or product configuration $\lambda \in \Lambda$ such that $\lambda(e) = \lambda(d) = \top$. This is due to the semantics of the feature diagram depicted in Fig. 3, which specifies that e, d are alternative features. As a result, the transition relation defined in Definition 4 can never have a choice between the actions $1e, 1d$.*

Although the above example suggests that the class of FTSs is expressive enough to specify the different inter-feature relationships, the notion of deriving valid products (Definition 4) by an FTS is syntactical in nature (e.g., compared to the notion of deriving valid products by an MTS). This syntax-driven notion of valid product derivation is too rigid for any semantic analysis such as testing. In particular, we note that Definition 4 is not even closed under strong bisimulation (see [33] for a formal definition). Next, we present a notion of deriving valid products from an FTS which generalizes Definition 4.

Definition 5. *Given an FTS* $(\mathbb{P}, A, F, \rightarrow, \Lambda)$*, an LTS* $(\mathbf{P}, A, \rightarrow)$*, and a product* $\lambda \in \Lambda$*. A family of binary relations* $\mathcal{R}_\lambda \subseteq \mathbb{P} \times \mathbf{P}$ *(parameterized by product configurations) are called* product-derivation relations *if and only if the following transfer properties are satisfied.*

1. $\forall_{P,Q,a,p,\phi}\ (P\mathcal{R}_\lambda p \wedge P \xrightarrow{a}_\phi Q \wedge \lambda \models \phi) \Rightarrow \exists_q\ p \xrightarrow{a} q \wedge Q\mathcal{R}_\lambda q$;
2. $\forall_{P,a,p,q}\ (P\mathcal{R}_\lambda p \wedge p \xrightarrow{a} q) \Rightarrow \exists_{Q,\phi}\ P \xrightarrow{a}_\phi Q \wedge \lambda \models \phi \wedge Q\mathcal{R}_\lambda q$.

A state $p \in \mathbf{P}$ *in an LTS derives the product* λ *from an FTS-specification* $P \in \mathbb{P}$*, denoted by* $P \vdash_\lambda p$*, if there exists an* $\mathcal{R}_\lambda$ *product-derivation relation such that* $P\mathcal{R}_\lambda p$.

We end this section on FTSs by highlighting two intuitive properties of the product derivation relation.

Lemma 1. *For any given feature specification* P *and the derived product* $\Delta_\lambda(P)$*, we have* $P \vdash_\lambda \Delta_\lambda(P)$.

Lemma 2. *Given any feature specification* P *and a derived product* p *with* $P \vdash_\lambda p$*, for some product configuration* λ*. If* q *is strongly bisimilar (in the sense of Park [33]) to* p*, then* $P \vdash_\lambda q$.

2.4. Product Line Process Algebras

Gruler et al. [17] extended Milner's Calculus of Communicating Systems (CCS) [25] into PL-CCS by introducing the "binary variant" operator $\oplus$ to represent the alternative relationship in feature diagrams.

Definition 6. *Let* $A = \Sigma \cup \bar{\Sigma} \cup \{\tau\}$ *be the alphabet, where* $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$*. The syntax of PL-CCS terms* e *is defined by the grammar* $\mathbf{Nil} \mid \alpha.e \mid e + e' \mid e \oplus e' \mid e \parallel e' \mid e[f] \mid e\backslash L$*, where* **Nil** *is the deadlocking process, for each* $\alpha \in A$*,* $\alpha.\_$ *denotes action prefixing,* $\_ + \_$ *denotes non-deterministic choice,* $\_ \oplus \_$ *denotes binary variant,* $\_ \parallel \_$ *denotes parallel composition, for each* $f : A \rightarrow A$*,* $\_[f]$ *denotes renaming by means of* f*, and for each* $L \subseteq A$*,* $\_\backslash L$ *denotes the restriction operator (blocking (co)actions in* L*). In addition, one may define recursive processes by means of process identifiers and equations.*

At the first sight, the variant operator $\oplus$ is reminiscent of the ordinary alternative composition operator + from CCS; however, they are substantially different, as the binary variant operator remembers the chosen alternative. For example, consider process terms $s = a.(b.s + c.s)$ and $t = a.(b.t \oplus c.t)$. Intuitively, the recursive process s keeps on making a choice between b and c upon performing a; whereas in t the choice is made at the first iteration after performing the action a and it is recorded for and respected in all future iterations, i.e., the process behaves deterministically once the choice between "features" b and c is made once and for all.

As syntactic sugar, a unary operator $\langle\_\rangle$, called the *optional* operator, was also introduced to represent the optional features of a feature diagram. It can be defined in PL-CCS as $\langle P\rangle = P \oplus \mathbf{Nil}$.

Example 7. *The following process definition specifies the vending machine product line in PL-CCS.*

$$
\begin{aligned}
s_1 &= \mathit{1e}.s_2 \oplus \mathit{1d}.s_2 & s_7 &= \mathit{pour\ sugar}.s_8\\
s_2 &= \mathit{sugar}.s_3 + \mathit{no\ sugar}.s_4 & s_8 &= \mathit{pour\ coffee}.s_{11}\\
s_3 &= \mathit{coffee}.s_5 + \langle \mathit{tea}.s_6 + \mathit{cappuccino}.s_7\rangle & s_9 &= \mathit{pour\ tea}.s_{12}\\
s_4 &= \mathit{coffee}.s_{10} + \langle \mathit{tea}.s_9 + \mathit{cappuccino}.s_8\rangle & s_{10} &= \mathit{pour\ coffee}.s_{12}\\
s_5 &= \mathit{pour\ sugar}.s_{10} & s_{12} &= \mathit{ring\ tone}.s_{13} + \mathit{skip}.s_{13}\\
s_6 &= \mathit{pour\ sugar}.s_9 & s_{13} &= \mathit{take\ cup}.s_1\ .
\end{aligned}
$$

The semantics of a PL-CCS term is defined in terms of product line labelled transition systems [17], recalled below, using a structural operational semantics. Roughly, the states and the transition relations of a product line labeled transition system are enriched with *configuration vectors* (i.e., functions of type $\{L, R, ?\}^I$ with I being an index set) that records the selection made in past about the alternative features.

Definition 7. *Let* $\{L, R, ?\}^I$ *denote the set of all total functions from an index set* I *to the set* $\{L, R, ?\}$*. A* product line labeled transition system *(PL-LTS) is a quadruple* $(\mathbb{P} \times \{L, R, ?\}^I, A, I, \rightarrow)$ *consisting of a set of states* $\mathbb{P} \times \{L, R, ?\}^I$*, a set of actions* A*, and a transition relation* $\rightarrow \subseteq (\mathbb{P} \times \{L, R, ?\}^I) \times (A \times \{L, R, ?\}^I) \times (\mathbb{P} \times \{L, R, ?\}^I)$ *satisfying the following restrictions:*

1. $\forall_{P,\nu,a,Q,\nu',\nu''}\ (P,\nu) \xrightarrow{a,\nu'} (Q,\nu'') \implies \nu' = \nu''$.
2. $\forall_{P,\nu,a,Q,\nu'}\ \Big((P,\nu) \xrightarrow{a,\nu'} (Q,\nu') \wedge \nu' \neq \nu\Big) \implies \exists_i\ \Big(\nu'(i) \neq \nu(i) \wedge \forall_{j\neq i}\ \nu'(j) = \nu(j)\Big)$.
3. $\forall_{P,\nu,a,Q,\nu',i}\ (P,\nu) \xrightarrow{a,\nu'} (Q,\nu') \wedge \nu(i) \neq ? \implies \nu'(i) = \nu(i)$.

Notice that, as a consequence of item 2 in Definition 7, for any transition $(P,\nu) \xrightarrow{a,\nu'} (Q,\nu')$, if $\nu \neq \nu'$, we can find a unique $i \in I$ such that $\nu'(i) \neq \nu(i) \wedge \forall_{j\neq i}\ \nu'(j) = \nu(j)$. Furthermore, it should also be noted that Conditions 1, 2, and 3 follow from the operational rules given by Gruler et al. to a PL-CCS term in [17].

Now in order to define when an LTS is a valid product of a given PL-LTS, we need the following notion of ordering on configurations and configuration vectors.

Definition 8. *The ordering relation $\sqsubseteq$ on the set $\{L, R, ?\}$ is defined in the following way:*

$$\sqsubseteq = \{(?,?), (L,L), (R,R), (?,L), (?,R)\}.$$

We lift this ordering relation to the level of configuration vectors by letting $\nu \sqsubseteq \nu' \iff \forall_{i\in I}\ \nu(i) \sqsubseteq \nu'(i)$, for any $\nu, \nu' \in \{L, R, ?\}^I$.

We end this section on PL-CCS by proposing a definition of product-derivation relations in a similar vein to Definition 5.

Definition 9. *Let $(\mathbb{P} \times \{L, R, ?\}^I, A, \rightarrow)$ be a PL-LTS and let $(\mathbf{P}, A, \rightarrow)$ be an LTS. A family of binary relations $\mathcal{R}_\theta \subseteq (\mathbb{P}\times\{L, R, ?\}^I)\times\mathbf{P}$ (parameterized by every product configuration $\theta \in \{L, R\}^I$) is a family of* product-derivation relations *if and only if the following transfer properties are satisfied:*

1. $\forall_{P,Q,a,\nu,\nu',p}\ \big((P,\nu)\mathcal{R}_\theta p \wedge (P,\nu) \xrightarrow{a}_{\nu'} (Q,\nu') \wedge \nu' \sqsubseteq \theta\big) \Rightarrow \exists_q\ p \xrightarrow{a} q \wedge (Q,\nu')\mathcal{R}_\theta q$,
2. $\forall_{P,a,\nu,p,q}\ \big((P,\nu)\mathcal{R}_\theta p \wedge p \xrightarrow{a} q\big) \Rightarrow \exists_{Q,\nu'}\ (P,\nu) \xrightarrow{a}_{\nu'} (Q,\nu') \wedge \nu' \sqsubseteq \theta \wedge (Q,\nu')\mathcal{R}_\theta q$.

A state $p \in \mathbf{P}$ in an LTS is (the initial state of) a product of a PL-LTS (P,ν) with respect to a configuration vector θ, denoted by $(P,\nu) \vdash_\theta p$, if $\nu \sqsubseteq \theta$ and there exists an $\mathcal{R}_\theta$ product-derivation relation such that $P\mathcal{R}_\theta p$.

3. Expressiveness Results

The goal of this section is to formally compare the expressiveness of the three product line formalisms as outlined in the previous section. Before we do so, let us bring all the three formalisms under one single definition of a *product line structure*. Intuitively, a product line structure consists of a product line specification and a semantic function $[\![\ ]\!]$ that maps a specification into a set of implementations (valid products) modeled as LTSs.

Definition 10. *Let $(\mathbf{P}, A, \rightarrow)$ be an LTS. A* product line structure *is a tuple $\mathbf{M} = (\mathbb{M}, [\![\ ]\!])$, where $\mathbb{M}$ is the class of all intended product line models or specifications (in our case: MTSs, FTSs, and PL-LTSs) and $[\![\ ]\!] : \mathbb{M} \rightarrow 2^{\mathbf{P}}$ is the semantic function mapping a product line specification to a set of LTSs.*

Definition 11. *An* encoding *$E : \mathbf{M} \rightarrow \mathbf{M}'$ from a product line structure $\mathbf{M} = (\mathbb{M}, [\![\ ]\!])$ into a product line structure $\mathbf{M}' = (\mathbb{M}', [\![\ ]\!]')$ is a function $E : \mathbb{M} \rightarrow \mathbb{M}'$ satisfying the* correctness *criterion $[\![\ ]\!] = E \circ [\![\ ]\!]'$.*

We say that the product line structure $\mathbf{M}'$ is at-least as expressive as *$\mathbf{M}$ if and only if there exists an encoding $E : \mathbf{M} \rightarrow \mathbf{M}'$.*

Furthermore, we say that the product line structure $\mathbf{M}'$ is less expressive than *$\mathbf{M}$, if and only if $\mathbf{M}$ is at-least as expressive as $\mathbf{M}'$, and $\mathbf{M}'$ is not at-least as expressive as $\mathbf{M}$, i.e., there does not exist any encoding $E : \mathbf{M} \rightarrow \mathbf{M}'$.*

In the remainder of this section, we explore the expressiveness among the classes of MTSs, FTSs, and PL-LTSs. In order to do this, we start with relating the two less expressive models, i.e., MTSs and PL-LTSs, to each other, and then move up in the lattice of expressiveness. Note that the "at-least as expressive as" relation is transitive (by the composition of the encoding functions) and hence, we can use the transitivity to relate the least- (i.e., MTSs) and the greatest (i.e., FTSs) points in the lattice, once we relate the middle-point (i.e., PL-LTSs) to each of them.

The following two theorems relate the expressiveness of PL-LTSs and MTSs.

Theorem 1. *The class of PL-LTSs is at-least as expressive as the class of MTSs.*

Proof. Consider the MTS $(\mathbb{P}, A, \rightarrow_\Diamond, \rightarrow_\Box)$ and some $P \in \mathbb{P}$. Let $\longrightarrow\!\!\!\!\!\rightarrow \subseteq \mathbb{P} \times A^* \times \mathbb{P}$ be the reachability relation defined as follows: $\frac{}{P \xrightarrow{\varepsilon} P}$ $\quad \frac{P \xrightarrow{s} P' \;\; P' \xrightarrow{a}_\Diamond P''}{P \xrightarrow{sa} P''}$. Let $\mathbf{tr}(P)$ be the set of traces generated by P, i.e., $\mathbf{tr}(P) = \{s \mid \exists_Q\, P \xrightarrow{s} Q\}$. For state P, we define a family of transition relations parameterized by the traces of P as follows: $\frac{Q \xrightarrow{a}_\Diamond Q' \;\; s \in \mathbf{tr}(P)}{Q \xrightarrow{a}_{s,P} Q'}$. We drop the subscript P from the family of transition relations whenever it is clear from the context.

Next, we construct a PL-LTS whose configuration vectors are functions of type

$$\mathbf{tr}(P) \rightarrow \left(\bigcup_{s \in \mathbf{tr}(P)} \rightarrow_s \rightarrow \{L, R, ?\} \right).$$

The transition relation between the states of PL-LTS is defined as the smallest relation satisfying:

$$\frac{Q \xrightarrow{a}_\Box Q' \quad s \in \mathbf{tr}(P) \quad \nu' =_s \nu \uparrow \{(Q, a, Q') \mapsto L\}}{(Q, \nu) \xrightarrow{a,\nu'} (Q', \nu')} \qquad \frac{Q \xrightarrow{a}_\Box Q' \quad s \in \mathbf{tr}(P) \quad \nu' =_s \nu \uparrow \{(Q, a, Q') \mapsto R\}}{(Q, \nu) \xrightarrow{a,\nu'} (Q', \nu')}$$

$$\frac{Q \xrightarrow{a}_{\Diamond\setminus\Box} Q' \quad s \in \mathbf{tr}(P) \quad \nu' =_s \nu \uparrow \{(Q, a, Q') \mapsto L\}}{(Q, \nu) \xrightarrow{a,\nu'} (Q', \nu')}$$

where $Q \xrightarrow{a}_{\Diamond\setminus\Box} Q' \iff Q \xrightarrow{a}_\Diamond Q' \wedge (Q, a, Q') \notin \rightarrow_\Box$ and the expression $\nu' =_s \nu \uparrow \{(Q, a, Q') \mapsto X\}$ (for $X \in \{L, R\}$) is defined in the following way:

$$\nu'(s)(\bar{Q} \xrightarrow{\bar{a}}_{s'} \bar{Q}') = \begin{cases} X & \text{if } s = s' \wedge \bar{Q} = Q \wedge \bar{a} = a \wedge \bar{Q}' = Q' \\ \nu(s)(\bar{Q} \xrightarrow{\bar{a}}_{s'} \bar{Q}') & \text{otherwise} \end{cases}.$$

We fix $E(P) = (P, \nu_0)$, where $\nu_0(s)(Q \xrightarrow{a}_{s'} Q') =?$ for $s, s' \in \mathbf{tr}(P)$. Furthermore, we let the symbols θ, θ' range over the total configuration vectors, i.e., the functions of type $\mathbf{tr}(P) \rightarrow \left(\bigcup_{s \in \mathbf{tr}(P)} \rightarrow_s \rightarrow \{L, R\}\right)$. Now we are in the position to show that $[\![P]\!] = [\![E(P)]\!]'$, where $[\![P]\!] = \{p \mid P \preceq p\}$ and $[\![E(P)]\!]' = \{p \mid \exists_\theta\, E(P) \vdash_\theta p\}$. We divide the proof obligation into two obligations: $[\![E(P)]\!]' \subseteq [\![P]\!]$ and $[\![P]\!] \subseteq [\![E(P)]\!]'$, which we prove below.

($[\![E(P)]\!]' \subseteq [\![P]\!]$). Let p be a state in an LTS such that $P \vdash_\theta p$, for some θ. Define a relation $Q \mathcal{R} q \iff \exists_{\nu,s}\, (Q, \nu) \vdash_\theta q \wedge \nu \sqsubseteq \theta \wedge P \xrightarrow{s} Q \wedge p \xrightarrow{s} q$. Next, we show that $\mathcal{R}$ is a modal refinement relation.

1. Let $Q \xrightarrow{a}_\Box Q'$ and $Q \mathcal{R} q$. Then, $(Q, \nu) \mathcal{R} q \wedge \nu \sqsubseteq \theta \wedge P \xrightarrow{s} Q \wedge p \xrightarrow{s} q$, for some ν, s. Clearly, $sa \in \mathbf{tr}(P)$. Let $\nu' =_{sa} \nu \uparrow \{(Q, a, Q') \mapsto \theta(sa)(Q \xrightarrow{a}_{sa} Q')\}$. Note that $\theta(sa)(Q \xrightarrow{a}_{sa} Q')$ can be either L or R. Then, we find $\nu' \sqsubseteq \theta$. Now from the transfer property of $\vdash_\theta$ we find a q' such that $q \xrightarrow{a} q' \wedge (Q', \nu') \vdash_\theta q'$. Hence, $Q' \mathcal{R} q'$.
2. Let $q \xrightarrow{a} q'$ and $Q \mathcal{R} q$. Trivial.

($[\![P]\!] \subseteq [\![E(P)]\!]'$). Let p be some state in an LTS such that $P \preceq p$. Let $\mathbf{tr}_m(p)$ denote the set of maximal traces from the state p. (Note that a maximal trace is either an infinite trace or a finite trace that leads to a deadlock state.) For every maximal trace $s \in \mathbf{tr}_m(p)$, we know that there is a unique execution e starting from $[p]_\leftrightarrow$ such that $\mathrm{dom}(e) = s$, where $[p]_\leftrightarrow$ is the transition system modulo strong bisimulation defined in the standard way. [1] Therefore, for any maximal trace $s \in \mathbf{tr}_m(p)$, we denote the corresponding unique execution by e_s. Define total configuration vectors θ_p (for every $P \preceq p$) in the following way:

$$\theta_p(s)(Q \xrightarrow{a}_{s'} Q') = \begin{cases} L & \text{if } s \in \mathbf{tr}_m(p) \wedge s = s' \wedge \exists_{s'' \preceq s}\, Q \preceq e_s(s'') \wedge Q' \preceq e_s(s''a) \\ R & \text{otherwise} \end{cases},$$

[1] An execution e starting from $[p]_\leftrightarrow$ is a function $e : \{s' \mid s' \preceq s\} \rightarrow 2^{\mathbf{P}}$ (for any $s \in \mathbf{tr}(p)$) such that $e(\varepsilon) = [p]_\leftrightarrow$ and $e(s') \xrightarrow{a} e(s'a) \iff \exists_{q,q'}\, q \in e(s') \wedge q' \in e(s'a) \wedge q \xrightarrow{a} q'$, for every $s' \preceq s$. Here $\preceq$ is the prefix relation between any two words.

where $Q \preceq X$ is defined as $\forall_q\, q \in X \implies Q \preceq q$. Next, we define a relation $\mathcal{R}_{\theta_p}$ as follows:

$$(Q,\nu)\mathcal{R}_{\theta_p} q \iff \nu \sqsubseteq \theta_p \wedge \exists_{s,s'}\ s \in \mathbf{tr}_m(p) \wedge s' \preceq s \wedge P \xrightarrow{s'}\!\!\!\!\!\rightarrow\ Q \wedge Q \preceq e_s(s') \wedge q \in e_s(s'),$$

Next, we show that $\mathcal{R}_{\theta_p}$ is a product derivation relation.

1. Let $(Q,\nu) \xrightarrow{a,\nu'} (Q',\nu')$, $\nu' \sqsubseteq \theta_p$, and $(Q,\nu)\mathcal{R}_{\theta_p} q$. Then, from the construction of $\mathcal{R}_{\theta_p}$ we have $\nu \sqsubseteq \theta_p \wedge s \in \mathbf{tr}_m(p) \wedge P \xrightarrow{s'}\!\!\!\!\!\rightarrow\ Q \wedge Q \preceq e_s(s') \wedge q \in e_s(s')$, for some $s \in \mathbf{tr}_m(p)$, $s' \preceq s$.
   (a) Let $Q \xrightarrow{a}_{\Box} Q'$. Then, $Q \preceq q$ (because $Q \preceq e_s(s') \wedge q \in e_s(s')$) and from the transfer property of modal refinement we find $q \xrightarrow{a} q' \wedge Q' \preceq q'$, for some q'. Thus, there is a maximal trace $\bar{s} \in \mathbf{tr}_m(p)$ such that $s'a \preceq \bar{s}$, $q \in e_{\bar{s}}(s')$, and $q' \in e_{\bar{s}}(s'a)$. Hence, due to the construction of $\mathcal{R}_{\theta_p}$, we get $(Q',\nu')\mathcal{R}_{\theta_p} q'$.
   (b) Let $Q \xrightarrow{a}_{\Diamond\setminus\Box} Q'$. Then, $\nu' =_s \nu \uparrow \{(Q,a,Q') \mapsto L\}$. But $\nu' \sqsubseteq \theta_p$, so $\theta_p(s)(Q \xrightarrow{a}_s Q') = L$. Due to the definition of θ_p, we find $q \xrightarrow{a} q', q' \in e_s(s'a), Q' \preceq e_s(s'a)$, for some q'. Thus, by the construction of $\mathcal{R}_{\theta_p}$, we have $(Q',\nu')\mathcal{R}_{\theta_p} q'$.
2. Let $q \xrightarrow{a} q'$ and $(Q,\nu)\mathcal{R}_{\theta_p} q$. Then, from the construction of $\mathcal{R}_{\theta_p}$ we have $\nu \sqsubseteq \theta_p \wedge s \in \mathbf{tr}_m(p) \wedge P \xrightarrow{s'}\!\!\!\!\!\rightarrow\ Q \wedge Q \preceq e_s(s') \wedge q \in e_s(s')$, for some $s \in \mathbf{tr}_m(p), s' \preceq s$. By using the fact $Q \preceq q$ and the transfer property of a modal refinement relation, we find $Q \xrightarrow{a}_{\Diamond} Q' \wedge Q' \preceq q'$, for some Q'. Furthermore, since q is reachable from p and $q \xrightarrow{a} q'$, so there is a maximal trace $\bar{s} \in \mathbf{tr}_m(p)$ such that $e_{\bar{s}}(\bar{s}') = q$ and $e_{\bar{s}}(\bar{s}'a) = q'$ (for some $\bar{s}' \preceq \bar{s}$). So let $\nu' =_{\bar{s}} \nu \uparrow (Q,a,Q') \mapsto L$ and thus $\nu' \sqsubseteq \theta_p$. Moreover, we find $(Q,\nu) \xrightarrow{a,\nu'} (Q',\nu')$. Thus, by the construction of $\mathcal{R}_{\theta_p}$, we have $(Q',\nu')\mathcal{R}_{\theta_p} q'$. □

Theorem 2. *The class of MTSs is less expressive than the class of PL-LTSs.*

Proof. Due to Theorem 1, we know that PL-LTS is at least as expressive as MTS. It hence remains to prove that MTS is not at least as expressive as PL-LTS, which we show by means of the example depicted in Figure 4.

We prove by contradiction that the PL-LTS depicted in Figure 4 (left), where $P = a.\mathbf{Nil} \oplus b.\mathbf{Nil}$ cannot be encoded using any sound encoding (satisfying Definition 11) to an MTS. To show this, observe the transition systems of the derived LTSs p and q drawn in Figure 4 under $\theta = L$ and $\theta' = R$.

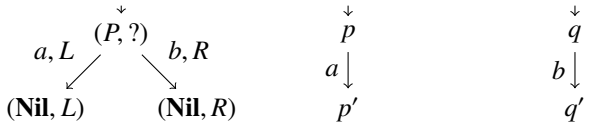


Figure 4. A PL-LTS (left) that cannot be encoded as an MTS.

Suppose there is an encoding E satisfying Definition 11. Clearly, $(P,?) \vdash_\theta p$ and $(P,?) \vdash_{\theta'} q$. Then by correctness of E we have $E(P,?) \preceq p$ and $E(P,?) \preceq q$. Thus, we can derive the following transitions (for some modal states P_a, P_b) from the transfer property of modal refinement:

$$E(P,?) \xrightarrow{a}_{\Diamond} P_a \qquad E(P,?) \xrightarrow{b}_{\Diamond} P_b\ .$$

Therefore, there exists a state r of the following form: $r \xrightarrow{a} r'$ and $r \xrightarrow{b} r''$ (for some r', r'') such that $E(P) \preceq r$. And by correctness of E we get $(P,?) \vdash_\theta r$ or $(P,?) \vdash_{\theta'} r$. However, $(P,?) \nvdash_\theta r$ and $(P,?) \nvdash_{\theta'} r$.

□

Now that we have related the expressiveness of PL-LTSs and MTSs, we move to the other end of the spectrum, namely to the comparison of PL-LTSs and FTSs, which is achieved by means of the following two theorems.

Theorem 3. *The class of FTSs is at-least as expressive as the class of PL-LTSs.*

Proof. Let $(\mathbb{P}\times\{L,R,?\}^I, A, \rightarrow)$ be a PL-LTS. The corresponding FTS is denoted by $(\mathbb{P}\times\{L,R,?\}^I, A, F, \rightarrow', \Lambda)$, where:

- $F = \bigcup_{i\in I}\{L_i, R_i\}$.
- $\Lambda = \bigwedge_{i\in I} \neg(L_i \wedge R_i)$.
- The transition relation $\rightarrow'$ is defined in the following way:

$$\frac{(P,\nu) \xrightarrow{a,\nu} (Q,\nu)}{(P,\nu) \overset{a,\top}{\rightarrow'} (Q,\nu)} \qquad \frac{(P,\nu) \xrightarrow{a,\nu'} (Q,\nu') \quad \phi = \nu'(i)_i \quad \Xi(i,\nu,\nu')}{(P,\nu) \overset{a,\phi}{\rightarrow'} (Q,\nu')},$$

where $\Xi(i,\nu,\nu') \iff \nu'(i) \neq \nu(i) \wedge \forall_{j\neq i}\, \nu'(j) = \nu(j)$.

For any $(P,\nu) \in \mathbb{P}\times\{L,R,?\}^I$, we fix $E(P,\nu) = (P,\nu)$. Let $[\![(P,\nu)]\!] = \{p \mid \exists_{\theta\in\{L,R\}^I}\ (P,\nu) \vdash_\theta p\}$ and $[\![(P,\nu)]\!]' = \{p \mid \exists_{\lambda\in\Lambda}\ (P,\nu) \vdash_\lambda p\}$. In the next step, we need to show that $[\![(P,\nu)]\!] = [\![(P,\nu)]\!]'$.
($[\![(P,\nu)]\!] \subseteq [\![(P,\nu)]\!]'$): Let $p \in [\![(P,\nu)]\!]$. Then $(P,\nu) \vdash_\theta p$, for some $\theta \in \{L,R\}^I$. Define a configuration $\lambda_\theta \in \Lambda$ as follows: $\lambda_\theta(L_i) = \top \iff \theta(i) = L$ and $\lambda_\theta(R_i) = \top \iff \theta(i) = R$. Furthermore, consider the following relation $\mathcal{R}_{\lambda_\theta}$ such that $(Q,\nu')\mathcal{R}_{\lambda_\theta} q \iff (Q,\nu') \vdash_\theta q$. It is straightforward to show that $\mathcal{R}_{\lambda_\theta}$ is a product derivation relation.

($[\![(P,\nu)]\!]' \subseteq [\![(P,\nu)]\!]$): Let $p \in [\![(P,\nu)]\!]'$. Then $P \vdash_\lambda p$ for some $\lambda \in \Lambda$. Let $\theta \in \{L,R\}^I$ be a configuration vector defined as $\theta_\lambda(i) = L \iff \lambda(L_i) = \top$ and $\theta_\lambda(i) = R \iff \lambda(R_i) = \top$. Define a relation $\mathcal{R}_{\theta_\lambda}$ such that $(Q,\nu')\mathcal{R}_{\theta_\lambda} q \iff (Q,\nu') \vdash_{\theta_\lambda} q$. It is straightforward to verify that $\mathcal{R}_{\theta_\lambda}$ is a product derivation relation for PL-LTS. □

Theorem 4. *The class of PL-LTSs is less expressive than the class of FTSs.*

Proof. Due to Theorem 3, we know that FTSs are at least as expressive as PL-LTSs. It remains to show that PL-LTSs are not at-least as expressive as FTSs.

Consider the FTS as shown in Fig 5, where f_a, f_b, f_c are three distinct features and the set of valid products Λ is defined as the smallest set of functions satisfying the following constraint:

$$(f_a \implies (\neg f_b \wedge \neg f_c)) \wedge (f_b \implies (\neg f_a \wedge \neg f_c)) \wedge (f_c \implies (\neg f_a \wedge \neg f_b)).$$

Through a proof by contradiction, we show that there is no encoding E that can transform the FTS P in the correct way. Suppose otherwise there is an encoding E of P into an PL-LTS whose configuration vectors are of type $\{L,R,?\}^I$ (for some index set I) such that $[\![P]\!] = [\![E(P)]\!]'$, where $[\![P]\!] = \{p \mid \exists_{\lambda\in\Lambda}\ P \vdash_\lambda p\}$, $E(P) = (Q,\nu_0)$ (for some state Q and configuration vector $\nu_0 \in \{L,R,?\}^I$ in an PL-LTS), and $[\![E(P)]\!]' = \{p \mid \exists_{\theta\in\{L,R\}^I}(Q,\nu) \vdash_\theta p\}$.

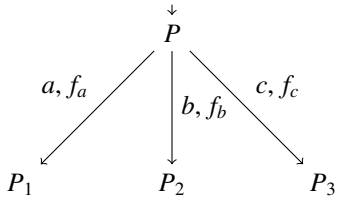


Figure 5. An FTS that cannot be encoded as an PL-LTS.

Clearly, the transition systems $(\{p_x, p'_x\}, \{x\}, \{(p_x, x, p'_x)\}$ (for $x \in \{a,b,c\}$) are three valid products of the given FTS P, i.e., $\{p_a, p_b, p_c\} \subseteq [\![P]\!]$. So from the correctness requirement of E we have $\{p_a, p_b, p_c\} \subseteq [\![E(P)]\!]'$. Let $\theta_a, \theta_b, \theta_c$ be the corresponding total configuration vectors that derives the products p_a, p_b, p_c, respectively. Thus, for every $x \in \{a,b,c\}$ we have $\nu_0 \sqsubseteq \theta_x$. Furthermore, from the transfer property of product derivation we find $(Q,\nu_0) \xrightarrow{x,\nu_x} (Q_x,\nu_x)$ such that $\nu_x \sqsubseteq \theta_x$, for $x \in \{a,b,c\}$. Clearly, $\nu_x \neq \nu_0$ (for any $x \in \{a,b,c\}$) because otherwise we can derive a transition system which contains choices of a,b or b,c or a,c. For instance, if $\nu_a = \nu_0$ then the transfer property of product derivation ensures that $\exists_q\ p_b \xrightarrow{a} q$ because $\nu_a = \nu_0 \sqsubseteq \theta_b$. Therefore, let $i_a, i_b, i_c \in I$ be the unique elements such that for every $x \in \{a,b,c\}$ we have $\nu_x(i_x) \neq \nu_0(i_x) \wedge \forall_{i\neq i_x}\, \nu_x(i) = \nu_0(i)$ (recall Condition (7)(2)).

Next, we show that if $i_a \neq i_c \wedge i_b \neq i_c \wedge i_a \neq i_b$ then we can derive a product which has a choice between a, c. Since i_a, i_b are the only elements whose values are changed by ν_a, ν_b, so from Condition (7)(3) we have $\nu_0(i_a) = \nu_0(i_b) =?$. Define a function θ'_c as follows:

$$\theta'_c(i) = \begin{cases} \theta_c(i) & \text{if } i \neq i_a \wedge i \neq i_b \\ \theta_a(i) & \text{if } i = i_a \\ R & \text{if } i = i_b \wedge \theta_b(i_b) = L \\ L & \text{if } i = i_b \wedge \theta_b(i_b) = R \end{cases}.$$

Next, we show that $\nu_c \sqsubseteq \theta'_c$. If $i \neq i_a \wedge i \neq i_b$ then clearly $\nu_c(i) \sqsubseteq \theta'_c(i)$ because $\theta'_c(i) = \theta_c(i)$. If $i = i_a$ or $i = i_b$ then $\nu_c(i) = \nu_0(i) =?$. Thus, $\nu_c(i) \sqsubseteq \theta'_c(i)$. Thus, $\nu_c \sqsubseteq \theta'_c$.

Next, we show that $\nu_a \sqsubseteq \theta'_c$. If $i \neq i_a \wedge i \neq i_b$ then $\nu_a(i) = \nu_0(i)$. Thus, $\nu_a(i) \sqsubseteq \theta'_c(i)$ because $\nu_0(i) \sqsubseteq \theta_c(i) \wedge \theta_c(i) = \theta'_c(i)$. If $i \neq i_a \wedge i = i_b$ then $\nu_a(i) = \nu_0(i) =?$. Thus, $\nu_a(i) \sqsubseteq \theta'_c(i)$. Lastly, if $i = i_a$ then $\nu_a(i) \sqsubseteq \theta'_c(i)$ because $\theta'_c(i) = \theta_a(i)$. Thus, $\nu_a \sqsubseteq \theta'_c$. As a result, we can derive a product that contains a choice between a, c by using θ'_c; however, such a product is clearly not a valid product of the given FTS P as it violates the condition Λ.

On the other hand, if $i_a = i_c$ then we show that using θ_b we can derive a product which has a choice between either a, b or b, c. It suffices to show that $\nu_a(i_a) \sqsubseteq \theta_b(i_a)$ or $\nu_c(i_a) \sqsubseteq \theta_b(i_a)$ because for every $i \neq i_a$ we have $\nu_a(i) = \nu_c(i) = \nu_0(i) \sqsubseteq \theta_b(i)$. We claim that $\nu_a(i_a) \neq?$. Suppose otherwise $\nu_a(i_a) =?$. Then, since $i_a = i_c$ we find $\nu_a \sqsubseteq \theta_c$. As a result, from the transfer property of $\vdash_{\theta_c}$ there must be a q such that $p_c \xrightarrow{a} q$ otherwise $E(P) \nvdash_{\theta_c} p_c$. Thus, $\nu_a(i_a) \neq?$. Likewise, we can prove that $\nu_c(i_a) \neq?$. Thus, $\nu_a(i_a), \nu_c(i_a) \in \{L, R\}$. And since θ_b is a function whose co-domain is the set $\{L, R\}$ we have either $\nu_a(i_a) \sqsubseteq \theta_b(i_a)$ or $\nu_c(i_a) \sqsubseteq \theta_b(i_a)$. Hence, we can derive a product that contains a choice between either a, b or b, c by using θ_b; however, such a product is clearly not a valid product of the given FTS P as it violates the condition Λ. Likewise, if $i_b = i_c$ (or $i_a = i_b$) then we can show that using θ_a (θ_c) we can derive a product which has a choice between either a, b (a, c) or a, c (b, c). □

We can hence summarize the results of this section by the following diagram:

$$\text{MTSs} \longrightarrow \text{PL-LTSs} \longrightarrow \text{FTSs},$$

where the arrow $\longrightarrow$ indicates the "less-expressive-than" relation, i.e., the existence of an encoding from one product line structure into another and the lack of encoding in the other directions. In other words, the class of MTSs (FTSs) is the least (most) expressive product line structure considered in this paper. The fact that MTSs are less expressive than FTSs follows from the transitivity of the "less-expressive-than" relation; to emphasize this fact, we give the evidence of the lack of encoding from FTSs to MTSs in the following example.

Example 8. *Consider the FTS drawn (left) in Figure 6 with the set of features $F = \{f, f'\}$ and the set of valid product configuration $\Lambda = \{\lambda, \lambda'\}$ with $\lambda(f) = \top, \lambda(f') = \bot$ and $\lambda'(f) = \bot, \lambda'(f') = \top$. The transition systems of the derived processes P and Q under λ and λ', respectively, are drawn in Figure 6. Now by contradiction we show that there is no encoding E satisfying Definition 11.*

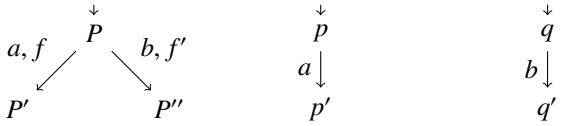


Figure 6. An FTS (left) that cannot be encoded as an MTS.

Suppose there is an encoding E satisfying Definition 11. Clearly, $P \vdash_\lambda p$ and $P \vdash_{\lambda'} q$. Then by correctness of E we have $E(P) \preceq p$ and $E(P) \preceq q$. Thus, we can derive the following transitions (for some modal states P_a, P_b) from the transfer property of modal refinement:

$$E(P) \xrightarrow{a}_\Diamond P_a \qquad E(P) \xrightarrow{b}_\Diamond P_b \ .$$

Therefore, there exists a state r of the following form: $r \xrightarrow{a} r'$ and $r \xrightarrow{b} r''$ (for some r', r'') such that $E(P) \preceq r$. And by correctness of E we get $P \vdash_\lambda r$ or $P \vdash_{\lambda'} r$. However, $P \nvdash_\lambda r$ and $P \nvdash_{\lambda'} r$.

4. Testing Pre-Orders for SPLs

In Section 2, we reviewed three different notions of product derivations based on a particular product line structure. These notions are intensional in nature, i.e., they require the products to be modeled completely as LTSs, and moreover, their models must be available in their entirety during testing. This assumption is rather unrealistic for practical systems. In practice, one needs an extensional notion of testing that can be used to generate a test-suite from a product-line specification (e.g., an MTS) in an offline or on-the-fly manner, in order to test a black-box implementation. Based on the foundational studies carried out in [20–22], such notions have been developed and extensively studied for various LTS-based formalisms [22, 34]; however, we are not aware of any such notion for MTSs, PL-LTSs, and FTSs (the only exceptions being our recent work [35, 36], as well as the recent work by Devroey et al. [37, 38]). In the remainder of this section, we adopt the testing framework of [23] and adapt its notion of test to MTSs, PL-LTSs, and FTSs to characterize the corresponding product derivation relation for the respective product-line structure.

The notions elaborated in this section lay the theoretical connection between the intensional (trace-based comparison) and the extensional (test-case execution) notions of comformance. In order to turn this theory into a practical testing scheme, some degrees of unboudedness have to be tamed: firstly, a fault-model (regarding the implementation) [39], a notion of coverage [37], or a test-selection algorithm [40] has to be adopted to choose a finite set of test cases. Moreover, some assumptions about valid products and the interaction of their features (combined with the aforementioned methods or a bound on the maximum length of test-cases) can be used to select a finite set of incremental test-suites for various products [35, 36].

4.1. Modal Refinement as a Testing Pre-Order

Consider a set of test expressions T, ranged over by t, generated by the following grammar [23]: [2]

$$t ::= \mathsf{SUCC} \mid \mathsf{FAIL} \mid at \mid \tilde{a}t \mid t_1 \wedge t_2 \mid t_1 \vee t_2 \mid \forall t \mid \exists t.$$

Intuitively, SUCC and FAIL denote the successful and the failed tests, respectively, i.e., for every MTS the test SUCC (FAIL) will always pass (fail). The expression at tests the existence of a must-transition labeled a and then examines the sub-test t. Furthermore, if an MTS refuses to perform the must-transition a, the verdict for this test is fail. The expression $\tilde{a}t$ tests the existence of a may-transition labeled a and then examines the sub-test t. Furthermore, if an MTS refuses to perform the may-transition a, then the verdict for this test is success (pass). The tests of the form $t_1 \wedge t_2$ and $t_1 \vee t_2$ represent testing different copies of a machine using the sub-tests and subsequently, combining the results [20]. The tests of the form $\forall t$ and $\exists t$ represent global testing by, respectively, quantifying universally and existentially over runs of sub-test t.

Given a modal specification P and an implementation p (modeled as an LTS), the main idea is to assert indirectly whether the implementation p is a valid product of the specification P, i.e., whether they are related by a modal refinement relation. (Throughout this section, we use P to denote a state of a modal specification and p to denote the state of an LTS implementation.) For this purpose, we need a concept of interaction between a test case and a state in an MTS (and an LTS). To this end, we recall the notion of *experiment expression* E [23], generated by the following grammars:

$$E ::= \top \mid \bot \mid (t \parallel P) \mid E_1 \wedge E_2 \mid E_1 \vee E_2 \mid \forall E \mid \exists E.$$
$$\mathcal{E} ::= \top \mid \bot \mid (t \parallel p) \mid \mathcal{E}_1 \wedge \mathcal{E}_2 \mid \mathcal{E}_1 \vee \mathcal{E}_2 \mid \forall \mathcal{E} \mid \exists \mathcal{E}.$$

Figure 7 provides the operational interpretation of experiment expressions over an MTS. Note that only the rules of the expressions $at \parallel P$ and $\tilde{a}t \parallel P$ (i.e., rules 3-6) are modified with respect to the original rules presented in [23], while the rest of the operational rules are quoted verbatim for the sake of completeness. In particular, we define a transition relation $\rightarrow$ between any two experiment expressions as the smallest relation satisfying the rules of Figure 7. Note that we do not need a separate set of rules to specify the experiment expressions interacting with an LTS, because they can also be derived from the rules of Figure 7 by considering the transitions of LTS as both may and must transitions.

[2] Throughout this section, we assume that the product line structure under investigation is image finite.

$$\mathsf{SUCC} \parallel P \to \top \quad (1) \qquad \mathsf{FAIL} \parallel P \to \bot \quad (2) \qquad \frac{P \xrightarrow{a}_{\Box} P'}{at \parallel P \to t \parallel P} \quad (3) \qquad \frac{\nexists_{P'}\ P \xrightarrow{a}_{\Box} P'}{at \parallel P \to \bot} \quad (4)$$

$$\frac{P \xrightarrow{a}_{\Diamond} P'}{\tilde{a}t \parallel P \to t \parallel P} \quad (5) \qquad \frac{\nexists_{P'}\ P \xrightarrow{a}_{\Diamond} P'}{\tilde{a}t \parallel P \to \top} \quad (6) \qquad (t_1 \wedge t_2) \parallel P \to (t_1 \parallel P) \wedge (t_2 \parallel P) \quad (7)$$

$$\frac{E_1 \to E_1'\ E_2 \to E_2'}{E_1 \wedge E_2 \to E_1' \wedge E_2'} \quad (8) \qquad \top \wedge E \to E \quad (9) \qquad \bot \wedge E \to \bot \quad (10) \qquad E \wedge \top \to E \quad (11)$$

$$E \wedge \bot \to \bot \quad (12) \qquad (t_1 \vee t_2) \parallel P \to (t_1 \parallel P) \vee (t_2 \parallel P) \quad (13) \qquad \frac{E_1 \to E_1'\ E_2 \to E_2'}{E_1 \vee E_2 \to E_1' \vee E_2'} \quad (14)$$

$$\top \vee E \to \top \quad (15) \qquad \bot \vee E \to E \quad (16) \qquad E \vee \top \to \top \quad (17) \qquad E \vee \bot \to E \quad (18)$$

$$\forall t \parallel P \to \forall (t \parallel P) \quad (19) \qquad \forall \bot \to \bot \quad (20) \qquad \forall \top \to \top \quad (21) \qquad \frac{\delta(E) = \{E_1, \cdots, E_n\}}{\forall E \to \bigwedge_{i=1}^{n} \forall E_i} \quad (22)$$

$$\exists t \parallel P \to \exists (t \parallel P) \quad (23) \qquad \exists \bot \to \bot \quad (24) \qquad \exists \top \to \top \quad (25) \qquad \frac{\delta(E) = \{E_1, \cdots, E_n\}}{\exists E \to \bigvee_{i=1}^{n} \exists E_i} \quad (26)$$

Figure 7. Operational Interpretation of Test Experiments

Once we have a transition system whose states are experiment expressions interacting with either a specification or an implementation, we can use this structure to define the set of results of evaluating a test on a process. The outcome of a single test is either successful or unsuccessful, which can be modeled as a two-point domain $\mathbb{O} = \bot \leq \top$. However, due to nondeterminism, sets of outcomes are required to get the results of all possible runs (cf. [23]). These outcomes are modeled using a set of truth values (or more precisely, using the Plotkin powerdomain $\mathcal{P}[\mathbb{O}] = \{\bot\} \sqsubseteq \{\bot, \top\} \sqsubseteq \{\top\}$). The semantics of the operators $\wedge, \vee, \forall, \exists$ over the Plotkin powerdomain $\mathcal{P}[\mathbb{O}]$ can be found in [23]. In addition, given an MTS $(\mathbb{P}, A, \to_{\Diamond}, \to_{\Box})$, we define the function $O : T \times \mathbb{P} \to \mathcal{P}[\mathbb{O}]$ in the following way:

$$O(t, P) = \{\top \mid (t \parallel P) \longrightarrow\!\!\!\!\!\rightarrow \top\} \cup \{\bot \mid (t \parallel P) \longrightarrow\!\!\!\!\!\rightarrow \bot\},$$

where $\longrightarrow\!\!\!\!\!\rightarrow$ is the reflexive and transitive closure of the transition relation $\to$ defined in Figure 7. Similar to the operational semantics, we re-use the same notation to denote the result of executing a test expression of a test on an LTS, i.e., for an LTS $(\mathbf{P}, A, \to)$, we write $O(t, p)$ to denote the results of executing test t on p.

The following property is immediate from the rules of Figure 7 (in particular, rules 19 and 23) on experiment expressions.

Lemma 3. *Let P be a modal specification. Then, for any test expression t we have*

$$O(\forall t, P) = \forall O(t, P) \qquad \text{and} \quad O(\exists t, P) = \exists O(t, P).$$

Before we turn our attention to the characterization of modal refinement as a testing pre-order, we first give a semantic preserving transformation $[\![]\!]_{\mathcal{A}}$ (Lemma 4) that transforms an HML formulae (interpreted over MTSs due to [41]) into the set of tests T. We will use this transformation in the proof of Theorem 5 to establish a link between testing pre-order $\sqsubseteq$ and the modal refinement relation $\preceq$.

Consider the set of all HML formulae Φ generated by the following grammar:

$$\varphi ::= \ \bot \mid \top \mid \langle a \rangle \varphi \mid [a]\varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi'.$$

The semantics of $\bot, \top, \wedge, \vee$ is standard, while the nonstandard semantics of $\langle a \rangle \varphi, [a]\varphi$ is given as follows [41]:

1. $P \models \langle a \rangle \varphi \ \Leftrightarrow \ \exists_{P'} \ P \xrightarrow{a}_{\Box} P' \wedge P' \models \varphi$.
2. $P \models [a]\varphi \ \Leftrightarrow \ \forall_{P'} \ P \xrightarrow{a}_{\Diamond} P' \Rightarrow P' \models \varphi$.

Following [23], we give a transformation $[\![\_]\!]_{\mathcal{A}} : \Phi \to T$ of HML formulae to the set of test expressions.

$$\begin{aligned} [\![\top]\!]_{\mathcal{A}} &= \mathsf{SUCC}, & [\![\bot]\!]_{\mathcal{A}} &= \mathsf{FAIL}, \\ [\![\varphi \wedge \varphi']\!]_{\mathcal{A}} &= [\![\varphi]\!]_{\mathcal{A}} \wedge [\![\varphi']\!]_{\mathcal{A}}, & [\![\varphi \vee \varphi']\!]_{\mathcal{A}} &= [\![\varphi]\!]_{\mathcal{A}} \vee [\![\varphi']\!]_{\mathcal{A}}, \\ [\![[a]\varphi]\!]_{\mathcal{A}} &= \forall \tilde{a} [\![\varphi]\!]_{\mathcal{A}}, & [\![\langle a \rangle \varphi]\!]_{\mathcal{A}} &= \exists a [\![\varphi]\!]_{\mathcal{A}}. \end{aligned}$$

By setting the above technical machinery, we now prove that an MTS satisfies a HML formula φ if and only if it passes the test $[\![\varphi]\!]_{\mathcal{A}}$.

Lemma 4. *Let P and p be a state in an MTS and an LTS, respectively. Then, for any $\varphi \in \Phi$ we have*

$$P \models \varphi \ \Leftrightarrow \ O([\![\varphi]\!]_{\mathcal{A}}, P) = \{\top\} \qquad \textit{and} \qquad p \models \varphi \ \Leftrightarrow \ O([\![\varphi]\!]_{\mathcal{A}}, p) = \{\top\}.$$

Proof. The proof is by induction on φ; the cases for $\bot, \top, \wedge, \vee$ are straightforward.

1. Let $\varphi = [a]\varphi'$ and $P \models \varphi$. Then,

$$\begin{aligned} & \forall_{P'} \ P \xrightarrow{a}_{\Diamond} P' \Rightarrow P' \models \varphi' \\ \Leftrightarrow \ & \forall_{P'} \ P \xrightarrow{a}_{\Diamond} P' \Rightarrow O([\![\varphi']\!]_{\mathcal{A}}, P') = \{\top\} && \text{(Induction hypothesis)} \\ \Leftrightarrow \ & O(\tilde{a}[\![\varphi']\!]_{\mathcal{A}}, P) = \{\top\} && \text{(Rule 5 and Definition of } O\text{)} \\ \Leftrightarrow \ & \forall O(\tilde{a}[\![\varphi']\!]_{\mathcal{A}}, P) = \{\top\} && \text{(Truth-table [23]: } \forall\{\top\} = \{\top\}\text{)} \\ \Leftrightarrow \ & O(\forall \tilde{a}[\![\varphi']\!]_{\mathcal{A}}, P) = \{\top\} && \text{(Lemma 3: } O(\forall t, p) = \forall O(t, P)\text{)} \\ \Leftrightarrow \ & O([\![[a]\varphi']\!]_{\mathcal{A}}, P) = \{\top\}. \end{aligned}$$

2. Let $\varphi = \langle a \rangle \varphi'$ and $P \models \varphi$. Then,

$$\begin{aligned} & \exists_{P'} \ P \xrightarrow{a}_{\Box} P' \wedge P' \models \varphi' \\ \Leftrightarrow \ & \exists_{P'} \ P \xrightarrow{a}_{\Box} P' \wedge O([\![\varphi']\!]_{\mathcal{A}}, P') = \{\top\} && \text{(Induction hypothesis)} \\ \Leftrightarrow \ & \top \in O(a[\![\varphi']\!]_{\mathcal{A}}, P) && \text{(Rule 5 and Definition of } O\text{)} \\ \Leftrightarrow \ & \exists O(a[\![\varphi']\!]_{\mathcal{A}}, P) = \{\top\} && \text{(Truth table [23]: } \exists\{\bot, \top\} = \{\top\} \text{ and } \exists\{\top\} = \{\top\}\text{)} \\ \Leftrightarrow \ & O(\exists a[\![\varphi']\!]_{\mathcal{A}}, P) = \{\top\} && \text{(Lemma 3: } \exists O(t, p) = O(\exists t, p)\text{)} \\ \Leftrightarrow \ & O([\![\langle a \rangle \varphi']\!]_{\mathcal{A}}, P) = \{\top\}. \end{aligned}$$

□

The following theorem is the first attempt towards characterization; namely, it shows that if the observations obtained from test-cases on an implementation are all allowed by the product line, then the implementation is a valid product (a modal refinement) of the product line.

Theorem 5. *Let P and p be states in an MTS and an LTS, respectively. Then,*

$$\forall_{t \in T} \ O(t, P) \sqsubseteq O(t, p) \ \Rightarrow \ P \preceq p.$$

Proof. Suppose $\forall_{t \in T} \ O(t, P) \sqsubseteq O(t, p)$. In lieu of the modal characterization given by Boudol and Larsen [41, Theorem 3.1], we show that $p \models \varphi$ whenever $P \models \varphi$, for any $\varphi \in \Phi$. Suppose $P \models \varphi$. Then, from Lemma 4 we know that $O([\![\varphi]\!]_{\mathcal{A}}, P) = \{\top\}$. Since the element $\{\top\}$ is the maximum in the Plotkin powerdomain $\mathcal{P}[\mathbb{O}]$ and $O([\![\varphi]\!]_{\mathcal{A}}, P) \sqsubseteq O([\![\varphi]\!]_{\mathcal{A}}, p)$ we know that $O([\![\varphi]\!]_{\mathcal{A}}, p) = \{\top\}$. From Lemma 4, we conclude that $p \models \phi$. □

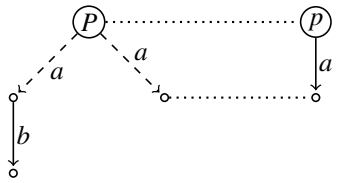


Figure 8. A counterexample for the converse of Theorem 5.

To see why the converse of Theorem 5 does not hold, consider the states P and p given in Fig. 8, where dashed transitions denote may transitions and solid transitions denote must transitions. The dotted lines show the witnessing refinement relation between P and p; thus, $P \preceq p$. Consider the test $t = \tilde{a}b\mathsf{SUCC}$. Clearly, $O(t, P) = \{\bot, \top\}$ and $O(t, p) = \{\bot\}$. However, $\{\bot, \top\} \not\sqsubseteq \{\bot\}$.

Thus, in order to obtain a full characterization of modal refinement, we need to restrict ourselves to the set of test expressions $T' \subseteq T$ generated by the grammar given below.

Corollary 1. *Let P and p be states in an MTS and an LTS, respectively. Let $T' \subseteq T$ be the set of tests generated by the following grammar:*

$$t ::= \mathsf{SUCC} \mid \mathsf{FAIL} \mid \exists a t \mid \forall \tilde{a} t \mid t_1 \wedge t_2 \mid t_1 \vee t_2.$$

Then, $\forall_{t \in T'}\ O(t, P) \sqsubseteq O(t, p) \ \Leftrightarrow\ P \preceq p$.

It follows also from Corollary 1 that if an LTS is not a valid product of an MTS, i.e., $P \not\preceq p$, then it is sufficient to find a test $t \in T$ such that $O(t, P) \not\sqsubseteq O(t, p)$. Moreover, for such an invalid product there always exists a test-case, which tells it apart from the product line.

Example 9. *Recall the MTS given in Fig. 2(a) and represent it by P. Consider an LTS given in Fig. 9 and represent it by p.*

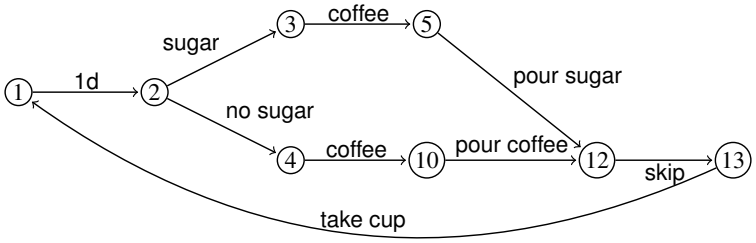


Figure 9. An invalid product for the MTS given in Fig. 2(a).

Observe that by adopting the test $t = \exists$1d sugar coffee poursugar pourcoffee SUCC we can show that p is not a valid product of P because $O(t, P) = \{\top\}$ and $O(t, p) = \{\bot\}$. Thus $\{\top\} \not\sqsubseteq \{\bot\}$; hence, $P \not\preceq p$.

4.2. Testing Pre-Orders for FTSs and PL-LTSs

Similar to the case of MTSs, we are not aware of any extensional notion of testing for FTSs and PL-LTSs (the product-derivation relation of the latter is similar to the product-derivation relation of FTSs). To fill in this gap, we modify the testing framework of MTS (given in the previous section) and show how to characterize our notion of product derivation of an FTS (PL-LTS) by a testing equivalence.

Recall the set of tests T generated from the grammar given in the previous subsection. We give now an interpretation of a test $t \in T$ over an FTS $(\mathbb{P}, A, F, \rightarrow, \Lambda)$. It should not be surprising that only the semantics of the tests of the form at and $\tilde{a}t$ needs to be modified, while the semantics of the remaining operators (from Figure 7) remains

unchanged. Formally, we first define a family of transition relations, parameterized by product configurations, by modifying the rules 3-6 in the following way.

$$\frac{P \xrightarrow{a}_{\phi} P' \quad \lambda \models \phi}{at \parallel P \rightarrow_{\lambda} t \parallel P} \quad (3') \qquad \frac{\nexists_{Q,\phi}\, P \xrightarrow{a}_{\phi} Q \quad \lambda \models \phi}{at \parallel P \rightarrow_{\lambda} \bot} \quad (4')$$

$$\frac{P \xrightarrow{a}_{\phi} P' \quad \lambda \models \phi}{\tilde{a}t \parallel P \rightarrow_{\lambda} t \parallel P} \quad (5') \qquad \frac{\nexists_{Q,\phi}\, P \xrightarrow{a}_{\phi} Q \quad \lambda \models \phi}{\tilde{a}t \parallel P \rightarrow_{\lambda} \top} \quad (6') \ .$$

Second, we define the family of observation functions (parameterised by the product configurations) which essentially evaluates an experiment expression interacting with a specification modeled as an FTS.

$$O_{\lambda}(t, P) = \{\top \mid (t \parallel P) \longrightarrow\!\!\!\!\!\rightarrow_{\lambda} \top\} \cup \{\bot \mid (t \parallel P) \longrightarrow\!\!\!\!\!\rightarrow_{\lambda} \bot\} \ .$$

In a similar vein, we also give an interpretation of a test $t \in T$ over a PL-LTS $(\mathbb{P} \times \{L, R, ?\}^I, A, \rightarrow)$, where only the rules 3-6 are modified and the remaining operational rules of the operators (except $at, \tilde{a}t$) are unchanged.

$$\frac{(P, \nu) \xrightarrow{a,\nu'} (P', \nu') \quad \nu' \sqsubseteq \theta}{at \parallel (P, \nu) \rightarrow_{\theta} t \parallel (P, \nu')} \quad (3'') \qquad \frac{\nexists_{Q,\nu'}\, (P, \nu) \xrightarrow{a,\nu'} (Q, \nu') \quad \nu' \sqsubseteq \theta}{at \parallel (P, \nu) \rightarrow_{\theta} \bot} \quad (4'')$$

$$\frac{(P, \nu) \xrightarrow{a,\nu'} (P', \nu') \quad \nu' \sqsubseteq \theta}{\tilde{a}t \parallel (P, \nu) \rightarrow_{\theta} t \parallel (P, \nu')} \quad (5'') \qquad \frac{\nexists_{Q,\nu'}\, (P, \nu) \xrightarrow{a,\nu'} (Q, \nu') \quad \nu' \sqsubseteq \theta}{\tilde{a}t \parallel P \rightarrow_{\theta} \top} \quad (6'') \ .$$

Lastly, we define a function $O_{\theta} : T \times (\mathbb{P} \times \{L, R, ?\}^I) \rightarrow \mathcal{P}[\mathbb{O}]$, parametrized by configuration vectors, as follows:

$$O_{\theta}(t, P, \nu) = \{\top \mid t \parallel (P, \nu) \longrightarrow\!\!\!\!\!\rightarrow_{\theta} \top\} \cup \{\bot \mid t \parallel (P, \nu) \longrightarrow\!\!\!\!\!\rightarrow_{\theta} \bot\} \ .$$

Just like in the case of MTSs, we have the following lemma.

Lemma 5. *Let P be a state in an FTS and let λ be a product configuration. Then, for any test expression t we have*

$$O_{\lambda}(\forall t, P) = \forall O_{\lambda}(t, P) \qquad \text{and} \quad O_{\lambda}(\exists t, P) = \exists O_{\lambda}(t, P).$$

Next, we give the main result of this subsection; namely that our notion of test-cases is both sound and complete for the generalized notion product derivation.

Theorem 6. *Let P be an FTS specification, p be state in an LTS, and λ be a product. Then,*

$$P \vdash_{\lambda} p \ \Leftrightarrow \ \forall_{t \in T}\, O_{\lambda}(t, P) = O(t, p).$$

Proof. ($\Leftarrow$) Suppose otherwise, $P \nvdash_{\lambda} p$ and for all tests t, we have $O_{\lambda}(t, P) = O(t, p)$. Then we distinguish the following cases:

1. Either, there exists a, Q such that $P \xrightarrow{a}_{\phi} Q, \lambda \models \phi$, and for all q, if $p \xrightarrow{a} q$ then $Q \nvdash_{\lambda} q$. Let $p(a) = \{q \mid p \xrightarrow{a} q\}$. Due to image finiteness assumption we know that the set $p(a)$ is finite. We identify the following cases:
   (a) Suppose $p(a) = \emptyset$. Then,

$$\begin{aligned} & \top \in O_{\lambda}(a\mathsf{SUCC}, P) && (\because P \xrightarrow{a,\phi} Q \wedge \lambda \models \phi) \\ \Rightarrow\ & \exists O_{\lambda}(a\mathsf{SUCC}, P) = \{\top\} && (\text{Truth table of } \exists) \\ \Rightarrow\ & O_{\lambda}(\exists a\mathsf{SUCC}, P) = \{\top\} && (\text{Lemma 5 : } O_{\lambda}(\exists t, s) = \exists O_{\lambda}(t, s)). \end{aligned}$$

   But, $O(a\mathsf{SUCC}, p) = \{\bot\}$; thus,

$$O(a\mathsf{SUCC}, p) = \exists O(a\mathsf{SUCC}, p) = O(\exists a\mathsf{SUCC}, p) = \{\bot\}.$$

   Hence, a contradiction follows.

(b) Suppose $p(a) = \{q_1, \cdots, q_n\}$. Then, by induction hypothesis there exists sub-tests $t_1, \cdots, t_n$ such that $O_\lambda(t_i, Q) \neq O(t_i, q_i)$.

i. Either, $O_\lambda(t_i, Q) = \{\top\} \wedge O(t_i, q_i) = \{\bot\}$. Let $t' = t_1 \wedge \cdots \wedge t_n$. Consequently,

$$\begin{aligned} & \top \in O_\lambda(at', P) && (\because O_\lambda(t_i, Q) = \{\top\}) \\ \Rightarrow\ & \exists O_\lambda(at', P) = \{\top\} && (\text{Truth table of } \exists) \\ \Rightarrow\ & O_\lambda(\exists at', P) = \{\top\} && (\text{Lemma 5 : } O_\lambda(\exists t, P) = \exists O_\lambda(t, P)). \end{aligned}$$

But,

$$\begin{aligned} & O(at', p) = \{\bot\} && (\because O(t_i, q_i) = \{\bot\}) \\ \Rightarrow\ & \exists O(at', p) = \{\bot\} && (\text{Truth table: } \exists\{\bot\} = \{\bot\}) \\ \Rightarrow\ & O(\exists at', p) = \{\bot\} && (O(\exists t, p) = \exists O(t, p)[23]). \end{aligned}$$

Hence, a contradiction.

ii. Or, $O_\lambda(t_i, Q) = \{\bot\} \wedge O(t_i, q_i) = \{\top\}$. Let $t' = t_1 \vee \cdots \vee t_n$. Consequently,

$$\begin{aligned} & \bot \in O_\lambda(at', Q) && (\because O_\lambda(t_i, Q) = \{\bot\}) \\ \Rightarrow\ & \forall O_\lambda(at', Q) = \{\bot\} && (\text{Truth table of } \forall) \\ \Rightarrow\ & O_\lambda(\forall at', Q) = \{\bot\} && (\text{Lemma 5 : } O_\lambda(\forall t, Q) = \forall O_\lambda(t, Q)). \end{aligned}$$

Furthermore,

$$\begin{aligned} & O(at', p) = \{\top\} && (\text{since } O(t'_i, q_i) = \{\top\}) \\ \Rightarrow\ & \forall O(at', p) = \{\top\} && (\forall\{\top\} = \{\top\}) \\ \Rightarrow\ & O(\forall at', p) = \{\top\}. \end{aligned}$$

Hence, a contradiction.

2. Or, there exists a, q such that $p \xrightarrow{a} q$ and for all Q, ϕ, if $P \xrightarrow{a}_\phi Q \wedge \lambda \models \phi$, then $Q \not\vdash_\lambda q$. Due to image finiteness assumption, we know that the set $P(a) = \{Q \mid \exists_\phi\ P \xrightarrow{a}_\phi Q \wedge \lambda \models \phi\}$ is finite.

(a) Suppose $P(a) = \emptyset$. Then,

$$\begin{aligned} & O_\lambda(\tilde{a}\mathsf{FAIL}, P) = \{\top\} && (\because P(a) = \emptyset) \\ \Rightarrow\ & \forall O_\lambda(\tilde{a}\mathsf{FAIL}, P) = \{\top\} && (\text{Truth table: } \forall\{\top\} = \{\top\}) \\ \Rightarrow\ & O_\lambda(\forall\tilde{a}\mathsf{FAIL}, P) = \{\top\} && (\text{Lemma 5 : } \forall O_\lambda(t, P) = O_\lambda(\forall t, P)). \end{aligned}$$

But, $O(\tilde{a}\mathsf{FAIL}, p) = \{\bot\}$ (since $p \xrightarrow{a} q, O(\mathsf{FAIL}, q) = \{\bot\}$). Thus, $O(\forall\tilde{a}, p) = \forall O(\tilde{a}\mathsf{FAIL}, p) = \{\bot\}$, which is a contradiction.

(b) Suppose $P(a) = \{Q_1, \cdots, Q_n\}$. Then, by induction hypothesis there exists sub-tests $t_1, \cdots, t_n$ such that $O_\lambda(t_i, Q_i) \neq O(t_i, q)$.

i. Either $O_\lambda(t_i, Q_i) = \{\top\}$ and $O(t_i, q) = \{\bot\}$. Let $t' = t_1 \vee \cdots \vee t_n$. Then, from truth table of $\vee$ we have $O_\lambda(t', Q_i) = \{\top\}$. Consequently,

$$\begin{aligned} & O_\lambda(\tilde{a}t', P) = \{\top\} && (\because O_\lambda(t', Q_i) = \{\top\}) \\ \Rightarrow\ & \forall O_\lambda(\tilde{a}t', P) = \{\top\} && (\text{Truth table: } \forall\{\top\} = \{\top\}) \\ \Rightarrow\ & O_\lambda(\forall\tilde{a}t', P) = \{\top\} && (\text{Lemma 5 : } \forall O_\lambda(t, P) = O_\lambda(\forall t, P)). \end{aligned}$$

Furthermore, $O(t', q) = O(t_1, q) \vee \cdots \vee O(t_n, q) = \{\bot\}$. Thus,

$$\begin{aligned} & \bot \in O(\tilde{a}t', p) \\ \Rightarrow\ & \forall O(\tilde{a}t', p) = \{\bot\} && (\text{Truth table of } \forall) \\ \Rightarrow\ & O(\forall\tilde{a}t', p) = \{\bot\} && (\forall O(t, p) = O(\forall t, p)[23]). \end{aligned}$$

Hence, a contradiction.

ii. Or $O_\lambda(t_i, Q_i) = \{\bot\}$ and $O(t_i, q') = \{\top\}$. Let $t' = t_1 \wedge \cdots \wedge t_n$. Then, $O_\lambda(t', Q_i) = \{\bot\}$. Consequently,

$$\begin{aligned} & O_\lambda(\tilde{a}t', P) = \{\bot\} && \text{(since } O_\lambda(t', Q_i) = \{\bot\}\text{)} \\ \Rightarrow\ & \exists O_\lambda(\tilde{a}t', P) = \{\bot\} && (\exists\{\bot\} = \{\bot\}) \\ \Rightarrow\ & O_\lambda(\exists\tilde{a}t', P) = \{\bot\}. \end{aligned}$$

Furthermore, $O(t', q) = O(t_1, q) \wedge \cdots \wedge O(t_n, q) = \{\top\}$. Thus,

$$\begin{aligned} & \top \in O(\tilde{a}t', p) && \text{(since } O(t', q) = \{\top\}\text{)} \\ \Rightarrow\ & \exists O(\tilde{a}t', p) = \{\top\} && \text{(Truth table of } \exists\text{)} \\ \Rightarrow\ & O(\exists\tilde{a}t', p) = \{\top\} && (\exists O(t, p) = O(\exists t, p)[23]). \end{aligned}$$

Hence, a contradiction follows.

($\Rightarrow$) Suppose $P \vdash_\lambda p$. We show by induction on t that $O_\lambda(t, P) = O(t, p)$. The cases when $t = \mathsf{SUCC}, \mathsf{FAIL}, t_1 \vee t_2, t_1 \wedge t_2, \forall t', \exists t'$ are straightforward. The interesting cases are the following:

1. Let $t = at'$.
   (a) Let $\top \in O_\lambda(at', P)$. Then,

$$\begin{aligned} & P \xrightarrow{a}_\phi Q \wedge \lambda \models \phi \wedge \top \in O_\lambda(t', Q) && \text{for some } \phi, Q \\ \Rightarrow\ & p \xrightarrow{a} q && (\because P \vdash_\lambda p) \\ \Rightarrow\ & \top \in O_\lambda(t', Q) \Rightarrow \top \in O(t', q) && \text{(Induction hypothesis)} \\ \Rightarrow\ & \top \in O(at', p). \end{aligned}$$

   (b) Let $\bot \in O_\lambda(at', P)$. Then we have following cases:
      i. Either $P \xrightarrow{a}_\phi Q \wedge \lambda \models \phi \wedge \bot \in O_\lambda(t', Q)$, for some Q, ϕ. Similar to Case (a).
      ii. Or, $\nexists_{Q,\phi}\ P \xrightarrow{a}_\phi Q \wedge \lambda \models \phi$. Then, $O_\lambda(at', P) = \{\bot\}$. Suppose otherwise, $\top \in O(at', p)$. Then, $\exists_q\ p \xrightarrow{a} q$. But, $P \vdash_\lambda p$. Thus, $\exists_{Q,\phi}\ P \xrightarrow{a}_\phi Q \wedge \lambda \models \phi$, which is a contradiction.
   (c) Let $\top \in O(at', p)$. Similar to Case (a).
   (d) Let $\bot \in O(at', p)$. Similar to Case (b).
2. Let $t = \tilde{a}t'$. Similar to the Case (1). □

Furthermore, it follows from Lemma 1 that the notion of test cases remains sound and complete for the traditional notion of product derivation.

Theorem 7. *Let (P, v) be a state in a PL-LTS, p be state in an LTS, and θ be a configuration vector. Then,*

$$(P, v) \vdash_\theta p \ \Leftrightarrow\ \forall_{t \in T}\ O_\theta(t, P, v) = O(t, p).$$

Proof. Similar to the proof of Theorem 6. □

5. Conclusions

In this paper, we studied three fundamental behavioral models for software product lines, namely, modal transition systems, featured transition systems, and product-line labeled transition systems. In particular, we studied the expressiveness of these models by comparing their sets of definable products, which are assumed to be expressible as labeled transition systems. We have shown that modal transition systems are the least expressive of all three, featured transition systems are the most expressive, and product-line labeled transition systems are strictly in between the two. Then we moved to define extensional notions of product derivation and adapted the notion of tests by Abramsky to this end. We proved that the intensional notions of product derivation coincide with the extensional notions defined in this paper for each and every formalism.

Compositionality (pre-congruence) is well-studied for modal refinement in the context of MTSs [24]. However, this problem is understudied for FTSs and this is a high priority item in our future-research agenda. We envisage that using the divide and congruence approach of [42, 43] could provide a solution in this regard (see [44] for our initial attempt in this direction). Another important topic in this area is defining a closed and finite notion of test-cases that can detect all faults, given a fault model (e.g., similar to the W-Method in FSM-based testing [39]). A third area of research, which builds upon the previously-mentioned topic, is to define an incremental procedure for testing different products of a product line.

A few other proposals for transition-system-based specifications of SPLs have been proposed that deserve further investigation. In [10, 11], (Generalized) Extended Modal Transition Systems (GEMTSs) have been introduced in order to specify SPLs. These are variants of disjunctive normal forms [45]. We conjecture that this formalism is strictly in between MTSs and FTSs in terms of expressiveness. Also, in [19], a multi-modal semantics for "Variant Process Algebra" has been introduced, which we conjecture, is as expressive as featured transition systems. We leave proving these conjectures, as well as devising the appropriate extensional notion of testing for GEMTSs for future work.

References

[1] 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.

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

[3] K. Schmid, R. Rabiser, P. Grünbacher, A comparison of decision modeling approaches in product lines, in: Proceedings of the International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS '11), ACM, 2011, pp. 119–126.

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

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

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

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

[8] P. Asirelli, M. H. ter Beek, A. Fantechi, S. Gnesi, A model-checking tool for families of services, in: Proc. on Formal techniques for distributed systems, Springer, 2011, pp. 44–58.

[9] P. Asirelli, M. H. ter Beek, A. Fantechi, S. Gnesi, A compositional framework to derive product line behavioural descriptions, in: Proceedings of the 5th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change (ISoLA '12), Vol. 7609 of Lecture Notes in Computer Science, Springer, 2012, pp. 146–161.

[10] A. Fantechi, S. Gnesi, A behavioural model for product families, in: Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC-FSE '07), ACM, 2007, pp. 521–524.

[11] A. Fantechi, S. Gnesi, Formal modeling for product families engineering, in: Proceedings of the 12th International of Software Product Line Conference (SPLC'08), 2008, pp. 193–202.

[12] M. Lochau, J. Kamischke, Parameterized preorder relations for model-based testing of software product lines, in: Proceedings of the 5th Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change (ISOLA' 12), Vol. 7609 of Lecture Notes in Computer Science, Springer, 2012, pp. 223–237.

[13] K. G. Larsen, U. Nyman, A. Wąsowski, Modal I/O automata for interface and product line theories, in: R. D. Nicola (Ed.), Proceedings of the 16th European Symposium on Programming Languages and Systems, 16th European Symposium on Programming (ESOP '07), Vol. 4421 of Lecture Notes in Computer Science, Springer, 2007, pp. 64–79.

[14] 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: Proceedings of the 32nd International Conference on Software Engineering (ICSE '10), Vol. 1, ACM, 2010, pp. 335–344.

[15] 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, Software Engineering, IEEE Transactions on 39 (8) (2013) 1069–1089. doi:10.1109/TSE.2012.86.

[16] M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, Beyond boolean product-line model checking: dealing with feature attributes and multi-features, in: D. Notkin, B. H. C. Cheng, K. Pohl (Eds.), Proceedings of the 35th International Conference on Software Engineering (ICSE '13), IEEE / ACM, 2013, pp. 472–481.

[17] A. Gruler, M. Leucker, K. Scheidemann, Modeling and model checking software product lines, in: Proceedings of the Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS '08), Vol. 5051 of Lecture Notes in Computer Science, Springer, 2008, pp. 113–131.

[18] M. H. ter Beek, A. Lluch-Lafuente, M. Petrocchi, Combining declarative and procedural views in the specification and analysis of product families, in: Proceedings of the 17th International Software Product Line Conference co-located workshops (SPLC '13 workshops), ACM, 2013, pp. 10–17.

[19] M. Tribastone, Behavioral relations in a process algebra for variants, in: S. Gnesi, A. Fantechi, P. Heymans, J. Rubin, K. Czarnecki, D. Dhungana (Eds.), Proceedings of the 18th International Software Product Line Conference (SPLC '14), ACM, 2014, pp. 82–91.
[20] R. De Nicola, M. Hennessy, Testing equivalences for processes, Theoretical Computer Science 34 (1-2) (1984) 83 – 133.
[21] I. Phillips, Refusal testing, Theoretical Computer Science 50 (1987) 241–284.
[22] J. Tretmans, Model based testing with labelled transition systems, in: Formal Methods and Testing, Vol. 4949 of Lecture Notes in Computer Science, Springer, 2008, pp. 1–38.
[23] S. Abramsky, Observation equivalence as a testing equivalence, Theor. Comput. Sci. 53 (2-3) (1987) 225–241.
[24] K. Larsen, B. Thomsen, A modal process logic, in: Proc. of the 3rd Annual Symposium on Logic in Computer Science (LICS '88), IEEE, 1988, pp. 203–210.
[25] R. Milner, A Calculus of Communicating Systems, Springer, 1982.
[26] S. Shoham, O. Grumberg, Multi-valued model checking games, J. Comput. Syst. Sci. 78 (2) (2012) 414–429.
[27] T. Kahsai, M. Roggenbach, B.-H. Schlingloff, Specification-based testing for software product lines, in: Proceedings of the 6th International Conference on Software Engineering and Formal Methods (SEFM '08), IEEE, 2008, pp. 149–158.
[28] S. Mishra, Specification based software product line testing: A case study, in: Concurrency, Specification and Programming, 2006, pp. 243–254.
[29] M. H. ter Beek, E. d. Vink, Using mCRL2 for the analysis of software product lines, in: S. Gnesi, N. Plat (Eds.), Proceedings of the 2nd FME Workshop on Formal Methods in Software Engineering (FormaliSE '14), FormaliSE 2014, ACM, 2014, pp. 31–37.
[30] R. Muschevici, J. Proença, D. Clarke, Feature Petri Nets, in: G. Botterweck, S. Jarzabek, T. Kishi, J. Lee, S. Livengood (Eds.), Proceedings of the International Conference on Software Product Lines Workshops (SPLC workshops '10), Lancaster University, 2010, pp. 99–106.
[31] R. Muschevici, J. Proença, D. Clarke, Modular modelling of software product lines with feature nets, in: Proceedings of the 9th International Conference on Software Engineering and Formal Methods (SEFM '11), Vol. 7041 of Lecture Notes in Computer Science, Springer, 2011, pp. 318–333.
[32] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, Feature diagrams: A survey and a formal semantics, in: Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE '06), IEEE, 2006, pp. 136–145.
[33] D. Park, Concurrency and automata on infinite sequences, in: Proceedings of the 5th GI-Conference on Theoretical Computer Science, Springer-Verlag, London, UK, 1981, pp. 167–183.
[34] R. M. Hierons, J. P. Bowen, M. Harman (Eds.), Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers, Vol. 4949 of Lecture Notes in Computer Science, Springer, 2008.
[35] H. Beohar, M. R. Mousavi, Spinal test suites for software product lines, in: Proceedings of the 9th Workshop on Model-Based Testing (MBT 2014), Vol. 141 of Electronic Proceedings in Theoretical Computer Science, 2014, pp. 44–55.
[36] H. Beohar, M. R. Mousavi, Input-output conformance testing based on featured transition systems, in: Proceedings of the 29th ACM Symposium on Applied Computing, Software Verification and Testing Track (SAC-SVT 2014), ACM Press, 2014, pp. 1272–1278.
[37] 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.), Proceedings of the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change (ISoLA '14), Vol. 8802 of Lecture Notes in Computer Science, Springer, 2014, pp. 336–350.
[38] X. Devroey, G. Perrouin, P. Schobbens, Abstract test case generation for behavioural testing of software product lines, in: S. Gnesi, A. Fantechi, M. H. ter Beek, G. Botterweck, M. Becker (Eds.), Proceedings of the 18th International Software Product Lines Conference - Companion Volume for Workshop, Tools and Demo papers (SPLC Workshops '14), ACM, 2014, pp. 86–93.
[39] A. d. S. Simão, A. Petrenko, Generating complete and finite test suite for ioco: Is it possible?, in: H. Schlingloff, A. K. Petrenko (Eds.), Proceedings Ninth Workshop on Model-Based Testing (MBT '14), Vol. 141 of EPTCS, 2014, pp. 56–70.
[40] M. Volpato, J. Tretmans, Towards quality of model-based testing in the ioco framework, in: Proceedings of the International Workshop on Joining AcademiA and Industry Contributions to testing Automation (JAMAICA '13), ACM, 2013, pp. 41–46.
[41] G. Boudol, K. G. Larsen, Graphical versus logical specifications, Theoretical Computer Science 106 (1) (1992) 3 – 20.
[42] B. Bloom, W. Fokkink, R. J. van Glabbeek, Precongruence formats for decorated trace semantics, ACM Trans. Comput. Log. 5 (1) (2004) 26–78.
[43] W. Fokkink, R. J. van Glabbeek, P. de Wind, Divide and congruence: From decomposition of modal formulas to preservation of branching and η-bisimilarity, Inf. Comput. 214 (2012) 59–85.
[44] H. Beohar, M. Mousavi, A precongruence format for XY-simulation, in: Proceedings of the 6th IPM International Conference on Foundations of Software Engineering (FSEN'15), Lecture Notes in Computer Science, Springer, 2015.
[45] K. G. Larsen, L. Xinxin, Equation solving using modal transition systems, in: Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), IEEE Computer Society, 1990, pp. 108–117.