On the expressiveness of modal transition systems with variability constraints

On the Expressiveness of Modal Transition Systems with Variability Constraints

Maurice H. ter Beek[a,*], Ferruccio Damiani[b], Stefania Gnesi[a], Franco Mazzanti[a], Luca Paolini[b]

[a] *ISTI–CNR, Pisa, Italy*
[b] *Università di Torino, Italy*

Abstract

We demonstrate that modal transition systems with variability constraints are equally expressive as featured transition systems, by defining a transformation of the latter into the former, a transformation of the former into the latter, and proving the soundness and completeness of both transformations. Modal transition systems and featured transition systems are widely recognised as fundamental behavioural models for software product lines and our results thus contribute to the expressiveness hierarchy of such basic models studied in other papers published in this journal.

Keywords: software product lines, formal specification, behavioural specification, modal transition systems, featured transition systems

1. Introduction

Modern software systems are often developed and managed as software product lines (SPLs) to allow for mass customisation of many individual product variants [1]. The variability among the instances of such highly-configurable, variant-rich systems is expressed in terms of features, which conceptualise pieces of functionality or aspects of a system that are relevant to the stakeholders [2]. Foundational formal models for the specification and verification of SPL behaviour have been the subject of extensive research

*Corresponding author. Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Via G. Moruzzi 1, 56124 Pisa, Italy.

Email addresses: `maurice.terbeek@isti.cnr.it` (Maurice H. ter Beek), `ferruccio.damiani@unito.it` (Ferruccio Damiani), `stefania.gnesi@isti.cnr.it` (Stefania Gnesi), `franco.mazzanti@isti.cnr.it` (Franco Mazzanti), `luca.paolini@unito.it` (Luca Paolini)

throughout the last decade [3–16]. Most fundamental behavioural models for SPLs are based on the superimposition of multiple labelled transition systems (LTSs), each of which represents a different variant (a product model), in one single LTS enriched with feature-based variability (a family model). Consequently, a family's products, i.e. ordinary LTSs, can be derived from the enriched LTS by *resolving* this variability. This boils down to deciding which 'variable' (i.e. optional) behaviour to include in a specific product and which not, based on the combination of features defining the product.

In [17], some of the most fundamental behavioural models for SPLs were compared with respect to their expressiveness, which was defined as the set of (product) variants (modelled as LTSs) that can be derived from these models according to some (product derivation) refinement relation. In particular, it was demonstrated that modal transition systems (MTSs) are less expressive than featured transition systems (FTSs). Furthermore, an FTS was provided for which it was demonstrated that it cannot be encoded as an MTS.

In [18], we informally presented an automatic technique to transform an FTS into an MTS with variability constraints (MTSυ), which is an extension of MTSs introduced in [15], and we sketched a proof of the soundness of this model transformation (cf. Theorem 1 in [18]). In this paper, we contribute to the expressiveness hierarchy of fundamental behavioural models for SPLs studied in [17], by proving that finite-state MTSυs are equally expressive as finite-state FTSs:

- We first prove that MTSυs are at least as expressive as FTSs by defining an algorithm that transforms any FTS into an MTSυ and proving its soundness and completeness (i.e. an MTSυ results with the same set of variant LTSs as the original FTS)—we thus formalise and improve the procedure sketched informally in [18]. Moreover, to illustrate our result, we transform both the aforementioned FTS from [17], reproduced in Example 25, and a more elaborate SPL example from [11], into MTSυs.

- Next, we prove that MTSυs are equally expressive as FTSs by defining an algorithm that transforms any MTSυ into an FTS and proving its soundness and completeness (i.e. an FTS results with the same set of variant LTSs as the original MTSυ). We illustrate this by an example.

Moreover, the transformation algorithm from FTS to MTSυ preserves the original (compact) branching structure, thus paving the way for using an

(optimised) version for family-based SPL model checking of FTSs with the variability model checker VMC [19, 20], which currently accepts only MTSυ.

The outline of the paper is as follows. In Section 2, we define LTSs and a few standard notions used in the sequel, after which we define FTSs and MTSυs in Sections 3 and 4, respectively. Our main contributions are presented next: in Section 5, we present an algorithm to transform any FTS into an MTSυ with a proof of soundness and completeness, followed in Section 6 by an algorithm to transform any MTSυ into an FTS together with its soundness and completeness proof. In Section 7, we embed our results in the literature, after which Section 8 concludes the paper and mentions possible future work.

2. Labelled Transition Systems

We start by introducing LTSs which are the common underlying (semantic) structure for FTSs and MTSs.

Definition 1 (Labelled transition system)**.** *A* labelled transition system *is a quadruple* $(Q, \Sigma, \bar{q}, \delta)$*, where* Q *is a finite (non-empty) set of states,* Σ *is a set of actions,* $\bar{q} \in Q$ *is an initial state, and* $\delta \subseteq Q \times \Sigma \times Q$ *is a transition relation. We call* $(q, a, q') \in \delta$ *an* a*-transition (from source* q *to target* q'*) and we may also write it as* $q \xrightarrow{a} q'$*.*

We formalise two standard notions concerning LTSs in the next definition.

Definition 2 (Path, reachable)**.** *Let* $\mathcal{L} = (Q, \Sigma, \bar{q}, \delta)$ *be an LTS. Then* σ *is a* path *of* $\mathcal{L}$ *if* $\sigma = \bar{q}$ *(empty path) or* $\sigma = q_1 a_1 q_2 a_2 q_3 \cdots$ *with* $q_1 = \bar{q}$ *and* $q_i \xrightarrow{a_i} q_{i+1}$ *for all* $i > 0$ *(possibly infinite path); its* i*th state is denoted by* $\sigma(i)$ *and its* i*th action is denoted by* $\sigma\{i\}$*. A state* $q \in Q$ *is* reachable *in* $\mathcal{L}$ *if there exists a path* σ *such that* $\sigma(i) = q$ *for some* $i > 0$*. An action* $a \in \Sigma$ *is* reachable *in* $\mathcal{L}$ *if there exists a path* σ *such that* $\sigma\{i\} = a$*, for some* $i > 0$*.*

Example 3. *In Fig. 1, we depict an LTS with 7 actions* (E, x, a, m, p, ℓ, e)*. Paths start from initial state* 1*, including infinite path* $1E2x3a6m7p8\ell9e1\cdots$ *which implies that all states and all actions are reachable.*

Since we will be studying the expressiveness of behavioural models, we restrict our attention to LTSs without unreachable states. In particular, when deriving an LTS from an FTS we will drop all the unreachable states and both their ingoing and their outgoing transitions. Note, however, that we will

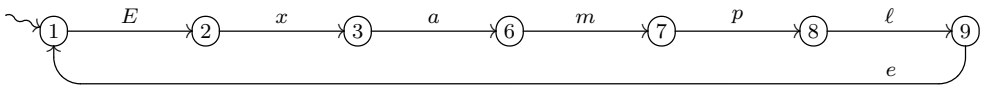


Figure 1: Example LTS $\mathcal{L}$

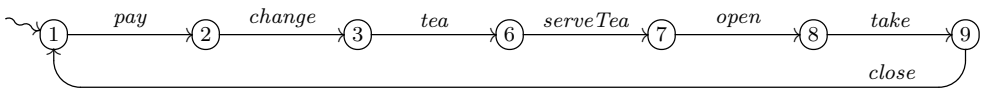


Figure 2: LTS $\rho(\mathcal{L})$: the ρ-relabelling of the LTS $\mathcal{L}$ of Example 3 depicted in Fig. 1

admit LTSs including unreachable actions (i.e. not labelling transitions) as is done, e.g., in [21]. This is because we will study sets of LTSs (i.e. product models) generated from a common set of actions (viz. of the family model).

We define an action relabelling for LTSs, which will be used in the sequel.

Definition 4 (Action relabelling)**.** *Let $\mathcal{L} = (Q, \Sigma_1, \bar{q}, \delta)$ be an LTS and let $\rho : \Sigma_1 \to \Sigma_2$ be a relabelling function. The ρ-relabelling of $\mathcal{L}$ is the LTS $\rho(\mathcal{L}) = (Q, \Sigma_2, \bar{q}, \{\, (q, \rho(a), q') \mid (q, a, q') \in \delta \,\})$.*

Relabelling is commonly adopted to reuse a given specification (model) with different action names.

It is worth noting that the above relabelling function is not required to be injective, in accord with similar operators defined in [21–23]. This choice allows us to collapse different actions to the same action (e.g. it is quite usual to collapse different actions on a generic (irrelevant) internal action).

Example 5. *The ρ-relabelling $\rho(\mathcal{L})$ of the LTS $\mathcal{L}$ of Example 3, with $\rho = \{(E, pay), (x, change), (a, tea), (m, serveTea), (p, open), (\ell, take), (e, close)\}$, is depicted in Fig. 2.*

3. Featured Transition Systems

An SPL is a set of (software-intensive) products, called *(product) variants* here, in a product portfolio of a manufacturer or software house that share substantial similarities and that are, ideally, generated from a common set of reusable (software) components by means of well documented variability [2]. A *feature* represents an abstract description of functionality, and a *feature model* can be used to provide an abstract description of (product) variants in terms of features: each (product) variant is identified by a set of features,

called a *(product) configuration* (cf. the example feature model discussed in Example 9 below). It is worth observing that a (product) configuration can be represented by a Boolean assignment to the features (i.e. selected $= \top$ and unselected $= \bot$), and a feature model can be represented by a *feature expression* (a Boolean formula over the features).

FTSs were introduced in [8, 11] to concisely model SPL behaviour, where the behaviour of its (product) variants is modelled by LTSs. An FTS is an LTS equipped with a function that labels each transition with a feature expression which needs to hold for this specific transition to be part of executable (product) variant behaviour (according to some feature model). An FTS captures a family of LTSs, one per (product) variant, which can be obtained by projection (pruning away transitions not belonging to the variant).

We largely follow the definitions from [17]. Let $\mathbb{B} = \{\top, \bot\}$ denote the Boolean constants true ($\top$) and false ($\bot$). Moreover, let $\mathbb{B}(F)$ denote the set of Boolean expressions over a set of features F (i.e. using features as propositional variables); its elements are called feature expressions.

Definition 6 (Featured transition system)**.** *A* featured transition system *is a sextuple* $(Q, \Sigma, \bar{q}, \delta, F, \Lambda)$*, where* Q *is a finite (non-empty) set of states,* Σ *is a set of actions,* $\bar{q} \in Q$ *is an initial state,* $\delta \subseteq Q \times \Sigma \times \mathbb{B}(F) \times Q$ *is a transition relation,* F *is a set of features, and* $\Lambda \subseteq \{\, \lambda : F \to \mathbb{B} \,\}$ *is a set of* (product) configurations*. We call* $(q, a, \phi, q') \in \delta$ *an* a*-transition (limited to configurations satisfying* ϕ*) and we may also write it as* $q \xrightarrow{a/\phi} q'$*.*

The notions from Definition 2 (path, reachable) are carried over as usual.

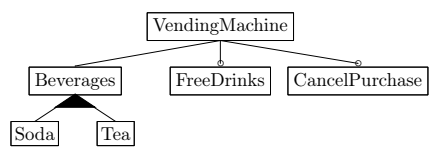
Remark 7. *Definition 6 is slightly different than the definition of FTSs given in [17], where any pair of* a*-transitions, for some* a*, between two states is required to be labelled with the same feature expression, no initial state is distinguished, and—more importantly—the set of states may be infinite. The latter is explicitly used to obtain the expressiveness results reported in [17] (cf. Section 8).*

We say that a configuration $\lambda \in \Lambda$ *satisfies* a feature expression $\phi \in \mathbb{B}(F)$, denoted by $\lambda \models \phi$, whenever the interpretation of ϕ via λ is true, i.e. if the result of substituting the value of the features occurring as variables in ϕ according to λ is $\top$. The variant LTS defined by a particular (product) configuration $\lambda \in \Lambda$ of an FTS is obtained from the latter by first removing all transitions whose feature expressions are not satisfied by λ (this operation is called *projection*) and then removing all the unreachable states.

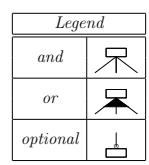
Definition 8 ((Product) variant of FTS)**.** *Let* $\mathcal{F} = (Q, \Sigma, \bar{q}, \delta, F, \Lambda)$ *be an FTS. The projection of* F *on its (product) configuration* $\lambda \in \Lambda$ *is the LTS* $\mathcal{L} = (Q, \Sigma, \bar{q}, \delta')$*, where* $\delta' = \{\, (q, a, q') \mid (q, a, \phi, q') \in \delta \text{ and } \lambda \models \phi \,\}$*. Let* $\mathcal{F}|_\lambda$ *denote the LTS that is obtained from* $\mathcal{L}$ *by removing all unreachable states and their outgoing transitions. Then* $\mathcal{F}|_\lambda$ *is called a (product)* variant *of* $\mathcal{F}$*.*

The set of variants derived from an FTS $\mathcal{F}$ is denoted by $\mathsf{lts}(\mathcal{F})$. Note that all variants of an FTS are thus LTSs, without unreachable states and without transitions that cannot be executed as a part of any path; however, they may contain unreachable actions. Furthermore, the variants do not contain states or actions that were not already present in the FTS.

Example 9. *In Fig. 3, we reproduce the FTS behaviour of a beverage vending machine SPL example from [11]. It has the following 12 actions: pay, free, change, cancel, return, soda, tea, serveSoda, serveTea, take, open and close. According to its feature model, reproduced on the right, it has six features: VendingMachine (v), Beverages (b), FreeDrinks (f), CancelPurchase (c), Soda (s) and Tea (t).*



Its feature model defines the 12 product configurations in Λ*.*

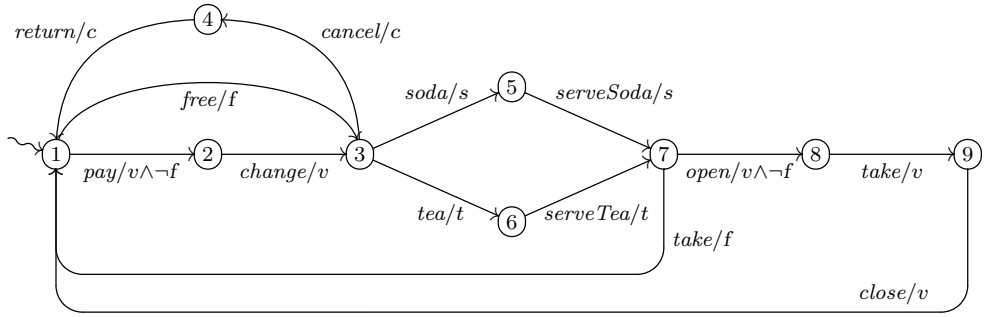


The LTS behaviour of the variant $\mathcal{F}|_\lambda$*, with* $\lambda = \{v, b, t\}$ *(i.e.* $\lambda(v) = \top$*,* $\lambda(b) = \top$*,* $\lambda(s) = \bot$*,* $\lambda(t) = \top$*,* $\lambda(f) = \bot$*,* $\lambda(c) = \bot$*), of this FTS* $\mathcal{F}$ *coincides with the LTS* $\rho(\mathcal{L})$ *depicted in Fig. 2.*

4. Modal Transition Systems with variability constraints

An MTS is an LTS that distinguishes between admissible (may) and necessary (must) transitions. MTSs were introduced in [24] to capture the refinement of a partial description into a more detailed one, reflecting increased knowledge on the admissible (but not necessary) behaviour. We follow the definitions from [15].

Definition 10 (Modal transition system)**.** *A* modal transition system *is a quintuple* $(Q, \Sigma, \bar{q}, \delta^\Diamond, \delta^\Box)$ *such that* $\delta^\Box \subseteq \delta^\Diamond$ *and* $(Q, \Sigma, \bar{q}, \delta^\Diamond)$ *is an LTS. We distinguish the transition relation* $\delta^\Diamond$ *expressing* admissible (may) *transitions and the transition relation* $\delta^\Box$ *expressing* necessary (must) *transitions, whereas the transitions in* $\delta^\Diamond \setminus \delta^\Box$ *are called* optional *transitions. We may also write* $q \xrightarrow{a}_\Diamond q'$ *for* $(q, a, q') \in \delta^\Diamond$*,* $q \xrightarrow{a}_\Box q'$ *for* $(q, a, q') \in \delta^\Box$ *and* $q \dashrightarrow q'$ *for* $(q, a, q') \in \delta^\Diamond \setminus \delta^\Box$*.*



$$F = \{v, b, s, t, f, c\}$$
$$\Lambda = \{\{v, b, t\}, \{v, b, t, f\}, \{v, b, t, c\}, \{v, b, t, f, c\}, \{v, b, s\}, \{v, b, s, f\}, \{v, b, s, c\}, \{v, b, s, f, c\}, \{v, b, s, t\}, \{v, b, s, t, f\}, \{v, b, s, t, c\}, \{v, b, s, t, f, c\}\}$$

Figure 3: FTS $\mathcal{F}$ of the beverage vending machine SPL example from [11]

Note that any (may) transition of an MTS is either a must transition or an optional transition. When drawing MTSs in this paper, we will depict their must transitions (as solid lines) and their optional transitions (as dashed lines). Again, the notions from Definition 2 (path, reachable) are carried over in the usual way.

MTSs describe all possible behaviour by means of variability modelled through optional transitions, i.e. admissible (may) but not necessary (must) transitions. Concrete variant behaviour in the form of LTSs can be obtained by resolving this variability, i.e. deciding for each optional transition whether or not it is included (executable) in a particular variant. This implies a notion of conformance to define when an LTS conforms to an MTS. We know from [3] that the traditional (strong and weak refinement) semantics of MTSs is not capable of capturing a notion of conformance that is suitable for SPL modelling. One of the problems is that MTS behaviour might not be preserved in a *consistent* way in variant LTSs, in the sense that it is in principle possible to decide that some occurrences of optional a-transitions are included while other occurrences are not—we consider such decisions inconsistent since, in SPL terminology, features are either included or not.

Another problem, illustrated in [11, 25], is that the optional transitions of an MTS are all independently optional, in the sense that the decision to include an optional a-transition is by definition independent of the decision to include an optional b-transition. In other words, there is no inherent mechanism to declare such transitions to be in an alternative (xor) relation.

We provide examples of the issues raised above. Consider the MTS in Fig. 4(left). The only variants that we consider to be consistent (formalised in Definition 15(3) below) are the four LTSs on the right: hence an LTS with the a-transition and only one of the b-transitions does *not* model acceptable product behaviour. Furthermore, if a and b were alternative (i.e. a xor b), then both the leftmost LTS and the rightmost LTS would no longer model acceptable product behaviour.

We refer the reader to [15] for a more detailed discussion and for further examples (cf. also Example 25 in Section 5.3 below).

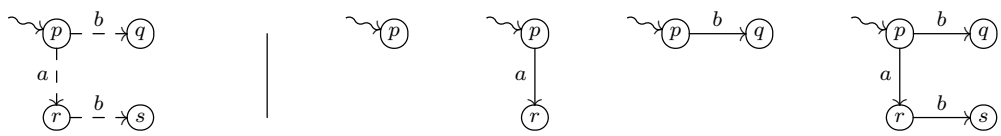


Figure 4: An MTS and four valid variants; only the two central LTSs are valid if a xor b

Coming from the desire to express intuitive specification requirements like persistent choices, in [26] so-called *parametric* MTSs are introduced, which allow to choose in a consistent (persistent) way whether or not to implement a transition in a product by using parameters with a priori fixed (Boolean) values that settle this choice for the entire product.

In [15], it was shown how to make MTSs amenable to SPL modelling and analysis by equipping so-called *coherent* MTSs with an additional set of variability constraints over actions. In the following definitions, we recall the notion of *MTS with variability constraints* and a syntactic operational definition to derive (product) variant LTSs which, in [15], were shown to be equivalent—modulo bisimulation (cf., e.g., [21, 27])—to the LTSs obtained by means of a special-purpose semantic refinement relation.

Definition 11 (Coherent MTS)**.** *An MTS* $\mathcal{M} = (Q, \Sigma, \bar{q}, \delta^{\Diamond}, \delta^{\Box})$ *is* coherent *whenever for all* $(p, a, p') \in \delta^{\Box}$ *and* $(q, b, q') \in \delta^{\Diamond} \setminus \delta^{\Box}$*, it holds that* $a \neq b$*.*

From now on, we consider only coherent MTSs, i.e. MTSs such that the set of actions labelling necessary transitions and the set of actions labelling optional transitions are disjoint. We inherit this from [15], where MTSυs are only defined for coherent MTSs. The motivation given in [15] is that an (often implicit) assumption underlying most of the fundamental behavioural models for SPLs based on LTSs [6], FTSs [8, 11, 28], MTSs [3, 9, 25, 29], I/O automata [4, 7] and mCRL2 [30] is that an action models a piece of functionality (or, a feature) which by definition either is optional or is not.

Definition 12 (Variability constraints)**.** *Let $\mathcal{L} = (Q, \Sigma, \overline{q}, \delta)$ be an LTS. We define the syntax and semantics of the following six types of* variability constraints *on the reachability of actions of $\mathcal{L}$ formalised as propositional formulae*[1] *over the actions of Σ interpreted as propositional variables. Let $a_i \in \Sigma$, for all $i \geq 1$, and let $m \geq 2$ and $n \geq 3$.*

1. *$b_1 \vee b_2 \vee \cdots \vee b_m$, where $b_i \in \{a_i, \neg a_i\}$ for all $1 \leq i \leq m$ (i.e. each atom b_i is either equal to a_i or to its negation): for at least one b_i, $1 \leq i \leq m$, it holds that either*
   - *$b_i = a_i$ and a_i is reachable in $\mathcal{L}$, or*
   - *$b_i = \neg a_i$ and a_i is not reachable in $\mathcal{L}$*
2. *$a_1 \oplus a_2 \oplus \cdots \oplus a_m$: precisely one among $a_1, \ldots, a_m$ is reachable in $\mathcal{L}$*
3. *$a_1 \uparrow a_2$: at most one among a_1 and a_2 is reachable in $\mathcal{L}$*
4. *$a_1 \rightarrow a_2$: a_2 is reachable in $\mathcal{L}$ whenever a_1 is reachable in $\mathcal{L}$*
5. *$a_1 \rightarrow (a_2 \oplus a_3 \oplus \cdots \oplus a_n)$: precisely one among $a_2, \ldots, a_n$ is reachable in $\mathcal{L}$ whenever a_1 is reachable in $\mathcal{L}$*
6. *$a_1 \rightarrow (a_2 \vee a_3 \vee \cdots \vee a_n)$: at least one among $a_2, \ldots, a_n$ is reachable in $\mathcal{L}$ whenever a_1 is reachable in $\mathcal{L}$*

Any propositional logic formula can be converted into an equivalent formula in conjunctive normal form (CNF), i.e. a conjunction of disjunctions of literals[2] by applying the laws of distribution, De Morgan's laws, and by removing double negations, possibly requiring exponential time [31]. Hence, the variability constraint of type 1 in Definition 12 suffices to define all propositional formulae over Σ; more precisely, any propositional formula ϕ over the actions of Σ interpreted as propositional variables can be defined by a set of disjunctive formulae of the form $\{b_{1_1} \vee b_{1_2} \vee \cdots \vee b_{1_n}, \ldots, b_{m_1} \vee b_{m_2} \vee \cdots \vee b_{m_n}\}$, with $b_{i_j} \in \{\, a_j, \neg a_j \mid \Sigma = \{a_1, \ldots, a_n\} \text{ and } 1 \leq j \leq n \,\}$ for some $m \geq 1$ and $1 \leq i \leq m$, which all need to hold to satisfy ϕ. However, we provide the other five types of variability constraints from [15], which stem directly from feature models and are all accepted by the model checker VMC [19, 20], as syntactic sugar.

[1] Note that $\oplus$ is the *exclusive or* and $\uparrow$ is the negation of conjunction (a.k.a. *not and*).

[2] A literal is a propositional variable or its negation, which are also called positive and negative literals, respectively.

Remark 13. *In the sequel, we will freely use any type of propositional formula over a set of actions since we know that it can always be converted into a set of disjunctive formulae (cf. type 1 from Definition 12).*

Definition 14 (MTS with variability constraints)**.** *A* modal transition system with variability constraints *is a sextuple* $(Q, \Sigma, \bar{q}, \delta^{\Diamond}, \delta^{\Box}, \Upsilon)$ *such that* $(Q, \Sigma, \bar{q}, \delta^{\Diamond}, \delta^{\Box})$ *is a coherent MTS and* Υ *is a set of variability constraints on actions from* Σ*.*

Intuitively, a variant (LTS) is derived from an MTSυ by including all must transitions of the MTSυ, together with a subset of its optional transitions. More precisely, the variant LTS has the same set of actions and the same initial state as the MTSυ, but a subset of the set of states of the MTSυ and a subset of its set of transitions such that the following four conditions are satisfied: (i) all states of the LTS are reachable from its initial state; (ii) all must transitions of the MTSυ are included in the LTS (except those must transitions whose source states are not reachable in the LTS); (iii) for any action a, whenever an a-transition of the MTSυ is included in the LTS, then any other (optional) a-transition in the MTSυ (from a state that is reachable in the LTS) is also included (i.e. the decision to turn one optional a-transition into a necessary a-transition must be *consistently* repeated for all other optional a-transitions); (iv) all variability constraints are satisfied. This operational derivation procedure is formalised in the next definition.

Definition 15 ((Product) variant of MTSυ)**.** *Let* $\mathcal{M} = (Q, \Sigma, \bar{q}, \delta^{\Diamond}, \delta^{\Box}, \Upsilon)$ *be an MTSυ. Then the LTS* $\mathcal{L} = (Q^v, \Sigma, \bar{q}, \delta^v)$ *is a (product)* variant *derived from* $\mathcal{M}$ *whenever* $Q^v \subseteq Q$ *and* $\delta^v \subseteq \delta^{\Diamond}$ *are such that the following holds:*

1. *Every* $q \in Q^v$ *is reachable in* $\mathcal{L}$
2. *There exists no* $(q, a, q') \in \delta^{\Box} \setminus \delta^v$ *such that* $q \in Q^v$
3. *For any* $a \in \Sigma$*, whenever* $(p, a, p') \in \delta^v$ *for some* $p, p' \in Q^v$*, then for all* $q, q' \in Q^v$ *such that* $(q, a, q') \in \delta^{\Diamond}$ *it must be the case that also* $(q, a, q') \in \delta^v$ *(consistent inclusion proviso)*
4. $\mathcal{L}$ *satisfies all variability constraints in* Υ

The set of variants derived from an MTSυ $\mathcal{M}$ is denoted by $\mathsf{lts}(\mathcal{M})$. Note that the variants do not contain states or actions that were not already present in the MTSυ. This is due to the syntactic operational definition to derive variants, basically pruning away transitions not belonging to the variant, which differs fundamentally from the modal refinement relation of

MTSs, which results in LTSs that need not preserve an MTS's branching structure, as noted in [3]. This moreover implies that any MTSυ has a finite number of variants, while the number of variants in which no more refinement is possible (usually called implementations) of an MTS is in general infinite.

We now provide a small example to illustrate these differences. Consider the MTS in Fig. 5(left) and some of its infinite number of implementations on the right. Seen as an MTSυ (with an empty set of variability constraints) only the two LTSs with a single state (initial state p) are variants by Definition 15.

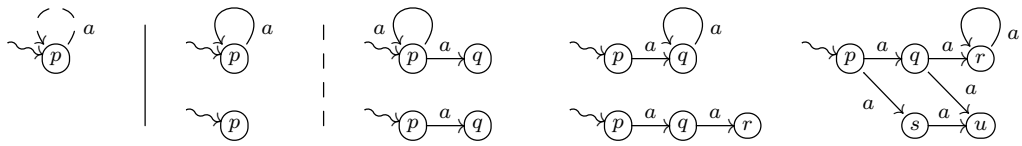
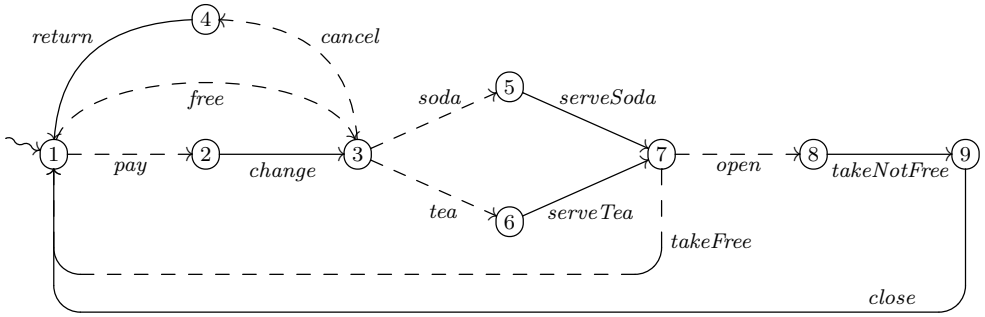


Figure 5: An MTS(υ) and some of its implementations (variants)

Example 16. *In Fig. 6, we depict an MTSυ $\mathcal{M}$ that is intended to model the behaviour of the beverage vending machine SPL example from [11], using essentially the same actions as the FTS $\mathcal{F}$ of Example 9 (except that here we distinguish the actions takeFree and takeNotFree instead of a single action take in $\mathcal{F}$). The set of variability constraints Υ is included in the figure.*

Note that the LTS $\mathcal{F}|_{\{v,b,t\}}$ of Example 9, depicted in Fig. 2, is a variant of the MTSυ $\mathcal{M}$ up to a relabelling of action *take* in *takeNotFree. In particular, every state of $\mathcal{F}|_{\{v,b,t\}}$ is reachable, $\mathcal{F}|_{\{v,b,t\}}$ has no must transition that is not a transition in $\mathcal{M}$ and $\mathcal{F}|_{\{v,b,t\}}$ satisfies all variability constraints. Finally, due to the action relabelling, $\mathcal{M}$ does not have two different transitions with the same action label (meaning that the consistent inclusion proviso is trivially satisfied). Hence, all conditions of Definition 15 are indeed satisfied.*

Note that the transitions labelled with return, serveSoda and serveTea could also have been designated optional rather than necessary if at the same time the variability constraints cancel $\leftrightarrow$ return, soda $\leftrightarrow$ serveSoda and tea $\leftrightarrow$ serveTea were included in Υ. This would not change $\mathsf{lts}(\mathcal{M})$ *since the respective pairs of transitions (e.g.* $(3, \mathit{cancel}, 4)$ *and* $(4, \mathit{return}, 1)$*) would still always either both be present or both be absent in any variant of $\mathcal{M}$. The same does not hold for the two pairs of transitions* $(1, \mathit{pay}, 2)$ *and* $(7, \mathit{open}, 8)$ *and* $(1, \mathit{free}, 3)$ *and* $(7, \mathit{takeFree}, 1)$*: constraints pay $\leftrightarrow$ open and free $\leftrightarrow$ takeFree, respectively, are needed to guarantee for each pair of transitions that either both its transitions are present or both are absent in variants of $\mathcal{M}$.*



$$\Upsilon = \{pay \oplus free,\ pay \leftrightarrow open,\ free \leftrightarrow takeFree,\ soda \vee tea\}$$

Figure 6: MTSυ of the beverage vending machine SPL example from [11]

Before turning our attention to the main contributions of this paper in Sections 5 and 6, viz. transformations from FTSs into MTSυs and vice versa that preserve the set of variants, it is important to note that the formal semantics of the variability constraints of an MTSυ is defined in terms of the *reachability* of actions in its variants. This follows directly from Definitions 12 and 15(4), both inherited from [15]. This differs fundamentally from FTSs, where a variant is obtained by projecting on the variant's configuration, i.e. pruning away transitions labelled with feature expressions that are not satisfied by the configuration defining the variant, and subsequently removing all unreachable states and their outgoing transitions. The transformations we will define in the next sections need to take this difference into account.

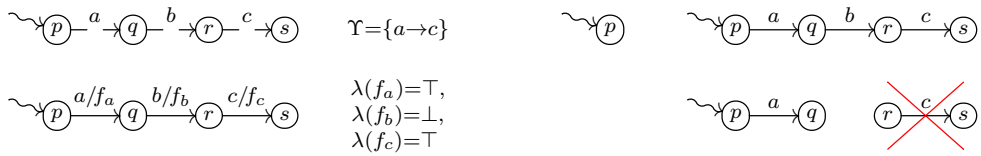


Figure 7: An MTSυ with two variants (top) and an FTS with a single variant (bottom)

Consider the MTSυ $\mathcal{M}$ in Fig. 7(top left). It is easy to see that its only two variants are those depicted on its right, which are obtained by either pruning all optional transitions or turning all of them into necessary transitions. Both satisfy the variability constraint that action c is reachable whenever action a is. Now note, in particular, that turning the optional a-transition into a necessary transition while pruning the optional b-transition, will *not* result in an LTS that satisfies the variability constraint $a \rightarrow c$ (independent of the decision taken for the optional c-transition).

Next consider the FTS $\mathcal{F}$ depicted in Fig. 7(bottom left). Its only variant, obtained by first projecting on the given configuration λ and then removing the unreachable states r and s and the c-transition from r to s, is depicted on its right. It is important to note that this variant is not a variant of the MTSυ $\mathcal{M}$, even though λ satisfies $f_a \to f_c$, where f_a and f_c are the feature expressions associated with the actions a and c, respectively. In order to obtain the rightmost variant of $\mathcal{M}$ from an FTS $\mathcal{F}'$ obtained from $\mathcal{F}$ by replacing λ with λ', the latter configuration needs to quantify over the feature expression f_b associated with the action b (even though b is not quantified over by the variability constraint $a \to c$), viz. $\lambda'(f_a) = \lambda'(f_b) = \lambda'(f_c) = \top$. Instead, the other variant of $\mathcal{M}$, consisting only of the initial state, can be obtained by replacing λ with any configuration λ'' such that $\lambda''(f_a) = \bot$.

5. From FTS to MTSυ

In the previous sections, we have presented the behavioural models for SPLs considered in this paper. In Section 5.1, we define an algorithm to transform any FTS into an MTSυ with the same variants and we prove the soundness and completeness of this transformation in Section 5.2, i.e. MTSυ are at least as expressive as FTSs. Formally, a behavioural SPL formalism M' is said to be *at least as expressive as* a behavioural SPL formalism M if and only if there exists a transformation from M into M', denoted by $\tau\colon M \to M'$, such that for all models $\mathcal{M} \in M$, the sets of derived variants $\mathsf{lts}(\mathcal{M})$ and $\mathsf{lts}(\tau(\mathcal{M}))$ coincide, possibly up to dummy transitions and action relabelling (both of which can be ignored for behavioural analyses); M is said to be *less expressive than* M' if and only if no such transformation from M' to M exists (i.e. M is not at least as expressive as M' while M' is at least as expressive as M). We thus formalise and improve the procedure sketched informally in [18]. In Section 5.3, we pinpoint the specific features of MTSυs that make them at least as expressive as FTSs, by illustrating the transformation of the FTS that cannot be encoded as an MTS (as demonstrated in [17]) into an MTSυ.

5.1. Model transformation

In this section, we define an algorithm to transform an FTS into an MTSυ. Basically, from an FTS we create an MTSυ in the following way. We define a new action for each combination of action and feature expression that effectively occurs as label of a transition in the FTS. We create dummy transitions

for all newly-defined actions and for all features. All these transitions lead from the initial state to a newly-defined deadlock state, which has no outgoing transitions (a.k.a. sink state). We create variability constraints to relate actions and features that are involved in the same FTS transition and others to encode the product configurations (i.e. the 'feature model') of the FTS. This transformation allows us to guarantee that each variant $\mathcal{L}$ derived from the MTSυ is such that its feature-labelled dummy transitions define a valid product configuration λ of the FTS and $\mathcal{L}$ (properly relabelled) is moreover equal to the variant of the FTS obtained by projecting on λ (duly augmented with some additional dummy transitions that all lead to a deadlock state). We explain the details of the definition afterwards.

Definition 17 (FTS2MTSυ transformation algorithm)**.** *Consider an FTS* $\mathcal{F} = (Q, \Sigma, \bar{q}, \delta, F, \Lambda)$*. Without loss of generality we may assume that* $\Sigma \cap F = \varnothing$*. We build an MTS$\upsilon$* $\mathcal{M}_* = (Q_*, \Sigma_*, \bar{q}, \delta_*^{\Diamond}, \delta_*^{\Box}, \Upsilon_*)$ *as follows.*

- $Q_* = Q \cup \{s\}$*, where* $s \notin Q$ *is a fresh (sink) state*
- $\Sigma_* = \{\, (a, \varphi) \mid (q, a, \varphi, q') \in \delta \,\} \cup (\Sigma \setminus \{\, a \mid (q, a, \varphi, q') \in \delta \,\}) \cup F$
- $\delta_*^{\Diamond} = \{\, (q, (a, \varphi), q') \mid (q, a, \varphi, q') \in \delta \,\} \cup \{\, (\bar{q}, \sigma, s) \mid \sigma \in \Sigma_* \setminus \Sigma \,\}$
- $\delta_*^{\Box} = \varnothing$
- $\Upsilon_* = \{\, (a, \varphi) \leftrightarrow \varphi \mid (q, a, \varphi, q') \in \delta \,\} \cup \{\bigvee_{\lambda \in \Lambda} (\bigwedge_{f \in F} \chi_\lambda(f))\}$*, where* χ_λ *is the function from* F *to* $\{\, f, \neg f \mid f \in F \,\}$ *defined as:*

$$\chi_\lambda(f) = \begin{cases} f & \text{if } \lambda(f) = \top \text{ (i.e. } \lambda \models f) \\ \neg f & \text{if } \lambda(f) = \bot \text{ (i.e. } \lambda \not\models f) \end{cases}$$

Each state of the FTS is added to the set of states of the MTSυ, together with a new (sink) state. For each feature f of the FTS, an action f is added to the set of actions of the MTSυ. For each transition (q, a, φ, q') in the FTS, an action (a, φ) is added to the set of actions of the MTSυ, a may (a, φ)-transition is added to the set of may transitions of the MTSυ and a variability constraint of type $(a, \varphi) \leftrightarrow \varphi$ is added to the set of variability constraints of the MTSυ. Moreover, if $\Sigma \setminus \{\, a \mid (q, a, \varphi, q') \in \delta \,\} \neq \varnothing$ (i.e. the FTS contains 'unused' actions), then such actions are copied into the set of actions of the MTSυ, in order to ease the comparison of their variants. For each newly-defined action of the MTSυ, a may transition from the initial state to the newly-defined sink state, labelled with that action, is added to the set of transitions of the MTSυ.

Note that the FTS2MTSv transformation creates an MTSυ without must transitions, which we recall to be a specific type of may transitions. Deciding which of the may transitions could safely be turned into a must transition without changing the behaviour nor the set of derived variants is left as an optimisation for future work. We come back to this at the end of Section 5.3. For now, we note that the second transition of a sequence of two may transitions whose labels contain the same feature expression (recall that transitions of an MTSυ are labelled with a combination of an action and a feature expression) can always safely be turned into a must transition. This can be seen as follows: whenever the first may transition is (meant to be) present in a variant, then so is the second. An even simpler case concerns may transitions whose labels contain a feature expression that is always true (either by definition, i.e. $\top$, or because it is a tautology with respect to all other constraints): also these can safely be turned into must transitions, since they are (meant to be) present in every variant.

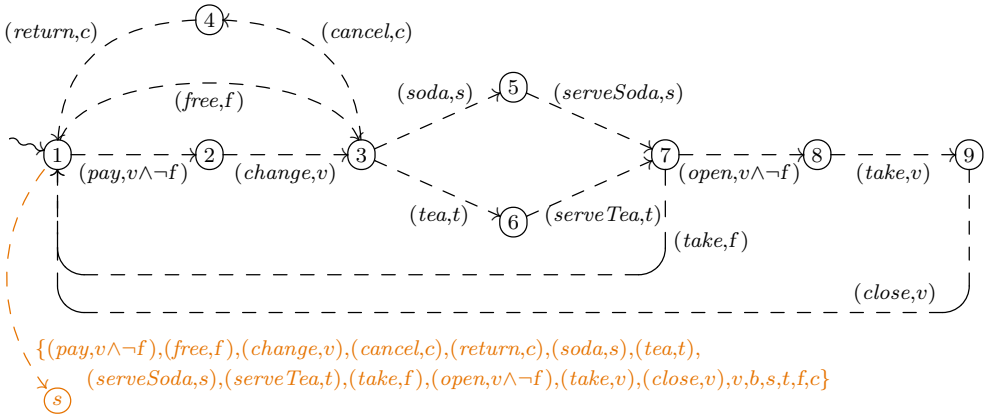
Finally, we explain in detail the creation of the set Υ_* of variability constraints. First of all, we remark that if φ is a feature expression then we can convert φ into a formula in disjunctive normal form (DNF), i.e. a disjunction of conjunctions of literals[3] (in this case, the literals are $\{ f, \neg f \mid f \in F \}$). Now φ is said to be reachable in an LTS whenever one of these conjunctions of literals is such that each of its positive literals is reachable while each of its negative literals is unreachable. Therefore, by adding a constraint of the form $(a, \varphi) \leftrightarrow \varphi$ to Υ_* for each transition (q, a, φ, q') in $\mathcal{F}$, we impose that: in all valid variants of $\mathcal{M}_*$, for each action (a, φ), the action (a, φ) is reachable if and only if φ is 'reachable'.

Next we explain the addition of the constraint $\bigvee_{\lambda \in \Lambda}(\bigwedge_{f \in F} \chi_\lambda(f))$ to Υ_*. Note that each (product) configuration $\lambda \in \Lambda$ of the FTS $\mathcal{F}$ gives rise to a characteristic formula over the set of features F, i.e. a conjunction of literals of the form $\{ f, \neg f \mid f \in F \}$, which together form a propositional formula over F in DNF. More precisely, $\bigvee_{\lambda \in \Lambda}(\bigwedge_{f \in F} \chi_\lambda(f))$ is a DNF formula of the form $(g_{1_1} \wedge g_{1_2} \wedge \cdots \wedge g_{1_n}) \vee \cdots \vee (g_{m_1} \wedge g_{m_2} \wedge \cdots \wedge g_{m_n})$, with $g_{i_j} \in \{ f_j, \neg f_j \mid F = \{f_1, \ldots, f_n\} \text{ and } 1 \leq j \leq n \}$ for some $m \geq 1$ and $1 \leq i \leq m$. Each clause $g_{i_1} \wedge g_{i_2} \wedge \cdots \wedge g_{i_n}$ characterises a valid product variant of the FTS $\mathcal{F}$,

[3] As was the case for CNF, it is well known that any propositional logic formula can be converted into an equivalent DNF formula by applying the laws of distribution, De Morgan's laws, and by removing double negations, possibly requiring exponential time [32].

viz. it contains the selected features as positive literals and the unselected features as negative literals (cf. Example 18 below).

Example 18. *In Fig. 8, we depict the MTSυ $\mathcal{M}_* = (Q_*, \Sigma_*, \bar{q}, \delta_*^{\Diamond}, \delta_*^{\Box}, \Upsilon_*)$ that is obtained by applying the transformation of Definition 17 to the FTS $\mathcal{F}$ of Example 9. The sink state and all dummy transitions*[4] *of $\mathcal{M}_*$ are coloured to emphasise their special status. Note that its underlying MTS is coherent.*



$$
\begin{aligned}
\Upsilon_* = \{&(pay, v \wedge \neg f) \leftrightarrow (v \wedge \neg f),\ (free, f) \leftrightarrow f,\ (change, v) \leftrightarrow v,\\
&(cancel, c) \leftrightarrow c,\ (return, c) \leftrightarrow c,\ (soda, s) \leftrightarrow s,\ (tea, t) \leftrightarrow t,\\
&(serveSoda, s) \leftrightarrow s,\ (serveTea, t) \leftrightarrow t,\ (take, f) \leftrightarrow f,\\
&(open, v \wedge \neg f) \leftrightarrow (v \wedge \neg f),\ (take, v) \leftrightarrow v,\ (close, v) \leftrightarrow v\}\\
\cup\ \{&(v \wedge b \wedge \neg s \wedge t \wedge \neg f \wedge \neg c) \vee (v \wedge b \wedge \neg s \wedge t \wedge f \wedge \neg c) \vee\\
&(v \wedge b \wedge \neg s \wedge t \wedge \neg f \wedge c) \vee (v \wedge b \wedge \neg s \wedge t \wedge f \wedge c) \vee\\
&(v \wedge b \wedge s \wedge \neg t \wedge \neg f \wedge \neg c) \vee (v \wedge b \wedge s \wedge \neg t \wedge f \wedge \neg c) \vee\\
&(v \wedge b \wedge s \wedge \neg t \wedge \neg f \wedge c) \vee (v \wedge b \wedge s \wedge \neg t \wedge f \wedge c) \vee\\
&(v \wedge b \wedge s \wedge t \wedge \neg f \wedge \neg c) \vee (v \wedge b \wedge s \wedge t \wedge f \wedge \neg c) \vee\\
&(v \wedge b \wedge s \wedge t \wedge \neg f \wedge c) \vee (v \wedge b \wedge s \wedge t \wedge f \wedge c)\}
\end{aligned}
$$

Figure 8: MTSυ obtained by transforming the FTS depicted in Fig. 3

In Fig. 9, we depict an LTS $\mathcal{L}^*$. Note that $\mathcal{L}^*$ is a variant of $\mathcal{M}_*$ because every state of $\mathcal{L}^*$ is reachable, $\mathcal{L}^*$ has no must transition that is not a

[4]Here, and likewise in the sequel, we use a set notation $(1, \{(pay, v \wedge \neg f), \ldots, c\}, s) \in \delta_*^{\Diamond}$ as shorthand for $(1, (pay, v \wedge \neg f), s), \ldots, (1, c, s) \in \delta_*^{\Diamond}$.

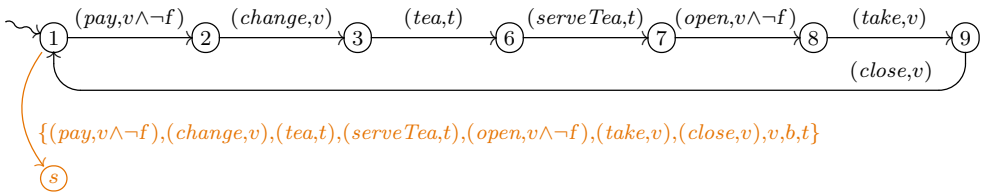


Figure 9: LTS $\mathcal{L}^*$ that is a variant derived from the MTSυ $\mathcal{M}_*$

transition in $\mathcal{M}_*$, $\mathcal{M}_*$ does not have two different transitions with the same action label (meaning that the consistent inclusion proviso is trivially satisfied) and $\mathcal{L}^*$ satisfies all variability constraints. The latter can be seen by realising that $v \wedge b \wedge \neg s \wedge t \wedge \neg f \wedge \neg c$ is true, since actions v, b and t are reachable (by means of dummy transitions) while actions s, f and c are not; $(pay, v \wedge \neg f) \leftrightarrow (v \wedge \neg f)$ is true since actions $(pay, v \wedge \neg f)$ and v are reachable while action f is not, $(free, f) \leftrightarrow f$ is true since neither $(free, f)$ nor f is reachable, etc. All conditions of Definition 15 are thus satisfied.

Note furthermore that $\mathcal{L}^*$ equals the LTS $\mathcal{F}|_{\{v,b,t\}}$ of Example 9, depicted in Fig. 2, once all dummy transitions of $\mathcal{L}^*$ are removed and all actions of $\mathcal{L}^*$ are relabelled according to the function ρ defined by $\rho((a, \varphi)) = a$ for all $(a, \varphi) \in \Sigma_* \setminus F$ and $\rho(f) = f$ for all $f \in F$.

Note that the algorithm in Definition 17 transforms feature names (of the FTS) into action names used in dummy transitions (of the MTSυ $\mathcal{M}_*$). Each variant of the resulting MTSυ $\mathcal{M}_*$ includes a selection (possibly empty) of such (dummy) transitions that identifies a product configuration. Therefore, to be able to prove the soundness and completeness of the transformation of Definition 17 in the next section (viz. that for any given FTS, an MTS is constructed such that it has the same set of derived variants), we need to augment the variants of the FTS with some additional dummy transitions (which all lead to a deadlock state).

Definition 19 (Dummy-extended variants)**.** *Let $\mathcal{F} = (Q, \Sigma, \bar{q}, \delta, F, \Lambda)$ be an FTS, let $\lambda \in \Lambda$ be a valid product inducing the variant $\mathcal{F}|_\lambda = (Q_\lambda, \Sigma, \bar{q}, \delta_\lambda)$ and let $s \notin Q$ be a fresh state.*

Set $\delta_F^s = \{\, (\bar{q}, f, s) \mid f \in F \text{ and } \lambda(f) \models \top \,\}$. We define two extended LTSs:

1. *$\mathcal{F}|_\lambda^s = (Q_\lambda \cup \{s\}, \Sigma \cup F, \bar{q}, \delta_\lambda \cup \delta_F^s \cup \delta_\Sigma^s)$ is the s-*extension *of $\mathcal{F}|_\lambda$, where $\delta_\Sigma^s = \{\, (\bar{q}, a, s) \mid (q, a, \varphi, q') \in \delta \text{ and } \lambda \models \varphi \,\}$*
2. *$\mathcal{F}\|_\lambda^s = (Q_\lambda \cup \{s\}, \Sigma_*, \bar{q}, \delta_* \cup \delta_F^s \cup \delta_{\Sigma_*}^s)$ is the* action-extension *of $\mathcal{F}|_\lambda$, where*

- $\Sigma_* = \{ (a, \varphi) \mid (q, a, \varphi, q') \in \delta \} \cup (\Sigma \setminus \{ a \mid (q, a, \varphi, q') \in \delta \}) \cup F$
- $\delta_* = \{ (q, (a, \phi), q') \mid (q, a, \phi, q') \in \delta_\lambda \ \textit{and} \ \lambda \models \phi \}$
- $\delta^s_{\Sigma_*} = \{ (\bar{q}, (a, \varphi), s) \mid (q, a, \varphi, q') \in \delta \ \textit{and} \ \lambda \models \varphi \}$

Given a product configuration λ, the s-extension $\mathcal{F}|^s_\lambda$ adds to the transition relation of variant $\mathcal{F}|_\lambda$ two sets of dummy transitions: a transition for each feature in the product variant determined by λ (labelled by the feature itself) and one for each action label of a transition of $\mathcal{F}|_\lambda$. On the other hand, the action-extension $\mathcal{F}\|^s_\lambda$ picks up $\mathcal{F}|^s_\lambda$ up to an action relabelling: $\mathcal{F}|^s_\lambda$ can be obtained from $\mathcal{F}\|^s_\lambda$ via a relabelling $\rho : \Sigma_* \to \Sigma \cup F$; it collapses all actions of the form (a, φ) on the same a. Indeed, the Algorithm in Definition 17 uses more informative action names than those used in the original FTS and such information has to be erased to prove the soundness of the transformation. This is illustrated below (Example 20) and proved afterwards (Lemma 21). It is worthwhile noticing that (i) all transitions included in $\mathcal{F}|_\lambda$ and $\mathcal{F}\|_\lambda$ are reachable, because those in δ_λ are reachable by construction (cf. Definition 8) and (ii) the actions $\Sigma \setminus \{ a \mid (q, a, \varphi, q') \in \delta \}$ do not label any transition.

Example 20. *In Fig. 10, we depict the s-extension $\mathcal{F}|^s_{\{v,b,t\}}$ of variant $\mathcal{F}|_{\{v,b,t\}}$, depicted in Fig. 2, of the FTS $\mathcal{F}$ from Example 9. Moreover, the LTS $\mathcal{L}^*$ depicted in Fig. 9 is its action-extension $\mathcal{F}\|^s_{\{v,b,t\}}$. Note that $\mathcal{F}|^s_{\{v,b,t\}}$ is equal to the ρ-relabelling of $\mathcal{F}\|^s_{\{v,b,t\}}$, where ρ is defined by $\rho((a, \varphi)) = a$ for all $(a, \varphi) \in \Sigma_* \setminus F$ and $\rho(f) = f$ for all $f \in F$ (i.e. it is the identity otherwise).*

Lemma 21. *Let $\mathcal{F} = (Q, \Sigma, \bar{q}, \delta, F, \Lambda)$ be an FTS, let $\lambda \in \Lambda$ be a valid product inducing the s-extended variant $\mathcal{F}|^s_\lambda$ and the action-extended variant $\mathcal{F}\|^s_\lambda$ such that $s \notin Q$ is a fresh state. If ρ is the relabelling defined by $\rho((a, \varphi)) = a$ for all $(a, \varphi) \in \Sigma_* \setminus F$ and it behaves as the identity in all other cases, then $\rho(\mathcal{F}\|^s_\lambda) = \mathcal{F}|^s_\lambda$.*

Proof. Following Definition 19, let $\mathcal{F}|^s_\lambda = (Q_\lambda \cup \{s\}, \Sigma \cup F, \bar{q}, \delta_\lambda \cup \delta^s_F \cup \delta^s_\Sigma)$ and let $\mathcal{F}\|^s_\lambda = (Q_\lambda \cup \{s\}, \Sigma_*, \bar{q}, \delta_* \cup \delta^s_F \cup \delta^s_{\Sigma_*})$. It is immediate that $\rho(\Sigma_*) = \Sigma \cup F$ and that $\rho(\delta_* \cup \delta^s_F \cup \delta^s_{\Sigma_*}) = \rho(\delta_*) \cup \delta^s_F \cup \rho(\delta^s_{\Sigma_*}) = \delta_\lambda \cup \delta^s_F \cup \delta^s_\Sigma$. □

Some hints concerning the negligibility of dummy-transitions (e.g. for model-checking purposes) can be found in Example 25 below.

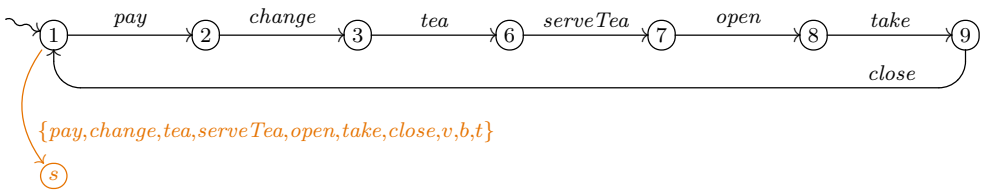


Figure 10: The s-extension $\mathcal{F}|^{s}_{\{\upsilon,b,t\}}$ of variant $\mathcal{F}|_{\{\upsilon,b,t\}}$

5.2. Soundness and completeness of the transformation

In this section, we prove that the transformation from FTS to MTSυ that we defined by the algorithm given in Definition 17 is sound and complete: it always results in an MTSυ with the same set of derived variants as the original FTS (up to dummy-extensions).

We first prove that, given an MTSυ $\mathcal{M}_*$ generated from an FTS $\mathcal{F}$, any variant derived from $\mathcal{M}_*$ is an action-extension of a variant of $\mathcal{F}$.

Lemma 22 (Each variant of the MTSυ corresponds to a variant of the FTS)**.** *Let $\mathcal{F} = (Q, \Sigma, \bar{q}, \delta, F, \Lambda)$ be an FTS and let $\mathcal{M}_* = (Q_*, \Sigma_*, \bar{q}, \delta_*^{\Diamond}, \delta_*^{\Box}, \Upsilon_*)$ be the MTSυ generated from $\mathcal{F}$ according to the FTS2MTSυ Algorithm in Definition 17. If $\mathcal{L}^\star$ is a variant derived from $\mathcal{M}_*$ and $s \in Q_* \setminus Q$, then there exists a $\lambda \in \Lambda$ such that $\mathcal{L}^\star = \mathcal{F}\|^{s}_{\lambda}$.*

Proof. Let $\mathcal{L}^\star = (Q^\star, \Sigma_*, \bar{q}, \delta^\star)$ be a variant derived from $\mathcal{M}_*$. $\mathcal{L}^\star$ thus satisfies all variability constraints in Υ_* (cf. Definition 15) and, in particular, $\bigvee_{\lambda\in\Lambda}(\bigwedge_{f\in F} \chi_\lambda(f))$ (included in Υ_* by Definition 17). These constraints express which of the features (considered as actions in Σ_*) must be reachable in each variant of $\mathcal{M}_*$; more precisely, their satisfiability requires that the set of features that is reachable in a variant of $\mathcal{M}_*$ identifies a set X of features such that there is a unique $\lambda \in \Lambda$ for which $f \in X$ iff $\lambda(f) = \top$. Let $\lambda^\star \in \Lambda$ be the unique configuration identified by the reachable features of $\mathcal{L}^\star$. We aim to prove that $\mathcal{L}^\star = \mathcal{F}\|^{s}_{\lambda^\star}$.

Let $\mathcal{F}\|^{s}_{\lambda^\star} = (Q^s_{\lambda^\star}, \Sigma^s_{\lambda^\star}, \bar{q}, \delta^s_{\lambda^\star})$. By Definitions 8, 17 and 19, $\mathcal{F}\|^{s}_{\lambda^\star}$ satisfies the following three conditions: (i) $Q^s_{\lambda^\star} = \{\, q \in Q \mid q \text{ is reachable in } \mathcal{F}|_{\lambda^\star} \,\} \cup \{s\}$; (ii) $\Sigma^s_{\lambda^\star} = \{\, (a, \varphi) \mid (q, a, \varphi, q') \in \delta \,\} \cup (\Sigma \setminus \{\, a \mid (q, a, \varphi, q') \in \delta \,\}) \cup F$; (iii) $\delta^s_{\lambda^\star} = \delta_* \cup \delta^s_F \cup \delta^s_{\Sigma_*}$, with $\delta_* = \{\, (q, (a, \varphi), q') \mid (q, a, \varphi, q') \in \delta,\ \lambda^\star \models \varphi$ and q is reachable in $\mathcal{F}|_{\lambda^\star} \,\}$, $\delta^s_F = \{\, (\bar{q}, f, s) \mid f \in F \text{ and } \lambda^\star \models f \,\}$ and $\delta^s_{\Sigma_*} = \{\, (\bar{q}, (a, \varphi), s) \mid (q, a, \varphi, q') \in \delta \text{ and } \lambda^\star \models \varphi \,\}$. We aim to show that $\delta^s_{\lambda^\star} = \delta^\star$, from which $Q^s_{\lambda^\star} = Q^\star$ follows.

We exhaustively explore transitions, by first considering the two types of dummy-transitions.

- $(\bar{q}, f, s) \in \delta_F^s$ iff $\lambda^\star(f) \models \top$ iff $\chi_{\lambda^\star}(f) = \top$ iff f is reachable in $\mathcal{L}^\star$ iff $(\bar{q}, f, s) \in \delta^\star$.

- $(\bar{q}, (a, \varphi), s) \in \delta_{\Sigma_*}^s$ iff $\lambda^\star \models \varphi$ and $(a, \varphi) \in \Sigma_*$ iff (a, φ) is reachable in $\mathcal{L}^\star$ (because $\lambda^\star \models \varphi$ means that there exists an equivalent formula in DNF such that each of its positive literals is a reachable feature and each of its negative literals is an unreachable feature, moreover $(a, \varphi) \leftrightarrow \varphi \in \Upsilon_*$) iff $(\bar{q}, (a, \varphi), s) \in \delta^\star$.

We conclude by considering transitions $(\tilde{q}, (\tilde{a}, \tilde{\varphi}), \tilde{q}')$ such that $\tilde{q}' \neq s$.

- We assume that $(\tilde{q}, (\tilde{a}, \tilde{\varphi}), \tilde{q}') \in \delta_*$, thus $(q, a, \varphi, q') \in \delta$ is a reachable transition and $\lambda^\star \models \varphi$. The reachability means that there is a path starting in $\bar{q}$ that reach $\tilde{q}$, viz. for some $n \geq 1$ there must exist $(q_{i-1}, a_i, \varphi_i, q_i) \in \delta$, with $1 \leq i \leq n$, such that $q_1 = \bar{q}$, $q_n = \tilde{q}$ and $\lambda^\star \models \varphi_i$. For all $1 \leq i \leq n$, $(a_i, \varphi_i) \leftrightarrow \varphi_i \in \Upsilon_*$ by the Algorithm in Definition 17, so that our choice of λ^*, and the consistent inclusion proviso of Definition 15 imply that $(\tilde{q}, (\tilde{a}, \tilde{\varphi}), \tilde{q}') \in \delta^\star$.

- We assume $(\tilde{q}, (\tilde{a}, \tilde{\varphi}), \tilde{q}') \in \delta^\star$ such that $\tilde{q}' \neq s$. The reachability of $\tilde{q}$ in $\mathcal{L}^\star$ means that for some $n \geq 1$ there must exist transitions $(q_i, (a_i, \varphi_i), q_{i+1}) \in \delta_*^\Diamond$, with $1 \leq i \leq n$, such that $q_0 = \bar{q}$ and $q_n = \tilde{q}'$. Furthermore, the reachability constraints $(a_i, \varphi_i) \leftrightarrow \varphi_i$ in Υ_* imply that $\lambda^\star \models \varphi_i$. Therefore, we can conclude $(\tilde{q}, (\tilde{a}, \tilde{\varphi}), \tilde{q}') \in \delta_*$ and the proof is done. $\square$

Next we prove that, given an MTSυ $\mathcal{M}_*$ generated from an FTS $\mathcal{F}$, any action-extension of a variant of $\mathcal{F}$ is a variant derived from $\mathcal{M}_*$.

Lemma 23 (Each variant of the FTS corresponds to a variant of the MTSυ)**.** *Let $\mathcal{F} = (Q, \Sigma, \bar{q}, \delta, F, \Lambda)$ be an FTS and let $\mathcal{M}_* = (Q_*, \Sigma_*, \bar{q}, \delta_*^\Diamond, \delta_*^\Box, \Upsilon_*)$ be the MTSυ generated from $\mathcal{F}$ according to the FTS2MTSυ Algorithm in Definition 17. If $\lambda \in \Lambda$ and $s \in Q_* \backslash Q$, then $\mathcal{F}\|_\lambda^s$ is a variant derived from $\mathcal{M}_*$.*

Proof. Let $\lambda \in \Lambda$ and $s \in Q_* \setminus Q$. We have to prove that $\mathcal{F}\|_\lambda^s$, i.e. the action-extended projection of $\mathcal{F}$ on $\lambda \in \Lambda$, is a variant derived from $\mathcal{M}_*$, i.e. that it respects Definition 15. Note that the initial state of $\mathcal{M}_*$ equals that of $\mathcal{F}$ (cf. Algorithm in Definition 17) and it is moreover the same for all their variants (cf. Definitions 8 and 15), which implies that $\bar{q}$ is the initial state of all LTSs involved in this proof (and $\bar{q}$ is always reachable).

Let $\mathcal{F}\|^s_\lambda = (Q^s_\lambda, \Sigma^s_\lambda, \bar{q}, \delta^s_\lambda)$. By Definitions 8 and 19, $\mathcal{F}|^s_\lambda$ satisfies the following three conditions: (i) $Q^s_\lambda = \{\, q \in Q \mid q \text{ is reachable in } \mathcal{F}|_\lambda \,\} \cup \{s\}$; (ii) $\Sigma^s_\lambda = \{\, (a,\varphi) \mid (q,a,\varphi,q') \in \delta \,\} \cup (\Sigma \setminus \{\, a \mid (q,a,\varphi,q') \in \delta \,\}) \cup F$; (iii) $\delta^s_\lambda = \delta_* \cup \delta^s_F \cup \delta^s_{\Sigma_*}$, with $\delta_* = \{\, (q,(a,\varphi),q') \mid (q,a,\varphi,q') \in \delta,\ \lambda \models \varphi$ and q is reachable in $\mathcal{F}|_\lambda \,\}$, $\delta^s_F = \{\, (\bar{q},f,s) \mid f \in F \text{ and } \lambda(f) \models \top \,\}$ and $\delta^s_{\Sigma_*} = \{\, (\bar{q},(a,\varphi),s) \mid (q,a,\varphi,q') \in \delta \text{ and } \lambda \models \varphi \,\}$.

We also note the following four relations, directly from Definition 17: $Q^s_\lambda \subseteq Q_* = Q \cup \{s\}$; $\Sigma^s_\lambda = \Sigma_*$ (and by Definition 15 this is also the set of actions of all variants derived from $\mathcal{M}_*$); for all transitions $(q,(a,\varphi),q') \in \delta_*$ such that $(q,a,\varphi,q') \in \delta$, there exists a transition $(q,(a,\varphi),q') \in \delta^\diamond_*$; $\delta^s_F \cup \delta^s_{\Sigma_*} \subseteq \delta^\diamond_*$ (and by Definition 19 all transitions in $\delta^s_F \cup \delta^s_{\Sigma_*}$ are outgoing transitions of a reachable state in $\mathcal{F}|^s_\lambda$, viz. $\bar{q}$).

We now check that $\mathcal{F}|^s_\lambda$ satisfies all four conditions of Definition 15.

1. As shown above, Q^s_λ is by construction restricted to reachable states.
2. Trivially $\delta^\square_* \setminus \delta^s_\lambda = \varnothing$, since $\delta^\square_* = \varnothing$ by Definition 17.
3. For any $\tilde{a} \in \Sigma^s_\lambda$, the consistent inclusion proviso is satisfied:
   - $\tilde{a} \in (\Sigma \setminus \{\, a \mid (q,a,\varphi,q') \in \delta \,\})$ makes the proviso trivially true.
   - If $(\bar{q},\tilde{a},s) \in \delta^s_F$, then $\tilde{a} \in F$ and for all $q,q' \in Q_*$ such that $(q,\tilde{a},q') \in \delta^\diamond_*$, it is the case that $q = \bar{q}$ and $q' = s$, thus the proof is immediate.
   - If $(\bar{q},(\tilde{a},\tilde{\varphi}),s) \in \delta^s_{\Sigma_*}$, then $(q,\tilde{a},\tilde{\varphi},q') \in \delta$ and $\lambda \models \tilde{\varphi}$. Now assume there exist $r,r' \in Q^s_\lambda$ such that $(r,(\tilde{a},\tilde{\varphi}),r') \in \delta^\diamond_*$. Then it remains to show that also $(r,(\tilde{a},\tilde{\varphi}),r') \in \delta^s_\lambda$. Since $(r,(\tilde{a},\tilde{\varphi}),r') \in \delta^\diamond_*$, by Definition 17, we have $(r,\tilde{a},\tilde{\varphi},r') \in \delta$. Hence, by Definitions 8 and 19, $(r,(\tilde{a},\tilde{\varphi}),r') \in \delta^s_\lambda$.
4. $\mathcal{F}\|^s_\lambda$ satisfies all variability constraints in Υ_*:
   - The variability constraints $\{\, (a,\varphi) \leftrightarrow \varphi \mid (q,a,\varphi,q') \in \delta \,\}$ are satisfied once we show that for any $(q,a,\varphi,q') \in \delta$, the action (a,φ) is reachable in $\mathcal{F}\|^s_\lambda$ whenever $\lambda \models \varphi$ and vice versa. Easily, (a,φ) is reachable iff $(q',(a,\varphi),q'') \in \delta^s_\lambda$ for some $q',q'' \in Q^s_\lambda$ iff either $(q',(a,\varphi),q'') \in \delta_*$ or $(q',(a,\varphi),q'') \in \delta^s_{\Sigma_*}$ (and both of them include only transitions such that $\lambda \models \varphi$ by definition).
   - The variability constraints $\{\bigvee_{\lambda\in\Lambda}(\bigwedge_{f\in F}\chi_\lambda(f))\}$ are satisfied once we prove that $\bigwedge_{f\in F}\chi_\lambda(f)$ is satisfied for our given $\lambda \in \Lambda$. Recall from Definition 17 that $\chi_\lambda(f) = f$ if $\lambda(f) = \top$ and $\chi_\lambda(f) = \neg f$ if

$\lambda(f) = \bot$. By the definition of δ^s_F, for all $f \in F$, f is reachable in $\mathcal{F}|^s_\lambda$ iff $\lambda(f) = \top$. $\square$

Together with Lemma 21, Lemmata 22 and 23 imply the soundness and completeness of the transformation proposed in Definition 17.

Theorem 24 (Soundness and completeness of FTS2MTSυ transformation)**.** *Let $\mathcal{F} = (Q, \Sigma, \bar{q}, \delta, F, \Lambda)$ be an FTS and let $\mathcal{M}_* = (Q_*, \Sigma_*, \bar{q}, \delta^\diamond_*, \delta^\Box_*, \Upsilon_*)$ be the MTSυ generated from $\mathcal{F}$ according to the FTS2MTSυ Algorithm in Definition 17. The sets of derived variants $\mathsf{lts}(\mathcal{F})$ and $\mathsf{lts}(\mathcal{M}_*)$ coincide, up to dummy transitions and action relabelling.*

Proof. Straightforward, by Lemmata 21, 22 and 23. $\square$

5.3. Discussion

To pinpoint the specific features of MTSυs that make them at least as expressive as FTSs, we illustrate the transformation into an MTSυ of the FTS that was introduced in [17] and for which it was demonstrated that it cannot be encoded as an MTS.

Example 25. *In Fig. 11 (left), we have drawn the FTS $\mathcal{F}$ from Example 8 of [17], with features $F = \{f, f'\}$ and product configurations $\Lambda = \{\lambda, \lambda'\}$ with $\lambda(f) = \top$, $\lambda(f') = \bot$ and $\lambda'(f) = \bot$, $\lambda'(f') = \top$, in the notation of this paper. Its variants $\mathsf{lts}(\mathcal{F}) = \{\mathcal{F}|_\lambda, \mathcal{F}|_{\lambda'}\}$ are depicted in Fig. 11 (right).*

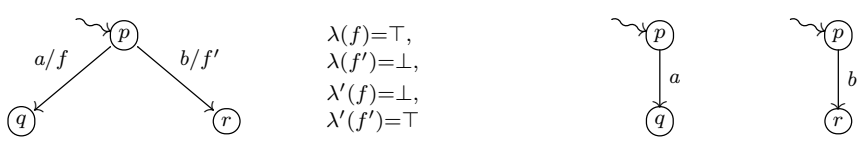


Figure 11: FTS $\mathcal{F}$ (left) and $\mathsf{lts}(\mathcal{F}) = \{\mathcal{F}|_\lambda, \mathcal{F}|_{\lambda'}\}$ (right) from Example 8 of [17]

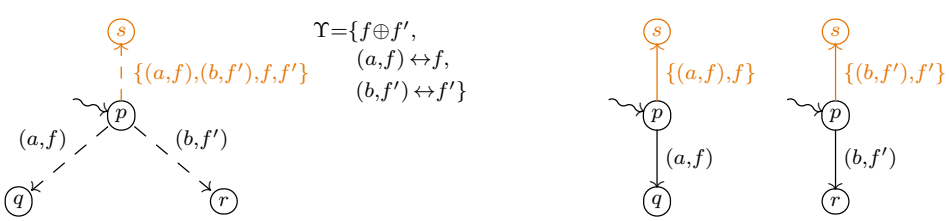


Figure 12: MTSυ $\mathcal{M}$ (left) and $\mathsf{lts}(\mathcal{M})$ (right) obtained by transforming FTS $\mathcal{F}$

In Fig. 12 (left), we have drawn the MTSυ $\mathcal{M}$, with variability constraints $\Upsilon = \{(a, f) \leftrightarrow f, (b, f') \leftrightarrow f', f \oplus f'\}$, that is obtained by transforming the FTS $\mathcal{F}$ and by using the fact that $f \oplus f'$ is equivalent to $\bigvee_{\lambda \in \Lambda}(\bigwedge_{f \in F} \chi_\lambda(f)) = (f \wedge \neg f') \vee (\neg f \wedge f')$. Its variants $\mathsf{lts}(\mathcal{M})$ are depicted in Fig. 12 (right).

It is straightforward to see that $\mathsf{lts}(\mathcal{F})$ and $\mathsf{lts}(\mathcal{M})$ coincide, up to dummy transitions (from the initial state to a sink state) and action relabelling (viz. $\rho((a, f)) = a$ and $\rho((b, f')) = b$). At the same time, it is immediately clear that MTSυ, as well as the variants that can be derived from them, can be used as is *(i.e. without modifications) for model checking. It basically suffices to realise that it is straightforward to ignore all dummy transitions when traversing the transition system. In [18], we demonstrated how to do so with the variability model checker VMC [19, 20].*

Note that the variability constraints together with the dummy transitions prohibit the derivation of a variant that contains both transitions $(p, (a, f), q)$ and $(p, (b, f'), r)$, which instead cannot be avoided in the case of MTSs without *variability constraints, as was demonstrated in [17]. In general, the consistency proviso guarantees that whenever an optional transition of the MTSυ is included in a variant LTS, then also any other reachable optional transition labelled with the same action is included. This implies in particular the inclusion of the appropriate dummy transitions, since these are all optional transitions from the initial state (i.e. reachable by definition).*

The FTS2MTSυ transformation algorithm in Definition 17 leads to an MTSυ that generally could be optimised in several ways without changing its behaviour nor its variants, such as reducing the so-called descriptional complexity of the MTSυ (like the number of variability constraints) or improving the efficiency of model checking properties over the MTSυ or deriving variants from it. For instance, it is not difficult to see that the set of variability constraints of the MTSυ $\mathcal{M}_*$ of Example 18 could be reduced in size and several of its optional transitions could be turned into must transitions (cf. the MTSυ $\mathcal{M}$ of Example 16). In [18], we discussed two such optimisations. Note, however, that the FTS2MTSυ transformation algorithm preserves the original (compact) branching structure of the FTS. To be precise, the resulting MTSυ has one additional state and dummy transitions to that state. This paves the way for using (optimised) versions of the MTSυs for family-based SPL model checking of FTSs with the variability model checker VMC [19, 20], which currently accepts only MTSυ.

6. From MTSυ to FTS

In Section 6.1, we define an algorithm to transform any MTSυ into an FTS with the same variants, after which we prove the soundness and completeness of this transformation in Section 6.2. Hence FTSs are at least as expressive as MTSυ. Together with the results from Section 5, this proves that MTSυs are equally expressive as FTSs. Formally, a formalism M' is said to be *equally expressive as* a formalism M if and only if M' is at least as expressive as M and M is at least as expressive as M'. In Section 6.3, we briefly discuss the complexity of the transformation algorithm defined in Section 6.2.

6.1. Model transformation

To simplify the next definition, we first introduce two auxiliary functions.

- The function $A : Q \times \Sigma \times Q \to \Sigma$ maps a transition (q, a, q') into its action a; when applied to a transition relation δ the function A returns the set of actions $A(\delta) = \{\, a \in \Sigma \mid (q, a, q') \in \delta \,\}$.
- The function $\digamma$ maps each action a to a distinguished feature f_a; when applied to a set of actions Σ, the function $\digamma$ returns a set of features $\digamma(\Sigma) = \{\, f_a \mid a \in \Sigma \,\}$ containing a distinguished feature for each action in Σ.

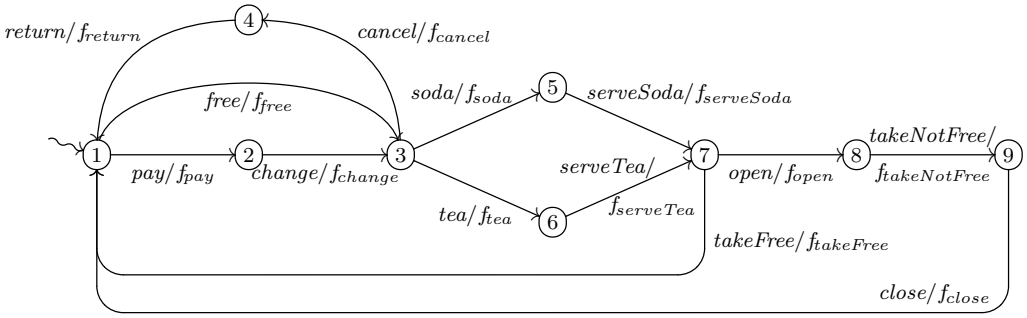
Given an MTSυ $\mathcal{M}$ with actions Σ, we define an FTS $\mathcal{F}_\circ$ with actions Σ and features $F_\circ = \digamma(\Sigma_\circ)$, where $\Sigma_\circ \subseteq \Sigma$ is the set of actions that are actually used as labels of transitions of the variants $\mathsf{lts}(\mathcal{M})$ of $\mathcal{M}$.

Definition 26 (MTSυ2FTS transformation algorithm)**.** *Consider an MTSυ $\mathcal{M} = (Q, \Sigma, \bar{q}, \delta^\diamond, \delta^\square, \Upsilon)$. We build an FTS $\mathcal{F}_\circ = (Q, \Sigma, \bar{q}, \delta_\circ, F_\circ, \Lambda_\circ)$ in which $F_\circ = \digamma(A(\delta^\diamond))$, $\delta_\circ = \{\, (q, a, f_a, q') \mid (q, a, q') \in \delta^\diamond \,\}$ and $\Lambda_\circ = \{\, \digamma(A(\delta^\upsilon)) \mid (Q^\upsilon, \Sigma, \delta^\upsilon, \bar{q}) \in \mathsf{lts}(\mathcal{M}) \,\}$.*

Example 27. *In Fig. 13, we depict the FTS $\mathcal{F}_\circ = (Q, \Sigma, \bar{q}, \delta_\circ, F_\circ, \Lambda_\circ)$ that is obtained by applying the transformation of Definition 26 to the MTSυ $\mathcal{M}$ depicted in Fig. 6. The sets of features $F_\circ$ and configurations $\Lambda_\circ$ are included in the figure. Recall that $\mathcal{M}$ was designed from scratch to have the same variants as the FTS $\mathcal{F}$ of Example 9, depicted in Fig. 3.*

It is worth observing that the output of the algorithm in Definition 26 depends only on the set of variants of the MTSυ in input. For instance,

consider the modifications of the MTSυ $\mathcal{M}$ of Fig. 6 proposed in the last paragraph of Example 16. Since none of the suggested modifications changes $\mathcal{M}$'s set of variants, the application of the algorithm in Definition 26 to the modified MTSυ would still produce the FTS depicted in Fig. 13.



$F_\circ = \{f_{pay}, f_{change}, f_{return}, f_{cancel}, f_{tea}, f_{soda}, f_{serveTea}, f_{serveSoda}, f_{open}, f_{takeFree}, f_{takeNotFree}, f_{close}\}$

$\Lambda_\circ = \{\{f_{pay}, f_{change}, f_{tea}, f_{serveTea}, f_{open}, f_{takeFree}, f_{close}\},$
$\{f_{free}, f_{tea}, f_{serveTea}, f_{takeNotFree}, f_{close}\},$
$\{f_{pay}, f_{change}, f_{return}, f_{cancel}, f_{tea}, f_{serveTea}, f_{open}, f_{takeFree}, f_{close}\},$
$\{f_{free}, f_{return}, f_{cancel}, f_{tea}, f_{serveTea}, f_{takeNotFree}, f_{close}\},$
$\{f_{pay}, f_{change}, f_{soda}, f_{serveSoda}, f_{open}, f_{takeFree}, f_{close}\},$
$\{f_{free}, f_{soda}, f_{serveSoda}, f_{takeNotFree}, f_{close}\},$
$\{f_{pay}, f_{change}, f_{return}, f_{cancel}, f_{soda}, f_{serveSoda}, f_{open}, f_{takeFree}, f_{close}\},$
$\{f_{free}, f_{return}, f_{cancel}, f_{soda}, f_{serveSoda}, f_{takeNotFree}, f_{close}\},$
$\{f_{pay}, f_{change}, f_{tea}, f_{soda}, f_{serveTea}, f_{serveSoda}, f_{open}, f_{takeFree}, f_{close}\},$
$\{f_{free}, f_{tea}, f_{soda}, f_{serveTea}, f_{serveSoda}, f_{takeNotFree}, f_{close}\},$
$\{f_{pay}, f_{change}, f_{return}, f_{cancel}, f_{tea}, f_{soda}, f_{serveTea}, f_{serveSoda}, f_{open}, f_{takeFree}, f_{close}\},$
$\{f_{free}, f_{return}, f_{cancel}, f_{tea}, f_{soda}, f_{serveTea}, f_{serveSoda}, f_{takeNotFree}, f_{close}\}\}$

Figure 13: FTS obtained by transforming the MTSυ depicted in Fig. 6

Lemma 28. *Let $\mathcal{M} = (Q, \Sigma, \bar{q}, \delta^\diamond, \delta^\square, \Upsilon)$ be an MTSυ such that $\mathcal{L}_0, \mathcal{L}_1 \in \mathsf{lts}(\mathcal{M})$. If $\mathcal{L}_0 = (Q_0, \Sigma, \delta_0, \bar{q})$, $\mathcal{L}_1 = (Q_1, \Sigma, \delta_1, \bar{q})$ and $A(\delta_0) = A(\delta_1)$, then $\mathcal{L}_0$ and $\mathcal{L}_1$ are the same LTS.*

Proof. The proof is straightforward, because the variants $\mathcal{L}_0$ and $\mathcal{L}_1$ have to satisfy the consistent inclusion proviso (cf. Definition 15(3)). $\square$

6.2. Soundness and completeness of the transformation

In this section, we prove that the transformation from MTSυ to FTS that we defined by the algorithm in Definition 26 is sound and complete: it always results in an FTS with the same set of derived variants as the original MTSυ.

Theorem 29 (Soundness and completeness of MTSυ2FTS transformation)**.** *Let $\mathcal{M} = (Q, \Sigma, \bar{q}, \delta^{\Diamond}, \delta^{\Box}, \Upsilon)$ be an MTSυ and let $\mathcal{F}_{\circ} = (Q, \Sigma, \bar{q}, \delta_{\circ}, F_{\circ}, \Lambda_{\circ})$ be the FTS generated from $\mathcal{M}$ according to the MTSυ2FTS Algorithm in Definition 26. The sets of derived variants $\mathsf{lts}(\mathcal{M})$ and $\mathsf{lts}(\mathcal{F}_{\circ})$ coincide, i.e. $\mathsf{lts}(\mathcal{M}) = \mathsf{lts}(\mathcal{F}_{\circ})$.*

Proof. Each variant $\mathcal{L}^{v} = (Q^{v}, \Sigma, \delta^{v}, \bar{q}) \in \mathsf{lts}(\mathcal{M})$ of $\mathcal{M}$ is uniquely identified by the actions in $A(\delta^{v})$, by Lemma 28. Since $\digamma$ is patently a bijection, it follows that $\Lambda_{\circ}$ (i.e. the set of configurations of $\mathcal{F}_{\circ}$) is in bijection with the set of $\mathsf{lts}(\mathcal{M})$, viz. for each $\lambda^{v} \in \Lambda_{\circ}$ there is a unique associated $\mathcal{L}^{v} \in \mathsf{lts}(\mathcal{M})$. Moreover, by Definition 8, configurations in $\Lambda_{\circ}$ bijectively identify the variants of $\mathcal{F}_{\circ}$. Hence, $\mathsf{lts}(\mathcal{M})$ and $\mathsf{lts}(\mathcal{F}_{\circ})$ are in bijection via a suitable extension of $\digamma$.

If $\lambda^{v} \in \Lambda_{\circ}$ is the configuration generated by $\mathcal{L}^{v}$, in accord to Definition 26, then λ^{v} is the configuration in which all and only the features $\digamma(A(\delta^{v}))$ occur as positive literals, viz. the features associated to actions occurring in $A(\delta^{v})$. Concluding, $\mathcal{F}|_{\lambda^{v}} = \mathcal{L}^{v} = (Q^{v}, \Sigma, \delta^{v}, \bar{q})$ by Definition 8. $\Box$

6.3. Discussion

Note that the MTSυ2FTS transformation algorithm in Definition 26 has a worst-case exponential complexity, since the set of configurations of the resulting FTS is built using the set of LTSs derived by the MTSυ, which may be of size exponential in the number of features. Since one of the aims of this paper is to complement the expressiveness hierarchy of fundamental behavioural models for SPLs studied in [17], the theoretical result that MTSυs are equally expressive as FTSs, based on the transformations in both directions presented in Sections 5.1 and 6.1, is of interest. For what concerns efficient implementations of these transformations, a second aim of this paper is an FTS2MTSυ transformation algorithm that preserves the original (compact) branching structure of FTSs to pave the way for using (optimised) versions of the resulting MTSυs for family-based SPL model checking of FTSs with the variability model checker VMC, which currently accepts only MTSυ. This goal is also the reason for using the format of variability constraints in Definition 12 rather than propositional formulae (cf. Remark 13).

We consider an efficient implementation of the MTSυ2FTS transformation to be out of the scope of this paper.

7. Related work

The related literature was partially cited throughout the paper. We add here further comparisons and remarks. In [17], three fundamental behavioural models for SPLs were compared with respect to their expressive power. In addition to the models studied in the present paper, i.e. FTSs and MTSυs, in [17] also product line LTSs (PL-LTSs) were taken into consideration. PL-LTSs form the semantic model of the product line process algebra PL-CCS introduced in [6]. PL-CCS extends Milner's calculus of communicating systems (CCS [33]) with a variants operator $\oplus$ enabling the modelling of alternative behaviour, in the form of alternative processes of which only one is intended to exist at runtime (i.e. compared to CCS's standard non-deterministic choice operator $+$, the variants' choice is made once and for all).

The expressiveness results in [17] state that MTSs are less expressive than PL-LTSs, which in turn are less expressive than FTSs. To emphasise the fact that MTSs are (thus) less expressive than FTSs, in [17] it was demonstrated that there exists no encoding from the FTS $\mathcal{F}$ reproduced in Example 25 into an MTS. In the same example, we showed that $\mathcal{F}$ *can* be transformed into an MTSυ, and we pinpointed the specific features of MTSυs responsible for this difference in expressiveness between MTSs and MTSυs.

In a very recent corrigendum to [17], contained in [34], the authors of [17] reported that their definition of PL-LTSs is more restrictive than the one originally introduced in [6], upon which they have proved that adopting the original and more liberal definition, PL-LTSs are equally expressive as FTSs.

It is important to note that the results in [17, 34] are based on LTS-based SPL models with a possibly infinite number of states. Moreover, the product-derivation relation for FTSs defined in [17] generalises the one from the SPL literature as used also in the present paper. Most notably, in [17] products derived from an FTS do not need to preserve the FTS' branching structure (viz. a product may contain more states than the FTS it is derived from) and they may be infinite in number. For these two reasons, this paper complements the expressiveness hierarchy in [17] with an expressiveness result for *finite-state* behavioural SPL models.

We provide an example to illustrate the different product-derivation relation, which also illustrates the effect of a possibly infinite number of states.

Consider the FTS in Fig. 14(left) and on the right some of the infinite number of products that can be derived from it according to the product-derivation relation from [17]. According to the classical projection of an FTS on a product configuration, only the LTS with a single state (initial state p) is a variant (cf. Definition 8 and [8, 11]). Clearly, the others are bisimilar to this one. Note, however, that the MTS in Fig. 15(left), reproduced from Section 4, has an infinite number of variants (implementations in MTS terminology), some of which are depicted on the right, but that are clearly not all bisimilar.

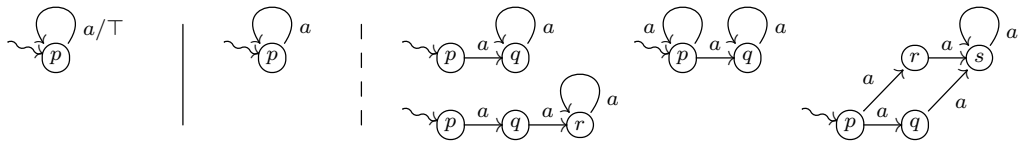


Figure 14: An FTS and some of its products according to [17, 34]

Figure 15: An MTS and some of its infinite number of implementations (variants)

In fact, MTSs and FTSs as originally introduced in [24] and [8], respectively, are equipped with two fundamentally different ways of deriving LTSs as (product) variants. In case of MTSs, an infinite number of variants (implementations in MTS terminology) is allowed also for finite-state MTSs (cf. Fig. 15). On the other hand, products derived from an FTS (through projection, cf. Definition 8) preserve *by definition* the FTS' branching structure (i.e. a product may *not* contain more states than the FTS it is derived from), and are thus finite in number.

Hence, to obtain the statement from [17] that any MTS can be encoded as an FTS, the authors of [17] chose to adapt the definition of product derivation for FTSs (and, moreover, to allow FTSs with an infinite number of states, cf. Remark 7), whereas we chose to compare finite-state FTSs (with the classical definition of product derivation from [8, 11]) with finite-state MTSυs, which are an adaptation of MTSs with, amongst others, a definition of product derivation that differs from the modal refinement relation of MTSs [24, 35].

Both are valid strategies. Advantages of the approach pursued in [17, 34] are: (i) an infinite number of states (and features) allows to encode the infinite number of variants of MTSs, such as those depicted in Fig. 15(right), and (ii) the generalised product-derivation relation is closed under (strong) bisimulation, making the formalisms amenable to testing equivalences (as exploited in [17, 34]). An advantage of our approach is that dedicated (family-based) model-checking algorithms and SPL model checkers exist for both formalisms (i.e. with the respective product-derivation relations adopted in this paper).

8. Conclusion and future work

In this paper, we proved that finite-state MTSυs are equally expressive as finite-state FTSs. This result complements the expressiveness results that were reported in [17, 34] for behavioural SPL formalisms with possibly infinite states, viz. MTSs are less expressive than FTSs (with a generalised product-derivation relation), which are equally expressive as PL-LTSs.

In the future, we plan to implement an optimised version (e.g. creating must transitions whenever possible) of the FTS2MTSυ transformation algorithm of Definition 17 as a front-end of the variability model checker VMC, a tool for the modelling and analysis of behavioural SPL models [19, 20]. VMC is the most recent member of the KandISTI product line of model checkers developed at ISTI–CNR over the past decades, including UMC [36] and CMC [37]. KandISTI's model checkers offer explicit-state on-the-fly model checking of functional properties expressed in specific action- and state-based branching-time temporal logics derived from ACTL [38], the action-based version of CTL [39]. Their common model-checking engine has been highly optimised, due to which millions of states can now be verified in minutes.

Currently, the only input model accepted by VMC is an MTSυ defined as an MTS specified in a high-level modal process algebra, together with a set of variability constraints specified according to Definition 12. The envisioned front-end would allow VMC to offer SPL model checking of temporal logic properties against either FTSs or MTSυs. At present, efficient SPL model checking against FTSs can be achieved by using dedicated family-based model checkers such as the ProVeLines [40] tool suite or, alternatively, by using one of the highly optimised off-the-shelf model checkers such as mCRL2 or SPIN, which have recently been made amenable to family-based SPL model checking against FTSs [30, 41].

Finally, in [42] a unified approach is presented to evaluate the relative expressive power of process calculi. In that approach, a calculus is specified by a triple $(\mathcal{P}, \longmapsto, \asymp)$ in which (i) $\mathcal{P}$ is the set of language terms (i.e. processes) that is built up from the terminated process and the success process by using at least the parallel composition operator, which is assumed to be unique; (ii) $\longmapsto$ is the operational semantics (usually a binary relation on processes) that specifies how processes compute; and (iii) $\asymp$ is a behavioural equivalence that specifies when two processes have corresponding (abstract) behaviour. Our transformations from FTSs to MTSυs and vice versa are aimed at preserving the sets of (product) configurations and (product) variants of the SPL. We see no straightforward way to apply the criteria proposed in [42] to our transformations. In future work, we would like to investigate how to lift the approach proposed in [42] for process calculi to the setting of behavioural SPL formalisms considered in this paper, e.g. by considering (i) $\mathcal{P}$ to be a behavioural SPL model; (ii) $\longmapsto$ to be a variant (product) derivation relation, which is not a binary relation on $\mathcal{P}$, since it maps a behavioural SPL model to its (product) variants (a set of LTSs); and (iii) $\asymp$ to be a bijection between sets of LTSs that associates each LTS to another (somehow) equivalent LTS. For this lifted approach to work, we likely need to revisit the parallel composition operators for MTSs and FTSs defined in [11, 15], which are not unique but depend on whether the composed models refer to the same feature model (intra-SPL) or not (inter-SPL). To this aim, we would probably need to exploit notions of feature model composition (cf., e.g., [43]) and SPL composition (cf., e.g., [44]).

Acknowledgements

We would like to thank the anonymous reviewers for insightful comments and suggestions for improving the presentation.

References

[1] K. Pohl, G. Böckle, F. J. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, 2005. doi:10.1007/3-540-28901-1.

[2] S. Apel, D. S. Batory, C. Kästner, G. Saake, Feature-Oriented Software Product Lines: Concepts and Implementation, Springer, 2013. doi:10.1007/978-3-642-37521-7.

[3] D. Fischbein, S. Uchitel, V. A. Braberman, A Foundation for Behavioural Conformance in Software Product Line Architectures, in: R. M. Hierons, H. Muccini (Eds.), Proceedings of the ISSTA Workshop on Role of Software Architecture for Testing and Analysis (ROSATEA'06), ACM, 2006, pp. 39–48. doi:10.1145/1147249.1147254.

[4] K. G. Larsen, U. Nyman, A. Wąsowski, Modal I/O Automata for Interface and Product Line Theories, in: R. De Nicola (Ed.), Proceedings of the 16th European Symposium on Programming (ESOP'07), Vol. 4421 of LNCS, Springer, 2007, pp. 64–79. doi:10.1007/978-3-540-71316-6_6.

[5] A. Fantechi, S. Gnesi, A behavioural model for product families, in: Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE'07), ACM, 2007, pp. 521–524. doi:10.1145/1287624.1287700.

[6] A. Gruler, M. Leucker, K. D. Scheidemann, Modeling and Model Checking Software Product Lines, in: G. Barthe, F. S. de Boer (Eds.), Proceedings of the 10th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'08), Vol. 5051 of LNCS, Springer, 2008, pp. 113–131. doi:10.1007/978-3-540-68863-1_8.

[7] K. Lauenroth, K. Pohl, S. Töhning, Model Checking of Domain Artifacts in Product Line Engineering, in: Proceedings of the 24th International Conference on Automated Software Engineering (ASE'09), IEEE, 2009, pp. 269–280. doi:10.1109/ASE.2009.16.

[8] 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), ACM, 2010, pp. 335–344. doi:10.1145/1806799.1806850.

[9] P. Asirelli, M. H. ter Beek, A. Fantechi, S. Gnesi, Formal Description of Variability in Product Families, in: Proceedings of the 15th Interna-

tional Software Product Lines Conference (SPLC'11), IEEE, 2011, pp. 130–139. doi:10.1109/SPLC.2011.34.

[10] M. Erwig, E. Walkingshaw, The Choice Calculus: A Representation for Software Variation, ACM Trans. Softw. Eng. Methodol. 21 (1) (2011) 6:1–6:27. doi:10.1145/2063239.2063245.

[11] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, J.-F. Raskin, Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking, IEEE Trans. Softw. Eng. 39 (8) (2013) 1069–1089. doi:10.1109/TSE.2012.86.

[12] M. Tribastone, Behavioral Relations in a Process Algebra for Variants, in: Proceedings of the 18th International Software Product Line Conference (SPLC'14), ACM, 2014, pp. 82–91. doi:10.1145/2648511.2648520.

[13] M. Lochau, S. Mennicke, H. Baller, L. Ribbeck, DeltaCCS: A Core Calculus for Behavioral Change, in: T. Margaria, B. Steffen (Eds.), Proceedings of the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'14), Vol. 8802 of LNCS, Springer, 2014, pp. 320–335. doi:10.1007/978-3-662-45234-9_23.

[14] R. Muschevici, J. Proença, D. Clarke, Feature Nets: behavioural modelling of software product lines, Softw. Sys. Model. 15 (4) (2016) 1181–1206. doi:10.1007/s10270-015-0475-z.

[15] M. H. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints, J. Log. Algebr. Meth. Program. 85 (2) (2016) 287–315. doi:10.1016/j.jlamp.2015.11.006.

[16] M. H. ter Beek, A. Legay, A. Lluch Lafuente, A. Vandin, A framework for quantitative modeling and analysis of highly (re)configurable systems, IEEE Transactions on Software Engineering (2018). doi:10.1109/TSE.2018.2853726.

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

[18] M. H. ter Beek, F. Damiani, S. Gnesi, F. Mazzanti, L. Paolini, From Featured Transition Systems to Modal Transition Systems with Variability Constraints, in: R. Calinescu, B. Rumpe (Eds.), Proceedings of the 13th International Conference on Software Engineering and Formal Methods (SEFM'15), Vol. 9276 of LNCS, Springer, 2015, pp. 344–359. doi:10.1007/978-3-319-22969-0_24.

[19] M. H. ter Beek, F. Mazzanti, A. Sulova, VMC: A Tool for Product Variability Analysis, in: D. Giannakopoulou, D. Méry (Eds.), Proceedings of the 18th International Symposium on Formal Methods (FM'12), Vol. 7436 of LNCS, Springer, 2012, pp. 450–454. doi:10.1007/978-3-642-32759-9_36.

[20] M. H. ter Beek, F. Mazzanti, VMC: Recent Advances and Challenges Ahead, in: Proceedings of the 18th International Software Product Line Conference (SPLC'14), Vol. 2, ACM, 2014, pp. 70–77. doi:10.1145/2647908.2655969.

[21] R. Gorrieri, C. Versari, Introduction to Concurrency Theory: Transition Systems and CCS, Texts in Theoretical Computer Science: An EATCS Series, Springer, 2015. doi:10.1007/978-3-319-21491-7.

[22] R. Gorrieri, Process Algebras for Petri Nets: The Alphabetization of Distributed Systems, Monographs in Theoretical Computer Science. An EATCS Series, Springer, 2017. doi:10.1007/978-3-319-55559-1_2.

[23] R. Milner, Communication and Concurrency, Prentice Hall, 1989.

[24] K. G. Larsen, B. Thomsen, A Modal Process Logic, in: Proceedings of the 3rd Symposium on Logic in Computer Science (LICS'88), IEEE, 1988, pp. 203–210. doi:10.1109/LICS.1988.5119.

[25] P. Asirelli, M. H. ter Beek, A. Fantechi, S. Gnesi, A Logical Framework to Deal with Variability, in: D. Méry, S. Merz (Eds.), Proceedings of the 8th International Conference on Integrated Formal Methods (IFM'10), Vol. 6396 of LNCS, Springer, 2010, pp. 43–58. doi:10.1007/978-3-642-16265-7_5.

[26] N. Benes, J. Kretínský, K. G. Larsen, M. H. Møller, J. Srba, Parametric Modal Transition Systems, in: T. Bultan, P.-A. Hsiung (Eds.), Proceedings of the 9th International Symposium on Automated Technology

for Verification and Analysis (ATVA'11), Vol. 6996 of LNCS, Springer, 2011, pp. 275–289. doi:10.1007/978-3-642-24372-1_20.

[27] R. De Nicola, Extensional Equivalences for Transition Systems, Acta Inf. 24 (2) (1987) 211–237. doi:10.1007/BF00264365.

[28] A. Classen, M. Cordy, P. Heymans, A. Legay, P.-Y. Schobbens, Formal semantics, modular specification, and symbolic verification of product-line behaviour, Sci. Comput. Program. 80 (B) (2014) 416–439. doi:10.1145/2499777.2499781.

[29] A. Fantechi, S. Gnesi, Formal modeling for product families engineering, in: Proceedings of the 12th International Conference on Software Product Line Engineering (SPLC'08), IEEE, 2008, pp. 193–202. doi:10.1109/SPLC.2008.45.

[30] M. H. ter Beek, E. P. de Vink, T. A. C. Willemse, Family-Based Model Checking with mCRL2, in: M. Huisman, J. Rubin (Eds.), Proceedings of the 20th International Conference on Fundamental Approaches to Software Engineering (FASE'17), Vol. 10202 of LNCS, Springer, 2017, pp. 387–405. doi:10.1007/978-3-662-54494-5_23.

[31] B. Pfahringer, Conjunctive Normal Form, in: C. Sammut, G. I. Webb (Eds.), Encyclopedia of Machine Learning, Springer, 2010, pp. 209–210. doi:10.1007/978-0-387-30164-8_158.

[32] B. Pfahringer, Disjunctive Normal Form, in: C. Sammut, G. I. Webb (Eds.), Encyclopedia of Machine Learning, Springer, 2010, pp. 371–372. doi:10.1007/978-1-4899-7687-1_223.

[33] R. Milner, A Calculus of Communicating Systems, Vol. 92 of LNCS, Springer, 1980. doi:10.1007/3-540-10235-3.

[34] M. Varshosaz, Test Models and Algorithms for Model-Based Testing of Software Product Lines, Licentiate thesis, Vol. 30 of Halmstad University Dissertations, Halmstad University Press, 2017.
URL http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-33893

[35] J. Křetínský, 30 Years of Modal Transition Systems: Survey of Extensions and Analysis, in: L. Aceto, G. Bacci, G. Bacci, A. Ingólfsdóttir, A. Legay, R. Mardare (Eds.), Models, Algorithms, Logics and

Tools, Vol. 10460 of LNCS, Springer, 2017, pp. 36–74. doi:10.1007/978-3-319-63121-9_3.

[36] M. H. ter Beek, A. Fantechi, S. Gnesi, F. Mazzanti, A state/event-based model-checking approach for the analysis of abstract system properties, Sci. Comput. Program. 76 (2) (2011) 119–135. doi:10.1016/j.scico.2010.07.002.

[37] A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, F. Tiezzi, A Logical Verification Methodology for Service-Oriented Computing, ACM Trans. Softw. Eng. Methodol. 21 (3) (2012) 16:1–16:46. doi:10.1145/2211616.2211619.

[38] R. De Nicola, F. W. Vaandrager, Action versus State based Logics for Transition Systems, in: I. Guessarian (Ed.), Semantics of Systems of Concurrent Processes, Vol. 469 of LNCS, Springer, 1990, pp. 407–419. doi:10.1007/3-540-53479-2_17.

[39] E. M. Clarke, E. A. Emerson, A. P. Sistla, Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM Trans. Program. Lang. Sys. 8 (2) (1986) 244–263. doi:10.1145/5397.5399.

[40] M. Cordy, A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, ProVeLines: A Product Line of Verifiers for Software Product Lines, in: Proceedings of the 17th International Software Product Line Conference (SPLC'13), Vol. 2, ACM, 2013, pp. 141–146. doi:10.1145/2499777.2499781.

[41] A. S. Dimovski, A. S. Al-Sibahi, C. Brabrand, A. Wąsowski, Family-Based Model Checking Without a Family-Based Model Checker, in: B. Fischer, J. Geldenhuys (Eds.), Proceedings of the 22nd International SPIN Symposium on Model Checking of Software (SPIN'15), Vol. 9232 of LNCS, Springer, 2015, pp. 282–299. doi:10.1007/978-3-319-23404-5_18.

[42] D. Gorla, Towards a unified approach to encodability and separation results for process calculi, Inf. Comput. 208 (9) (2010) 1031–1053. doi:10.1016/j.ic.2010.05.002.

[43] M. Acher, P. Collet, P. Lahire, R. B. France, Composing Feature Models, in: M. van den Brand, D. Gasevic, J. Gray (Eds.), Proceedings of the 2nd International Conference on Software Language Engineering (SLE'09), Vol. 5969 of LNCS, Springer, 2010, pp. 62–81. doi:10.1007/978-3-642-12107-4_6.

[44] F. Damiani, M. Lienhardt, L. Paolini, A Formal Model for Multi SPLs, in: M. Dastani, M. Sirjani (Eds.), Proceedings of the 7th International Conference on Fundamentals of Software Engineering (FSEN'17), Vol. 10522 of LNCS, Springer, 2017, pp. 67–83. doi:10.1007/978-3-319-68972-2_5.