This is the published version of a paper published in Science of Computer Programming.

Citation for the original published paper (version of record):

Basic behavioral models for software product lines: Expressiveness and testing pre-orders.
Science of Computer Programming, 123: 42-60
http://dx.doi.org/10.1016/j.scico.2015.06.005

Access to the published version may require subscription.

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-29104
Basic behavioral models for software product lines: Expressiveness and testing pre-orders

Harsh Beohar, Mahsa Varshosaz, Mohammad Reza Mousavi *

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

A R T I C L E   I N F O

Article history:
Received 10 October 2014
Received in revised form 11 June 2015
Accepted 12 June 2015
Available online 9 July 2015

Keywords:
Software product lines
Formal specification
Behavioral specification
Labeled transition systems
Featured transition systems
Modal transition systems
Calculus of communicating systems (CCS)
Product line CCS (PL-CCS)

A B S T R A C T

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.

© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

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.

E-mail addresses: harsh.beohar@hh.se (H. Beohar), mahsa.varshosaz@hh.se (M. Varshosaz), m.r.mousavi@hh.se (M.R. Mousavi).

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 Kunskap- och Kompetensutveckling) in the context of the AUTO-CAAS Hög project with award number 20140312*.

http://dx.doi.org/10.1016/j.scico.2015.06.005
0167-6423/© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).
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 (1€) 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.

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 [13,12] 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*.

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.

**Definition 1.** A modal transition system (MTS) [24] is a quadruple \( (P, A, \rightarrow_{\Box}, \rightarrow_{\Diamond}) \), where \( P \) is a set of states or processes, \( A \) is a set of actions, \( \rightarrow_{\Box} \subseteq P \times A \times P \) is the so-called may transition relation, and \( \rightarrow_{\Diamond} \subseteq \rightarrow_{\Box} \) 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 \( \rightarrow_{\Diamond} = \rightarrow_{\Box} \), 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 $R \subseteq P \times P$ is a modal refinement [24] relation if and only if the following transfer properties are satisfied.

1. $\forall p, p', q \in P, a \in A \quad (P \rightarrow Q \land p \stackrel{a}{\rightarrow} p') \implies \exists q' \in P \quad q \stackrel{a}{\rightarrow} q' \land p' \rightarrow R q'$.
2. $\forall p, q, q' \in P, a \in A \quad (P \rightarrow Q \land q \stackrel{a}{\rightarrow} q') \implies \exists p' \in P \quad p \stackrel{a}{\rightarrow} p' \land p' \rightarrow R q'$.

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

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.

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 \( B = \{ \top, \bot \} \) 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 \( e \land \neg d \) asserts the presence of euro coin and the absence of dollar coin payment features. We let \( \phi, \phi' \) range over the set \( \mathbb{B}(F) \).

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

1. \( \mathbb{P} \) is the set of states,
2. \( A \) is the set of actions,
3. \( F \) is a set of features,
4. \( \to \subseteq S \times A \times \mathbb{B}(F) \times S \) is the transition relation satisfying the following condition:
   \[
   \forall P, a, P', \phi, \phi' \ (P, a, \phi, P') \in \to \land (P, a, \phi', P') \in \to \implies \phi = \phi',
   \]
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 \( P \overset{a, \phi}{\to} Q \) 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.

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 \overset{t_e}{\to} s_2 )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( s_1 \overset{t_d}{\to} s_2 )</td>
<td>( d )</td>
</tr>
<tr>
<td>( s_2 \overset{\text{coffee}}{\to} s_5 )</td>
<td>( c )</td>
</tr>
<tr>
<td>( s_2 \overset{\text{tea}}{\to} s_6 )</td>
<td>( t )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_2 \overset{\text{cappuccino}}{\to} s_7 )</td>
<td>( p )</td>
</tr>
<tr>
<td>( s_{12} \overset{\text{ring a tone}}{\to} s_{13} )</td>
<td>( p \Rightarrow r )</td>
</tr>
<tr>
<td>Remaining transitions</td>
<td>( m )</td>
</tr>
</tbody>
</table>

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]:

\[
\Lambda = \{ \{ m, o, b, c, e \}, \{ m, o, b, c, e, r \}, \{ m, o, b, c, e, t \}, \{ m, o, b, c, e, r, t \}, \{ m, o, b, c, e \}, \{ m, o, b, c, d \}, \{ m, o, b, c, d, r \}, \{ m, o, b, c, d, t \}, \{ m, o, b, c, d, r, t \}, \}
\]

**2.3.2. Deriving valid products**

In [15] a class of operators \( \Delta_{\lambda}(\_) \), each parameterized by product configurations \( \lambda \in \Lambda \), is introduced to project an FTS into various LTSSs, 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 \( \lambda \).
**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:

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

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 $1 \xrightarrow{e} 2$. 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 $\lambda(e) = \lambda(d) = T$. 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 understrong 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 $(P, A, F, \rightarrow, \Lambda)$, an LTS $(P, A, \rightarrow)$, and a product $\lambda \in \Lambda$. A family of binary relations $\mathcal{R}_\lambda \subseteq P \times P$ (parameterized by product configurations) are called *product-derivation relations* if and only if the following transfer properties are satisfied.

1. $\forall_{P, a, p, \phi} \left( P \mathcal{R}_\lambda, p \land P \xrightarrow{\phi} Q \land \lambda \models \phi \Rightarrow \exists_{q} \ P \xrightarrow{e} q \land Q \mathcal{R}_\lambda q \right)$.
2. $\forall_{P, a, p, q} \left( P \mathcal{R}_\lambda, p \land P \rightarrow q \Rightarrow \exists_{Q, \phi} \ P \xrightarrow{\phi} Q \land \lambda \models \phi \land Q \mathcal{R}_\lambda q \right)$.

A state $p \in P$ in an LTS derives the product $\lambda$ from an FTS-specification $P \in \mathcal{P}$, 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 $\lambda$. If $q$ is strongly bisimilar (in the sense of Park [33]) to $p$, then $P \vdash \lambda \Delta_\lambda(p)$.

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 $A = \Sigma \cup \sum \cup \{\tau\}$ be the alphabet, where $\sum = \{a \mid a \in \Sigma\}$. The syntax of PL-CCS terms $e$ is defined by the grammar $\text{Nil} | \alpha. e | e + e' | e \oplus e' | e \parallel e' | e[f] | e(L)$, where $\text{Nil}$ is the deadlock process, for each $\alpha \in A$, $\alpha_+$ denotes action prefixing, $\alpha_- \oplus_-$ denotes non-deterministic choice, $\alpha_- \oplus$ denotes binary variant, $\alpha_- \parallel$ denotes parallel composition, for each $f : A \rightarrow A$, $\alpha_- [f]$ denotes renaming by means of $f$, and for each $L \subseteq A$, $\alpha_- \setminus L$ denotes the restriction operator (blocking $\alpha$ 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 $s = a(b \cdot s + c.s)$ 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 $\langle P \rangle = P \oplus \text{Nil}$. 

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

\[
\begin{align*}
  s_1 &= 1e.s_2 \oplus 1d.s_2 & s_7 &= \text{pour sugar}.s_8 \\
  s_2 &= \text{sugar}.s_3 + \text{no sugar}.s_4 & s_8 &= \text{pour coffee}.s_{11} \\
  s_3 &= \text{coffee}.s_5 + (\text{tea}.s_6 + \text{cappuccino}.s_7) & s_9 &= \text{pour tea}.s_{12} \\
  s_4 &= \text{coffee}.s_{10} + (\text{tea}.s_9 + \text{cappuccino}.s_8) & s_{10} &= \text{pour coffee}.s_{12} \\
  s_5 &= \text{pour sugar}.s_{10} & s_{12} &= \text{ring tone}.s_{13} + \text{skip}.s_{13} \\
  s_6 &= \text{pour sugar}.s_9 & s_{13} &= \text{take cup}.s_1.
\end{align*}
\]

The semantics of a PL-CCS term is defined in terms of product line labeled 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 \((\mathcal{P} \times [L, R, ?]^I, A, I, \rightarrow)\) consisting of a set of states \(\mathcal{P} \times [L, R, ?]^I\), a set of actions \(A\), and a transition relation \(\rightarrow \subseteq (\mathcal{P} \times [L, R, ?]^I) \times (A \times [L, R, ?]^I) \times (\mathcal{P} \times [L, R, ?]^I)\) satisfying the following restrictions:

1. \(\forall_{P, v, A, Q, v', v''} (P, v) \xrightarrow{A, v'} (Q, v') \implies v' = v''\).
2. \(\forall_{P, v, A, Q, v', v''} ((P, v) \xrightarrow{A, v'} (Q, v') \land v'' \neq v) \implies \exists i \ (v'(i) \neq v(i) \land \forall_{j \neq i} v'(j) = v(j)).\)
3. \(\forall_{P, v, A, Q, v', v''} ((P, v) \xrightarrow{A, v'} (Q, v') \land v'(i) \neq ?) \implies v'(i) = v(i)\).

Notice that, as a consequence of item 2 in Definition 7, for any transition \((P, v) \xrightarrow{A, v'} (Q, v')\), if \(v \neq v'\), we can find a unique \(i \in I\) such that \(v'(i) \neq v(i) \land \forall_{j \neq i} v'(j) = v(j)\). 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 \([L, R, ?]^I\) 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 \(v \sqsubseteq v' \iff \forall_{i \in I} v(i) \sqsubseteq v'(i)\), for any \(v, v' \in [L, R, ?]^I\).

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 \((\mathcal{P} \times [L, R, ?]^I, A, \rightarrow)\) be a PL-LTS and let \((P, A, \rightarrow)\) be an LTS. A family of binary relations \(\mathcal{R}_\theta \subseteq (\mathcal{P} \times [L, R, ?]^I) \times P\) (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. \(\forall_{P, Q, A, v, v', p} ((P, v)\mathcal{R}_\theta p \land (P, v) \xrightarrow{A, v'} (Q, v') \land v' \sqsubseteq \theta) \implies \exists q \ p \xrightarrow{\theta} q \land (Q, v') \mathcal{R}_\theta q\).
2. \(\forall_{A, v, p, q} ((P, v)\mathcal{R}_\theta p \land p \xrightarrow{\theta} q) \implies \exists Q, v' \ (P, v) \xrightarrow{\theta} (Q, v') \land v' \sqsubseteq \theta \land (Q, v') \mathcal{R}_\theta q\).

A state \(p \in P\) in an LTS is (the initial state of) a product of a PL-LTS \((P, v)\) with respect to a configuration vector \(\theta\), denoted by \((P, v) \models_{\theta} p\), if \(v \sqsubseteq \theta\) 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 \((P, A, \rightarrow)\) be an LTS. A product line structure is a tuple \(M = (\mathcal{M}, [\ ]\)\), where \(\mathcal{M}\) is the class of all intended product line models or specifications (in our case: MTSS, FTSS, and PL-LTSs) and \([\ ]\ : \mathcal{M} \rightarrow 2^P\) is the semantic function mapping a product line specification to a set of LTSs.
Definition 11. An encoding \( E : \mathcal{M} \rightarrow \mathcal{M}' \) from a product line structure \( \mathcal{M} = (\mathcal{M}', [\ ]') \) into a product line structure \( \mathcal{M}' = (\mathcal{M}', [\ ']') \) is a function \( E : \mathcal{M} \rightarrow \mathcal{M}' \) satisfying the correctness criterion \([\ ] = E \circ [\ ]'\).

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

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 \((\mathcal{P}, A, \rightarrow, \), \(\mathcal{O} \rightarrow \mathcal{O} \rightarrow \)) and some \( \mathcal{P} \in \mathcal{P} \). Let \( \rightarrow \subseteq \mathcal{P} \times \mathcal{A}^+ \times \mathcal{P} \) be the reachability relation defined as follows: \( p \rightarrow p' \stackrel{s}{\rightarrow} p'' \). Let \( \mathfrak{tr}(\mathcal{P}) \) be the set of traces generated by \( \mathcal{P} \), i.e., \( \mathfrak{tr}(\mathcal{P}) = \{ s \mid \exists Q \mathcal{P} \rightarrow Q \} \). For state \( \mathcal{P} \), we define a family of transition relations parameterized by the traces of \( \mathcal{P} \) as follows: \( Q \stackrel{a,\mathcal{O}}{\rightarrow} Q' \). We drop the subscript \( \mathcal{P} \) 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 \( \mathfrak{tr}(\mathcal{P}) \rightarrow (\bigcup_{s \in \mathfrak{tr}(\mathcal{P})} \rightarrow s [ (Q, L, R, ?) ] \).

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

\[
Q \stackrel{a}{\rightarrow} Q' \text{ s \in } \mathfrak{tr}(\mathcal{P}) \quad v' = v \uparrow \{(Q, a, Q') \rightarrow L\} \\
(\mathfrak{Q}, v) \stackrel{a,v'}{\rightarrow} (\mathfrak{Q'}, v') \\
Q \stackrel{\mathfrak{O},\mathcal{O}}{\rightarrow} Q' \text{ s \in } \mathfrak{tr}(\mathcal{P}) \quad v' = v \uparrow \{(Q, a, Q') \rightarrow R\} \\
(\mathfrak{Q}, v) \stackrel{a,v'}{\rightarrow} (\mathfrak{Q'}, v') \\
\]

where \( Q \stackrel{a}{\rightarrow} Q', \mathcal{O} \rightarrow \mathcal{O}, Q \land (Q, a, Q') \notin \mathcal{O} \) and the expression \( v' = v \uparrow \{(Q, a, Q') \rightarrow X\} \) (for \( X \in \{L, R\} \)) is defined in the following way:

\[
v'(s)(\mathfrak{Q}, a, \mathfrak{O}, \mathfrak{Q'}) = \begin{cases} 
X & \text{if } s = s' \land \mathfrak{Q} = Q \land a = a \land \mathfrak{Q}' = Q' \\
\text{otherwise}
\end{cases}
\]

We fix \( E(\mathcal{P}) = (\mathcal{P}, v_0) \), where \( v_0(s)(\mathcal{Q} \stackrel{a}{\rightarrow} s', Q') = ? \) for \( s, s' \in \mathfrak{tr}(\mathcal{P}) \). Furthermore, we let the symbols \( \mathcal{O}, \mathcal{O}' \) range over the total configuration vectors, i.e., the functions of type \( \mathfrak{tr}(\mathcal{P}) \rightarrow (\bigcup_{s \in \mathfrak{tr}(\mathcal{P})} \rightarrow s [ (Q, L, R, ?) ] \). Now we are in the position to show that \( [\mathcal{P}] = [E(\mathcal{P})] \), where \( [\mathcal{P}] = [p : p \leq p] \) and \( [E(\mathcal{P})] = [p : \exists \mathcal{O}, E(\mathcal{P}) \rightarrow p] \). We divide the proof obligation into two obligations: \( [E(\mathcal{P})]' \subseteq [\mathcal{P}] \) and \([E(\mathcal{P})]' \subseteq [E(\mathcal{P})]' \), which we prove below.

Let \( \mathcal{P} \) be a state in an LTS such that \( \mathcal{P} \rightarrow \mathcal{P} \), for some \( \mathcal{O} \). Define a relation \( q \rightarrow r \) if \( q \rightarrow r \) and \( q \rightarrow r \). Next, we show that \( R \) is a modal refinement relation.

1. Let \( q \rightarrow r \) and \( q \rightarrow r \). Then, \( (Q, v) \rightarrow (Q, v) \) for some \( v \). Clearly, \( q \rightarrow r \). Let \( v' = v \uparrow \{(Q, a, Q') \rightarrow \mathcal{O}(Q, a, Q') \} \). Note that \( \mathcal{O}(Q, a, Q') \) can be either \( L \) or \( R \). Then, we find \( v' \) for \( v \). Now from the transfer property of \( \rightarrow \) we find a \( q' \) such that \( q \rightarrow q' \) and \( q' \) for \( q' \). Hence, \( q \rightarrow r \).

2. Let \( q \rightarrow r \) and \( q \rightarrow r \). Trivial.

\[
[\mathcal{P}] \subseteq [E(\mathcal{P})]' \]. Let \( p \) be some state in an LTS such that \( p \leq p \). Let \( \mathfrak{tr}(p) \) 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 traces \( s \in \mathfrak{tr}(p) \), we know that there is a unique execution \( e \) starting from \( [p]_e \), such that \( dom(e) = s \), where \( [p]_e \) is the transition system modulo strong bisimulation defined in the standard way.\(^2\) Therefore, for any maximal trace

\[^2\text{An execution } e \text{ starting from } [p]_e \text{ is a function } e : [s'] \rightarrow 2^p \text{ for any } s \in \mathfrak{tr}(p) \text{ such that } e(s) = [p]_e \text{ and } e(s') \rightarrow e(s'u) \rightarrow 3_{q'q} \text{ for every } s' \leq s \text{. Here } \leq \text{ is the prefix relation between any two words.}\)
s ∈ tr_m(p), we denote the corresponding unique execution by e_s. Define total configuration vectors θ_p (for each P ⊆ L) in the following way:

\[
\theta_p(s)(Q \overset{a}{\Rightarrow} s', Q') = \begin{cases} L & \text{if } s \in tr_m(p) \land s = s' \land \exists s'' \leq s \ Q \leq e_s(s'') \land Q' \leq e_s(s''a) \\ R & \text{otherwise} \end{cases}
\]

where Q ≤ X is defined as ∀q q ∈ X ⇒ Q ≤ q. Next, we define a relation R_{θ_p} as follows:

\[(Q, ν) R_{θ_p} q \iff ν \subseteq θ_p \land \exists s, s' s \in tr_m(p) \land s' ≤ s \land P \overset{s'}{\Rightarrow} Q \land Q ≤ e_s(s') \land q \in e_s(s').\]

Next, we show that R_{θ_p} is a product derivation relation.

1. Let \((Q, ν) R_{θ_p} q\). Then, from the construction of R_{θ_p} we have ν ⊆ θ_p and s ∈ tr_m(p) ∧ P ⇒ Q ∧ Q ≤ e_s(s') ∧ q ∈ e_s(s'), for some s ∈ tr_m(p), s' ≤ s.

2. Let Q \overset{a}{\Rightarrow} q' and (Q, ν) R_{θ_p} q. Then, from the construction of R_{θ_p} we have ν \subseteq θ_p ∧ s ∈ tr_m(p) ∧ P ⇒ Q ∧ Q ≤ e_s(s') ∧ q ∈ e_s(s'), for some s ∈ tr_m(p), s' ≤ s.

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 Fig. 4.

We prove by contradiction that the PL-LTS depicted in Fig. 4 (left), where P = a.Nil ⊕ b.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 Fig. 4 under θ = L and θ' = R.

Suppose there is an encoding E satisfying Definition 11. Clearly, (P, ?) ⇒_θ p and (P, ?) ⇒_θ q. Then by correctness of E we have E(P, ?) ≤ p and E(P, ?) ≤ q. Thus, we can derive the following transitions (for some modal states P_a, P_b) from the transfer property of modal refinement:

\[E(P, ?) \overset{a}{\Rightarrow} P_a \quad E(P, ?) \overset{b}{\Rightarrow} P_b.\]

Therefore, there exists a state r of the following form: r \overset{a}{\Rightarrow} r' and r \overset{b}{\Rightarrow} r'' (for some r', r'') such that E(P) ≤ r. And by correctness of E we get (P, ?) ⇒_θ r or (P, ?) ⇒_θ r. However, (P, ?) ∉_θ r and (P, ?) ∉_θ 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 \((P \times [L, R, ?]^I, A, \rightarrow)\) be a PL-LTS. The corresponding FTS is denoted by \((P \times [L, R, ?]^I, A, F, \rightarrow', \Lambda)\), where:

- \[F = \bigcup_{i \in I} \{L_i, R_i\}.\]
- \[\Lambda = \bigwedge_{i \in I} \bar{\Lambda}(L_i \land R_i).\]
- The transition relation →' is defined in the following way:

\[
\begin{align*}
(P, v) \overset{a}{\rightarrow'} (Q, v) \quad & (P, v) \overset{a, v'}{\rightarrow'} (Q, v') \\
(P, v) \overset{i}{\rightarrow'} (Q, v) \quad & (P, v) \overset{\phi}{\rightarrow'} (Q, v') \quad \Xi(i, v, v').
\end{align*}
\]

where \(\Xi(i, v, v') \iff v'(i) \neq v(i) \land \forall j \neq i v'(j) = v(j).\)
For any \((P, v) \in P \times \{L, R, ?\}\), we fix \(E(P, v) = (P, v)\). Let \(\llbracket (P, v) \rrbracket = \{p | 3p e (L, R) (P, v) \vdash_0 p\}\) and \(\llbracket (P, v) \rrbracket = \{p | 3p e (P, v) \vdash_0 p\}\). In the next step, we need to show that \(\llbracket (P, v) \rrbracket \subseteq \llbracket (P, v) \rrbracket\).

\((\llbracket (P, v) \rrbracket \subseteq \llbracket (P, v) \rrbracket\): Let \(p \in \llbracket (P, v) \rrbracket\). Then \((P, v) \vdash_0 p\) for some \(\theta \in (L, R)\). Define a configuration \(\lambda_0 \in \Lambda\) as follows:

\[\lambda_0(L_i) = T \iff \theta(i) = L \text{ and } \lambda_0(R_i) = T \iff \theta(i) = R.\]

Furthermore, consider the following relation \(\mathcal{R}_0\) such that \((Q, \nu)^{\mathcal{R}_0} q \iff (Q, \nu^{'}) \vdash_0 q\). It is straightforward to show that \(\mathcal{R}_0\) is a product derivation relation.

\((\llbracket (P, v) \rrbracket \subseteq \llbracket (P, v) \rrbracket\): Let \(p \in \llbracket (P, v) \rrbracket\). Then \(P \vdash_\lambda p\) for some \(\lambda \in \Lambda\). Let \(\theta \in (L, R)\). We have \((Q, \nu^{'})^{\mathcal{R}_0} q \iff (Q, \nu^{'}) \vdash p\).

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

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

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 \(\Lambda\) is defined as the smallest set of functions satisfying the following constraint:

\[(f_a \rightarrow (\neg f_b \land \neg f_c)) \land (f_b \rightarrow (\neg f_a \land \neg f_c)) \land (f_c \rightarrow (\neg f_a \land \neg f_b)).\]

Through a proof by contradiction, we show that there is no encoding \(E\) that shows the FTS \(P\) in the correct way. Suppose there is otherwise an encoding of \(P\) into an PL-LTS whose configuration vectors are of type \((L, R, ?)\) (for some index set \(I\)) such that \(\llbracket P \rrbracket = \llbracket E(P) \rrbracket\), where \(\llbracket P \rrbracket = \{p | 3p e \Lambda (P, v) \vdash_0 p\}\), \(E(P) = (Q, \nu_0)\) (for some state \(Q\) and configuration \(v_0 \in (L, R, ?)\) in an PL-LTS), and \(\llbracket E(P) \rrbracket = \{p | 3p e \Lambda (Q, \nu_0) \vdash_0 p\}\).

Clearly, the transition systems \({\llbracket (p_x, p_y) | x, (p_x, x) | (x, a, c)\}\) (for \(x \in \{a, b, c\}\)) are three valid products of the given FTS \(P\), i.e., \(\{p_a, p_b, p_c\} \subseteq \llbracket P \rrbracket\). So from the correctness requirement of \(E\) we have \(\{p_a, p_b, p_c\} \subseteq \llbracket E(P) \rrbracket\). Let \(\theta_a, \theta_b, \theta_c\) be the corresponding total configuration vectors that derive the products \(p_a, p_b, p_c\), respectively. Thus, for every \(x \in \{a, b, c\}\) we have \(v_0 = \theta_x\). Furthermore, from the transfer property of product derivation we find \((Q, v_0) \xrightarrow{\lambda_{\nu_0}} (Q, \nu_0)\) such that \(v_0 = \theta_x\), for \(x \in \{a, b, c\}\). Clearly, \(v_x \neq v_0\) (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 \(v_0 = \nu_0\) then the transfer property of product derivation ensures that \(3p e \Lambda (\nu_0) \vdash q\) because \(v_0 = \theta_x\). Therefore, let \(i, j, k \in I\) be the unique elements such that for every \(x \in \{a, b, c\}\) we have \(v_x(i) \neq v_0(i)\) and \(\nu_0(j) = v_0(k)\) (recall Condition (7)(2)).

Next, we show that if \(i_0 \neq i_0\) \& \(i_0 \neq k\), \(i_0 \neq l_0\) then we can derive a product which has a choice between \(a, c\). Since \(i_0, l_0\) are the only elements whose values are changed by \(v_0, \nu_0\), so from Condition (7)(3) we have \(v_0(i_0) = v_0(l_0) = v_0(?)\). Define a function \(\theta_0^c\) as follows:

\[
\theta_0^c(i) = \begin{cases} 
\theta_0(i) & \text{if } i \neq i_0 \land i \neq l_0, \\
i_0 & \text{if } i = i_0, \\
R & \text{if } i = l_0 \land \theta_0(l_0) = L, \\
L & \text{if } i = l_0 \land \theta_0(l_0) = R.
\end{cases}
\]

Next, we show that \(v_x \subseteq \theta_0^c\). If \(i \neq i_0 \land i \neq l_0\) then clearly \(v_x(i) \subseteq \theta_0^c(i)\) because \(\theta_0^c(i) = \theta_0(i)\). If \(i = i_0\) or \(i = l_0\) then \(v_x(i) = v_0(?)\). Thus, \(v_x \subseteq \theta_0^c\).

Next, we show that \(v_0 \subseteq \theta_0^c\). If \(i \neq i_0 \land i \neq l_0\) then \(v_0(i) = v_0(?)\). Thus, \(v_0(i) \subseteq \theta_0^c(i)\) because \(v_0(i) \subseteq \theta(i) \land \theta(i) = \theta(i)\). If \(i \neq i_0 \land i = l_0\) then \(v_0(i) = v_0(?)\). Thus, \(v_0 \subseteq \theta_0^c\).

As a result, we can derive a product that contains a choice between \(a, c\) by using \(\theta_0^c\); however, such a product is clearly not a valid product of the given FTS \(P\) as it violates the condition \(\Lambda\).

On the other hand, if \(i_0 = i_0\) then we show that using \(\theta_0\) we can derive a product which has a choice between either \(a, b\) or \(b, c\). It suffices to show that \(v_0(i_0) \subseteq \theta_0(i_0)\) or \(v_0(i_0) \subseteq \theta_0(l_0)\) because for every \(i = i_0\) we have \(v_0(i) = v_0(?) = v_0(l_0)\). We claim that \(v_0(i_0) \neq ?.\) Suppose otherwise \(v_0(i_0) = ?.\) Then, since \(i_0 = i_0\) we find \(v_x \subseteq \theta_0\). As a result, from the transfer property of \(\lambda_{\nu_0}\) there must be a \(q\) such that \(p_0 e \Lambda (\nu_0, p_0) \vdash \lambda_0\). Thus, \(v_0(?) \neq ?.\) Likewise, we can prove that \(v_0(i_0) \neq ?.\) Thus, \(v_0(i_0) \neq (L, R)\). And since \(\theta_0\) is a function whose co-domain is the set \((L, R)\) we have either \(v_0(i_0) \subseteq \theta_0(i_0)\) or \(v_0(i_0) \subseteq \theta_0(l_0)\). Hence, we can derive a product that contains a choice between either \(a, b\) or \(b, c\) by using \(\theta_0\); however, such a product is clearly not a valid product of the given FTS \(P\) as it violates the condition \(\Lambda\). Likewise, if \(l_0 = i_0\)
(or \(i_a = i_b\)) then we can show that using \(\theta_a \ (\theta_c)\) we can derive a product which has a choice between either \(a, b \ (a, c)\) or \(a, c \ (b, c)\). \(\square\)

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

\[
\text{MTSs} \rightarrow \text{PL-LTSs} \rightarrow \text{FTSs},
\]

where the arrow \(\rightarrow\) 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 Fig. 6 with the set of features \(F = \{f, f'\}\) and the set of valid product configuration \(\Lambda = \{\lambda, \lambda'\}\) with \(\lambda(f) = T, \lambda(f') = \bot\) and \(\lambda'(f) = \bot, \lambda'(f') = T\). The transition systems of the derived processes \(P\) and \(Q\) under \(\lambda\) and \(\lambda'\), respectively, are drawn in Fig. 6. Now by contradiction we show that there is no encoding \(E\) satisfying Definition 11.

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 \(E(P) \leq p\) and \(E(P) \leq q\). Thus, we can derive the following transitions (for some modal states \(Pa, Pb\)) from the transfer property of modal refinement:

\[
E(P) \overset{a}{\rightarrow} \diamond Pa \quad E(P) \overset{b}{\rightarrow} \diamond Pb.
\]

Therefore, there exists a state \(r\) of the following form: \(r \overset{a}{\rightarrow} r'\) and \(r \overset{b}{\rightarrow} r''\) (for some \(r', r''\)) such that \(E(P) \leq r\). And by correctness of \(E\) we get \(P \vdash_{\lambda} r\) or \(P \vdash_{\lambda'} r\). However, \(P \not\vdash_{\lambda} r\) and \(P \not\vdash_{\lambda'} r\).

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 intensional 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 [22,21,20], 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 Devroye 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 conformance. In order to turn this theory into a practical testing scheme, some degrees of unboundedness 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]:

\[
t ::= \text{SUCC} \mid \text{FAIL} \mid at \mid \bar{at} \mid t_1 \land t_2 \mid t_1 \lor t_2 \mid \forall t \mid \exists t.
\]

\(^3\) Throughout this section, we assume that the product line structure under investigation is image finite.
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 $\text{at}$ 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 $\text{at}$ 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 $t_1 \land t_2$ and $t_1 \lor t_2$ 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:

$$E ::= \top \mid \bot \mid (t \parallel P) \mid E_1 \land E_2 \mid E_1 \lor E_2 \mid \forall E \mid \exists E.$$  

$$E ::= \top \mid \bot \mid (t \parallel p) \mid E_1 \land E_2 \mid E_1 \lor E_2 \mid \forall E \mid \exists E.$$  

Fig. 7 provides the operational interpretation of experiment expressions over an MTS. Note that only the rules of the expressions $\mathbf{at} P$ and $\mathbf{at} (t \parallel P)$ (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 Fig. 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 Fig. 7 by considering the transitions of LTS as both may and must transitions.

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 $\mathcal{O} = \bot \cup \top$. 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 $\mathcal{P}[\mathcal{O}] = (\bot) \subseteq (\bot, \top) \subseteq (\top)$). The semantics of the operators $\land, \lor, \forall, \exists$ over the Plotkin powerdomain $\mathcal{P}[\mathcal{O}]$ can be found in [23]. In addition, given an MTS ($\mathcal{P}, \mathcal{A}, \rightarrow_0, \rightarrow_\square$), we define the function $O : T \times \mathcal{P} \rightarrow \mathcal{P}[\mathcal{O}]$ in the following way:

$$O(t, P) = \{\top \mid (t \parallel P) \rightarrow \top\} \cup \{\bot \mid (t \parallel P) \rightarrow \bot\}.$$
Lemma 3. Let $P$ be a modal specification. Then, for any test expression $t$ we have

$$O(∀t, P) = ∀O(t, P)$$ and $$O(∃t, P) = ∃O(t, P).$$

Before we turn our attention to the characterization of modal refinement as a testing pre-order, we first give a semantic preserving transformation $⟦⟦⟧⟧_A$ (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 $⊆$ and the modal refinement relation $≤$.

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

$$Φ ::= ⊥ | T | (a)φ | [a]φ | φ ∧ φ′ | φ ∨ φ′.$$  

The semantics of $⊥, T, ∧, ∨$ is standard, while the nonstandard semantics of $(a)φ, [a]φ$ is given as follows [41]:

1. $P ⊨ (a)φ ⇔ ∃P′. P ⊢P′ T ∧ P′ ⊨ φ.$
2. $P ⊨ [a]φ ⇔ ∀P′. P ⊢P′ T ⇒ P′ ⊨ φ.$

Following [23], we give a transformation $⟦⟦⟧⟧_A : Φ → T$ of HML formulae to the set of test expressions.

$$⟦T⟧_A = SUCC,$$

$$⟦φ ∧ φ′⟧_A = ⟦φ⟧_A ∧ ⟦φ′⟧_A,$$

$$⟦φ ∨ φ′⟧_A = ⟦φ⟧_A ∨ ⟦φ′⟧_A,$$

$$⟦[a]φ⟧_A = ∀a⟦φ⟧_A,$$

$$⟦(a)φ⟧_A = 3a⟦φ⟧_A.$$

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 $⟦φ⟧_A$.

Lemma 4. Let $P$ and $p$ be a state in an MTS and an LTS, respectively. Then, for any $φ ∈ Φ$ we have

$$P ⊨ φ ⇔ O(⟦φ⟧_A, P) = {T}$$ and $$p ⊨ φ ⇔ O(⟦φ⟧_A, p) = {T}.$$  

Proof. The proof is by induction on $φ$; the cases for $⊥, T, ∧, ∨$ are straightforward.

1. Let $φ = [a]φ′$ and $P ⊨ φ$. Then,

$$∀P′. P ⊢P′ T ∧ P′ ⊨ φ′$$

$$(\text{Induction hypothesis})$$

$$⊥ ≡ ∀P′. P ⊢P′ T ∧ P′ ⊨ φ′$$

$$O(⟨φ′⟧_A, P) = {T}$$

$$(\text{Rule 5 and Definition of } O)$$

$$(\text{Lemma 3: } O(∀t, P) = ∀O(t, P))$$

$$O(∀a[φ′⟧_A, P) = {T}$$

$$O(3a[φ′⟧_A, P) = {T}$$

$$(\text{Truth table } [23]: 3(⊥) = {T} \text{ and } 3(T) = {T})$$

2. Let $φ = (a)φ′$ and $P ⊨ φ$. Then,

$$∀P′. P ⊢P′ T ∧ P′ ⊨ φ′$$

$$(\text{Induction hypothesis})$$

$$∃P′. P ⊢P′ T ∧ O(⟨φ′⟧_A, P′) = {T}$$

$$(\text{Rule 5 and Definition of } O)$$

$$O(∃a[φ′⟧_A, P) = {T}$$

$$(\text{Truth table } [23]: 3(⊥) = {T} \text{ and } 3(T) = {T})$$

$$O(3a[φ′⟧_A, P) = {T}$$

$$O(∃(a)φ′⟧_A, P) = {T}.$$  

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

$$\forall t \in T \ O(t, P) \subseteq O(t, p) \Rightarrow P \leq p.$$  

Proof. Suppose $\forall t \in T \ O(t, P) \subseteq O(t, p)$. 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 $O([\varphi]_A, P) = \{\top\}$. Since the element $\{\top\}$ is the maximum in the Plotkin powerdomain $\mathcal{P}[\mathcal{O}]$ and $O([\varphi]_A, P) \subseteq O([\varphi]_A, p)$ we know that $O([\varphi]_A, p) = \{\top\}$. From Lemma 4, we conclude that $p \models \varphi$. \qed

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, $P \leq p$. Consider the test $t = \#\#SUCC$. Clearly, $O(t, P) = \{\perp, \top\}$ and $O(t, p) = \{\perp\}$. However, $\{\perp, \top\} \not\subseteq \{\perp\}$.

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:

$$t ::= SUCC \mid FAIL \mid \exists at \mid \forall at \mid t_1 \land t_2 \mid t_1 \lor t_2.$$  

Then, $\forall t \in T' \ O(t, P) \subseteq O(t, p) \iff P \leq p$.

It follows also from Corollary 1 that if an LTS is not a valid product of an MTS, i.e., $P \not\leq p$, then it is sufficient to find a test $t \in T$ such that $O(t, P) \not\subseteq O(t, p)$. 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 $p$.

Observe that by adopting the test $t = 31d$ sugar coffee pour sugar pour coffee SUCC we can show that $p$ is not a valid product of $P$ because $O(t, P) = \{\top\}$ and $O(t, p) = \{\perp\}$. Thus $\{\top\} \not\subseteq \{\perp\}$; hence, $P \not\leq p$.

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 FTSSs). 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 $(\mathcal{P}, A, F, \rightarrow, \Lambda)$. It should not be surprising that only the semantics of the tests of the form $t = \#\#SUCC$.
Theorem
Proof.
\text{(the modified In Second, way.)}

Lemma
1. Either,
Next, Just \text{(a) Suppose image needs \( \tilde{\lambda} \) like \( \lambda \implies \phi \). We define \( O_{\lambda} \) at \( \lambda \implies \phi \). But, we have a contradiction.}

\text{Hence, give \( \theta \) at \( \theta \implies \lambda \implies \phi \). Due to finiteness assumption we know that the set \( p(\alpha) \) is finite. We identify the following cases:}

1. Either, there exists \( a, Q \) such that \( P \stackrel{a, \phi}{\longrightarrow} Q, \lambda \implies \phi \), and for all \( q \), if \( P \stackrel{a, \phi}{\longrightarrow} q \) then \( Q \not\vDash q \). Let \( p(\alpha) = \{ q \mid P \stackrel{a, \phi}{\longrightarrow} q \} \). Due to finite test-case assumption we know that the set \( p(\alpha) \) is finite. We identify the following cases:

(a) Suppose \( p(\alpha) = \emptyset \). Then,

\text{\( \top \in O_{\lambda}(a\text{Succ}) \) we \( (\cdot : P \stackrel{a, \phi}{\longrightarrow} Q \land \lambda \implies \phi) \).}

Thus, \text{O\text{(aSucc, p)} = \{ \top \} \text{(Truth table of) \text{(Lemma 5: \text{O\text{(3t, s)} = \{ \top \}}).)}}

But, \text{O\text{(aSucc, p)} = \{ \bot \}; thus,}

\text{O\text{(aSucc, p)} = \exists \text{O\text{(aSucc, p)}} \text{= O\text{(aSucc, p)} = \{ \bot \}.}}

Hence, a contradiction follows.

(b) Suppose \( p(\alpha) = \{ q_1, \ldots, q_n \} \). Then, by induction hypothesis there exists sub-tests \( t_1, \ldots, t_n \) such that \( O_{\lambda}(t_i, Q) \neq O(t, q_i) \).}
2. Either, \( O_\lambda(t_i, Q) = \{\top\} \land O(t_i, q_i) = \{\bot\} \). Let \( t' = t_1 \land \cdots \land t_n \). Consequently,

\[
\top \in O_\lambda(at', P) \quad \text{(since } O_\lambda(P) = \{\top\})
\]

\[
\implies \exists O_\lambda(at', P) = \{\top\} \quad \text{(Truth table of } \exists \)
\]

\[
\implies O_\lambda(\exists at', P) = \{\top\} \quad \text{(Lemma 5: } O_\lambda(\exists t, P) = \exists O_\lambda(t, P)) \) .
\]

But,

\[
O(at', p) = \{\bot\} \quad \text{(since } O_\lambda(Q) = \{\bot\})
\]

\[
\implies \exists O(at', p) = \{\bot\} \quad \text{(Truth table: } \exists \{\bot\} = \{\bot\})
\]

\[
\implies O(\exists at', p) = \{\bot\} \quad (O(\exists t, p) = \exists O(\exists t, p) \ [23]).
\]

Hence, a contradiction.

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

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

\[
O_\lambda(\tilde{a}FAIL, P) = \{\top\} \quad \text{(since } P(a) = \emptyset) \)
\]

\[
\implies \forall O_\lambda(\tilde{a}FAIL, P) = \{\top\} \quad \text{(Truth table: } \forall \{\top\} = \{\top\})
\]

\[
\implies O_\lambda(\forall \tilde{a}FAIL, P) = \{\top\} \quad \text{(Lemma 5: } O_\lambda(\forall t, P) = \forall O_\lambda(t, P)) \) .
\]

But, \( O(\tilde{a}FAIL, p) = \{\bot\} \) (since \( p \not\rightarrow q \), \( O(FAIL, q) = \{\bot\} \). Thus, \( O(\forall a, p) = \forall O(\tilde{a}FAIL, p) = \{\bot\} \), which is a contradiction.

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

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

\[
O_\lambda(\tilde{a}t', P) = \{\top\} \quad \text{(since } O_\lambda(t', Q_i) = \{\top\})
\]

\[
\implies \forall O_\lambda(\tilde{a}t', P) = \{\top\} \quad \text{(Truth table: } \forall \{\top\} = \{\top\})
\]

\[
\implies O_\lambda(\forall \tilde{a}t', P) = \{\top\} \quad \text{(Lemma 5: } O_\lambda(\forall t, P) = O_\lambda(\forall t, P)) \) .
\]

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

\[
\bot \in O(\tilde{a}t', p) \quad \text{(Truth table of } \forall \)
\]

\[
\implies O(\forall \tilde{a}t', p) = \{\bot\} \quad \text{(Truth of } O(\forall t, p) = O(\forall t, p) \ [23]).
\]

Hence, a contradiction.

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

\[
O_\lambda(\tilde{a}t', P) = \{\bot\} \quad \text{(since } O_\lambda(t', Q_i) = \{\bot\})
\]

\[
\implies \exists O_\lambda(\tilde{a}t', P) = \{\bot\} \quad (\exists \{\bot\} = \{\bot\})
\]

\[
\implies O_\lambda(\exists \tilde{a}t', P) = \{\bot\} .
\]
Furthermore, \( O(t', q) = O(t_1, q) \land \cdots \land O(t_n, q) = \{ \top \} \). Thus,
\[
\begin{align*}
\top & \in O(a t', p) \\
\Rightarrow & \exists O(a t', p) = \{ \top \} \\
\Rightarrow & O(\exists a t', p) = \{ \top \} \\
\Rightarrow & \exists O(t, p) = O(\exists t, p) \quad [23].
\end{align*}
\]
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 \( t = \text{SUCC}, \text{FAIL}, t_1 \lor t_2, t_1 \land t_2, \forall t', \exists t' \) are straightforward. The interesting cases are the following:

1. Let \( t = a t' \).
   (a) Let \( \top \in O_\lambda(a t', P) \). Then,
   \[
   \begin{align*}
   P & \not\vdash_\phi Q, \wedge \lambda \models \phi \wedge \top \in O_\lambda(t', Q) \\
   \Rightarrow & p \not\vdash_\phi q \\
   \Rightarrow & \top \in O_\lambda(t', Q) \Rightarrow \top \in O(t', q) \\
   \Rightarrow & \top \in O(\exists t, P).
   \end{align*}
   \]
   (b) Let \( \bot \in O_\lambda(a t', P) \). Then we have the following cases:
   i. Either \( P \not\vdash_\phi Q, \wedge \lambda \models \phi \wedge \bot \in O_\lambda(t', Q) \), for some \( Q, \phi \). Similar to Case (a).
   ii. Or, \( \exists Q, \phi P \not\vdash_\phi Q, \wedge \lambda \models \phi \). Then, \( O_\lambda(a t', P) = \{ \bot \} \). Suppose otherwise, \( \top \in O(\exists t', P) \). Then, \( \exists q \not\vdash_\phi q \). But, \( P \vdash_\lambda p \). Thus, \( \exists Q, \phi P \not\vdash_\phi Q, \wedge \lambda \models \phi \), which is a contradiction.
   (c) Let \( \top \in O(\exists t', P) \). Similar to Case (a).
   (d) Let \( \bot \in O(\exists t', P) \). Similar to Case (b).

2. Let \( t = \bar{a} t' \). Similar to the Case (1). \( \square \)

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 \( \theta \) be a configuration vector. Then,
\[
(P, v) \vdash_\theta p \iff \forall t \in T O_\theta(t, P, v) = O(t, p).
\]

Proof. Similar to the proof of Theorem 6. \( \square \)

5. Conclusions

In this paper, we studied three fundamental behavioral models for software product lines, namely, modal transition systems, feature 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, feature 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 feature transition systems. We leave proving these conjectures, as well as devising the appropriate extensional notion of testing for GEMTSs for future work.


