Behavioral Feature Models: Technical Report

JULIANE PÄSSLER, University of Oslo, Norway
SOPHIE FORTZ, King's College, UK
MAURICE H. TER BEEK, CNR–ISTI, Italy
FERRUCCIO DAMIANI, University of Turin, Italy
EINAR BROCH JOHNSEN, University of Oslo, Norway
MOHAMMAD REZA MOUSAVI, King's College, UK
SILVIA LIZETH TAPIA TARIFA, University of Oslo, Norway

Modeling software product lines (SPLs) involves addressing two challenges: representing the structure of the product line, called structural variability, and capturing its behavior, called behavioral variability. For structural variability, several established compositional modeling languages have been proposed, most notably feature models (FMs). For behavioral variability, there are no proposals that offer the same level of compositionality as offered by structural models. Moreover, these two types of models are traditionally developed in isolation. This separation often necessitates the maintenance of different models for structural and behavioral aspects, requiring synchronized updates to prevent discrepancies.

In this paper, we propose a novel formalism called *Behavioral Feature Model* (BFM) that integrates structural and behavioral aspects within a single unified model. A BFM is an FM in which each feature has an associated behavioral model, namely an event structure, that models the behavior of the feature; constraints between behavior of different features can be modeled as constraints between their associated events; and the behavior of each product of the SPL consists of the combined behavior of its features and the constraints between their behavior. Thus, the proposed formalism integrates both structural and behavioral variability in a compositional model, while preserving the separation of concerns typical of feature modeling. We illustrate the concept of a BFM through a running example of a cleaning robot SPL and provide a theoretical and an empirical evaluation where we compare the expressiveness and succinctness of BFMs with Featured Transition Systems (FTSs), the state-of-the-art formalism for behavioral modeling of SPLs. We show that BFMs are as expressive as a subclass of FTSs, and that, in practice, BFMs can reduce the size of FTSs by up to 98.63%, with a median reduction of 70.02% and a standard deviation of 32.66%.

CCS Concepts: • **Software and its engineering** → **Software product lines**; **Formal methods**; • **Computer systems organization** → Robotics.

Additional Key Words and Phrases: software product lines, feature models, behavioral feature models, featured event structures, robotics

1 INTRODUCTION

Software product line engineering [38, 85] is a software development paradigm concerned with developing a family of software products that share a common core and differ in some of their functionalities, called a *Software Product Line* (SPL). An important aspect of SPLs is modeling the variability of their products [28] in the form of *structural models* and *behavioral models*, where the former captures structural and the latter behavioral variability. This introduces a dichotomy with the consequence that the two types of models are traditionally kept apart. Furthermore, they are organized differently; while structural models offer compositional modeling techniques, current behavioral models lack them. This makes the maintenance of SPL models challenging. The models evolve in isolation, and cumbersome, synchronized updates are needed to prevent discrepancies.

Authors' addresses: Juliane Päßler, julipas@ifi.uio.no, University of Oslo, Oslo, Norway; Sophie Fortz, sophie.fortz@kcl.ac.uk, King's College, London, UK; Maurice H. ter Beek, maurice.terbeek@isti.cnr.it, CNR–ISTI, Pisa, Italy; Ferruccio Damiani, ferruccio.damiani@unito.it, University of Turin, Turin, Italy; Einar Broch Johnsen, einarj@ifi.uio.no, University of Oslo, Oslo, Norway; Mohammad Reza Mousavi, mohammad.mousavi@kcl.ac.uk, King's College, London, UK; Silvia Lizeth Tapia Tarifa, sltarifa@ifi.uio.no, University of Oslo, Oslo, Norway.

Structural models of SPLs, such as Feature Models (FMs) [6, 41, 69, 89], are used to organize the configuration space of the SPL in terms of *features*, where a feature is a name associated to a functionality. FMs contain composition operators to describe how different features are related and can be combined. Thereby, the FM describes the product variants of the SPL, which are identified by valid combinations of features, called *products*. *Behavioral models* of SPLs, such as Featured Transition Systems (FTSs) [35, 40], an extension of transition systems [10], describe the behavior of all products of an SPL in one single model and allow answering questions about the behavior of single products. However, these behavioral models do not describe the behavior of their features. In fact, in many practical cases, the behavior of a single feature cannot be described as a transition system or a state machine since the dependencies among its events can cut across the FM structure. As a result, the behavioral models are often flattened, which means that the understanding of the SPL, gained from the structural models, is not reflected in the behavioral models and the separation of concerns expressed in terms of feature decomposition is lost.

To address the dichotomy in SPL models, this paper introduces *Behavioral Feature Models (BFMs)* that unify the modeling of structural and behavioral variability. BFMs define the behavior of the products of an SPL *compositionally* from the behavior of its features. We propose to structure the behavioral models of products using the composition operators of FMs, which are well-understood operators to describe variability in a configuration space. As a result, we obtain a compositional model for the behavior of the products of an SPL, assembled from behavioral models of its individual features according to how these compose into products.

By integrating the structural and behavioral aspects of the configuration space within a single unified model, BFMs facilitate reasoning over the behavior of partial feature models, including the behavior of its individual features, as well as over the reuse and composition of features. Ultimately, the compositionality inherent to BFMs has the potential to make SPLs easier to develop and maintain: instead of modeling the behavior of the whole SPL monolithically, only the behavior of single features and their relationship to the behavior of other features need to be modeled. Furthermore, if a new feature is introduced into an SPL, the new behavior of the SPL can be obtained by just adding the behavior of this feature to the BFM of the old behavior of the SPL, easing model extension. In addition, the behavior of one feature can be reused across different SPLs, enabling the development of behavioral feature libraries: features with associated behavior define reusable patterns of behavior that can be combined into new SPLs or used to extend existing SPLs when needed.

Throughout the paper, we use the running example of an SPL of cleaning robots to illustrate the necessary background and our proposal.

Example 1.1 (Cleaning robot SPL). The running example is a simple SPL of cleaning robots, which we call the *cleaning robot SPL*. The behavior of a product of the SPL can be summarized as follows: the robot starts by optionally *mapping* the cleaning area, followed by a *cleaning* procedure consisting of *moving straight* to the next position, if there is no obstacle, and *cleaning* there. In case the robot *detects an obstacle*, using either a *lidar* or a *camera*, it cannot move straight. Instead, the robot has to *bypass the obstacle* and then continue cleaning. After the cleaning procedure, the robot goes to *charge*.

Contributions. In summary, the main contributions of this paper are as follows:

(1) *Behavioral Feature Models (BFMs)*, a unifying and compositional model for structural and behavioral variability of SPLs, which combines FMs and Bundle Event Structures (BESs) [71], where the BESs are used to model both the behavior of individual features and the behavior of products;
(2) *Featured Event Structures (FESs)*, an isomorphic view of BFMs, that are more similar to FTSs;

(3) trace-preserving algorithms for translating an FES to an FTS and for translating a linear FTS, i.e., an FTS whose traces contain every event at most once, to an FES; allowing techniques and tools developed for FTSs to be applied to systems modeled using BFMs; and
(4) a theoretical and empirical evaluation showing that BFMs are as expressive and as succinct as linear FTSs in theory, and more succinct than parallel compositions of linear FTSs in practice.

Paper organization. The remainder of this paper is organized as follows. In Sect. 2, we provide the background definitions needed for the rest of the paper. In Sect. 3, we introduce BFMs informally to give the reader a first overview over the used concepts, before defining them formally in Sect. 4 where we also define FESs and prove that BFMs and FESs are isomorphic. In Sect. 5, we provide algorithms between FESs and (a specific class of) FTSs and prove that they provide trace-equivalent structures. Section 6 contains a theoretical and empirical evaluation of our approach. In Sect. 7, we position our contribution within the state of the art and review related work. In Sect. 8, we discuss our findings and possible shortcomings of BFMs and FTSs. Finally, in Sect. 9, we conclude the paper and present the directions of our ongoing research. We illustrate all concepts in Sects. 2 to 4 using the running example of the cleaning robot SPL presented in Example 1.1.

How to read this article?

This article contains an extensive account of our newly introduced concept of BFMs. To help the reader navigate this article, we provide guidance on which parts to read for which purposes. Furthermore, concepts marked with ⋆ can safely be ignored when first reading the article, as they will only be needed later on.

Getting an overview. To get an overview of BFMs and their isomorphic structure FESs, we suggest first getting familiar with feature models and bundle event structures in Sects. 2.1 and 2.4 (while ignoring everything marked with ⋆) before reading the motivation and overview of BFMs in Sect. 3. This should provide the reader with enough information to get a feeling for what BFMs are, before reading their formal definition in Sect. 4.1 (again ignoring everything marked with ⋆) and getting to know FESs in Def. 4.7 and Example 4.8, an isomorphic structure to BFMs.

Understanding how BFMs and FTSs compare. To get a basis for understanding how BFMs and FTS compare, we suggest getting familiar with transition systems and featured transition systems in Sects. 2.2 and 2.3 first (ignoring everything marked with ⋆). The reader should then get acquainted with (behavioral) products, traces and configurations of BFMs in Sect. 4.3 (again ignoring everything marked with ⋆) to understand what we mean by a BFM and an FTS being trace-equivalent in Sect. 4.6. In Sect. 5, the reader will then find the algorithms translating between BFMs and (a particular class of) FTSs. The proofs for their trace-equivalence can be skipped by readers only interested in how the translation between these structures works. A theoretical and empirical evaluation of BFMs, where we, among others, show how compositional models of FTSs and BFMs compare on existing benchmarks of FTSs, can be found in Sect. 6. For a more theoretical exposition of composition in BFMs and FTSs, we refer to Sects. 4.5 and 8.2, respectively.

Understanding BFMs and FESs better. Section 4.2 discusses why and how BFMs and FESs are isomorphic and Sect. 8.3 includes a discussion about the location of events in BFMs.

2 BACKGROUND

In this section, we present the necessary background to motivate our proposal. Namely, we use the running example in Example 1.1 to illustrate the concepts of a *feature model* (in Sect. 2.1), a *transition system* (in Sect. 2.2), a *featured transition system* (in Sect. 2.3) and a *bundle event structure*

(in Sect. 2.4). The reader familiar with these concepts may want to read only the parts devoted to illustrating the running example (i.e., Examples 2.5, 2.7, 2.10, 2.13, 2.18, 2.21, 2.23, 2.25, 2.29 and 2.32) and the peculiarities of our formalization (i.e., Remarks 2.3, 2.15 and 2.33 and Notations 2.12, 2.20 and 2.27). However, we suggest reading the entire text to become familiar with our notations.

2.1 Feature Models

A *Feature Model* (FM) allows a designer to organize functionalities of a system in a structured way by providing a means to specify (hierarchical) relations and constraints between functionalities, enabling the structured specification of products of an SPL. In an FM, a *feature* is a name representing some functionality, a *configuration* is a set of features, and each *valid configuration* (i.e., a configuration that satisfies the constraints in the FM) is called a *product*. There are many different definitions of what a feature is [37, 97], we use the definition of Apel et al. [6]: "a feature is a characteristic or end-user-visible behavior of a software system".

Before defining FMs, we define feature expressions over a set of features, i.e., Boolean formulae which have feature names as their atomic propositions. We will use these feature expressions in feature models to denote constraints between features.

Definition 2.1 (Feature expressions). Let N be a set of feature names and let x range over N. The set of *feature expressions over* N, denoted $\mathrm{FExp}(N)$, includes all Boolean expressions over elements of N (modulo logical equivalence) generated by the following grammar:

$$\phi ::= x \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \phi \leftrightarrow \phi \mid \neg\phi \mid \mathit{ff} \mid \mathit{tt}.$$

In the sequel, let I^n denote a finite index set I with minimal cardinality n (so I^2 denotes an index set I such that $|I| \geq 2$).

Definition 2.2 (Feature Model (FM)). Let N be a finite set of feature names with $f \in N$. A *feature model* (FM) $\mathcal{M}$ over N has the following syntax:

$$\begin{array}{lll} \mathcal{M} & ::= (f, \{\mathcal{G}_i\}_{i \in I^0}, \mathrm{FC}) & \mathrm{Op} ::= \textbf{or} \mid \textbf{xor} \\ \mathcal{G} & ::= \mathrm{Uop}\,(\mathcal{M}) \mid \mathrm{Op}\,(\mathcal{M}_j)_{j \in I^2} & \mathrm{FC} ::= \phi \\ \mathrm{Uop} & ::= \textbf{opt} \mid \textbf{mnd} & \end{array}$$

Here, f is the *root feature* of $\mathcal{M}$; the $\mathcal{G}_i, i \in I^0$ are *sub-FM groups*, each consisting of an indexed set of *child FMs* of $\mathcal{M}$; a sub-FM group can be a unary operator (Uop) or an n-ary operator (Op) applied to FMs, indicating the *kind* of the sub-FM group; a unary operator can be of kind *optional* (**opt**) or of kind *mandatory* (**mnd**), while an n-ary operator is a choice between FMs, either of kind *inclusive or* (**or**) or of kind *exclusive or* (**xor**); *cross-tree feature constraints* FC can express constraints among features in the form of a feature expression $\phi \in \mathrm{FExp}(N)$.

In Def. 2.2, FMs are defined compositionally, allowing decomposition. We will use this in Sect. 4.5 to define composition and decomposition of FMs and showing that BFMs allow the same.

Remark 2.3 (Key assumptions on FMs).

(1) Given an FM $\mathcal{M}$ over N (see Def. 2.2) we assume (without loss of generality) that each feature in N is present exactly once in $\mathcal{M}$; and it is the root feature of one of the sub-FMs of $\mathcal{M}$.
(2) Definition 2.2 allows cross-tree constraints FC to be situated inside sub-FMs;[1] without loss of generality, we assume that cross-tree constraints are situated in the smallest sub-FM that contains the mentioned features. This facilitates composition, decomposition and reusability.

[1] In the literature on FMs, cross-tree constraints are often presented as a single set associated to the whole FM.

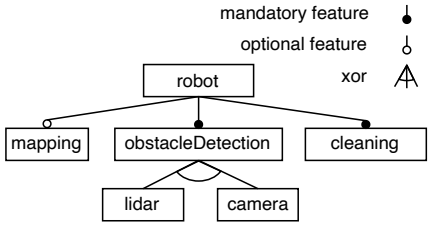

Fig. 1. A graphical representation of the FM of the cleaning robot SPL

The following notations are useful for working with FMs. We use the same notations for working with BFMs, see Not. 4.2.

Notation 2.4 (★ On FMs). We use features($\mathcal{M}$) to denote the set of the features of the FM $\mathcal{M}$, we use root($\mathcal{M}$) to denote its root feature and we use subTree$_f(\mathcal{M})$ to denote its sub-FM with root f. Moreover, given an FM $\mathcal{M}$ and a feature f such that subTree$_f(\mathcal{M}) = (f, \{\mathcal{G}_i\}_{i \in I^0}, \text{FC})$, we use

- FC$_f(\mathcal{M})$ to denote the set of *cross-tree feature constraints* FC *of* f;
- Uop(f) and Op($f_1, ..., f_m$) (with $m \geq 2$) to denote that subTree$_f(\mathcal{M})$ and subTree$_{f_j}(\mathcal{M})$ ($1 \leq j \leq m$) are in sub-FM groups of $\mathcal{M}$ of kind Uop and Op, respectively;
- $\text{gfc}_f(\mathcal{M}) \doteq \{\text{Uop}(\text{root}(\mathcal{M}_j)) \mid \text{Uop}(\mathcal{M}_j) = \mathcal{G}_i \text{ for some } i \in I^0\}$
  $\cup \{\text{Op}(\text{root}(\mathcal{M}_{j_1}), \ldots, \text{root}(\mathcal{M}_{j_{m_i}})) \mid \text{Op}(\mathcal{M}_{j_1}, \ldots, \mathcal{M}_{j_{m_i}}) = \mathcal{G}_i \text{ for some } i \in I^2\}$
  to denote the set of *group feature constraints of* f which describe how the sub-FMs $\mathcal{M}_j$ of subTree$_f(\mathcal{M})$ are grouped into sub-FM groups.

To ease the understanding of FMs, an FM is often represented graphically as a tree in which the features are the nodes of the tree and the tree's edges are annotated with a graphical representation of the kinds of the sub-FM group represented by the sub-tree connected by the edge; cross-tree constraints are written below the FM.

To construct an FM for the cleaning robot SPL, we consider the capabilities of the cleaning robot, as described above, to be features.

Example 2.5 (FM for the cleaning robot SPL). Let N_r be a set of feature names containing the features *robot*, *mapping*, *cleaning*, *obstacleDetection*, *lidar* and *camera*, represented as r, m, c, o, li and ca, respectively. We define the FM $\mathcal{M}_r$ of the cleaning robot SPL as follows:

$$\begin{array}{lll} \mathcal{M}_r = (r, \{\mathbf{opt}(\mathcal{M}_m), \mathbf{mnd}(\mathcal{M}_c), \mathbf{mnd}(\mathcal{M}_o)\}, tt) & & \mathcal{M}_o = (o, \{\mathbf{xor}(\mathcal{M}_{li}, \mathcal{M}_{ca})\}, tt) \\ \mathcal{M}_m = (m, \emptyset, tt) & & \mathcal{M}_{li} = (li, \emptyset, tt) \\ \mathcal{M}_c = (c, \emptyset, tt) & & \mathcal{M}_{ca} = (ca, \emptyset, tt) \end{array}$$

The graphical representation of the feature model is shown in Fig. 1.

Another way of representing an FM is by providing a feature expression expressing the FM's structure and feature constraints. Below, we show how to translate the FM syntax defined in Def. 2.2 into a feature expression. This is sometimes called the logical semantics of an FM [6], which is why we denote it using $[\![\cdot]\!]$.

Definition 2.6 (Propositional representation of the FM). Let $\mathcal{M}$ and $\mathcal{M}_i$ be FMs over sets of feature names N and N_i, respectively, with $N_i \subseteq N$, and let $f, f_i \in N$ be such that $f = \text{root}(\mathcal{M})$ and

$f_i = \text{root}(\mathcal{M}_i)$. The *propositional representation* of the FM $\mathcal{M}$ is the pair $[\![\mathcal{M}]\!] = (N, f \wedge \Phi(\mathcal{M}))$, in which $\Phi(\mathcal{M})$ is the feature expression over N defined inductively as:

$$
\begin{array}{ll}
\Phi((f, \{\mathcal{G}_i\}_{i\in I^0}, \text{FC})) & \doteq \bigwedge_{i\in I^0} \Phi((f, \{\mathcal{G}_i\}, tt)) \wedge \Phi(\text{FC}) \\
\Phi((f, \{\mathbf{opt}(\mathcal{M}_1)\}, tt)) & \doteq (f_1 \rightarrow f) \wedge \Phi(\mathcal{M}_1) \\
\Phi((f, \{\mathbf{mnd}(\mathcal{M}_1)\}, tt)) & \doteq (f_1 \leftrightarrow f) \wedge \Phi(\mathcal{M}_1) \\
\Phi((f, \{\mathbf{or}(\mathcal{M}_i)_{i\in I^2}\}, tt)) & \doteq \left(\left(\bigvee_{i\in I^2} f_i\right) \leftrightarrow f\right) \wedge \bigwedge_{i\in I^2} \Phi(\mathcal{M}_i) \\
\Phi((f, \{\mathbf{xor}(\mathcal{M}_i)_{i\in I^2}\}, tt)) & \doteq \left(\left(\bigvee_{i\in I^2} f_i\right) \leftrightarrow f\right) \wedge \bigwedge_{k<j\in I^2} \neg(f_j \wedge f_k) \wedge \bigwedge_{i\in I^2} \Phi(\mathcal{M}_i) \\
\Phi(\text{FC}) & \doteq \text{FC}
\end{array}
$$

Example 2.7 (Propositional representation of the FM of the cleaning robot SPL). The propositional representation of the FM $\mathcal{M}_r$ of the cleaning robot defined in Example 2.5 is the pair (N_r, ϕ_r), with $\phi_r = r \wedge \Phi(\mathcal{M}_r)$ and

$$
\begin{array}{rcl}
\Phi(\mathcal{M}_r) & = & ((m \rightarrow r) \wedge \Phi(\mathcal{M}_m) \wedge \Phi(tt)) \wedge ((c \leftrightarrow r) \wedge \Phi(\mathcal{M}_c) \wedge \Phi(tt)) \\
& & \wedge\ ((o \leftrightarrow r) \wedge \Phi(\mathcal{M}_o) \wedge \Phi(tt)) \wedge \Phi(tt) \\
& & = (m \rightarrow r) \wedge (c \leftrightarrow r) \wedge (o \leftrightarrow r) \wedge ((li \vee ca) \leftrightarrow o) \wedge \neg(li \wedge ca) \\
\Phi(\mathcal{M}_m) & = & \Phi(tt) = tt \\
\Phi(\mathcal{M}_c) & = & \Phi(tt) = tt \\
\Phi(\mathcal{M}_o) & = & ((li \vee ca) \leftrightarrow o) \wedge \neg(li \wedge ca) \wedge \Phi(\mathcal{M}_{li}) \wedge \Phi(\mathcal{M}_{ca}) \wedge \Phi(tt) \\
& & = ((li \vee ca) \leftrightarrow o) \wedge \neg(li \wedge ca) \\
\Phi(\mathcal{M}_{li}) & = & \Phi(tt) = tt \\
\Phi(\mathcal{M}_{ca}) & = & \Phi(tt) = tt
\end{array}
$$

The products of an FM $\mathcal{M}$ are defined in terms of its propositional representation. They are exactly the sets of features satisfying the propositional representation of a feature model.

Definition 2.8 (Products of an FM). Let N be a set of feature names, and let $pr \subseteq N$. An *evaluation over pr* is a function $\text{eval}_{pr} : \text{FExp}(N) \longrightarrow \mathbb{B}$, which maps all variables $x \in N$ of a feature expression ϕ to tt if $x \in pr$ and to ff otherwise. The *products described by a feature expression* $\phi \in \text{FExp}(N)$ are the sets of feature names pr such that eval_{pr} satisfies ϕ. Formally, $\text{products}((N, \phi)) \doteq \{\, pr \mid \text{eval}_{pr}(\phi) \text{ and } pr \in \mathcal{P}(N) \,\}$. The *products of an FM* $\mathcal{M}$ are defined as $\text{products}(\mathcal{M}) = \text{products}([\![\mathcal{M}]\!])$.

We sometimes also write $pr \models \phi$ for $\text{eval}_{pr}(\phi) = tt$. For a set of features pr, we write $\text{fexp}(pr)$ for the *feature expression of pr*, i.e., for $\bigwedge_{f\in pr} f \wedge \bigwedge_{g\notin pr} \neg g$. In particular, we do so for products of an FM.

Remark 2.9 (★ Features in products). By the definition of products and propositional semantics of an FM, the following always holds: If a product pr contains a feature f, then it also contains its parent features. Inductively, this implies that pr also contains the parent of the parent of f and so on until the root, i.e., that for all $g \in N$ with $f \in \text{features}(\text{subTree}_g(\mathcal{M}))$ it holds that $g \in pr$.

Example 2.10 (Products of the FM of the cleaning robot SPL). Consider the FM $\mathcal{M}_r$ of the cleaning robot SPL given in Example 2.5, graphically represented in Fig. 1, and its propositional representation $[\![\mathcal{M}_r]\!] = (N_r, \phi_r)$ given in Example 2.7. We have $\text{products}(\mathcal{M}_r) = \{pr_1, \ldots, pr_4\}$ with $pr_1 = \{r, c, o, li\}$, $pr_2 = \{r, c, o, ca\}$, $pr_3 = \{r, m, c, o, li\}$ and $pr_4 = \{r, m, c, o, ca\}$.

2.2 Transition Systems

A Transition System (TS) [10] is a behavioral model consisting of a directed graph whose edges are labeled with events. It can be used to model the behavior of an individual product of an SPL.

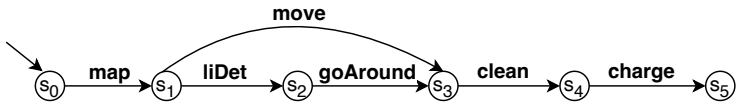

Fig. 2. A graphical representation of the TS for the product pr_3 of the cleaning robot SPL.

Definition 2.11 (Transition system (TS)). A *Transition System (TS)* over the events E is a quadruple $\mathcal{T} = (S, E, s_0, \delta)$, where:

- S is a finite (non-empty) set of *states*,
- E is a finite set of *events*, also called *actions*,[2]
- $s_0 \in S$ is the initial state, and
- $\delta \subseteq S \times E \times S$ is is a transition relation.

Notation 2.12 (On TSs). We denote a transition $t = (s, e, s') \in \delta$ as $s \xrightarrow{e} s'$. We use events($\mathcal{T}$) to denote the set of events E of the TS $\mathcal{T}$.

Also a TS is often represented graphically, with states represented by circles, the initial state indicated by an incoming arrow and the transition relation between states represented by arrows between the states, labeled with the event included in the transition relation between those states.

Example 2.13 (TS for the product pr_3 of the cleaning robot SPL). The TS that describes the behavior of product pr_3 in Example 2.10 is $\mathcal{T}_3 = (S_3, E_3, s_0, \delta_3)$, where

$S_3 = \{s_0, s_1, s_2, s_3, s_4, s_5\}$,
$E_3 = \{map, liDet, goAround, move, clean, charge\}$,
$\delta_3 = \{(s_0, map, s_1), (s_1, liDet, s_2), (s_1, move, s_3), (s_2, goAround, s_3), (s_3, clean, s_4)(s_4, charge, s_5)\}$.

It can be represented graphically as depicted in Fig. 2.

Before defining the traces of a TS, we define what it means for a state to be *reachable*.

Definition 2.14 (Reachable states of a TS). Let $\mathcal{T} = (S, E, s_0, \delta)$ be a TS and $s_1, s_2 \in S$. We say that s_2 *is reachable from* s_1, denoted reachable(s_1, s_2) if there exists a sequence of transitions that leads from s_1 to s_2, i.e., if

- $\exists e \in E.\ (s_1, e, s_2) \in \delta$, or
- $\exists s \in S\ \exists e \in E.$ reachable$(s_1, s) \wedge (s, e, s_2) \in \delta$.

By a slight abuse of notation, we denote by reachable(e_1, e_2) that an event e_2 is reachable from an event e_1 in a TS $\mathcal{T}$ (defined similarly as in Def. 2.14).

Remark 2.15 (Assumption on reachability of states of a TS). We assume that all the states of a TS are reachable from the initial state and that for every event e, there exists a transition $(s, e, s') \in \delta$.

The traces of a TS, i.e., the sequences of events specified by it, are defined as follows.

Definition 2.16 (Traces of a TS). A *trace* of a TS $\mathcal{T} = (S, E, s_0, \delta)$ is a sequence $\langle e_1, \ldots, e_n \rangle$ $(n \geq 1)$ such that for all $i \in \{1 \ldots, n\}$ we have $e_i \in E$ and there exists a transition $s_{i-1} \xrightarrow{e_i} s_i \in \delta$. We denote by Tr($\mathcal{T}$) the set of traces of a TS $\mathcal{T}$.

The following notations will make it easier to work with traces of a TS in Sect. 5.

Notation 2.17 (⋆ On traces of TSs). For a trace $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ with $(s_{i-1}, e_i, s_i) \in \delta$, $i \in \{1, \ldots, n\}$, we write events(tr) for the set $\{e_1, \ldots, e_n\}$ and $s_0 \xrightarrow{tr} s_n$ for $s_0 \xrightarrow{e_1} s_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} s_n$.

[2]Events and actions are different concepts, but for the purpose of this paper, we use them equivalently.

Example 2.18 (Traces of the TS for the product pr_3 of the cleaning robot SPL). The set of traces of $\mathcal{T}_3$ of Example 2.13 consists of all prefixes of the traces $\langle map \cdot move \cdot clean \cdot charge \rangle$ and $\langle map \cdot liDet \cdot goAround \cdot clean \cdot charge \rangle$, i.e.,

$$\mathrm{Tr}(\mathcal{F}_3) = \mathrm{prefixClosure}(\{\langle map \cdot move \cdot clean \cdot charge \rangle, \langle map \cdot liDet \cdot goAround \cdot clean \cdot charge \rangle\}),$$

where prefixClosure is the prefix-closure operator, which takes a set of traces and adds to it all prefixes of its traces (thus making it prefix-closed). Note that this also includes the empty trace.

2.3 Featured Transition Systems

The notion of a *featured transition system* (FTS) [35, 40] is built on top of the notion of a TS (see Def. 2.11). Namely, given a feature model $\mathcal{M}$, an FTS is a TS where, additionally to the event labels on transitions, transitions are also labeled with feature expressions over the features of $\mathcal{M}$. Thereby, an FTS describes the behavior of all products of an SPL in one model. The behavior of a product pr of $\mathcal{M}$ is obtained by taking all transitions of the FTS that are labeled by feature expressions satisfied by pr, i.e., transitions labeled by feature expressions ϕ such that $\mathrm{eval}_{pr}(\phi) = tt$ (see Def. 2.8 for the definition of eval). Modeling the behavior of all products in one model facilitates so called *family-based analysis* according to which all products of an SPL are analyzed in one run [99].

Definition 2.19 (Featured transition system (FTS)). A *Featured Transition System (FTS)* over the features N and the events E is a triple $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$, where:

- $\mathcal{M}$ is an FM over N, called *the FM of* $\mathcal{F}$,
- $\mathcal{T} = (S, E, s_0, \delta)$ is a TS over E, called *the TS of* $\mathcal{F}$,
- $\mu : S \times E \times S \to \mathrm{FExp}(N)$ is a mapping that associates each transition of $\mathcal{T}$ with a feature expression over N, that we call the *product (configuration) knowledge of* $\mathcal{F}$ in this paper.

Note that a transition in an FTS can only be taken if there exists a product satisfying the feature expression associated to this transition.

Notation 2.20 (On FTSs). The pair $(\mathcal{T}, \mu)$ is called the *annotated TS of* $\mathcal{F}$. We use $\mathrm{features}(\mathcal{F})$, $\mathrm{events}(\mathcal{F})$ and $\mathrm{prodKnow}(\mathcal{F})$ to denote the set of features, the set of events and product knowledge of the FTS $\mathcal{F}$, respectively.

A graphical representation of an annotated TS is obtained from the graphical representation of its TS by additionally labeling each transition with with the propositional representation.

Example 2.21 (FTS of the cleaning robot SPL). The behavior of the cleaning robot SPL can be modeled by the FTS $\mathcal{F}_r = (\mathcal{M}_r, (S_r, E_r, s_0, \delta_r), \mu_r)$, where $\mathcal{M}_r$ is the FM of Example 2.5 and

$$\begin{aligned}
S_r &= \{s_0, s_1, s_2, s_3, s_4, s_5\},\\
E_r &= \{map, liDet, goAround, move, clean, charge, caDet\},\\
\delta_r &= \{(s_0, map, s_1), (s_0, liDet, s_2), (s_0, caDet, s_2), (s_0, move, s_3), (s_1, liDet, s_2),\\
&\quad (s_1, caDet, s_2), (s_1, move, s_3), (s_2, goAround, s_3), (s_3, clean, s_4), (s_4, charge, s_5)\},\\
\mu_r(t) &= \begin{cases}
m & \text{if } t \in \{(s_0, map, s_1), (s_1, move, s_3), (s_2, goAround, s_3)\},\\
\neg m \wedge li & \text{if } t = (s_0, liDet, s_2),\\
\neg m \wedge ca & \text{if } t = (s_0, caDet, s_2),\\
\neg m & \text{if } t = (s_0, move, s_3),\\
m \wedge li & \text{if } t = (s_1, liDet, s_2),\\
m \wedge ca & \text{if } t = (s_1, caDet, s_2),\\
tt & \text{if } t \in \{(s_3, clean, s_4), (s_4, charge, s_5)\},
\end{cases}
\end{aligned}$$

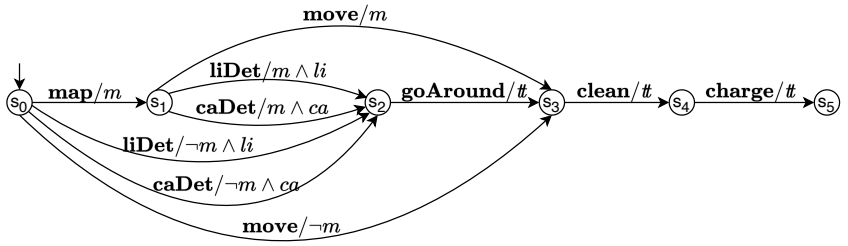

Fig. 3. A graphical representation of the annotated TS (i.e., the TS and product knowledge) of the FTS for the cleaning robot.

where $S_r = S_3$ and $E_r = E_3 \cup \{caDet\}$ for S_3, E_3 in Example 2.13. It can be graphically represented as shown in Fig. 1 (which depicts the FM) and in Fig. 3 (which depicts its annotated TS).

The *products of an FTS* are the products of its FM, and the *behavioral products of an FTS* are the TSs associated its products. Namely, a behavioral product of an FTS $\mathcal{F}$ for a product pr is the so called *projection* of the TS of $\mathcal{F}$ onto the sub-TS $\mathcal{T}'$ such that for every transition in $\mathcal{T}'$, the feature expression associated to this transition in $\mathcal{F}$ is satisfied by pr. Products and behavioral products are formally defined as follows.

Definition 2.22 (Products and behavioral products of an FTS). Given an FTS $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$, the *products of* $\mathcal{F}$ are the products of $\mathcal{M}$. For each of the FTS's products pr, the *behavioral product* of $\mathcal{F}$ for pr is the TS $\text{behPr}(\mathcal{F}, pr)$ obtained from the TS $\mathcal{T} = (S, E, s_0, \delta)$ by:

(1) first removing all the transitions $t \in \delta$ such that $\text{eval}_{pr}(\mu(t)) = ff$; and
(2) then removing all the states and events that are no longer reachable from the initial state.

We write $\text{products}(\mathcal{F})$ to denote the set of products of $\mathcal{F}$, and $\text{behPr}(\mathcal{F})$ to denote the set of behavioral products of $\mathcal{F}$.

We assume that for a product pr, all transitions satisfied by the product are included in the behavioral product of $\mathcal{F}$ for pr, i.e., $e \in \text{events}(\text{behPr}(\mathcal{F}, pr))$ if and only if $(s, e, s') \in \delta$ with $\text{eval}_{pr}(\mu((s, e, s')) = tt$.

Example 2.23 (Behavioral product of the FTS $\mathcal{F}_r$ for the product pr_3). Consider the FTS $\mathcal{F}_r$ given in Example 2.21 (and depicted in Fig. 3) and the product $pr_3 = \{r, m, c, o, li\}$ in Example 2.10. The behavioral product of $\mathcal{F}_r$ for pr_3 is $\text{behPr}(\mathcal{F}_r, pr_3) = \mathcal{T}_3$, where $\mathcal{T}_3$ is the TS given in Example 2.13 (and depicted in Fig. 2).

The traces of an FTS for a product are obtained from its behavioral product as defined below.

Definition 2.24 (Traces of an FTS for a product). The traces of an FTS $\mathcal{F}$ for a product pr are the traces of its behavioral product for pr, i.e., we define the set of traces $\text{Tr}(\mathcal{F}, pr)$ of $\mathcal{F}$ for pr as $\text{Tr}(\mathcal{F}, pr) \doteq \text{Tr}(\text{behPr}(\mathcal{F}, pr))$. More concretely, for an FTS $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$, a product pr of $\mathcal{F}$ with behavioral product $\text{behPr}(\mathcal{F}, pr) = (S, E, s_0, \delta)$, a trace in $\text{Tr}(\mathcal{F}, pr)$ is a sequence $\langle e_1, \ldots, e_n \rangle$ $(n > 1)$ of events $e_i \in E$, $i \in \{1, \ldots, n\}$, satisfying for all $i \in \{1, \ldots, n\}$:

(1) $\exists s_{i-1}, s_i \in S.\ (s_{i-1}, e_i, s_i) \in \delta$; and
(2) $\forall (s_{i-1}, e_i, s_i) \in \delta.\ \text{eval}_{pr}(\mu((s_{i-1}, e_i, s_i))) = tt$.

Example 2.25 (Traces of the behavioral products of the FTS of the cleaning robot SPL). Consider the FTS $\mathcal{F}_r$ given in Example 2.21 (with annotated TS depicted in Fig. 3). The products of $\mathcal{F}_r$ are pr_1, pr_2,

pr_3 and pr_4 as defined in Example 2.10. The traces of its behavioral products $\mathcal{T}_i = \text{behPr}(\mathcal{F}, pr_i)$ $(1 \leq i \leq 4)$ are the following:

$\text{Tr}(\mathcal{T}_1) = \text{prefixClosure}(\{\langle move \cdot clean \cdot charge\rangle, \langle liDet \cdot goAround \cdot clean \cdot charge\rangle\})$
$\text{Tr}(\mathcal{T}_2) = \text{prefixClosure}(\{\langle move \cdot clean \cdot charge\rangle, \langle caDet \cdot goAround \cdot clean \cdot charge\rangle\})$
$\text{Tr}(\mathcal{T}_3) = \text{prefixClosure}(\{\langle map \cdot move \cdot clean \cdot charge\rangle, \langle map \cdot liDet \cdot goAround \cdot clean \cdot charge\rangle\})$
$\text{Tr}(\mathcal{T}_4) = \text{prefixClosure}(\{\langle map \cdot move \cdot clean \cdot charge\rangle, \langle map \cdot caDet \cdot goAround \cdot clean \cdot charge\rangle\})$

2.4 Bundle Event Structures

Event structures [81, 105] were first introduced to capture models of true concurrency: systems in which multiple events can happen "at the same time". For this, they use the concepts of *conflict* and *causality* to denote events that exclude each other and events that enable other events, respectively. Like a TS, an event structure specifies a set of traces (i.e., sequences of events); so it can be used to model the behavior of an individual product of an SPL. *Bundle event structures (BESs)* [71], defined next, are an extension of event structures in which the causality relation (a.k.a. bundle set or enabling relation) allows both disjunctive and conjunctive enabling relations to be specified. The flexible behavioral models provided by event structures, as well as the combination of disjunctive and conjunctive enabling constraints provided by bundle event structures, are exactly what we need to match the composition structure of FMs to address behavior.

Definition 2.26 (Bundle event structures (BESs)). A *bundle event structure* (BES) over the set of events E is a triple $\mathcal{S} = (E, \#, \mapsto)$, where

- E is a finite set of events,
- $\# \subseteq E \times E$ is an irreflexive and symmetric relation, called the *conflict relation*, and
- $\mapsto \subseteq (\mathcal{P}(E) \setminus \{\emptyset\}) \times E$ is a relation, called the *causality relation*, satisfying:
  (1) $\forall (X, e) \in \mapsto \forall e_1, e_2 \in X. e_1 \neq e_2 \Rightarrow e_1 \# e_2$; and
  (2) $\forall (X, e), (Y, e) \in \mapsto. X \neq Y \Rightarrow (X \not\subset Y \wedge Y \not\subset X)$.

Notation 2.27 (On BESs). We use $\text{events}(\mathcal{S})$ to denote the set of events of the BES $\mathcal{S}$ and, for events e, e_1, e_2 and a set of events X, we write $e_1 \# e_2$ for $(e_1, e_2) \in \#$ and $X \mapsto e$ for $(X, e) \in \mapsto$.

The behavior of a BES can be specified by set of events, called *configurations*, conforming to the conflict and causality relation. A configuration C contains non-conflicting events such that if an event $e \in E$ is included in C and $(X, e) \in \mapsto$, then C contains exactly one element of X. Configurations consist of events that constitute (partial) runs of the represented system. Therefore, a causality $(X, e) \in \mapsto$ specifies *disjunctive causality* of events where exactly one event in X has to occur in a run of a system before e an occur. The condition enforcing X to contain only conflicting events ensures that an enabling event can be uniquely identified. *Conjunctive causality*, i.e., that a set of events (instead of a single event) has to occur in a run of the system before another event, can be specified by using several causality constraints. For example, $\{e_1\} \mapsto e$ and $\{e_2\} \mapsto e$ implies that both e_1 and e_2 are required for e to happen.

Definition 2.28 (Configurations of a BES). Let $\mathcal{S} = (E, \#, \mapsto)$ be a BES. A *configuration* $C \subseteq E$ of $\mathcal{S}$ is a set of distinct events satisfying:

(1) C is conflict free, i.e., $\forall e, e' \in C. (e, e') \notin \#$; and
(2) $\forall X \subseteq E \; \forall e \in C. (X, e) \in \mapsto \Rightarrow C \cap X \neq \emptyset$.

Example 2.29 (BES and configurations for the product pr_3 of the cleaning robot SPL). Consider the product pr_3 of the cleaning robot SPL as described in Example 2.10. The BES $\mathcal{S}_3 = (E_3, \#_3, \mapsto_3)$

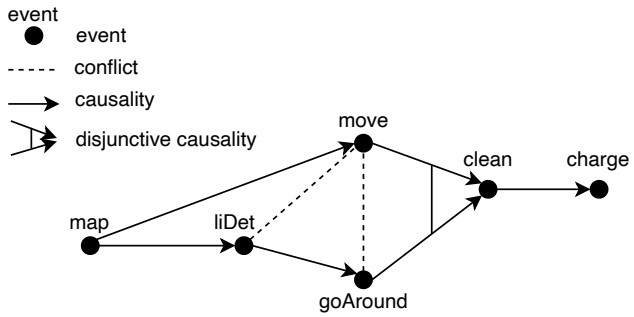

Fig. 4. The BES for the product pr_3 of the cleaning robot

describes the behavior of pr_3, where

$$\begin{aligned} E_3 &= \{map, liDet, goAround, move, clean, charge\},\\ \#_3 &= \{(liDet, move), (goAround, move)\}, \text{ and}\\ \mapsto_3 &= \{(\{map\}, liDet), (\{map\}, move), (\{move, goAround\}, clean), (\{clean\}, charge),\\ &\quad (\{liDet\}, goAround)\}. \end{aligned}$$

It can be represented graphically as depicted in Fig. 4. Its configurations are the sets $C_0 = \emptyset, C_1 = \{map\}, C_2 = \{map, liDet\}, C_3 = \{map, liDet, goAround\}, C_4 = \{map, liDet, goAround, clean\}$, $C_5 = \{map, liDet, goAround, clean, charge\}$, $C_6 = \{map, move\}$, $C_7 = \{map, move, clean\}$, $C_8 = \{map, move, clean, charge\}$.

A trace of a BES is a sequence of events that is conflict free and respects the causality relation, i.e., for every event e in a trace t and every $X \subseteq E$ that causes e, an event of X is included in t; more specifically, it comes before e.

Definition 2.30 (Traces of a BES). Let $\mathcal{S} = (E, \#, \mapsto)$ be a BES. A *trace of* $\mathcal{S}$ is a sequence $\langle e_1 \cdot \ldots \cdot e_n \rangle$ $(n \geq 1)$ of distinct events $e_i \in E, i \in \{1, \ldots, n\}$, satisfying:

(1) $\{e_1, \ldots, e_n\}$ is conflict free, i.e., $\forall i, j \in \{1, \ldots, n\} \,.\, (e_i, e_j) \notin \#$;
(2) $\forall X \subseteq E \; \forall i \in \{2, \ldots, n\} \,.\, X \mapsto e_i \Rightarrow \{e_1, \ldots, e_{i-1}\} \cap X \neq \emptyset$; and
(3) $\nexists X \subseteq E. X \mapsto e_1$.

We denote by $\mathrm{Tr}(\mathcal{S})$ the set of traces of $\mathcal{S}$.[3]

Note that for a configuration C, an element $e \in C$ and $(X, e) \in \mapsto$, the set $C \cap X$ contains exactly one element because all elements in X are in conflict and C is conflict-free. Therefore, the causality relation $\mapsto$ introduces a partial order $<_C$ on the events of C: for two events $e, e' \in C$ it holds that $e' <_C e$ if and only if there exists $(X, e) \in \mapsto$ with $e' \in C$ (and thus $\{e'\} = C \cap X$). This partial order allows us to connect the notions of configuration and trace of a BES.

Lemma 2.31 (★ Configurations forming traces of a BES). *Let $\mathcal{S} = (E, \#, \mapsto)$ be a BES and $C \subseteq E$ be a set of events. Then C is a configuration of $\mathcal{S}$ if and only if there exists a trace $tr \in \mathrm{Tr}(\mathcal{S})$ such that $C = \mathrm{events}(tr)$.*

Proof. "$\Rightarrow$" Let C be a configuration of $\mathcal{S}$, i.e., C is conflict free and for all $X \subseteq E$ and all $e \in C$, it holds that $(X, e) \in \mapsto$ implies $C \cap X \neq \emptyset$. Now let $\langle e_1, \ldots, e_n \rangle$ be an ordering of the elements of C that respects the partial order $<_C$ introduced by $\mapsto$. Then $\langle e_1, \ldots, e_n \rangle$ satisfies Condition (1) of Def. 2.30 because events $e_1, \ldots, e_n$ are conflict free since they are elements of C, and it satisfies

[3] In Langerak [71, Def. 2.3], which is credited to Boudol and Castellani [26, page 10], a *trace* is called a *proving sequence*.

Condition (2) of Def. 2.30 because the $e_i, i \in \{1, \ldots, n\}$, are ordered according to the partial order induced by $\mapsto$. Thus, $\langle e_1, \ldots, e_n \rangle \in \mathrm{Tr}(\mathcal{S})$.

"$\Leftarrow$" Let $tr \in \mathrm{Tr}(\mathcal{S})$ and $C = \mathrm{events}(tr)$. Then C is conflict free by Condition (1) of Def. 2.30, and for all $e \in C$ and all $X \subseteq E$, it holds that $(X, e) \in \mapsto$ implies $X \cap C = \emptyset$ by Conditions (2) and (3) of Def. 2.30. □

Note that the definition of traces of a BES (Def. 2.30) contains one more condition than the definition of configurations of a BES (Def. 2.28). A trace of a BES is a sequence of events, so we have to ensure that the events are ordered according to the causality relation and that the first event is not caused by another event, ensured by Conditions (2) and (3) of Def. 2.30), respectively. We do not need the second condition to define configurations because a configuration is a set.

Example 2.32 (Traces of the BES for the product pr_3 of the cleaning robot SPL). Consider product pr_3 of the cleaning robot SPL as described in Example 2.10 and BES $\mathcal{S}_3$ describing its behavior as shown in Example 2.29. The set of traces of $\mathcal{S}_3$ consists of all prefixes of the traces $\langle map \cdot move \cdot clean \cdot charge \rangle$ and $\langle map \cdot liDet \cdot goAround \cdot clean \cdot charge \rangle$ and its configurations are the sets $\emptyset$, $\{map\}$, $\{map, move\}$, $\{map, move, clean\}$, $\{map, move, clean, charge\}$, $\{map, liDet\}$, $\{map, liDet, goAround\}$, $\{map, liDet, goAround, clean\}$, $\{map, liDet, goAround, clean, charge\}$. It is worth observing that it is equal to the set of traces of the TS for the product pr_3 given in Example 2.18.

Remark 2.33 (On the definition of BESs). To simplify the presentation in Sect. 4, we slightly modified the definition of BESs given in [71, Def. 2.1] by: (i) dropping the labeling function (since we do not need it); (ii) excluding impossible events, i.e., events e with $\emptyset \mapsto e$ (which, as pointed out in [71], can always be removed and were included because they are technically convenient in expressing some operations in a concise way); and (iii) not allowing (thanks to Condition (2)) causalities $X \mapsto e$ such that there exists a causality $Y \mapsto e$ such that $X \subset Y$, because the causality $X \mapsto e$ would disallow traces using events in $Y \setminus X$.

In Sects. 5.1 and 5.2 we develop algorithms between BESs and (a class of) TSs that preserve trace equivalence. We now define what it means for a TS and a BES to be trace equivalent.

Definition 2.34 (★ Trace equivalence for TSs and BESs). A TS $\mathcal{T}$ and a BES $\mathcal{S}$ are *trace equivalent* if

(1) they have the same events; and
(2) it holds that $\mathrm{Tr}(\mathcal{T}) = \mathrm{Tr}(\mathcal{S})$.

3 BEHAVIORAL FEATURE MODELS: MOTIVATION AND OVERVIEW

Behavioral Feature Models (BFMs) facilitate compositional modeling of the structure and behavior of SPLs in one single model, similarly to the compositional modeling of the structure of an SPL in an FM. To do so, the structural model of an SPL in the form of an FM is extended by attaching behavior to every feature and specifying constraints between behaviors of different features. Thereby, BFMs are able to not only specify the behavior of all products of an SPL, but also the behavior of single features. We designed BFMs according to the following **Design Choices (DCs)**:

DC1 Features are organized into an FM.
DC2 The behavior of each product of the BFM is described in terms of events by a BES.
DC3 The behavior of each feature is described in terms of events by a BES, and BESs associated with different features have disjoints sets of events (so that each event in the BFM is associated with exactly one feature).
DC4 Constraints between behaviors associated with different features can be specified using conflicts and causalities.
DC5 Dependencies between behavior and features can be specified using feature expressions.

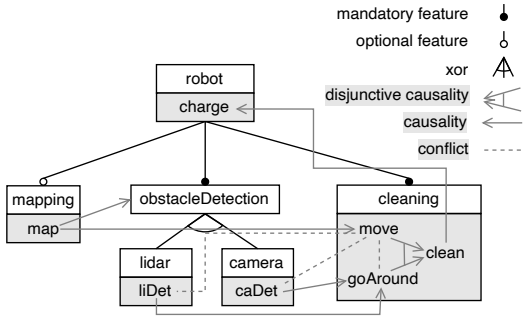

Fig. 5. A graphical representation of the BFM for the cleaning robot SPL

A BFM can be understood as an FM (see DC1) in which each feature f is associated with a BES that models the behavior of f, and the BESs associated with different features contain disjoint sets of events (see DC3). So called *inter-feature event constraints* specify conflicts and causalities between events of BESs associated with different features (see DC4), while *product constraints* specify that certain behavior depends on the selection of features (see DC5). Thereby, a BFM contains *structural* cross-tree constraints in the form of cross-tree feature constraints (as an FM), *behavioral* cross-tree constraints in the form of inter-feature event constraints, and a mix of the two in the form of product constraints. The behavior of a product of the BFM is described by a BES (see DC2) that can be composed from the BESs associated to the product's features and the involved inter-feature event constraints.

Example 3.1 (Creating a BFM for cleaning robot SPL). As an example, we consider the cleaning robot SPL as defined in Example 1.1. We now describe how a BFM for the cleaning robot SPL can be created. The behavior of the cleaning robot SPL is determined by the events $E_r = \{map, liDet, goAround, move, clean, charge, caDet\}$, as already defined in Example 2.21. The first step towards creating a BFM for the cleaning robot SPL is deciding which events should be associated to which feature of the FM of the cleaning robot SPL, where the FM is the same as the one in Example 2.5. The events should be associated with the lowest possible feature of the FM such that the behavior of a feature can be described using the events associated with it. Therefore, we decided to associate the event *map* with the feature *mapping*, the event *liDet* with the feature *lidar*, the event *caDet* with the feature *camera*, the events *move*, *goAround*, and *clean* with the feature *cleaning*, and the event *charge* with the feature *robot*. These are of course design choices and other assignments of events with features would be possible.

Next, we define the constraints among the events, both within the events associated with a feature as well as between events associated with different features. The information for how to define them is included in the textual description of the cleaning robot SPL in Example 1.1: The event *map* causes the events *liDet*, *caDet*, and *move*; the events *liDet* and *caDet* are in conflict with the event *move* and both cause the event *goAround*; the events *move* and *goAround* are in conflict, and either of them causes the event *clean*; and the event *clean* causes the event *charge*. Note that, as noted in Sect. 2.4, we used two different types of causality here: conjunctive causality (*liDet* and *caDet* both cause *move*) and disjunctive causality (either one of *move* and *goAround* causes *clean*). Combining the events associated with a feature and the conflicts and causalities between them gives the BES describing the behavior of a feature.

The final step is to specify dependencies of events on features. In our case, there are no such dependencies. A graphical representation of the BFM $\mathcal{B}_r$ for the cleaning robot SPL, which will be

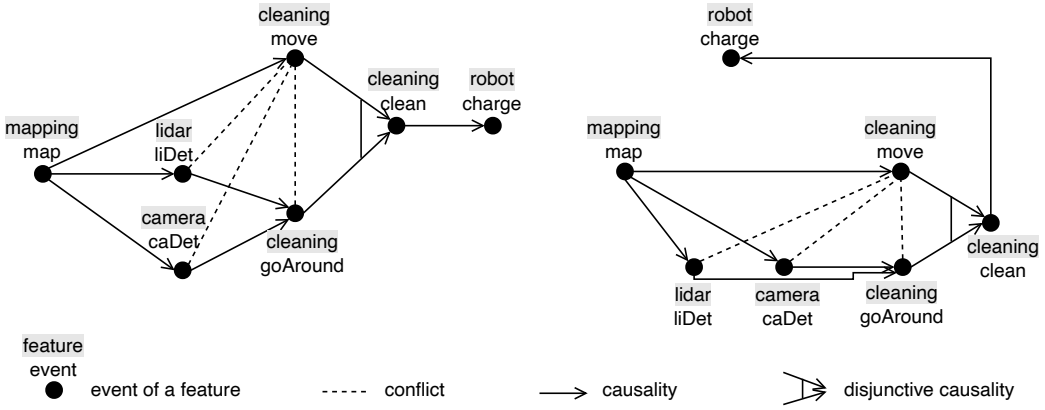

Fig. 6. Two graphical representations of the annotated BES (that is, the BES where each event is annotated by a feature, highlighted in gray) of the BFM in Fig. 5. *Left*: A graphical representation of the annotated BES where (to improve comprehensibility) events are positioned according to the topological ordering w.r.t. causality. *Right*: A graphical representation of the annotated BES where (to facilitate comparison with the graphical representation of the BFM in Fig. 5) events are positioned as in Fig. 5.

formally defined in Example 4.5, is shown in Fig. 5, where the behavioral part is highlighted with gray backgrounds and gray arrows.

Working with BFMs provides the additional advantage that its structure and behavior can easily be separated, providing an isomorphic view of BFMs called *featured event structures* because they associate features to (bundle) event structures. Thus, BFMs enjoy the advantages of a combined model of the structure and the behavior of an SPL (such as, e.g., ease of maintenance), while keeping a separation of concerns by separating the model of the structure from the model of the behavior (as is the case for FTSs). We provide an intuition of FESs using the cleaning robot SPL.

Example 3.2 (Isomorphic view of BFM for cleaning robot SPL). It is worth observing that, in the graphical representation of the BFM $\mathcal{B}_r$ in Fig. 5, we can clearly see three distinct entities. Namely:

(1) an FM, identical to the FM $\mathcal{M}_r$ depicted in Fig. 1, called *the FM of the BFM*;
(2) the BES depicted in Fig. 6 (please ignore, for now, the feature names highlighted in gray), which is depicted in two different ways (to improve comprehensibility and to facilitate comparison with Fig. 5, respectively), called *the BES of the BFM*;
(3) a mapping that associates each event with a feature, visualized by the feature names highlighted in gray that annotate each event in Fig. 6, called *event location knowledge of the BFM*; and
(4) when including product constraints, we would see them as an additional entity in Fig. 6, called *product knowledge of the BFM*.

Note that the focus of the representation of behavior in Fig. 6 is not on the behavior of single features. Instead, similar to FTSs, the focus is on the behavior of the whole product line. Using the four different entities above provides an isomorphic view of BFMs, the featured event structure $\mathcal{E}_r$.

As stated before, the behavior of a product of a BFM can be obtained by combining the behavior of its features and the necessary constraints between them. The FES that is isomorphic to a BFM provides a convenient structure for defining the behavior of products. We illustrate this using the cleaning robot SPL.

Example 3.3 (Behavior of products of BFM for the cleaning robot SPL). The products of BFM $\mathcal{B}_r$ for the cleaning robot SPL are the products of its features model, as defined in Example 2.10. The behavior of each product pr can be obtained from the FES $\mathcal{E}_r$, depicted in Fig. 6, that is isomorphic to the BFM $\mathcal{B}_r$ by:

(1) removing all the events e that are labeled by a feature not included in pr;
(2) removing all the conflicts (depicted as dotted lines) that involve at least one removed event; and
(3) removing or transforming all the causalities (depicted as arrows) that involve at least one removed event, i.e., each causality $X \mapsto e$ is either: (i) removed if e is removed or all the events in X are removed; or otherwise (ii) transformed into $X' \mapsto e$, where e is an event labeled by a feature in pr and $X' \neq \emptyset$ is obtained from X by removing all the removed events.

For example, the behavioral of the product $pr_3 = \{r, m, c, o, li\}$ is the BES $\mathcal{S}_3$ given in Example 2.29 (and depicted in Fig. 4).

A large portion of this paper is devoted to showing that BFMs and (a certain class of) FTSs are trace-equivalent. We shortly illustrate this using the cleaning robot SPL.

Example 3.4 (Trace-equivalence of FTS and BFM for the cleaning robot SPL). The BFM $\mathcal{B}_r$ (given in Fig. 5) and the FTS $\mathcal{F}_r$ (given in Fig. 1, which depicts the FM of $\mathcal{F}_r$, and in Fig. 3, which depicts the annotated TS of $\mathcal{F}_r$) are *trace equivalent*, i.e.:

(1) they have the same features, events and products; and
(2) each of their products has the same behavior, i.e., for every product pr, the TS obtained from $\mathcal{F}_r$ which describes the behavior of pr and the BES obtained from $\mathcal{B}_r$ which describes the behavior of pr have the same traces.

However, unlike FTSs, both BFMs and FESs share the same kind of compositionality that is also present in FMs, namely:

- the behavior of a feature is described by a BES over events uniquely associated to the feature
- the behavior of a product can be obtained by combining the behavior of its features and the involved constraints, completely ignoring behavior from other features not included in the product;
- a sub-FM describes the behavior of all included features (and only of these features); and
- behavior can easily be reused between several BFMs by copying a sub-BFM to another BFM and including cross-tree feature constraints and inter-feature event constraints.

Remark 3.5 (On the graphical representation of BFMs). To improve readability, we can remove the graphical representation of the inter-feature event constraints from the BFM's graphical representation and display them in textual form using, e.g., the syntax introduced in Def. 2.26 for the conflicts and causalities of a BES. In the literature, this is sometimes done similarly for cross-tree feature constraints in the graphical representation of FMs. They are provided with the FM as a set of propositional formulae over feature names (to be understood as a conjunction between them).

For instance, a graphical representation of the BFM $\mathcal{B}_r$ (depicted in Fig. 5) with conflicts and causalities displayed in textual form is given in Fig. 7, where

- EC_{robot} is the set of the intra-feature event constraints for the feature *robot*, which only contains conflicts and causalities between events associated with different features; and
- $E'_{ostacleDetection} = \{liDet, caDet\}$ is the set of the events associated with the features in the sub-BFM with root *obstacleDetection*, where

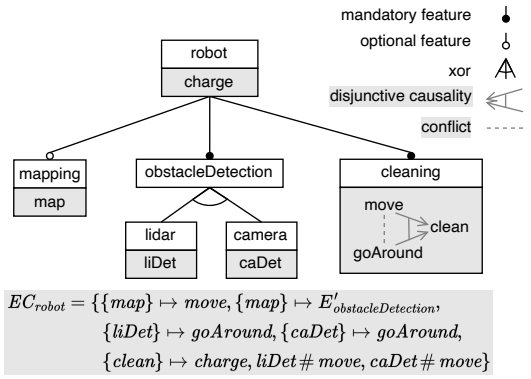

Fig. 7. A graphical representation of the BFM for the cleaning robot SPL where conflicts and causalities are displayed in textual form (cf. Fig. 5).

4 BEHAVIORAL FEATURE MODELS: A FORMAL ACCOUNT

This section provides a formal account of BFMs and of the isomorphic concept of Featured Event Structures (FESs). BFMs extend FMs with behavior, thus integrating structural and behavioral models of SPLs in one unified model. This unified model organizes behavior according to the structuring concepts of FMs, resulting in reusable and compositional building blocks for SPL development and maintenance.

4.1 Behavioral Feature Models

We start with introducing our main modeling concept, namely, Behavioral Feature Models. Throughout this section, we will use gray backgrounds to highlight extensions of FMs (introduced in Sect. 2.1, Def. 2.2) with behavior.

Definition 4.1 (Behavioral Feature Model (BFM)). Let N be a set of feature names with $f \in N$, and let E be a set of events with $e \in E$. A *Behavioral Feature Model (BFM)* $\mathcal{B}$ over N and E has the following syntax:

$$\begin{array}{lll} \mathcal{B} & ::= (f : \mathcal{S}, \{C_i\}_{i \in I^0}, \mathrm{FC}, \mathrm{EC}, \mathrm{PC}) & \\ C & ::= \mathrm{Uop}\,(\mathcal{B}) \mid \mathrm{Op}\,(\mathcal{B}_i)_{i \in I^2} & \mathrm{FC} ::= \phi \\ \mathrm{Uop} & ::= \textbf{opt} \mid \textbf{mnd} & \mathrm{EC} ::= \varnothing \mid e \# e \mid X \mapsto e \mid \mathrm{EC}\ \mathrm{EC} \\ \mathrm{Op} & ::= \textbf{or} \mid \textbf{xor} & \mathrm{PC} ::= \varnothing \mid (e, \mathrm{FC})\ \mathrm{PC} \end{array}$$

Here, f is the *root feature* of $\mathcal{B}$; $\mathcal{S} = (E_f, \#_f, \mapsto_f)$ is the BES associated with the feature f, containing the events E_f and the *intra-feature event constraints* $\#_f \cup \mapsto_f$ that describe conflicts ($\#_f$) and causalities ($\mapsto_f$) between events in E_f; the $C_i, i \in I^0$, are *sub-BFM groups*, each consisting of an indexed set of *child BFMs* of $\mathcal{B}$; a sub-BFM group can be a unary operator (Uop) or an n-ary operator (Op) of BFMs, indicating the *kind* of the sub-BFM group; a unary operator can be of kind *optional* (**opt**) or of kind *mandatory* (**mnd**), while an n-ary operator is a choice between BFMs, either of kind *inclusive or* (**or**) or of kind *exclusive or* (**xor**); *cross-tree feature constraints* FC can express constraints among features in the form of a feature expression $\phi \in \mathrm{FExp}(N)$; *inter-feature event constraints* EC associated with the feature f describe conflicts (#) and causalities ($\mapsto$) between either events of distinct sub-BFMs or between events of E_f and events of a sub-BFM, where # is an irreflexive and symmetric relation over $E \times E$, and $\mapsto$ is a relation over $(P(E) \setminus \{\varnothing\}) \times E$, satisfying

- $\forall (X, e) \in \mapsto\ \forall e_1, e_2 \in X . e_1 \neq e_2 \Rightarrow e_1 \# e_2$, and

- $\forall (X,e),(Y,e) \in \mapsto . X \neq Y \Rightarrow (X \not\subset Y \wedge Y \not\subset X)$;

and *product (configuration) constraints* PC associated with the feature f describe feature constraints FC associated with an event $e \in E_f$, where the feature constraints can be related to all features in N.

In Def. 4.1, BFMs are defined compositionally by mimicking the structure of FMs (cf. Sect. 2.1). We assume that each event is associated with exactly one feature, so that different features have disjoint sets of events.

As for FMs in Not. 2.4, we define some notations that facilitate referring to different parts of BFMs.

Notation 4.2 (⋆ On BFMs). We use $\text{features}(\mathcal{B})$ to denote the set of the features of the BFM $\mathcal{B}$, $\text{root}(\mathcal{B})$ to denote its root feature and $\text{subTree}_f(\mathcal{B})$ to denote its sub-BFM with root f. Furthermore, given a BFM $\mathcal{B}$ and a feature f such that $\text{subTree}_f(\mathcal{B}) = (f : \mathcal{S}, \{C_i\}_{i \in I^0}, \text{FC}, \text{EC}, \text{PC})$, with $\mathcal{S} = (E_f, \#_f, \mapsto_f)$ and $e \in E_f$, we use

- $\text{FC}_f(\mathcal{B})$ to denote the set of *cross-tree feature constraints* FC *of* f;
- $\text{Uop}(f)$ and $\text{Op}(f_1, ..., f_m)$ (with $m \geq 2$) to denote that $\text{subTree}_f(\mathcal{B})$ and $\text{subTree}_{f_j}(\mathcal{B})$, $j \in \{1, \ldots, m\}$, are in sub-BFM groups of $\mathcal{B}$ of kind Uop and Op, respectively;
- $\text{gfc}_f(\mathcal{B}) \doteq \{\text{Uop}(\text{root}(\mathcal{B}_j)) \mid \text{Uop}(\mathcal{B}_j) = C_i \text{ for some } i \in I^0\}$
  $\cup \{\text{Op}(\text{root}(\mathcal{B}_{j_1}), \ldots, \text{root}(\mathcal{B}_{j_{m_i}})) \mid \text{Op}(\mathcal{B}_{j_1}, \ldots, \mathcal{B}_{j_{m_i}}) = C_i \text{ for some } i \in I^2\}$
  to denote the set of *group feature constraints of* f which describe how the sub-BFMs $\mathcal{B}_j$ of $\text{subTree}_f(\mathcal{B})$ are grouped into sub-BFM groups.
- $\text{events}_f(\mathcal{B}) \doteq E_f$ to denote the set of *events of* f;
- $\text{events}(\mathcal{B}) \doteq \bigcup_{f \in \text{features}(\mathcal{B})} \text{events}_f(\mathcal{B})$ to denote the set of *events of* $\mathcal{B}$;
- $\text{intraFtEc}_f(\mathcal{B}) \doteq \#_f \cup \mapsto_f$ to denote the set of *intra-feature event constraints of* f;
- $\text{interFtEc}_f(\mathcal{B}) \doteq \text{EC}$ to denote the set of *inter-feature event constraints of* f;
- $\text{confl}(\text{EC}) \doteq \{ (e_1, e_2) \mid e_1 \# e_2 \in \text{EC} \}$ to denote the set of *inter-feature conflicts*;
- $\text{cause}(\text{EC}) \doteq \{ (X, e) \mid X \mapsto e \in \text{EC} \}$ to denote the set of *inter-feature causalities*;
- $\text{ec}_f(\mathcal{B}) \doteq \text{intraFtEc}_f(\mathcal{B}) \cup \text{interFtEc}_f(\mathcal{B})$ to denote the set of *event constraints of* f;
- $\text{behFt}_f(\mathcal{B}) \doteq (f, \mathcal{S})$ to denote the *behavioral feature for* f;
- $\text{productC}_f(\mathcal{B})(e) \doteq \phi_e$, where $(e, \phi_e) \in \text{PC}$, to denote the *product constraints of* f *for* e; and
- $\text{productC}(\mathcal{B}) \doteq \bigcup_{f \in \text{features}(\mathcal{B})} \text{productC}_f(\mathcal{B})$ to denote the set of *product constraints of* $\mathcal{B}$.

Remark 4.3 (Key assumptions on BFMs). For every BFM $\mathcal{B}$ and every feature $f \in \text{features}(\mathcal{B})$ we assume the following:

- Inter-feature event constraints are situated in the smallest sub-BFM that contains the mentioned events. This assumption is in the same style as the second assumption on FMs in Rem. 2.3; we assume that information is situated as far down in the (B)FM as possible to facilitate composition, decomposition and reusability. In BFMs this measn that each conflict and causality in $\text{interFtEc}_f(\mathcal{B})$ involves only events of $\text{subTree}_f(\mathcal{B})$, and it either involves an event of f and (at least) one event of a feature in an immediate sub-BFM of f; or it involves events in (at least) two immediate sub-BFMs of f. That is, conflict and causality in $\text{interFtEc}_f(\mathcal{B})$ involve events from at least two of the following three disjoint sets: (i) events of f, (ii) events of a feature in an immediate sub-BFM of f, and (iii) events of a feature in another immediate sub-BFM of f.
- Given the set $\bigcup_{f \in \text{features}(\mathcal{B})} \text{ec}_f(\mathcal{B})$ of event constraints of $\mathcal{B}$, the included conflicts form a conflict relation over $\text{events}_f(\mathcal{B})$ and the included causalities form a causality relation over $\text{events}_f(\mathcal{B})$ (see Def. 2.26). This assumption is needed later to show that BFMs and the concept of featured event structures, introduced in Def. 4.7, are isomorphic.

Notation 4.4 (Shorthand notation for causality relation). In examples, we sometimes use the notation $X \mapsto Y$ to denote the causality relation $X \mapsto e$ for all events e in the set of events Y. In Example 4.5, e.g., "$\{map\} \mapsto_3 E'_o$" stands for "$\{map\} \mapsto_3 liDet$, $\{map\} \mapsto_3 caDet$" (cf. Remark 3.5).

Example 4.5 (BFM for the cleaning robot SPL). Consider the example of the cleaning robot SPL over the set of features N_r from Example 2.5 and the set of events $E'_r = \{charge, map, goAround, move, clean, liDet, caDet\}$. Then the BFM of the cleaning robot SPL, as depicted in Fig. 5, is defined as follows:

$$
\begin{array}{lcl}
\mathcal{B}_r & \doteq & (r : (E_r \doteq \{charge\}, \emptyset, \emptyset)\,, \{\mathbf{opt}(\mathcal{B}_m), \mathbf{mnd}(\mathcal{B}_c), \mathbf{mnd}(\mathcal{B}_o)\}, tt,\ \mathrm{EC}_r, \emptyset\,) \\
\mathcal{B}_m & \doteq & (m : (E_m \doteq \{map\}, \emptyset, \emptyset)\,, \emptyset, tt,\ \emptyset, \emptyset\,) \\
\mathcal{B}_c & \doteq & (c : (E_c, \#_c, \mapsto_c)\,, \emptyset, tt,\ \emptyset, \emptyset\,) \\
\mathcal{B}_o & \doteq & (o : (E_o \doteq \emptyset, \emptyset, \emptyset)\,, \{\mathbf{xor}(\mathcal{B}_{li}, \mathcal{B}_{ca})\}, tt,\ \emptyset, \emptyset\,) \\
\mathcal{B}_{li} & \doteq & (li : (E_{li} \doteq \{liDet\}, \emptyset, \emptyset)\,, \emptyset, tt,\ \emptyset, \emptyset\,) \\
\mathcal{B}_{ca} & \doteq & (ca : (E_{ca} \doteq \{caDet\}, \emptyset, \emptyset)\,, \emptyset, tt,\ \emptyset, \emptyset\,) \\
\mathrm{EC}_r & \doteq & \{\{map\} \mapsto move, \{map\} \mapsto E'_o, \{liDet\} \mapsto goAround, \{caDet\} \mapsto goAround, \\
 & & \quad \{clean\} \mapsto charge, liDet \,\#\, move, caDet \,\#\, move\} \\
E_c & \doteq & \{goAround, move, clean\} \\
\#_c & \doteq & \{(goAround, move)\} \\
\mapsto_c & \doteq & \{(\{goAround, move\}, clean)\} \\
E'_o & \doteq & \{liDet, caDet\}
\end{array}
$$

Notation 4.6 (On Example 4.5). In Example 4.5, E'_r and E'_o denote the sets of all events of the sub-BFM with root *robot* and *obstacleDetection*, respectively. They are primed, i.e, denoted by E'_r and E'_o, to distinguish them from the sets of events E_r and E_o associated with the BES of the feature *robot* and *obstacleDetection*, respectively. We will use primes throughout the paper to mark these differences.

4.2 Featured Event Structures: an Isomorphic Representation of BFMs

In this section, we first (in Sect. 4.2.1) define Featured Event Structures (FESs) which provide an isomorphic representation of BFMs that has a similar structure as FTSs in the following sense: it consists of a feature model, a behavioral model and product knowledge (a mapping annotating the behavioral model with feature expressions) to define behavior of products, while it additionally includes event location knowledge (a mapping associating behavior to features). Then (in Sect. 4.2.2) we define four mappings that, given a BFM $\mathcal{B}$, extract the FM, the BES, the product knowledge, and the event location knowledge of $\mathcal{B}$, respectively. Finally (in Sect. 4.2.3), we define an isomorphism that relates BFMs and FESs which contain the same information.

4.2.1 Featured Event Structures.

Definition 4.7 (Featured event structure (FES)). A *featured event structure (FES)* over the features N and the events E is a quadruple $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$, where

- $\mathcal{M}$ is a FM over N, called *the FM of* $\mathcal{E}$,
- $\mathcal{S} = (E, \#, \mapsto)$ is a BES, called *the BES of* $\mathcal{E}$,
- $\lambda : E \to N$ is a mapping that associates each event of $\mathcal{S}$ with a feature in N, which we call *event location knowledge of* $\mathcal{E}$,
- $\nu : E \to \mathrm{FExp}(N)$ is a mapping that associates each event of $\mathcal{E}$ with a feature expression over N, which we call the *product (configuration) knowledge of* $\mathcal{E}$.

The mapping λ shows the positioning of an event in a feature model while the mapping ν expresses which products of the feature model can include an event in their behavior. The triple $(\mathcal{S}, \lambda, \nu)$ is called *the annotated BES of* $\mathcal{E}$.

Example 4.8 (FES of the cleaning robot SPL). Consider the example of the cleaning robot SPL. We can construct an FES $\mathcal{E}'_r = (\mathcal{M}_r, \mathcal{S}'_r, \lambda'_r, \nu_r)$ over the features N_r (as defined in Example 2.5) and the events E'_r (as defined below) for it, where the FM $\mathcal{M}_r$ is as displayed in Fig. 1 and the annotated BES $(\mathcal{S}'_r, \lambda'_r)$ is as displayed in Fig. 6. Formally, the FES of the cleaning robot is defined as:

- $\mathcal{M}_r$ as in Example 2.5;
- $\mathcal{S}'_r = (E'_r, \#'_r, \mapsto'_r)$, with $E'_r = \bigcup_{i \in \{r,m,c,li,ca\}} E_i$, $\#'_r = \#_c \cup \text{confl}(\text{EC}_r)$ and $\mapsto'_r = \mapsto_c \cup \text{cause}(\text{EC}_r)$;
- $\lambda'_r : E'_r \to N_r$, with
$$\lambda'_r(e) = \begin{cases} \mathit{robot} & \text{if } e \in E_r, \\ \mathit{mapping} & \text{if } e \in E_m, \\ \mathit{cleaning} & \text{if } e \in E_c, \\ \mathit{lidar} & \text{if } e \in E_{li}, \\ \mathit{camera} & \text{if } e \in E_{ca} \text{ and} \end{cases}$$
- $\nu_r : E'_r \to \text{FExp}(N_r)$, with $\nu(e) = tt$ for all $e \in E'_r$;

where E_i, $\#_c$, $\mapsto_c$ and EC_r are as defined in Example 4.5 for $i \in \{r, m, c, o, li, ca\}$.

As for FMs and BFMs in Notations 2.4 and 4.2, respectively, we define some notations that will be helpful for working with FESs.

Notation 4.9 (★ On FESs). We use $\text{features}(\mathcal{E})$, $\text{events}(\mathcal{E})$, $\text{fm}(\mathcal{E})$, $\text{bes}(\mathcal{E})$, $\text{evLocKnow}(\mathcal{E})$ and $\text{prodKnow}(\mathcal{E})$ to denote the set of features, the set of events, the FM, the BES, the event location knowledge, and the product knowledge of the FES $\mathcal{E}$, respectively. Moreover, given an FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ with $\mathcal{S} = (E, \#, \mapsto)$ and one of its features f, we use

- $\text{FC}_f(\mathcal{E}) \doteq \text{FC}_f(\mathcal{M})$ to denote the set of *cross-tree feature constraints of* f;
- $\text{gfc}_f(\mathcal{E}) \doteq \text{gfc}_f(\mathcal{M})$ to denote the set of *group-feature constraints of* f;
- $\text{events}_f(\mathcal{E}) \doteq \{ e \in E \mid \lambda(e) = f \}$ to denote the set of *events of* f;
- $\text{ec}_f(\mathcal{E}) \doteq \{ e_1 \# e_2 \mid e_1, e_2 \in E' \} \cup \{ X \mapsto e \mid X \cup \{e\} \subseteq E' \}$, where $E' \doteq \{ e \mid \lambda(e) \in \text{features}(\text{subTree}_f(\mathcal{M})) \}$ to denote the set of *event constraints of* f;
- $\text{intraFtEc}_f(\mathcal{E}) \doteq \{ e_1 \# e_2 \mid e_1, e_2 \in \text{events}_f(\mathcal{E}) \} \cup \{ X \mapsto e \mid X \cup \{e\} \subseteq \text{events}_f(\mathcal{E}) \}$ to denote the set of *intra-feature event constraints of* f;
- $\text{interFtEc}_f(\mathcal{E}) \doteq \text{ec}_f(\mathcal{E}) \setminus \text{intraFtEc}_f(\mathcal{E})$ to denote the set of *inter-feature event constraints of* f;
- $\text{behFt}_f(\mathcal{E}) \doteq (f : (\text{events}_f(\mathcal{E}), \text{confl}(\text{intraFtEc}_f(\mathcal{E})), \text{cause}(\text{intraFtEc}_f(\mathcal{E}))))$ to denote the *behavioral feature for* f; and
- $\text{productC}_f(\mathcal{E}) \doteq \{ (e, \nu(e)) \mid e \in \text{events}_f(\mathcal{E}) \}$ to denote the *product constraints of* f.

To define the notion of a sub-FES with root f of an FES $\mathcal{E}$, denoted by $\text{subTree}_f(\mathcal{E})$, in Def. 4.13, we first recall a mathematical notation (in Not. 4.10) and define how to restrict a conflict relation, a causality relation and a BES to a smaller set of events (in Def. 4.11).

Notation 4.10 (Restriction of a mapping to a subdomain). The restriction of a mapping $map : Domain \to Range$ to the set $Subdomain \subseteq Domain$, denoted by $map|_{Subdomain} : Subdomain \to Range$, is defined as $map|_{Subdomain}(e) = map(e)$ for $e \in Subdomain$.

Definition 4.11 (Conflict, causality and BES event restriction). Let E and E' be sets of events such that $E' \subseteq E$.

(1) Let $\# \subseteq E \times E$ be a conflict relation. The *restriction* of $\#$ to E' is the conflict relation $\rho_{E'}(\#) \doteq \# \cap (E' \times E')$.
(2) Let $\mapsto \subseteq \mathcal{P}(E) \times E$ be a causality relation. The *restriction* of $\mapsto$ to E' is the causality relation $\rho_{E'}(\mapsto) \subseteq \mathcal{P}(E') \times E'$ such that

$$\rho_{E'}(\mapsto) \doteq \{ (X \cap E', e) \mid (X \mapsto e) \wedge (e \in E') \wedge (X \cap E' \neq \emptyset) \wedge (\neg \exists Y \subseteq E.\, (Y \mapsto e \wedge (X \cap E') \subset (Y \cap E'))) \}.$$

(3) Let $\mathcal{S} = (E, \#, \mapsto)$ be a BES. The *restriction* of $\mathcal{S}$ to E' is the BES $\rho_{E'}(\mathcal{S}) = (E', \rho_{E'}(\#), \rho_{E'}(\mapsto))$.

Since the definition of the restriction of a causality relation is not as straightforward as the one of a conflict relation, we explain the rationale in the following remark.

Remark 4.12 (On Def. 4.11.(2)). The condition

$$\neg \exists Y \subseteq E.\, (Y \mapsto e \wedge (X \cap E') \subset (Y \cap E')) \tag{$*$}$$

in the definition of $\rho_{E'}(\mapsto)$ in Def. 4.11.(2) is needed to ensure that Point (2) of Def. 2.26 still holds for the resulting causality relation $\mapsto' \doteq \rho_{E'}(\mapsto)$: there could be causalities $X \mapsto e$ and $Y \mapsto e$, with $X \not\subset Y$ and $X \cap E' \subset Y \cap E'$, that would produce causalities $(X \cap E') \mapsto' e$ and $(Y \cap E') \mapsto' e$. According to Point (2) of Def. 2.26, the causality $(X \cap E') \mapsto' e$ must be excluded from $\mapsto'$ for $\mapsto'$ to be a causality relation. Therefore, Condition ($*$) is needed in Def. 4.11.(2).

As an example, consider the BES $\mathcal{S} = (E, \#, \mapsto)$ over the events $E = \{e, e_1, e_2, e_3\}$ with $\# = \{(e_1, e_2), (e_2, e_3)\}$ and $\mapsto = \{(\{e_1, e_2\}, e), (\{e_2, e_3\}, e)\}$. Furthermore, let $X = \{e_2, e_3\}$, $Y = \{e_1, e_2\}$ and $E' = \{e, e_1, e_2\}$. By using the definition of $\rho_{E'}(\mapsto)$ without Condition ($*$), we would get $\rho_{E'}(\mapsto) = \{(\{e_1, e_2\}, e), (\{e_2\}, e)\}$, which violates the condition in Point (2) of Def. 2.26. Condition ($*$) ensures that $\rho_{E'}(\mapsto) = \{(\{e_1, e_2\}, e)\}$ and, therefore, that $\rho_{E'}(\mapsto)$ is a causality relation.

A sub-FES with root f of a FES $\mathcal{E}$ is defined as the restriction of $\mathcal{E}$ on the set of events associated with features in the feature model with root f.

Definition 4.13 (Sub-FESs). Given an FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ and a feature $f \in \text{features}(\mathcal{M})$, the *sub-FES of $\mathcal{E}$ with root f* is

$$\text{subTree}_f(\mathcal{E}) \doteq (\text{subTree}_f(\mathcal{M}), \rho_{E'}(\mathcal{S}), \lambda|_{E'}, \nu|_{E'}),$$

where $E' \doteq \{ e \in E \mid \lambda(e) \in \text{features}(\text{subTree}_f(\mathcal{M})) \}$.

Example 4.14 (Sub-FES with root cleaning *of the FES for the cleaning robot SPL).* Consider the FES $\mathcal{E}'_r = (\mathcal{M}_r, \mathcal{S}'_r, \lambda'_r, \nu_r)$ of the cleaning robot SPL over the features N_r and the events E'_r, as defined in Example 4.8, and the feature *cleaning* $\in \text{features}(\mathcal{M}_r)$. Then the sub-FES of $\mathcal{E}'_r$ with root *cleaning* is $\text{subTree}_c(\mathcal{E}'_r) = (\mathcal{M}_c, \mathcal{S}_c, \lambda_c, \nu_c)$, in which

- $\mathcal{M}_c$ is as in Example 2.5;
- $\mathcal{S}_c = (E_c, \#_c, \mapsto_c)$, where E_c, $\#_c$ and $\mapsto_c$ are as in Example 4.5;
- $\lambda_c : E_c \to N_r$ maps all events in E_c to the feature *cleaning*; and
- $\nu : E_c \to \text{FExp}(N_r)$ maps all events in E_c to tt.

4.2.2 FM, BES, Event Location Knowledge and Product Knowledge of a BFM.

To define the isomorphism between BFMs and FESs, we first define the four extraction mappings that we will use in the definition. A BFM is an FM in which behavior is associated with the features. Thus, the FM of a BFM can be extracted by removing the behavioral part of the BFM.

Definition 4.15 (FM of a BFM). The FM of a BFM $\mathcal{B}$, written $\mathrm{fm}(\mathcal{B})$, is recursively defined as follows (where, to improve readability, the behavioral parts are highlighted in gray).

$$\begin{array}{ll}
\mathrm{fm}((f : \mathcal{S}, \{C_i\}_{i\in I^0}, \mathrm{FC}, \boxed{\mathrm{EC}}, \boxed{\mathrm{PC}})) & \doteq (f, \{\mathrm{fm}(C_i)\}_{i\in I^0}, \mathrm{FC}) \\
\mathrm{fm}(\mathrm{Uop}(\mathcal{B})) & \doteq \mathrm{Uop}(\mathrm{fm}(\mathcal{B})) \\
\mathrm{fm}(\mathrm{Op}(\mathcal{B}_i)_{i\in I^2}) & \doteq \mathrm{Op}(\mathrm{fm}(\mathcal{B}_i))_{i\in I^2}
\end{array}$$

Example 4.16 (FM of the BFM of the cleaning robot SPL). Let $\mathcal{B}_r$ be the BFM of the cleaning robot SPL as defined in Example 4.5 (and depicted in Fig. 5). Then the FM of $\mathcal{B}_r$ is extracted as follows:

$$\begin{array}{lll}
\mathrm{fm}(\mathcal{B}_r) & = \mathcal{M}_r & = (r, \{\mathbf{opt}(\mathrm{fm}(\mathcal{B}_m)), \mathbf{mnd}(\mathrm{fm}(\mathcal{B}_c)), \mathbf{mnd}(\mathrm{fm}(\mathcal{B}_o))\}, \mathit{tt}) \\
\mathrm{fm}(\mathcal{B}_m) & = \mathcal{M}_m & = (m, \varnothing, \mathit{tt}) \\
\mathrm{fm}(\mathcal{B}_c) & = \mathcal{M}_c & = (c, \varnothing, \mathit{tt}) \\
\mathrm{fm}(\mathcal{B}_o) & = \mathcal{M}_o & = (o, \{\mathbf{xor}(\mathrm{fm}(\mathcal{B}_{li}), \mathrm{fm}(\mathcal{B}_{ca}))\}, \mathit{tt}) \\
\mathrm{fm}(\mathcal{B}_{li}) & = \mathcal{M}_{li} & = (li, \varnothing, \mathit{tt}) \\
\mathrm{fm}(\mathcal{B}_{ca}) & = \mathcal{M}_{ca} & = (ca, \varnothing, \mathit{tt})
\end{array}$$

where the FMs $\mathcal{M}_f$, for $f \in \{r, m, c, o, li, ca\}$, are the FMs obtained in Example 2.5. In particular, $\mathcal{M}_r$ (which is the FM of $\mathcal{B}_r$) is depicted in Fig. 1.

The following proposition states that, for every BFM $\mathcal{B}$, the underlying FM of the sub-BFM with root f of $\mathcal{B}$ is the same as the sub-FM with root f of the underlying FM of $\mathcal{B}$.

Proposition 4.17 (On FMs of Sub-BFMs). *Let $\mathcal{B}$ be a BFM and $f \in \mathrm{features}(\mathcal{B})$. Then it holds that* $\mathrm{fm}(\mathrm{subTree}_f(\mathcal{B})) = \mathrm{subTree}_f(\mathrm{fm}(\mathcal{B}))$.

Proof. Straightforward using the definition of the sub-BFM with root f and by Def. 4.15. □

The BES of a BFM $\mathcal{B}$ can be obtained by collecting all events associated with the features of $\mathcal{B}$ as well as all conflicts and causalities associated with the features of $\mathcal{B}$ and all inter-feature conflicts and causalities.

Definition 4.18 (BES of a BFM). The *BES of a BFM* $\mathcal{B}$, denoted by $\mathrm{bes}(\mathcal{B})$, is defined inductively as follows:

$\mathrm{bes}((f : (E_f, \#_f, \mapsto_f), \varnothing, \mathrm{FC}, \mathrm{EC}, \mathrm{PC})) \doteq (E_f, \#_f \cup \mathrm{confl}(\mathrm{EC}), \mapsto_f \cup \mathrm{cause}(\mathrm{EC}))$

$\mathrm{bes}((f : (E_f, \#_f, \mapsto_f), \{\mathrm{Uop}(\mathcal{B}_1)\}, \mathrm{FC}, \mathrm{EC}, \mathrm{PC}))$
$\quad \doteq (E_1 \cup E_f, \#_1 \cup \#_f \cup \mathrm{confl}(\mathrm{EC}), \mapsto_1 \cup \mapsto_f \cup \mathrm{cause}(\mathrm{EC}))$, where $(E_1, \#_1, \mapsto_1) = \mathrm{bes}(\mathcal{B}_1)$

$\mathrm{bes}((f : (E_f, \#_f, \mapsto_f), \{\mathrm{Op}(\mathcal{B}_i)_{i\in I^2}\}, \mathrm{FC}, \mathrm{EC}, \mathrm{PC}))$
$\quad \doteq (\bigcup_{i\in I^2} E_i \cup E_f, \bigcup_{i\in I^2} \#_i \cup \#_f \cup \mathrm{confl}(\mathrm{EC}), \bigcup_{i\in I^2} \mapsto_i \cup \mapsto_f \cup \mathrm{cause}(\mathrm{EC}))$,
$\quad$ where $(E_i, \#_i, \mapsto_i) = \mathrm{bes}(\mathcal{B}_i)$

$\mathrm{bes}((f : (E_f, \#_f, \mapsto_f), \{C_i\}_{i\in I^2}, \mathrm{FC}, \mathrm{EC}, \mathrm{PC}))$
$\quad \doteq (\bigcup_{i\in I^0} E_i \cup E_f, \bigcup_{i\in I^0} \#_i \cup \#_f \cup \mathrm{confl}(\mathrm{EC}), \bigcup_{i\in I^0} \mapsto_i \cup \mapsto_f \cup \mathrm{cause}(\mathrm{EC}))$,
$\quad$ where $(E_i, \#_i, \mapsto_i) = \mathrm{bes}((f : \varnothing, \{C_i\}, \varnothing, \varnothing))$

Example 4.19 (BES of the BFM of the cleaning robot SPL). Let $\mathcal{B}_r$ be the BFM of the cleaning robot SPL as defined in Example 4.5 (and depicted in Fig. 5). Then the BES of $\mathcal{B}_r$ is extracted as follows:

$$\begin{aligned}
\mathcal{S}'_r \doteq \mathrm{bes}(\mathcal{B}_r) &= (E'_r = \{charge\} \cup E_m \cup E_c \cup E_o,\\
&\quad \#'_r = \{(liDet, move), (caDet, move)\} \cup \#_c,\\
&\quad \mapsto'_r = \{(\{map\}, move), (\{map\}, E_o), (\{liDet\}, goAround),\\
&\qquad (\{caDet\}, goAround), (\{clean\}, charge)\} \cup \mapsto_c)\\
\mathcal{S}_m \doteq \mathrm{bes}(\mathcal{B}_m) &= (E_m = \{map\}, \emptyset, \emptyset)\\
\mathcal{S}_c \doteq \mathrm{bes}(\mathcal{B}_c) &= (E_c = \{goAround, move, clean\},\\
&\quad \#_c = \{(goAround, move)\},\\
&\quad \mapsto_c = \{(\{goAround, move\}, clean)\})\\
\mathcal{S}'_o \doteq \mathrm{bes}(\mathcal{B}_o) &= (E'_o = \{caDet, liDet\}, \emptyset, \emptyset)\\
\mathcal{S}_{li} \doteq \mathrm{bes}(\mathcal{B}_{li}) &= (E_{li} = \{liDet\}, \emptyset, \emptyset)\\
\mathcal{S}_{ca} \doteq \mathrm{bes}(\mathcal{B}_{ca}) &= (E_{ca} = \{caDet\}, \emptyset, \emptyset)
\end{aligned}$$

A graphical representation of $\mathrm{bes}(\mathcal{B}_r)$ can be found in Fig. 6 by ignoring the feature annotations highlighted in gray (which represent event location knowledge). Note that the BES $\mathrm{bes}(\mathcal{B}_r)$ of the BFM $\mathcal{B}_r$ of the cleaning robot SPL is equal to the BES $\mathcal{S}'_r$ of the FES $\mathcal{E}'_r$ of the cleaning robot SPL in Example 4.8.

The first behavioral mapping we need is the one extracting event location knowledge of a BFM, which maps an event to the feature it is associated with in the BFM.

Definition 4.20 (Event location knowledge of a BFM). Let N be a set of features and E be a set of events. *Event location knowledge of a BFM* $\mathcal{B}$ over features N and events E is a mapping $\mathrm{evLocKnow}(\mathcal{B}) : E \to N$ defined by $\mathrm{evLocKnow}(\mathcal{B})(e) = f$, where $\mathrm{behFt}_f(\mathcal{B}) = (f : (E_f, \#_f, \mapsto_f))$ with $e \in E_f$.

Example 4.21 (Event location knowledge of the BFM of the cleaning robot SPL). Let $\mathcal{B}_r$ be the BFM of the cleaning robot SPL as defined in Example 4.5 (and depicted in Fig. 5). Then the event location knowledge of $\mathcal{B}_r$ is the mapping $\mathrm{evLocKnow}(\mathcal{B}_r) : E'_r \to N_r$ (where E'_r is as defined in Example 4.19) defined by:

$$\mathrm{evLocKnow}(\mathcal{B})_r(e) = \begin{cases} mapping & \text{if } e = map,\\ cleaning & \text{if } e \in \{goAround, clean, move\},\\ lidar & \text{if } e = liDet,\\ camera & \text{if } e = caDet,\\ robot & \text{if } e = charge. \end{cases}$$

Note that the event location knowledge $\mathrm{evLocKnow}(\mathcal{B})_r$ of the BFM $\mathcal{B}_r$ of the cleaning robot SPL is equal to the event location knowledge λ'_r of the FES $\mathcal{E}'_r$ of the cleaning robot SPL in Example 4.8.

The second behavioral mapping is the one extracting product (configuration) knowledge from a BFM, representing the feature restrictions of events.

Definition 4.22 (Product (configuration) knowledge of a BFM). Let N be a set of features and E be a set of events. *Product (configuration) knowledge of a BFM* $\mathcal{B}$ over features N and events E, denoted by $\mathrm{prodKnow}(\mathcal{B})$, is a mapping $\mathrm{prodKnow}(\mathcal{B}) : E \to N$ defined by

$$\mathrm{prodKnow}(\mathcal{B})(e) = \begin{cases} \phi_e & \text{if } \exists (e, \phi_e) \in \mathrm{productC}(\mathcal{B}),\\ \not\!tt & \text{otherwise.} \end{cases}$$

Example 4.23 (Product knowledge of the BFM of the cleaning robot SPL). Consider the BFM $\mathcal{B}_r$ of the cleaning robot SPL over the events E'_r and the features N_r as in Example 4.5. Then $\text{prodKnow}(\mathcal{B}_r) : E'_r \to N_r$ is the mapping $\text{prodKnow}(\mathcal{B}_r)(e) = tt$ for all $e \in E'_r$.

4.2.3 FES View of a BFM and BFM View of an FES.

Now we have all the ingredients in place to define the mapping between BFMs and FESs that maps a BFM to the tuple containing its extracted FM, its extracted BES, its extracted event location knowledge and its extracted product knowledge.

Definition 4.24 (FES view of a BFM). Let $\mathbb{B}$ and $\mathbb{E}$ be the sets of BFMs and FESs, respectively, over a set of features N and a set of events E. The mapping

$$\text{fes} : \mathbb{B} \to \mathbb{E},\ \mathcal{B} \mapsto (\text{fm}(\mathcal{B}), \text{bes}(\mathcal{B}), \text{evLocKnow}(\mathcal{B}), \text{prodKnow}(\mathcal{B}))$$

takes a BFM $\mathcal{B}$ over N and E and returns the *FES view of* $\mathcal{B}$, which is the FES over N and E consisting of the extracted FM, BES, event location knowledge and product knowledge of $\mathcal{B}$.

Example 4.25 (FES view of the BFM of the cleaning robot SPL). The FES view of the BFM $\mathcal{B}_r$ of the cleaning robot SPL as defined in Example 4.5 (and depicted in Fig. 5) is the FES $\mathcal{E}'_r = (\mathcal{M}_r, \mathcal{S}'_r, \lambda'_r, \nu_r)$, where $\mathcal{M}_r$, $\mathcal{S}'_r$, λ'_r and ν_r are given in Examples 4.16, 4.19, 4.21 and 4.23, respectively. Thus, it is the FES defined in Example 4.8. In particular, the FM $\mathcal{M}_r$ is depicted in Fig. 1 and the annotated BES $(\mathcal{S}'_r, \lambda'_r)$ is depicted in Fig. 6.

We furthermore define a mapping that takes an FES and returns a BFM that is inductively constructed from the structure of the FES's FM using the notations defined in Not. 4.9, i.e., using the events associated with features via the event location knowledge, the product knowledge, and the different event and feature constraints obtained from the BES and the FM, respectively.

Definition 4.26 (BFM view of FES). Let $\mathbb{B}$ and $\mathbb{E}$ be the sets of BFMs and FESs, respectively, over a set of features N and a set of events E. The mapping $\text{bfm} : \mathbb{E} \to \mathbb{B}$ takes an FES $\mathcal{E}$ over N and E and returns the *BFM view of* $\mathcal{E}$, which is the BFM over N and E

$$\text{bfm}(\mathcal{E}) \doteq (f : \mathcal{S}_f, \text{GFC}_f, \text{FC}_f(\mathcal{E}), \text{interFtEc}_f(\mathcal{E}), \text{productC}_f(\mathcal{E})),$$

where

$$\begin{aligned} f &= \text{root}(\text{fm}(\mathcal{E})), \\ \mathcal{S}_f &= (\text{events}_f(\mathcal{E}), \text{confl}(\text{intraFtEc}_f(\mathcal{E})), \text{cause}(\text{intraFtEc}_f(\mathcal{E}))), \text{ and} \\ \text{GFC}_f &= \{\, \text{Uop}(\text{bfm}(\text{subTree}_{f_i}(\mathcal{E}))) \mid \text{Uop}(f_i) \in \text{gfc}_f(\mathcal{E}) \,\} \\ &\quad \cup \{\, \text{Op}(\text{bfm}(\text{subTree}_{f_{j_1}}(\mathcal{E})), \ldots, \text{bfm}(\text{subTree}_{f_{j_m}}(\mathcal{E}))) \mid \text{Op}(f_{j_1}, \ldots, f_{j_m}) \in \text{gfc}_f(\mathcal{E}) \,\}. \end{aligned}$$

Example 4.27 (BFM view of the FES of the cleaning robot SPL). The BFM view of the FES $\mathcal{E}'_r$ of the cleaning robot SPL as defined in Example 4.8 (and depicted in Figs. 1 and 6) is the BFM $\mathcal{B}_r$ as defined in Example 4.5 (and depicted in Fig. 5).

The following proposition states that the mappings fes and bfm are isomorphisms.

Proposition 4.28 (Mappings fes and bfm are isomorphisms). *Let $\mathbb{B}$ and $\mathbb{E}$ be the sets of BFMs and FESs, respectively, over a set of features N and a set of events E. Let $\mathcal{B} \in \mathbb{B}$, $\mathcal{E} \in \mathbb{E}$ and $f \in N$. Then it holds that:*

(1) $\text{fm}(\mathcal{B}) = \text{fm}(\text{fes}(\mathcal{B}))$ *and* $\text{fm}(\mathcal{E}) = \text{fm}(\text{bfm}(\mathcal{E}))$*;*
(2) $\text{bes}(\mathcal{B}) = \text{bes}(\text{fes}(\mathcal{B}))$ *and* $\text{bes}(\mathcal{E}) = \text{bes}(\text{bfm}(\mathcal{E}))$*;*
(3) $\text{evLocKnow}(\mathcal{B}) = \text{evLocKnow}(\text{fes}(\mathcal{B}))$ *and* $\text{evLocKnow}(\mathcal{E}) = \text{evLocKnow}(\text{bfm}(\mathcal{E}))$*;*
(4) $\text{prodKnow}(\mathcal{B}) = \text{prodKnow}(\text{fes}(\mathcal{B}))$ *and* $\text{prodKnow}(\mathcal{E}) = \text{prodKnow}(\text{bfm}(\mathcal{E}))$*;*
(5) $\text{behFt}_f(\mathcal{B}) = \text{behFt}_f(\text{fes}(\mathcal{B}))$ *and* $\text{behFt}_f(\mathcal{E}) = \text{behFt}_f(\text{bfm}(\mathcal{E}))$*;*

(6) $\text{events}_f(\mathcal{B}) = \text{events}_f(\text{fes}(\mathcal{B}))$ *and* $\text{events}_f(\mathcal{E}) = \text{events}_f(\text{bfm}(\mathcal{E}))$;
(7) $\text{intraFtEc}_f(\mathcal{B}) = \text{intraFtEc}_f(\text{fes}(\mathcal{B}))$ *and* $\text{intraFtEc}_f(\mathcal{E}) = \text{intraFtEc}_f(\text{bfm}(\mathcal{E}))$;
(8) $\text{interFtEc}_f(\mathcal{B}) = \text{interFtEc}_f(\text{fes}(\mathcal{B}))$ *and* $\text{interFtEc}_f(\mathcal{E}) = \text{interFtEc}_f(\text{bfm}(\mathcal{E}))$;
(9) $\text{FC}_f(\mathcal{B}) = \text{FC}_f(\text{fes}(\mathcal{B}))$ *and* $\text{FC}_f(\mathcal{E}) = \text{FC}_f(\text{bfm}(\mathcal{E}))$;
(10) $\text{gfc}_f(\mathcal{B}) = \text{gfc}_f(\text{fes}(\mathcal{B}))$ *and* $\text{gfc}_f(\mathcal{E}) = \text{gfc}_f(\text{bfm}(\mathcal{E}))$;
(11) $\text{ec}_f(\mathcal{B}) = \text{ec}_f(\text{fes}(\mathcal{B}))$ *and* $\text{ec}_f(\mathcal{E}) = \text{ec}_f(\text{bfm}(\mathcal{E}))$;
(12) $\text{productC}_f(\mathcal{B}) = \text{productC}_f(\text{fes}(\mathcal{B}))$ *and* $\text{productC}_f(\mathcal{E}) = \text{productC}_f(\text{bfm}(\mathcal{E}))$;
(13) $\text{subTree}_f(\mathcal{B}) = \text{bfm}(\text{subTree}_f(\text{fes}(\mathcal{B})))$ *and* $\text{subTree}_f(\mathcal{E}) = \text{fes}(\text{subTree}_f(\text{bfm}(\mathcal{E})))$; *and*
(14) $\text{fes} \circ \text{bfm} = id_{\mathbb{B}}$ *and* $\text{bfm} \circ \text{fes} = id_{\mathbb{E}}$, *where* $id_{\mathbb{B}}$ *and* $id_{\mathbb{E}}$ *are the identity mappings on* $\mathbb{B}$ *and* $\mathbb{E}$, *respectively.*

Therefore, the mappings fes *and* bfm *are isomorphisms.*

Proof. Straightforward from Definitions 4.24 and 4.26 and Notations 4.2 and 4.9. □

4.3 Products and Behavioral Products of a BFM/FES

As for FMs, we also want to define a notion of products for BFMs and FESs. Furthermore, since (different from FMs), BFMs and FESs contain behavior, we also define the notion of a behavioral product of a BFM and an FES, corresponding to a product and its behavior.

Definition 4.29 (Products and behavioral products of a BFM/FES). Given an FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$, the *products of* $\mathcal{E}$ are the products of $\mathcal{M}$, and for each of its products pr, the *behavioral product of* $\mathcal{E}$ *for* pr is the BES $\text{behPr}(\mathcal{E}, pr) = \rho_{E'}(\mathcal{S})$, where $E' = \{ e \mid \lambda(e) \in pr \wedge pr \models \nu(e) \}$. We write $\text{products}(\mathcal{E})$ to denote the set of products of $\mathcal{E}$ and $\text{behPr}(\mathcal{E})$ to denote the set of behavioral products of $\mathcal{E}$.

The products and behavioral products of a BFM are defined using its FES view: For a BFM $\mathcal{B}$ we define its products as $\text{products}(\mathcal{B}) = \text{products}(\text{fes}(\mathcal{B}))$ and a behavioral product with respect to $pr \in \text{products}(\mathcal{B})$ as $\text{behPr}(\mathcal{B}, pr) = \text{behPr}(\text{fes}(\mathcal{B}), pr)$. We denote by $\text{behPr}(\mathcal{B})$ the set of all behavioral products of $\mathcal{B}$.

Example 4.30 (Behavioral products of the cleaning robot SPL). Consider the BFM $\mathcal{B}_r$ of the cleaning robot SPL defined in Example 4.5 (and depicted in Fig. 5) and its FES view $\mathcal{E}_r$ defined in Example 4.25 (its FM is depicted in Figure 1 and its annotated BES is depicted in Fig. 6). The products of $\mathcal{B}_r$ are the products pr_1, pr_2, pr_3 and pr_4 as defined in Example 2.10. Its behavioral products are the following:

$$
\begin{aligned}
\mathcal{S}_1 \doteq \text{behPr}(\mathcal{E}_r, pr_1) &= (E_1, \#_1, \mapsto_1), \text{ with} \\
E_1 &= \{charge, goAround, clean, move, liDet\}, \\
\#_1 &= \{(goAround, move), (liDet, move)\}, \\
\mapsto_1 &= \{(\{goAround, move\}, clean), (\{liDet\}, goAround), (\{clean\}, charge)\}; \\
\mathcal{S}_2 \doteq \text{behPr}(\mathcal{E}_r, pr_2) &= (E_2, \#_2, \mapsto_2), \text{ with} \\
E_2 &= \{charge, goAround, clean, move, caDet\}, \\
\#_2 &= \{(goAround, move), (caDet, move)\}, \\
\mapsto_2 &= \{(\{goAround, move\}, clean), (\{caDet\}, goAround), (\{clean\}, charge)\}; \\
\mathcal{S}_3 \doteq \text{behPr}(\mathcal{E}_r, pr_3) &= (E_3 = E_1 \cup \{map\}, \#_3 = \#_1, \mapsto_3 = \mapsto_1 \cup \{(\{map\}, move), (\{map\}, liDet)\}); \\
\mathcal{S}_4 \doteq \text{behPr}(\mathcal{E}_r, pr_4) &= (E_4 = E_2 \cup \{map\}, \#_4 = \#_2, \mapsto_4 = \mapsto_2 \cup \{(\{map\}, move), (\{map\}, caDet)\});
\end{aligned}
$$

where $\mathcal{S}_3$ is as in Example 2.29.

4.4 Configurations and Traces of a BFM/FES

The configurations of a BFM/FES are defined in terms of its behavioral products, using the definition of a configuration of a BES, see Def. 2.28.

Definition 4.31 (Configuration of a BFM/FES). Let $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ be an FES with $\mathcal{S} = (E, \#, \mapsto)$. A set $C \subseteq E$ is called a *configuration* of $\mathcal{E}$ if there exists a product pr of $\mathcal{E}$ such that C is a configuration of $\text{behPr}(\mathcal{E}, pr)$. More concretely, a set $C \subseteq E$ is a configuration of $\mathcal{E}$ if

(0) there exists a product pr of $\mathcal{E}$ such that $\lambda(e) \in pr$ and $pr \models \nu(e)$ for all $e \in C$;
(1) C is conflict free, i.e., $\forall e, e' \in C.\ (e, e') \notin \#$; and
(2) $\forall X \subseteq E'\ \forall e \in C.\ X \mapsto' e \Rightarrow C \cap X \neq \emptyset$ for $\text{behPr}(\mathcal{E}, pr) = (E', \#', \mapsto')$.

Analogously, a configuration of a BFM $\mathcal{B}$ is a configuration of one of the behavioral products of $\mathcal{B}$. We write $\text{Config}(\mathcal{X}, pr)$ for the set of configurations of a BFM/FES $\mathcal{X}$ for the product pr and $\text{Config}(\mathcal{X})$ for $\bigcup_{pr \in \text{products}(\mathcal{X})} \text{Config}(\mathcal{X}, pr)$.

Note that Condition (2) is slightly different from Condition (2) in Def. 2.28 because we only consider the events and causality relation of the behavioral product for pr, not all events and the causality relation of $\mathcal{E}$.

Example 4.32 (Configurations of the product pr_3 of the cleaning robot SPL). Consider the BFM $\mathcal{B}_r$ of the cleaning robot SPL defined in Example 4.5 (and depicted in Fig. 5) and its FES view $\mathcal{E}_r$ defined in Example 4.25 (its FM is depicted in Figure 1 and its annotated BES is depicted in Fig. 6). Then the configurations of the behavioral product $\mathcal{S}_3 = \text{behPr}(\mathcal{E}_r, pr_3)$ as defined in Example 4.30 are the sets $\emptyset$, $\{map\}$, $\{map, move\}$, $\{map, move, clean\}$, $\{map, move, clean, charge\}$, $\{map, liDet\}$, $\{map, liDet, goAround\}$, $\{map, liDet, goAround, clean\}$, $\{map, liDet, goAround, clean, charge\}$ that are shown in Example 2.32.

Similarly, the traces of BFMs and FESs are defined via the traces of their behavioral products, i.e., using the definition of a trace of a BES as defined in Def. 2.30.

Definition 4.33 (Traces of a BFM/FES for a product). The *traces* of an FES $\mathcal{E}$ for one of its products pr are the traces of its behavioral product for pr, i.e., they are traces in the set $\text{Tr}(\mathcal{E}, pr) \doteq \text{Tr}(\text{behPr}(\mathcal{E}, pr))$. More concretely, for an FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \mu)$ and one of its products pr with behavioral product $\text{behPr}(\mathcal{E}, pr) = (E, \#, \mapsto)$, a trace of $\mathcal{E}$ for pr is a sequence $\langle e_1 \cdot \ldots \cdot e_n \rangle$ $(n \geq 1)$ of distinct events $e_i \in E, i \in \{1, \ldots, n\}$, satisfying

(0) $\forall i \in \{1, \ldots, n\}.\ \lambda(e_i) \in pr$ and $pr \models \nu(e_i)$;
(1) $\{e_1, \ldots, e_n\}$ is conflict-free;
(2) $\forall X \subseteq E = \{e \mid \lambda(e) \in pr \wedge pr \models \nu(e)\}\ \forall i \in \{2, \ldots, n\}.\ X \mapsto e_i \Rightarrow \{e_1, \ldots, e_{i-1}\} \cap X \neq \emptyset$; and
(3) $\nexists X \subseteq E.\ X \mapsto e_1$.

The traces of a BFM $\mathcal{B}$ for one of its products pr are defined using its FES view: $\text{Tr}(\mathcal{B}, pr) \doteq \text{Tr}(\text{fes}(\mathcal{B}), pr)$.

Note that Conditions (1)–(3) are as in Def. 2.30, slightly modified for BFMs/FESs.

Example 4.34 (Traces of the behavioral products of the BFM of the cleaning robot SPL). Consider the BFM $\mathcal{B}_r$ of the cleaning robot SPL defined in Example 4.5 (and depicted in Fig. 5) and its FES view $\mathcal{E}_r$ defined in Example 4.25 (its FM is depicted in Figure 1 and its annotated BES is depicted in Fig. 6). The products of $\mathcal{B}_r$ are pr_1, pr_2, pr_3 and pr_4 as defined in Example 2.10 and its behavioral products are $\mathcal{S}_i = \text{behPr}(\mathcal{E}_r, pr_i)$ $(1 \leq i \leq 4)$ as defined in Example 4.30. Their traces are the following:

$\text{Tr}(\mathcal{S}_1) = \text{prefixClosure}(\{\langle move \cdot clean \cdot charge \rangle, \langle liDet \cdot goAround \cdot clean \cdot charge \rangle\})$,
$\text{Tr}(\mathcal{S}_2) = \text{prefixClosure}(\{\langle move \cdot clean \cdot charge \rangle, \langle caDet \cdot goAround \cdot clean \cdot charge \rangle\})$,
$\text{Tr}(\mathcal{S}_3) = \text{prefixClosure}(\{\langle map \cdot move \cdot clean \cdot charge \rangle, \langle map \cdot liDet \cdot goAround \cdot clean \cdot charge \rangle\})$, and
$\text{Tr}(\mathcal{S}_4) = \text{prefixClosure}(\{\langle map \cdot move \cdot clean \cdot charge \rangle, \langle map \cdot caDet \cdot goAround \cdot clean \cdot charge \rangle\})$.

The following lemma is the equivalent of Lemma 2.31 for BFMs/FESs, showing that configurations form traces of a BFM/FES.

Lemma 4.35 (★ Configurations forming traces of a BFM/FES). *Let $\mathcal{X}$ be a BFM/FES over the events E, let $pr \in \text{products}(\mathcal{X})$, and let $e_1, \ldots, e_n \in E$ for $n > 1$. Then $\exists tr \in \text{Tr}(\mathcal{X}, pr)$ such that $\{e_1, \ldots, e_n\} = \text{events}(tr)$ if and only if $\{e_1, \ldots, e_n\} \in \text{Config}(\mathcal{X}, pr)$.*

Proof. We provide the proof for an FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$; for a BFM $\mathcal{B}$ the proof works exactly the same, using $\text{fes}(\mathcal{B})$. Let us first recall the definition of the behavioral product of $\mathcal{E}$ for pr, defined in Def. 4.29. It is the BES $\text{behPr}(\mathcal{E}, pr) = \rho_{E'}(\mathcal{S}) \doteq (E', \#', \mapsto')$ with events $E' = \{\, e \in E \mid \lambda(e) \in pr \wedge pr \models \nu(e) \,\}$.

"$\Rightarrow$" Let $tr \in \text{Tr}(\mathcal{E}, pr)$ with $\{e_1, \ldots, e_n\} = \text{events}(tr)$. Then Conditions (0)–(3) of Def. 4.33 hold for tr and imply Conditions (0')–(2') of Def. 4.31 for $\{e_1, \ldots, e_n\}$, i.e, they imply that $\{e_1, \ldots, e_n\}$ is a configuration of $\mathcal{E}$ for pr: Condition (0) implies Condition (0'), Condition (1) implies Condition (1'), and Conditions (2) and (3) imply Condition (2').

"$\Leftarrow$" Let $\{e_1, \ldots, e_n\} \in \text{Config}(\mathcal{E}, pr)$, and let $\langle e'_1, \ldots, e'_n \rangle$ be an ordering of $\{e_1, \ldots, e_n\}$ according to the partial order $<_{\{e_1,\ldots,e_n\}}$ introduced by $\mapsto'$ (for an explanation see the text above Lemma 2.31). Then the ordering ensures that $\langle e'_1, \ldots, e'_n \rangle$ satisfies Conditions (2) and (3) of being a trace of $\mathcal{X}$ for pr (see Def. 4.33), and $\langle e'_1, \ldots, e'_n \rangle$ satisfies Conditions (0) and (1) because $\{e'_1, \ldots, e'_n\} = \{e_1, \ldots, e_n\}$ is a configuration of $\mathcal{X}$ for pr (see Def. 4.31). Thus, $\langle e'_1, \ldots, e'_n \rangle$ is a trace of $\mathcal{X}$ for pr. □

Definition 4.36 (★ Feature expression associated with a configuration of a BFM/FES). Let $\mathcal{X}$ be a BFM/FES over the features N and let $C \in \text{Config}(\mathcal{X})$ be a configuration of $\mathcal{X}$. Then the *feature expression associated with* C is the feature expression $\phi_C \in \text{FExp}(N)$ with

$$\phi_C = \bigvee_{\{pr \in \text{products}(\mathcal{X}) \mid C \in \text{Config}(\mathcal{X}, pr)\}} \text{fexp}(pr)$$

That is, the feature expression associated with C is the disjunction of the feature expressions for all products that have C as a configuration.

The following definition will be needed to define an algorithm from an FES to an FTS in Sect. 5.4.

Example 4.37 (★ Feature expressions associated with configurations of the product pr_3 of the cleaning robot SPL). Consider the BFM $\mathcal{B}_r$ of the cleaning robot SPL defined in Example 4.5 (and depicted in Fig. 5) with products pr_1, pr_2, pr_3 and pr_4 as defined in Example 2.10 and the configurations of its product pr_3 as defined in Example 4.32. Then the feature expressions associated with the elements of $\text{Config}(\mathcal{B}_r, pr_3)$ are the following:

$$\begin{aligned}
\phi_\varnothing &= \text{fexp}(pr_1) \vee \text{fexp}(pr_2) \vee \text{fexp}(pr_3) \vee \text{fexp}(pr_4) \\
\phi_{\{map\}} = \phi_{\{map,move\}} &= \phi_{\{map,move,clean\}} = \phi_{\{map,move,clean,charge\}} = \text{fexp}(pr_3) \vee \text{fexp}(pr_4) \\
\phi_{\{map,liDet\}} = \phi_{\{map,liDet,goAround\}} &= \phi_{\{map,liDet,goAround,clean\}} = \phi_{\{map,liDet,goAround,clean,charge\}} = \text{fexp}(pr_3)
\end{aligned}$$

The following lemma will be needed to prove trace equivalence of an FES and an FTS in Sect. 5.4.

Lemma 4.38 (★ On feature expressions associated with configurations). *Let $\mathcal{X}$ be a BFM/FES over the features N, let $C \in \text{Config}(\mathcal{X})$ be a configuration of $\mathcal{X}$, and let $pr \in \text{products}(\mathcal{X})$ be a product of $\mathcal{X}$. Then $\text{eval}_{pr}(\phi_C) = tt$ if and only if $C \in \text{Config}(\mathcal{X}, pr)$.*

Proof. Recall that $\phi_C = \bigvee_{\{pr' \in \text{products}(\mathcal{X}) \mid C \in \text{Config}(\mathcal{X}, pr)\}} \text{fexp}(pr')$, where $\text{fexp}(pr') = \bigwedge_{f \in pr'} f \wedge \bigwedge_{g \notin pr'} g$ for a product $pr' \in \text{products}(\mathcal{X})$. That is, for every product $pr' \in \text{products}(\mathcal{X})$, the feature expression $\text{fexp}(pr')$ is a feature expression over all features of N; more concretely, it is a conjunction over all (negated or non-negated) features of N. Since ϕ_C is a disjunction of all these feature expressions, it evaluates to true if and only if at least one of its conjuncts evaluates to true.

"$\Rightarrow$" Let $\text{eval}_{pr}(\phi_C) = tt$. As we observed above, this means that there exists $pr' \in \text{products}(\mathcal{X})$ with $C \in \text{Config}(\mathcal{X}, pr')$ such that $\text{eval}_{pr}(\text{fexp}(pr')) = tt$. This implies $pr = pr'$ because a product does not satisfy another product's feature expression. Therefore, $C \in \text{Config}(\mathcal{X}, pr)$.

"$\Leftarrow$" Let $C \in \text{Config}(\mathcal{X}, pr)$. Then by definition $\text{fexp}(pr)$ is included in ϕ_C, so $\text{eval}_{pr}(\phi_C) = tt$. □

4.5 Composing and Decomposing BFMs and FESs

In Sect. 3, we motivated the design of BFMs (introducing behavior in FMs), among others, by the claim that BFMs feature the same kind of compositionality as FMs. In this section, we shortly give an overview of how FMs can be composed and decomposed, and then show that these concepts are also reflected in the structure of BFMs and FESs, thereby supporting the motivation given in Sect. 3.

FMs come with a natural notion of composition: the FMs of (sub)systems can be *composed* into a larger FM by making them immediate sub-FMs of a new feature (the root of the FM of the composed system) and adding group feature constraints and cross-tree constraints. Thereby, a (sub-)FM can be reused across different (composed) FMs.

Definition 4.39 (FM composition). Given

- a feature f;
- a number of $m \geq 1$ FMs $\mathcal{M}_i$, for $i \in \{1, \ldots, m\}$, such that the $m{+}1$ sets $\{f\}$, $\text{features}(\mathcal{B}_1), \ldots,$ $\text{features}(\mathcal{B}_m)$ are all pairwise disjoint;
- a set of group feature constraints which describe how to group the FMs $\mathcal{M}_i$ into sub-FM groups $\{\mathcal{G}_j\}_{j \in I^1}$; and
- cross-tree feature constraints FC, where the features included in the feature expression belong to at least two distinct $\mathcal{M}_i$'s;

their *composition* is the FM $\mathcal{M} = (f, \{\mathcal{G}_i\}_{i \in I^1}, \text{FC})$ over the set of features $\{f\} \cup (\bigcup_{i=1,\ldots,m} \text{features}(\mathcal{M}_i))$.

Since this composition is reversible, FMs can also be decomposed: sub-FMs can be extracted from a larger FM and analyzed or refactored in isolation, or be reused in other FMs. By interleaving composition and decomposition, a sub-FM of an FM can be replaced by other sub-FMs.

Definition 4.40 (FM decomposition). An FM $\mathcal{M}$ in which the root feature $f = \text{root}(\mathcal{M})$ has $m \geq 1$ child features $f_1, \ldots, f_m$ can be decomposed into

- the root feature f;
- its immediate sub-FMs $\mathcal{M}_i = \text{subTree}_{f_i}(\mathcal{M})$, for $i \in \{1, \ldots, m\}$;
- the set $\text{gfc}_f(\mathcal{M})$ of group feature constraints of the root feature f, which describe how the $\mathcal{M}_i$'s are grouped into sub-FM groups of $\mathcal{M}$; and
- the cross-tree feature constraints $\text{FC}_f(\mathcal{M})$ of the root feature f.

The following two definitions show how the notion of FM composition can be lifted to BFMs and FESs, respectively. Extensions to FMs with behavior are again highlighted with gray background in the definition of composition for BFMs.

Definition 4.41 (BFM composition). Given

(1) a behavioral feature $(f, \mathcal{S})$;
(2) a number of $m \geq 1$ BFMs $\mathcal{B}_i$, for $i \in \{1, \ldots, m\}$, such that the $m{+}1$ sets $\{f\}$, $\text{features}(\mathcal{B}_1), \ldots,$ $\text{features}(\mathcal{B}_m)$ are all pairwise disjoint and likewise the $m{+}1$ sets $\text{events}(\mathcal{S}), \text{events}(\mathcal{B}_1), \ldots,$ $\text{events}(\mathcal{B}_m)$ are all pairwise disjoint;
(3) a set of group feature constraints which describe how to group the BFMs $\mathcal{B}_i$ into sub-BFM groups $\{C_j\}_{j \in I^1}$;
(4) cross-tree feature constraints FC, where the features included in the feature expression belong to at least two distinct $\mathcal{M}_i$'s;
(5) a set of event constraints EC, where each conflict and causality in EC either involves at least one event in $\text{events}(\mathcal{S})$ and at least one event associated with a feature belonging to one of the $\mathcal{B}_i$'s, or involves events associated with features belonging to at least two distinct $\mathcal{B}_i$'s; and

(6) a set of product constraints PC that has an element (e, ϕ_e) for each $e \in \text{events}(\mathcal{S})$.

their *composition* is the BFM $\mathcal{B} = (f : \mathcal{S}_f, \{C_j\}_{j\in I^1}, \text{FC}_f, \text{EC}, \text{PC})$ over the set of events $\text{events}(\mathcal{S}_f) \cup (\bigcup_{i=1,\dots,m} \text{events}(\mathcal{B}_i))$ and the set of features $f \cup (\bigcup_{i=1,\dots,m} \text{features}(\mathcal{B}_i))$.

Definition 4.42 (FES composition). Given

(a) a behavioral feature $(f, (E_f, \#_f, \mapsto_f))$;
(b) a number of $m \geq 1$ FESs $\mathcal{E}_i = (\mathcal{M}_i, (E_i, \#_i, \mapsto_i), \lambda_i, \nu_i)$, for $i \in \{1, \dots, m\}$, with $\text{root}(\mathcal{M}_i) = f_i$ such that the sets $\{f\}, \text{features}(\mathcal{M}_1), \dots, \text{features}(\mathcal{M}_m)$ are all pairwise disjoint and likewise the sets $E_f, E_1, \dots, E_m$ are all pairwise disjoint;
(c) a set of group feature constrains which describes how to group the FMs $\mathcal{M}_i$ into sub-FM groups $\{\mathcal{G}_j\}_{j\in I^1}$;
(d) cross-tree feature constraints FC, where the features included in the feature expression belong to at least two distinct $\mathcal{M}_i$'s;
(e) a set of event constraints EC, where each conflict and causality in EC involves at least one event in E_f and one event in one of the E_i's, or involves events of at least two distinct E_i's; and
(f) a feature expression $\phi_e \in \text{FExp}(N)$, for each $e \in E_f$, where N is defined as below;

their *composition* is the FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ over the set of features $N = \{f\} \cup (\bigcup_{i=1\dots,m} \text{features}(\mathcal{M}_i))$ and the set of events $E = E_f \cup (\bigcup_{i=1,\dots,m} E_i)$, where

- $\mathcal{M} = (f, \{\mathcal{G}_j\}_{j\in I^1}, \text{FC})$;
- $\mathcal{S} = (E, \text{confl}(\text{EC}) \cup (\bigcup_{i=1,\dots,m} \#_i), \text{cause}(\text{EC}) \cup (\bigcup_{i=1,\dots,m} \mapsto_i))$;
- $\lambda : E \to N$ with $\lambda(e) = \begin{cases} \lambda_i(e) & \text{if } e \in E_i,\ i \in \{1, \dots, m\}, \\ f & \text{if } e \in E_f; \text{ and} \end{cases}$
- $\nu : E \to \text{FExp}(N)$ with $\nu(e) = \begin{cases} \nu_i(e) & \text{if } e \in E_i,\ i \in \{1, \dots, m\}, \\ \phi_e & \text{if } e \in E_f. \end{cases}$

The following definition shows how the notion of FM decomposition can be lifted to BFMs and FESs. The extensions to FMs with behavior are again highlighted with gray background for BFMs.

Definition 4.43 (BFM/FES decomposition). A BFM or FES $\mathcal{X}$, with $\mathcal{M} = \text{fm}(\mathcal{X})$, in which the root feature $f = \text{root}(\mathcal{M})$ has $m \geq 1$ child features $f_1, \dots, f_m$ can be decomposed into

(1) the behavioral feature $\text{behFt}_f(\mathcal{X})$ of the root feature f;
(2) its immediate sub-BFMs or sub-FESs $\mathcal{X}_i = \text{subTree}_{f_i}(\mathcal{X})$, for $i \in \{1, \dots, m\}$;
(3) the set $\text{gfc}_f(\mathcal{X})$ of group feature constraints of the root feature f, which describe how the $\mathcal{X}_i$'s are grouped into sub-BFM groups of $\mathcal{X}$ in case of a BFM, and how the $\text{fm}(\mathcal{X}_i)$'s are grouped into sub-FM groups of $\mathcal{M}$ in case of an FES;
(4) the cross-tree feature constraints $\text{FC}_f(\mathcal{X})$ of the root feature f;
(5) the inter-feature event constraints $\text{interFtEc}_f(\mathcal{X})$ of the root feature f; and
(6) the product constraints $\text{productC}_f(\mathcal{X})$ of the root feature f.

It is worth observing that, by design, decomposing a BFM or FES as described in Def. 4.43 can be reversed by re-composing it as described in Definitions 4.41 and 4.42, respectively.

Example 4.44 (Decomposing and composing the BFM of the cleaning robot SPL). Consider the BFM $\mathcal{B}_r = (r : (E_r, \varnothing, \varnothing), \{\textbf{opt}(\mathcal{B}_m), \textbf{mnd}(\mathcal{B}_c), \textbf{mnd}(\mathcal{B}_o)\}, \varnothing, \text{EC}_r, \varnothing)$ of the cleaning robot SPL as defined in Example 4.5. Then, using Def. 4.43, it can be decomposed into

- the behavioral feature $(r, (E_r, \varnothing, \varnothing))$;
- its immediate sub-BFMs $\mathcal{B}_m$, $\mathcal{B}_c$ and $\mathcal{B}_o$;

- the set $\mathrm{gfc}_r(\mathcal{B}_r) = \{\mathbf{opt}(m), \mathbf{mnd}(c), \mathbf{mnd}(o)\}$ of group feature constraints;
- the cross-tree feature constraints $\mathrm{FC}_r(\mathcal{B}_r) = tt$;
- the cross-tree event constraints $\mathrm{interFtEc}_r(\mathcal{B}_r) = \mathrm{EC}_r$; and
- the product constraints $\mathrm{productC}_r(\mathcal{B}_r) = \emptyset$.

Now, given the above, we can re-compose the BFM $\mathcal{B}_r$, by using Def. 4.41.

4.6 Trace Equivalence for BFMs, FESs and FTSs

Definition 4.45 (Trace equivalence for BFMs/FESs and FTSs). A BFM, FES or FTS $\mathcal{X}$ and a BFM, FES or FTS $\mathcal{X}'$ are *trace equivalent* if

- they have the same features, events and products; and
- for each of their products pr, it holds that $\mathrm{Tr}(\mathrm{behPr}(\mathcal{X}, pr)) = \mathrm{Tr}(\mathrm{behPr}(\mathcal{X}', pr))$.

Example 4.46 (Trace equivalence of FTS and BFM/FES for cleaning robot SPL). Consider the FTS $\mathcal{F}_r$ given in Example 2.21 (and depicted in Fig. 3) and the BFM $\mathcal{B}_r$ given in Example 4.5 (and depicted in Fig. 5) and its FES view $\mathcal{E}'_r$ given in Example 4.25 (its FM is depicted in Fig. 1 and its annotated BES is depicted in Fig. 6). Then $\mathcal{F}_r$ and $\mathcal{B}_r$ are trace-equivalent, as can be seen from the above mentioned examples and Examples 2.10, 2.25, 4.30 and 4.34.

5 FROM BFMs TO FTSs AND BACK BY PRESERVING TRACE EQUIVALENCE

Now that we have defined BFMs and their semantics, we show how to translate FTSs to BFMs and BFMs to FTSs in Sect. 5.5 and Sect. 5.6, respectively, and prove their trace equivalence.

As a first step towards these translations, we provide algorithms to translate TSs to BESs, BESs to TSs, FTSs to FESs, and FESs to FTSs in Sects. 5.1 to 5.4, and show their trace equivalence. However, for the algorithms from TS to BES and from FTS to FES, we restrict ourselves to *linear* TSs and to FTSs with underlying linear TSs, respectively. Intuitively, a linear TS is a TS $\mathcal{T}$ where every trace of $\mathcal{T}$ contains each event at most once. Assuming unique events simplifies the presentation but does not restrict expressiveness. Indeed, in the original presentation of event structures one can map events to action names and, hence, make fresh copies of repeated actions. This practice of introducing unique events (linear TSs in our case) also has reminiscence in reversible computation [45, 84]. In our evaluation we generalize this to include translations from a parallel composition of linear TSs to a BES and from a parallel composition of linear FTSs to a BFM, see Sect. 6. In the future, we plan to further investigate, which TSs we can translate into BESs while ensuring trace equivalence.

A schematic representation relating the different algorithms can be found in Fig. 8. It shows that

- the algorithm from a linear FTS to an FES uses the algorithm from a linear TS to a BES;
- the algorithm from an FES to an FTS uses the algorithm from a BES to a TS;
- the algorithms between (linear) FTSs and BFMs use the algorithms between (linear) FTSs and FESs;
- it is not possible to use the algorithms to go back and forth between FTSs and BFMs/FESs because the algorithm from a BFM/FES to an FTS produces a mostly non-linear FTS (and the same holds for going back and forth between TSs and BESs).

5.1 From Linear TSs to BESs by Preserving Traces

Let us first define formally what we mean by a linear TS.

Definition 5.1 (Linear TS). Let $\mathcal{T} = (S, E, s_0, \delta)$ be a TS. Then $\mathcal{T}$ is called *linear* if each event is associated with at most one transition and $\mathcal{T}$ does not contain cycles, i.e., if

- $\forall e \in E\ \exists!(s, s') \in S \times S.\ (s, e, s') \in \delta$; and
- $\forall s_1, s_2 \in S.\ \neg\,\mathrm{reachable}(s_1, s_2) \vee \neg\,\mathrm{reachable}(s_2, s_1)$.

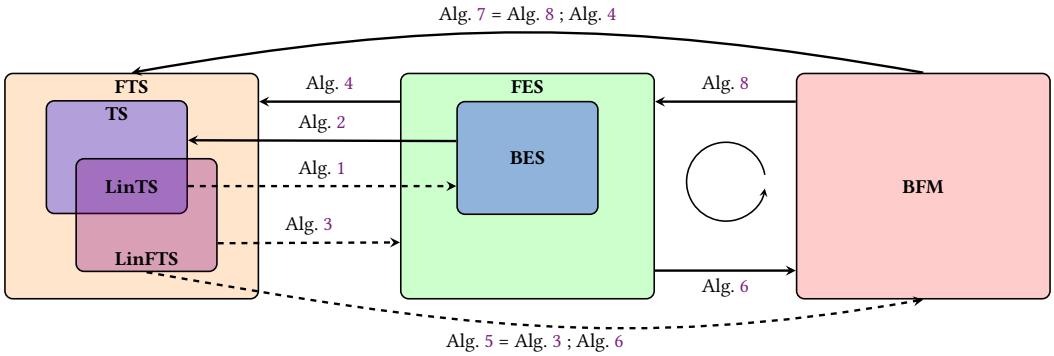

Fig. 8. A schematic representation relating the algorithms included in Sect. 5

Algorithm 1 LinTS2BES: from linear TS to BES

Input: linear TS (S, E, s_0, δ)
Output: BES $(E', \#, \mapsto)$

1: $E' := E$
2: $\# := \{ (e_1, e_2) \mid e_1, e_2 \in E \wedge e_1 \neq e_2 \wedge \neg \text{reachable}(e_1, e_2) \wedge \neg \text{reachable}(e_2, e_1) \}$
3: $\mapsto := \emptyset$
4: **for** $e \in E$ **do**
5: $\quad \mapsto := \{(X, e) \mid X = \{ e' \in E \mid \exists s_1, s_2, s_3 \in S. (s_1, e', s_2), (s_2, e, s_3) \in \delta \} \}$
6: $\quad \mapsto := \mapsto \setminus \{(\emptyset, e)\}$
7: **return** $(E', \#, \mapsto)$

Algorithm 1 defines a translation from a linear TS to a BES, as proven in Prop. 5.2. The algorithm works as follows. The events of the TS and its translation are the same, see Line 1. To define a conflict relation, we take all pairs of distinct events that cannot reach one another, see Line 2. The intuition behind this is that two events that are not reachable from one another in a TS cannot occur in the same trace of the TS, which is exactly what the conflict relation of a BES expresses. To define a causality relation, we include a pair (X, e) for every event e of the TS, where X contains all "incoming events to e", i.e., all events that happen directly before e in a trace of the TS, see Lines 3 to 5. Note that this step is only possible since we assume that the TS is linear. In this way, we inductively build up causalities such that the order of events in a trace of the TS is also preserved in a trace of its translation. Since this construction also includes a pair (X, e) in the causality relation for events e "exiting from the initial state", the causality relation includes $(\emptyset, e)$ for these e, which we need to exclude, see Line 6.

We first prove that Alg. 1 translates a linear TS to a BES.

Proposition 5.2 (LinTS2BES($\mathcal{T}$) is a BES). *Let $\mathcal{T}$ be a linear TS. Then* LinTS2BES($\mathcal{T}$) *is a BES.*

Proof. Let $\mathcal{T} = (S, E, s_0, \delta)$ and LinTS2BES($\mathcal{T}$) $= (E, \#, \mapsto)$. We have to show that:

(1) E is a finite set of events,
(2) $\# \subseteq E \times E$ is an irreflexive and symmetric relation, and
(3) $\mapsto \subseteq (\mathcal{P}(E) \setminus \{\emptyset\}) \times E$ is a relation, satisfying:
  (a) $\forall (X, e) \in \mapsto \forall e_1, e_2 \in X. e_1 \neq e_2 \Rightarrow e_1 \# e_2$; and
  (b) $\forall (X, e), (Y, e) \in \mapsto. X \neq Y \Rightarrow (X \not\subset Y \wedge Y \not\subset X)$.

Condition 1 holds by Line 1 of Alg. 1 and Condition 2 holds by Line 2 of Alg. 1, because we only include pairs of distinct elements in # (implying that # is irreflexive) that are not reachable from one another (implying that # is symmetric).

Ad 3: The causality relation $\mapsto$ is a subset of $\mathcal{P}(E) \times E$ because of Line 5 of Alg. 1, and it is a subset of $(\mathcal{P}(E) \setminus \{\varnothing\}) \times E$ because of Line 6 of Alg. 1. Now consider condition 3a. If $(X, e) \in \mapsto$, then all $e' \in X$ are immediate predecessors of e, i.e., if $(s, e, s') \in \delta$ and $X = \{e_1, \ldots, e_n\}$, then there exist $s_1, \ldots, s_n \in S$ with $(s_i, e_i, s) \in \delta$ for all $i \in \{1, \ldots, n\}$. Since $\mathcal{T}$ is linear, this also implies that $\neg$ reachable(e_i, e_j) for all $i, j \in \{1, \ldots, n\}$, with $i \neq j$, thus $(e_i, e_j) \in \#$ for all $i, j \in \{1, \ldots, n\}$, with $i \neq j$, by Line 2 of Alg. 1. Condition 3b holds because there is at most one $(X, e) \in \mapsto$ for every $e \in E$ by the construction of $\mapsto$ in Lines 4–6 of Alg. 1. □

Next we show that the TS and its translation to a BES via Alg. 1 are trace equivalent.

Proposition 5.3 (Trace equivalence $\mathcal{T}$ and LinTS2BES($\mathcal{T}$)). *Let $\mathcal{T} = (S, E, s_0, \delta)$ be a linear TS. Then $\mathcal{T}$ and* LinTS2BES($\mathcal{T}$) *are trace equivalent.*

Proof. By Def. 2.34, we have to show that $\mathcal{T}$ and LinTS2BES($\mathcal{T}$) have the same events and that $\mathrm{Tr}(\mathcal{T}) = \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$. By Line 1 of Alg. 1, they have the same events. We now show that $\mathrm{Tr}(\mathcal{T}) = \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$. To do so, let LinTS2BES($\mathcal{T}$) $= (E, \#, \mapsto)$.

"$\subseteq$" We first show that for all $tr \in \mathrm{Tr}(\mathcal{T})$, it holds that $tr \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$. We do this by induction on the length of tr for $tr \in \mathrm{Tr}(\mathcal{T})$.

Base cases: If tr is the empty trace, then $tr \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$ by the definition of traces of a BES (Def. 2.30). If $tr = \langle e \rangle$ for $e \in E$, then there exists $s \in S$ such that $(s_0, e, s) \in \delta$. By Line 5 of Alg. 1, there does not exist $X \subseteq E$ with $X \mapsto e$, which implies that $tr \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$.

Induction step: Assume that for all traces $tr \in \mathrm{Tr}(\mathcal{T})$ of length n, it holds that $tr \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$. Let $tr' \in \mathrm{Tr}(\mathcal{T})$ be a trace of length $n+1$ with $tr' = \langle e_1 \cdot \ldots \cdot e_n \cdot e \rangle$, for $e_i, e \in E, i \in \{1, \ldots, n+1\}$. By the definition of the traces of a TS, the sequence $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ is a trace of $\mathcal{T}$ of length n and there exist states $s, s' \in S$ with $s_0 \xrightarrow{tr} s \xrightarrow{e} s'$. The induction hypothesis implies that $tr \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$. We have to show that $tr' \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$.

By Def. 2.30 need to show (1) $(e_i, e_j) \notin \#$ for all $i \neq j \in \{1, \ldots, n+1\}$, (2) for all $i \in \{2, \ldots, n+1\}$ and for all $X \mapsto e_i$, it holds $\{e_1, \ldots, e_{i-1}\} \cap X \neq \varnothing$, and (3) there does not exist $X \subseteq E$ with $X \mapsto e_1$. Since tr, which is tr' without e, is a trace of LinTS2BES($\mathcal{T}$), it suffices to show that (1′) $(e_i, e) \notin \#$, for all $i \in \{1, \ldots, n\}$, and (2′) for all $X \subseteq E$ with $X \mapsto e$ it holds $\{e_1, \ldots, e_n\} \cap X \neq \varnothing$.

Ad (1′): Since tr' is a trace in $\mathcal{T}$, the event e is reachable from all e_i, $i \in \{1, \ldots, n\}$, i.e., reachable(e_i, e) for all $i \in \{1, \ldots, n\}$. Thus $(e_i, e) \notin \#$ by Line 2 of Alg. 1.

Ad (2′): Let $X \subset E$ with $(X, e) \in \mapsto$. By Line 5 of Alg. 1, it holds that $X = \{\, e' \in E \mid \exists s_1 \in S.\, (s_1, e', s) \in \delta \,\}$ (using the linearity of $\mathcal{T}$ and the fact that $(s, e, s') \in \delta$). Thus, by the definition of tr', it holds that $e_n \in X$, implying $e_n \in X \cap \{e_1, \ldots, e_n\}$.

"$\supseteq$" Now we show that for all $tr \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$, it holds that $tr \in \mathrm{Tr}(\mathcal{T})$. We do this by induction on the length of tr for $tr \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$.

Base case: If tr is the empty trace, then $tr \in \mathrm{Tr}(\mathcal{T})$. If $tr = \langle e \rangle$ for $e \in E$, then there does not exist $X \subseteq E$ with $X \mapsto e$ by the definition of traces of a BES (Def. 2.30). Thus, Lines 5 and 6 of Alg. 1 imply that there do not exist $s_1, s_2, s_3 \in S$ and $e' \in E$ with $(s_1, e', s_2), (s_2, e, s_3) \in \delta$. Since we assume that all events in $\mathcal{T}$ are reachable from the initial state (Rem. 2.15), it has to hold that there exists $s \in S$ with $(s_0, e, s) \in \delta$, which implies $tr \in \mathrm{Tr}(\mathcal{T})$.

Induction step: Assume that for all traces $tr \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$ of length n, it holds that $tr \in \mathrm{Tr}(\mathcal{T})$. Let $tr' \in \mathrm{Tr}(\text{LinTS2BES}(\mathcal{T}))$ be a trace of length $n+1$ with $tr' = \langle e_1 \cdot \ldots \cdot e_n \cdot e \rangle$ for $e_i, e \in E, i \in \{1, \ldots, n\}$. By the definition of traces of a BES, the sequence $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ is a trace of LinTS2BES($\mathcal{T}$) of length n. From the induction hypothesis we obtain that $tr \in \mathrm{Tr}(\mathcal{T})$, i.e., there

Algorithm 2 BES2TS: from BES to TS

Input: BES $\mathcal{S} = (E, \#, \mapsto)$
Output: TS (S, E', s_0, δ)
1: $S := \text{Config}(\mathcal{S}), E' := E, s_0 := \emptyset, \delta := \emptyset$
2: **for** $s, s' \in S, e \in E$ **such that** $s' \setminus s = \{e\}$ **do**
3: $\quad \delta := \delta \cup \{(s, e, s')\}$
4: **return** (S, E', s_0, δ)

exists $s \in S$ with $s_o \xrightarrow{tr} s$. We have to show that $tr' \in \text{Tr}(\mathcal{T})$, i.e., there exists a state $s' \in S$ with $(s, e, s') \in \delta$.

We first show that there exists $X \subseteq E$ with $(X, e) \in \mapsto$. Assume (towards contradiction) that there does not exist $X \subseteq E$ with $(X, e) \in \mapsto$. Then there does not exist $e' \in E$ and $s_1, s_2, s_3 \in S$ such that $(s_1, e', s_2), (s_2, e, s_3) \in \delta$ by Line 5of Alg. 1. In particular, it must hold that there exists $\hat{s} \in S$ with $(s_0, e, \hat{s}) \in \delta$. Since tr' is a trace of LinTS2BES($\mathcal{T}$), none of its elements are in conflict, i.e., by Line 2 of Alg. 1 it holds that reachable(e_i, e) or reachable(e, e_i), for all $i \in \{1, \ldots, n\}$. From the facts that $(s_0, e, \hat{s}) \in \delta$ and $\mathcal{T}$ is linear, we obtain that reachable(e, e_i) for all $i \in \{1, \ldots, n\}$. In particular, reachable(e, e_1). However, tr is a trace in $\mathcal{T}$, so there exists $s_1 \in S$ with $(s_0, e_1, s_1) \in \delta$, which is a contradiction to reachable(e, e_1) by the linearity of $\mathcal{T}$. Thus, there exists $X \subseteq E$ with $(X, e) \in \mapsto$.

Let $X \subseteq E$ with $(X, e) \in \mapsto$. Then there exists $k \in \{1, \ldots, n\}$ such that $e_k \in X \cap \{e_1, \ldots, e_n\}$, because tr' is a trace of LinTS2BES($\mathcal{T}$). That is, by the definition of $\mapsto$ in Line 5 of Alg. 1, there exist $s_1, s_2, s_3 \in S$ such that $(s_1, e_k, s_2), (s_2, e, s_3) \in \delta$. We have to show $k = n$, since this implies the lemma.

Since tr' is a trace of LinTS2BES($\mathcal{T}$), its events are not in conflict, in particular $(e_i, e) \notin \#$, for all $i \in \{1, \ldots, n\}$. By Line 2 of Alg. 1 this implies that either reachable(e_i, e) or reachable(e, e_i), for all $i \in \{1, \ldots, n\}$. We want to show reachable(e_i, e), for all $i \in \{1, \ldots, n\}$. Since tr is a trace in $\mathcal{T}$, this implies $k = n$ above, and thus implies the lemma.

Assume (towards contradiction) that there exists an $i \in \{1, \ldots, n\}$ with reachable(e, e_i), and let $\ell \in \{1, \ldots, n\}$ be the smallest integer with this property, i.e., for all $j < \ell$ it holds that reachable(e_j, e). Case 1: $\ell = 1$. Since tr is a trace in $\mathcal{T}$, there exists $s_1 \in S$ with $(s_0, e_1, s_1) \in \delta$. Since $\mathcal{T}$ is linear, this is a contradiction to reachable(e, e_1). Case 2: $\ell > 1$. Since tr is a trace in $\mathcal{T}$, there exist $s_{\ell-1}, s_\ell, s_{\ell+1} \in S$ with $(s_{\ell-1}, e_{\ell-1}, s_\ell), (s_\ell, e_\ell, s_{\ell+1}) \in \delta$. By the choice of ℓ, it must hold that reachable$(e_{\ell-1}, e)$ and reachable(e, e_ℓ). Thus, there exist $\hat{s}, \tilde{s} \in S$ with $(\hat{s}, e, \tilde{s}) \in \delta$ and reachable$(s_\ell, \hat{s})$ and reachable$(\tilde{s}, s_\ell)$, resulting in a loop in $\mathcal{T}$, which is a contradiction to $\mathcal{T}$ being linear. Therefore, it must hold, for all $i \in \{1, \ldots, n\}$, that reachable(e_i, e), implying that $k = n$, which concludes the proof. □

5.2 From BESs to TSs by Preserving Traces

Algorithm 2 shows how to translate a BES into a (mostly non-linear) TS, as proven in Prop. 5.4. It is not possible to translate it into a linear TS in general, because the structure of BESs using causalities allows many interleavings of events. The algorithm works as follows. The events of the BES and its translation are the same, see Line 1. The states of the translated BES are defined as the BES's configurations, since the configurations of a BES correspond to its traces, i.e., they contain all events of a trace, see Line 1. By defining the states of the BES in this way, we can define the initial state as the empty configuration, and the transition relation by including a transition (s, e, s') if s' can be obtained from s by adding e, see Lines 1 to 3. Thereby, the configurations (which are the states of the translated BES) determine the transition relation of the translated BES, ensuring that the BES and its translation are trace equivalent.

We first prove that the translation of a BES via Alg. 2 is a TS.

Proposition 5.4 (BES2TS($\mathcal{S}$) is a TS). *Let $\mathcal{S}$ be a BES. Then* BES2TS($\mathcal{S}$) *is a TS.*

Proof. For a BES $\mathcal{S}$, its translation BES2TS($\mathcal{S}$) is by construction a TS. □

Now we prove that a BES and its translation via Alg. 2 are trace equivalent.

Proposition 5.5 (Trace equivalence of $\mathcal{S}$ and BES2TS($\mathcal{S}$)). *Let $\mathcal{S} = (E, \#, \mapsto)$ be a BES. Then* BES2TS($\mathcal{S}$) *and $\mathcal{S}$ are trace equivalent.*

Proof. By Def. 2.34, we have to show that $\mathcal{S}$ and BES2TS($\mathcal{S}$) have the same events, and that $\mathrm{Tr}(\mathcal{S}) = \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$. By Line 2 of Alg. 2, they have the same events. We now show that $\mathrm{Tr}(\mathcal{S}) = \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$. Let $\mathrm{BES2TS}(\mathcal{S}) = (S, E, s_0, \delta)$.

"$\subseteq$" We first show that for all $tr \in \mathrm{Tr}(\mathcal{S})$, it holds that $tr \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$. We do this by induction on the length of tr, for $tr \in \mathrm{Tr}(\mathcal{S})$.

Base case: If tr is the empty trace, then $tr \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$ because $s_0 = \emptyset$. If $tr = \langle e \rangle$ for $e \in E$, then $\{e\}$ is a configuration of $\mathcal{S}$ by Lemma 2.31, so $tr \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$ by Lines 2 and 3 of Alg. 2.

Induction step: Assume that for all traces $tr \in \mathrm{Tr}(\mathcal{S})$ of length n, it holds that $tr \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$. Let $tr' \in \mathrm{Tr}(\mathcal{S})$ be a trace of length $n+1$ with $tr' = \langle e_1 \cdot \ldots \cdot e_n \cdot e \rangle$, for $e_i, e \in E$, with $i \in \{1, \ldots, n\}$. By the definition of traces of a BES, $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ is a trace in $\mathcal{S}$. By the induction hypothesis, tr is also a trace in BES2TS($\mathcal{S}$), i.e., there exists $s \in S$ with $s_0 \xrightarrow{tr} s$. We have to show that $tr' \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$, i.e., that there exists $s' \in S$ with $(s, e, s') \in \delta$.

By Lemma 2.31, the sets $C = \{e_1, \ldots, e_n\}$ and $C' = C \cup \{e\}$ are configurations of $\mathcal{S}$. Line 1 of Alg. 2 implies that there exist states $s_C, s_{C'} \in S$ such that $s_C = C$ and $s_{C'} = C'$. Because $C' = C \cup \{e\}$, it follows from Lines 2–3 of Alg. 2 that $(s_C, e, s_{C'}) \in \delta$. Furthermore, since $C = \mathrm{events}(tr)$, where tr is a trace from s_0 to s, Lemma 5.6 implies that $s = C$. Thus, $(s = s_C, e, s_{C'}) \in \delta$, which implies that $tr' \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$.

"$\supseteq$" Now we show that for all $tr \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$, it holds that $tr \in \mathrm{Tr}(\mathcal{S})$. We do this by induction on the length of tr for $tr \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$.

Base case: If tr is the empty trace, then $tr \in \mathrm{Tr}(\mathcal{S})$ by definition (Def. 2.30). If $tr = \langle e \rangle$, then there exists a state $s \in S$ with $(s_0, e, s) \in \delta$. By Lines 2 and 3 of Alg. 2 it holds that $\{e\} = s \setminus s_0 = s \setminus \emptyset = s$, which implies by Line 1 that $\{e\}$ is a configuration of $\mathcal{S}$ which is a trace of $\mathcal{S}$ by the definition of traces (Def. 2.30).

Induction step: Assume that for all traces $tr \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$ of length n, it holds that $tr \in \mathrm{Tr}(\mathcal{S})$. Let $tr' \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$ be a trace of length $n+1$ with $tr' = \langle e_1 \cdot \ldots \cdot e_n \cdot e \rangle$ for $e_i, e \in E, i \in \{1, \ldots, n\}$, and let $s \in S$ with $s_0 \xrightarrow{tr} s$. By the definition of traces of a TS, the sequence $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ is a trace of BES2TS($\mathcal{S}$) of length n. From the induction hypothesis we obtain that $tr \in \mathrm{Tr}(\mathcal{S})$. We have to show that $tr' \in \mathrm{Tr}(\mathcal{S})$.

Since tr is a trace in $\mathcal{S}$, it is enough to show (1′) $(e_i, e) \notin \#$ for all $i \in \{1, \ldots, n\}$ and (2′) $\forall X \subseteq E$ with $X \mapsto e$ it holds $X \cap \{e_1, \ldots, e_n\} \neq \emptyset$ (see Def. 2.30). By Lemma 5.6 it holds that $s = \mathrm{events}(tr')$ and by Line 1 of Alg. 2 it holds that $s \in \mathrm{Config}(\mathcal{S})$. Thus, the definition of a configuration of a BES (Def. 2.28) implies (1′) and (2′).

□

In the proof of Prop. 5.5, we need a lemma that determines, for a state of a BES translated via Alg. 2, the corresponding configuration of the BES.

Lemma 5.6 (On Alg. 2). *Let $\mathcal{S}$ be a BES and* BES2TS($\mathcal{S}$) $= (S, E, s_0, \delta)$ *its translation via Alg. 2. Let $s \in S$ and $tr \in \mathrm{Tr}(\mathrm{BES2TS}(\mathcal{S}))$ with $s_0 \xrightarrow{tr} s$. Then $s = \mathrm{events}(tr)$.*

Proof. We do the proof by induction on the length of tr.

Base case: tr is the empty trace. Then $\mathrm{events}(tr) = \emptyset = s_0$.

Algorithm 3 LinFTS2FES: from linear FTS to FES

Input: linear FTS $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$ where $\mathcal{T} = (S, E, s_0, \delta)$
Output: FES $(\mathcal{M}', \mathcal{S}, \lambda, \nu)$ with $\mathcal{S} = (E', \#, \mapsto)$
1: $\mathcal{M}' := \mathcal{M}$
2: $\mathcal{S} :=$ LinTS2BES($\mathcal{T}$)
3: **for** $e \in E$ **do**
4: $\quad \lambda(e) :=$ root($\mathcal{M}$)
5: $\quad \nu(e) := \mu((s, e, s'))$ for $(s, e, s') \in \delta$ ▷ (s, e, s') is unique because the FTS is linear
6: **return** $(\mathcal{M}', \mathcal{S}, \lambda, \nu)$

Induction step: Assume that for all traces $tr \in \text{BES2TS}(\mathcal{S})$ of length n the lemma holds. Let $tr' \in \text{Tr}(\text{BES2TS}(\mathcal{S}))$ be a trace of length $n+1$ with $tr' = \langle e_1 \cdot \ldots \cdot e_n \cdot e \rangle$, for $e_i, e \in E$, with $i \in \{1, \ldots, n\}$, and $s_0 \xrightarrow{tr'} s'$, for $s' \in S$. We have to show that $s' = \text{events}(tr')$.

By the definition of traces of a TS, the sequence $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ is a trace of BES2TS($\mathcal{S}$) of length n and there exists $s \in S$ with $s_0 \xrightarrow{tr} s \xrightarrow{e} s'$. By the induction hypothesis, it holds that $s = \text{events}(tr)$. Since $(s, e, s') \in \delta$, Lines 2–3 of Alg. 2 imply that $s' = s \cup \{e\}$. Thus, $s' = \text{events}(tr) \cup \{e\} = \{e_1, \ldots, e_n, e\} = \text{events}(tr')$. □

5.3 From Linear FTSs to FESs by Preserving Traces

Now that we have defined an algorithm for translating a linear TS to a BES, we can define an algorithm from linear FTSs to FESs based on the algorithm from linear TSs to BES. First, we define what a linear FTS is.

Definition 5.7 (Linear FTS). An FTS $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$ is called *linear* if $\mathcal{T}$ is a linear TS.

Algorithm 3 defines a translation from a linear FTS to an FES, as proven in Prop. 5.8. The algorithm works as follows. The feature model of the translated FTS is the same as the one of the FTS, see Line 1, and the BES is obtained using Alg. 1 with the underlying TS of the FTS, see Line 2. Since an FTS does not contain information on which behavior is associated with which feature, all events are associated with the root of the FM, see Line 4. Lastly, the product knowledge of an event e is defined as the feature expression associated with the transition labeled by e, see Line 5. Note that we can define the product knowledge as we do here only because we consider linear FTSs.

We first prove that Alg. 3 translates a linear FTS to an FES.

Proposition 5.8 (LinFTS2FES($\mathcal{F}$) is an FES). *Let $\mathcal{F}$ be a linear FTS. Then* LinFTS2FES($\mathcal{F}$) *is a FES.*

Proof. For a linear FTS $\mathcal{F}$, its translation LinFTS2FES($\mathcal{F}$) is by construction an FES. □

Now we can prove that a linear FTS and its translation via Alg. 3 are trace equivalent.

Proposition 5.9 (Trace equivalence of $\mathcal{F}$ and LinFTS2FES($\mathcal{F}$)). *Let $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$ be a linear FTS over the set of features N. Then* LinFTS2FES($\mathcal{F}$) *and $\mathcal{F}$ are trace equivalent.*

Proof. For $\mathcal{F}$ and LinFTS2FES($\mathcal{F}$) to be trace equivalent, they have to have the same features, events and products, and for every product pr it has to hold that $\text{Tr}(\mathcal{F}, pr) = \text{Tr}(\text{LinFTS2FES}(\mathcal{F}), pr)$. By construction, $\mathcal{F}$ and LinFTS2FES($\mathcal{F}$) have the same features, events and products.

Let $pr \in \text{products}(\mathcal{F}) = \text{products}(\text{LinFTS2FES}(\mathcal{F}))$, let $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$ with $\mathcal{T} = (S, E, s_0, \delta)$, let $\text{behPr}(\mathcal{F}, pr) = (S', E', s_0', \delta')$, let $\text{LinFTS2FES}(\mathcal{F}) = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ with $\mathcal{S} = (E, \#, \mapsto)$, and let $\text{behPr}(\text{LinFTS2FES}(\mathcal{F}), pr) = (E', \#', \mapsto')$ (the events of the behavioral products of $\mathcal{F}$ and

LinFTS2FES($\mathcal{F}$) for pr are the same by Lemma 5.10). It remains to show that the sets of traces of $\mathcal{F}$ for pr and of LinFTS2FES($\mathcal{F}$) for pr are the same, i.e, $\mathrm{Tr}(\mathcal{F}, pr) = \mathrm{Tr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$. This proof is similar to the one of Prop. 5.3.

"⊆" We first show that for all $tr \in \mathrm{Tr}(\mathcal{F}, pr)$, it holds that $tr \in \mathrm{Tr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$, i.e., that $tr \in \mathrm{Tr}(\mathrm{behPr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr))$. We do this by induction on the length of tr for $tr \in \mathrm{Tr}(\mathcal{F}, pr)$.

Base case: If tr is the empty trace, then $tr \in \mathrm{Tr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$ by the definition of traces of a BES (Def. 2.30). If $tr = \langle e \rangle$ for $e \in E$, then Conditions (1)–(3) of being a trace of FES2FTS($\mathcal{F}$) for pr (see Def. 4.33) follow in the same way as in the proof of Prop. 5.3. Condition (0) follows from Lemma 5.10 and the definition of a behavioral product (Def. 4.29) because e is an event of $\mathrm{behPr}(\mathcal{F}, pr)$.

Induction step: Assume that for all traces $tr \in \mathrm{Tr}(\mathcal{F}, pr)$ of length n, it holds that tr is also a trace in LinFTS2FES($\mathcal{F}$) for pr. Let $tr' \in \mathrm{Tr}(\mathcal{F}, pr)$ be a trace of length $n+1$ with $tr' = \langle e_1 \cdot \ldots \cdot e_n \cdot e \rangle$, for $e_i, e \in E$, with $i \in \{1, \ldots, n\}$. For all transitions t in tr', we have $\mathrm{eval}_{pr}(\mu(t)) = tt$ by the definition of the traces of an FTS for a product (Def. 2.24), i.e., $pr \models \mu(t)$. By the definition of traces of an FTS for a product, $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ is a trace of $\mathcal{F}$ for pr of length n and there exist states $s, s' \in S$ with $s_0 \xrightarrow{tr} s \xrightarrow{e} s'$. From the induction hypothesis we obtain that $tr \in \mathrm{Tr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$. We have to show that $tr' \in \mathrm{Tr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$.

Since tr is a trace of LinFTS2FES($\mathcal{F}$) for pr, it is enough to show that (0′) $\lambda(e) \in pr$ and $pr \models \nu(e)$, (1′) $(e, e_i) \notin \#'$, for all $i \in \{1, \ldots, n\}$, and (2′) for all $X \subseteq E'$, it holds that $X \mapsto' e$ implies $X \cap \{e_1, \ldots, e_n\} \neq \emptyset$, see Def. 4.33. (The proofs of (1′) and (2′) are very similar to the proofs of (1′) and (2′) in Prop. 5.3.)

Ad (0′): Since e is an event of $\mathrm{behPr}(\mathcal{F}, pr)$, it follows from Lemma 5.10 that it is also an event of $\mathrm{behPr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$, implying $\lambda(e) \in pr$ and $pr \models \nu(e)$.

Ad (1′): Since tr' is a trace of $\mathcal{F}$ for pr, the event e is reachable from all e_i, with $i \in \{1, \ldots, n\}$, i.e., $\mathrm{reachable}(e_i, e)$, for all $i \in \{1, \ldots, n\}$. Thus, $(e_i, e) \notin \#$ by Line 2 of Alg. 1, which implies that $(e_i, e) \notin \#'$, because $\#' = \# \cap (E' \times E')$.

Ad (2′): We need to show that for all $(X, e) \in \mapsto'$, there exists an $i \in \{1, \ldots, n\}$ such that $e_i \in X \cap \{e_1, \ldots, e_n\}$. Let $(X, e) \in \mapsto'$. By Line 5 of Alg. 1 and by the definition of the behavioral product (Def. 4.29), it holds that $X = \{\, e' \in E' \mid \exists s_1 \in S.\ (s_1, e', s) \in \delta \,\}$ (using the linearity of $\mathcal{F}$ and the fact that $(s, e, s') \in \delta$). Thus, by the definition of tr', it holds that $e_n \in X$, which implies that $e_n \in X \cap \{e_1, \ldots, e_n\}$.

"⊇" Now we show that for all $tr \in \mathrm{Tr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$, it holds that $tr \in \mathrm{Tr}(\mathcal{F}, pr) = \mathrm{Tr}(\mathrm{behPr}(\mathcal{F}, pr))$. We do this by induction on the length of tr for $tr \in \mathrm{Tr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$.

Base case: If tr is the empty trace, then $tr \in \mathrm{Tr}(\mathcal{F}, pr)$ by the definition of traces of an FTS for a product (Def. 2.24). If $tr = \langle e \rangle$ for $e \in E$, then it follows in the same way as in the proof of Prop. 5.3 that there exists $s \in S$ with $(s_0, e, s) \in \delta$. By Lemma 4.35, the set $\mathrm{events}(tr) = \{e\}$ is a configuration of LinFTS2FES($\mathcal{F}$) for pr, which implies that $\mathrm{eval}_{pr}(\mu((s_0, e, s))) = tt$, so tr is a trace of $\mathcal{F}$ for pr.

Induction step: Assume that for all traces $tr \in \mathrm{Tr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$ of length n, it holds that $tr \in \mathrm{Tr}(\mathcal{F}, pr)$. Let $tr' \in \mathrm{Tr}(\mathrm{LinFTS2FES}(\mathcal{F}), pr)$ be a trace of length $n+1$ with $tr' = \langle e_1 \cdot \ldots \cdot e_n \cdot e \rangle$ for $e_i, e \in E$ with $i \in \{1, \ldots, n\}$. By the definition of the traces of an FES for a product, the sequence $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ is a trace of LinFTS2FES($\mathcal{F}$) for pr, and by the induction hypothesis it holds that $tr \in \mathrm{Tr}(\mathcal{F}, pr)$. That is, there exists a state $s \in S$ with $s_0 \xrightarrow{tr} s$ and $\mathrm{eval}_{pr}(\mu(t)) = tt$, for all transitions t in tr. We have to show that $tr' \in \mathrm{Tr}(\mathcal{F}, pr)$. Since tr is a trace of $\mathcal{F}$ for pr, it is enough to show that there exists a state $s' \in S$ with (1) $(s, e, s') \in \delta$ and (2) $\mathrm{eval}_{pr}(\mu((s, e, s')) = tt$.

Ad (1): The proof works in the same way as in Prop. 5.3, using traces in $\mathcal{F}$ for pr and configurations of LinFTS2FES($\mathcal{F}$) for pr.

Algorithm 4 FES2FTS: from FES to FTS

Input: FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ with $\mathcal{S} = (E, \#, \mapsto)$
Output: FTS $\mathcal{F} = (\mathcal{M}', \mathcal{T}, \mu)$ with $\mathcal{T} = (S, E', s_0, \delta)$
1: $\mathcal{M}' := \mathcal{M}, S := \text{Config}(\mathcal{E}), E' := E, s_0 := \emptyset, \delta := \emptyset$
2: **for** $s, s' \in S, e \in E$ **such that** $s' \setminus s = \{e\}$ **do**
3: $\quad \delta := \delta \cup \{(s, e, s')\}$
4: $\quad \mu((s, e, s')) := \phi_s \wedge \phi_{s'}$
5: **return** $(\mathcal{M}, \mathcal{T}, \mu)$

We first show that there exists $X \subseteq E'$ with $(X, e) \in \mapsto'$. Assume (towards contradiction) that there exists no $X \subseteq E'$ with $(X, e) \in \mapsto'$. Then either (*i*) there exists no $X \subseteq E$ with $(X, e) \in \mapsto$ (in which case we proceed in the same way as in the proof of Prop. 5.3) or (*ii*) for all $X \subseteq E$ with $(X, e) \in \mapsto$, it holds that $X \cap E' = \emptyset$. In case (*ii*), there does not exist $e' \in E'$ and $s_1, s_2, s_3 \in S$ with $(s_1, e', s_2), (s_2, e, s_3) \in \delta$ by Line 5 of Alg. 1 (called in Line 2 of Alg. 4). However, since e is an event of behPr(LinFTS2FES($\mathcal{F}$), pr), it is also an event of behPr($\mathcal{F}, pr$), by Lemma 5.10. Thus, by the definition of behavioral products of an FTS (Def. 2.22), the event e must be reachable in behPr($\mathcal{F}, pr$). This is a contradiction, because there exist no $e' \in E' = \text{events}(\text{behPr}(\mathcal{F}, pr))$ and $s_1, s_2, s_3 \in S$ with $(s_1, e', s_2), (s_2, e, s_3) \in \delta$. Thus, it must hold that there exists $X \subseteq E'$ with $(X, e) \in \mapsto'$.

Let $X \subseteq E'$ with $(X, e) \in \mapsto'$. Then there exists $k \in \{1, \ldots, n\}$ such that $e_k \in X \cap \{e_1, \ldots, e_n\}$, because tr' is a trace of LinFTS2FES($\mathcal{F}$). That is, by the definition of $\mapsto$ in Line 5 of Alg. 1, there exist $s_1, s_2, s_3 \in S$ such that $(s_1, e_k, s_2), (s_2, e, s_3) \in \delta$. We have to show $k = n$.

Since tr' is a trace of LinFTS2FES($\mathcal{F}$), its events are not in conflict; in particular, $(e_i, e) \notin \#' \subseteq \#$, for all $i \in \{1, \ldots, n\}$. By Line 2 of Alg. 1, this implies that either reachable(e_i, e) or reachable(e, e_i), for all $i \in \{1, \ldots, n\}$. It is possible to show that reachable(e_i, e) holds for all $i \in \{1, \ldots, n\}$ in the same way as in the proof of Prop. 5.3. Since tr is a trace in $\mathcal{F}$ for pr, this implies $k = n$ above, which finishes the proof of (1).

Ad (2): By the definition of traces of an FES for a product (Def. 4.33), it holds $\lambda(e) \in pr$ and $pr \models \nu(e)$ because tr' is a trace of LinFTS2FES($\mathcal{F}$) for pr and $e \in \text{events}(tr')$. Since $\nu(e) = \mu((s, e, s'))$ by Line 5 of Alg. 3, it holds that $\text{eval}_{pr}(\mu((s, e, s'))) = tt$. □

In the proof of Alg. 3, we need the following lemma which states that the behavioral products of a linear FTS and its translation via Alg. 3 have the same events.

Lemma 5.10 (LinFTS2FES($\mathcal{F}$) preserves events of behavioral products). *Let $\mathcal{F}$ be an FTS, let* LinFTS2FES($\mathcal{F}$) *be its translation via Alg. 3, and let $pr \in \text{products}(\mathcal{F}) = \text{products}(\text{LinFTS2FES}(\mathcal{F}))$ be a product. Then* behPr($\mathcal{F}, pr$) *and* behPr(LinFTS2FES($\mathcal{F}$), pr) *have the same events.*

Proof. Let $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$ with $\mathcal{T} = (S, E, s_0, \delta)$ and let pr be a product of $\mathcal{F}$.

"⊆" Let $e \in \text{events}(\text{behPr}(\mathcal{F}, pr))$, and let $s, s' \in S$ with $(s, e, s') \in \delta$. Since e is an event of behPr($\mathcal{F}, pr$), it holds that $\text{eval}_{pr}(\mu((s, e, s'))) = tt$. We have to show that $\lambda(e) \in pr$ and $pr \models \nu(e)$.

By Lines 4 and 5 of Alg. 3, it holds that $\lambda(e) = \text{root}(\mathcal{M})$ and $\nu(e) = \mu((s, e, s'))$, respectively. Above, we concluded that $\text{eval}_{pr}(\mu((s, e, s'))) = tt$, i.e., $pr \models \mu((s, e, s')) = \nu(e)$. By Rem. 2.9, the root of $\mathcal{M}$ is a feature in pr, so $\lambda(e) \in pr$.

"⊇" Let $e \in \text{events}(\text{behPr}(\text{LinFTS2FES}(\mathcal{F}), pr))$. Then $\lambda(e) \in pr$ and $pr \models \nu(e)$. We have to show that $\text{eval}_{pr}(\mu((s, e, s'))) = tt$ for $(s, e, s') \in \delta$. Line 5 of Alg. 3 implies that $\nu(e) = \mu((s, e, s'))$ for $(s, e, s') \in \delta$. Thus, $pr \models \nu(e) = \mu((s, e, s'))$ and therefore $\text{eval}_{pr}(\mu((s, e, s'))) = tt$. □

5.4 From FESs to FTSs by Preserving Traces

Algorithm 4 defines a translation from an FES to a (mostly non-linear) FTS, as proven in Prop. 5.11. The algorithm works as follows. The feature model and the events of the translated FES are the same as the ones of the FES, see Line 1. As in Alg. 2, we define the set of states of the translated FES as the set of configurations, where we consider the configurations of the FES here, not the configurations of the underlying BES, see Line 1. Intuitively, this ensures that the traces of the FES are preserved in the FTS. To understand why we need to consider the configurations of the FES, recall that the traces of an FES are the traces of its behavioral products, which correspond to the configurations of its behavioral products. Every configuration of an FES is a configuration of a BES, obtained by restricting the BES of the FES to a set of events. To obtain an FTS that is trace equivalent to the FES, we need to ensure that all configurations of the FES's behavioral products, i.e., all configurations of the FES, are included. The transition relation is constructed in the same way as in Alg. 2, see Lines 1 to 3, and the feature expression associated with a transition is obtained from the feature expressions associated with the configurations from the source and target states (see Def. 4.36) of the transition, see Line 4.

We first prove that Alg. 4 translates an FES to an FTS.

Proposition 5.11 (FES2FTS($\mathcal{E}$) is an FTS). *Let $\mathcal{E}$ be an FES. Then* FES2FTS($\mathcal{E}$) *is an FTS.*

Proof. Let $\mathcal{E}$ be an FES over the features N and let FES2FTS($\mathcal{E}$) $= (\mathcal{M}, \mathcal{T}, \mu)$, with $\mathcal{T} = (S, E, s_0, \delta)$. We have to show that:

- $\mathcal{M}$ is an FM over N,
- $\mathcal{T} = (S, E, s_0, \delta)$ is a TS over E,
- $\mu : S \times E \times S \to \mathrm{FExp}(N)$ is a mapping.

By Line 1 of Alg. 4, $\mathcal{M}$ is an FM over N. The only difference in Lines 1–3 of Alg. 4 compared to Alg. 2 is that the set of states is defined as the set of configurations of the FES instead of the configurations of the underlying BES.

Thus, we can proceed as in the proof of Prop. 5.5 to see that $\mathcal{T}$ is a TS over E. Since S is defined as the set of configurations of $\mathcal{E}$, we know that $\mu : S \times E \times S \to \mathrm{FExp}(N)$ is a mapping because $\phi_s, \phi_{s'} \in \mathrm{FExp}(N)$, for $s, s' \in S$, see Def. 4.36. □

Now we can prove that an FES and its translation via Alg. 4 are trace equivalent.

Proposition 5.12 (Trace equivalence of $\mathcal{E}$ and FES2FTS($\mathcal{E}$)). *Let $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ be an FES. Then* FES2FTS($\mathcal{E}$) *and $\mathcal{E}$ are trace equivalent.*

Proof. For $\mathcal{E}$ and FES2FTS($\mathcal{E}$) to be trace equivalent, they have to have the same features, events and products, and for every product pr it has to hold that $\mathrm{Tr}(\mathcal{E}, pr) = \mathrm{Tr}(\mathrm{FES2FTS}(\mathcal{E}), pr)$. By construction, $\mathcal{E}$ and FES2FTS($\mathcal{E}$) have the same features, events and products. Let $pr \in \mathrm{products}(\mathcal{E}) = \mathrm{products}(\mathrm{FES2FTS}(\mathcal{E}))$, let FES2FTS($\mathcal{E}$) $= (\mathcal{M}, \mathcal{T}, \mu)$ with $\mathcal{T} = (S, E, s_0, \delta)$, let $\mathrm{behPr}(\mathrm{FES2FTS}(\mathcal{E}), pr) = (S', E', s_0', \delta')$, and let $\mathrm{behPr}(\mathcal{E}, pr) = (E', \#', \mapsto')$ (the events of the behavioral products of $\mathcal{E}$ and FES2FTS($\mathcal{E}$) for pr are the same by Lemma 5.14). We have to show that $\mathrm{Tr}(\mathcal{E}, pr) = \mathrm{Tr}(\mathrm{FES2FTS}(\mathcal{E}), pr)$. The proof uses the proof of Prop. 5.5.

"⊆" We first show that for all $tr \in \mathrm{Tr}(\mathcal{E}, pr)$, it holds that $tr \in \mathrm{Tr}(\mathrm{FES2FTS}(\mathcal{E}), pr)$. We do this by induction on the length of tr for $tr \in \mathrm{Tr}(\mathcal{E}, pr)$.

Base case: If tr is the empty trace, then $tr \in \mathrm{Tr}(\mathrm{FES2FTS}(\mathcal{E}), pr)$, because the empty trace is a trace for every product of an FTS. If $tr = \langle e \rangle$ for $e \in E$, then $\{e\}$ is a configuration of $\mathrm{behPr}(\mathcal{E}, pr)$ by Lemma 4.35, so $tr \in \mathrm{Tr}(\mathrm{FES2FTS}(\mathcal{E}), pr)$ by Alg. 4.

Induction step: Assume that for all traces $tr \in \mathrm{Tr}(\mathcal{E}, pr)$ of length n, it holds that tr is also a trace of FES2FTS($\mathcal{E}$) for pr. Let $tr' \in \mathrm{Tr}(\mathcal{E}, pr)$ be a trace of length $n + 1$ with $tr = \langle e_1 \cdot \ldots \cdot e_n \cdot e \rangle$

for $e_i, e \in E$ with $i \in \{1, \ldots, n\}$. By the definition of the traces of an FES, $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ is a trace in $\mathcal{E}$ for pr. By the induction hypothesis, tr is also a trace in FES2FTS($\mathcal{E}$) for pr, i.e., there exists $s \in S$ with $s_0 \xrightarrow{tr} s$ and $\text{eval}_{pr}(\mu(t)) = tt$, for all transitions t of tr. We have to show that $tr' \in \text{Tr}(\text{FES2FTS}(\mathcal{E}), pr)$. Since tr is a trace in FES2FTS($\mathcal{E}$) for pr, it suffices to show that (1) there exists $s' \in S$ with $(s, e, s') \in \delta$ and (2) $\text{eval}_{pr}(\mu((s, e, s'))) = tt$, i.e., $pr \models \mu((s, e, s'))$.

Ad (1): By Lemma 4.35, the sets $C = \{e_1, \ldots, e_n\}$ and $C' = C \cup \{e\}$ are configurations of $\mathcal{E}$ for pr. Line 1 of Alg. 4 implies that there exist states $s_C, s_{C'} \in S$ such that $s_C = C$ and $s_{C'} = C'$. Since $C' = C \cup \{e\}$, it follows from Lines 2–3 of Alg. 4 that $(s_C, e, s_{C'}) \in \delta$. Furthermore, since $C = \text{events}(tr)$, where tr is a trace from s_0 to s, Lemma 5.13 implies that $s = C$. Thus, $(s = s_C, e, s_{C'}) \in \delta$.

Ad (2): Since C and C' are configurations of $\mathcal{E}$ for pr, it holds that $pr \models \phi_C$ and $pr \models \phi_{C'}$, where ϕ_C and $\phi_{C'}$ are the feature expressions associated with C and C', respectively, see Def. 4.36. Furthermore, by (1) we obtain $(s, e, s') \in \delta$ with $s = C$ and $s' = C'$. Thus, $pr \models \phi_C \wedge \phi_{C'} = \mu((s, e, s'))$ (see Line 4 of Alg. 4).

"$\supseteq$" Now we show that for all $tr \in \text{Tr}(\text{FES2FTS}(\mathcal{E}), pr)$, it holds that $tr \in \text{Tr}(\mathcal{E}, pr)$.

Base case: If tr is the empty trace, then $tr \in \text{Tr}(\mathcal{E}, pr)$ by definition (Def. 4.33). If $tr = \langle e \rangle$, then there exists a state $s \in S'$ with $(s_0, e, s) \in \delta'$, i.e., $\text{eval}_{pr}(\mu((s_0, e, s)) = tt$. By Lines 2 and 3 of Alg. 4 it holds that $\{e\} = s \backslash s_0 = s \backslash \varnothing = s$, which implies by Line 1 (and since $\text{eval}_{pr}(\mu((s_0, e, s)) = tt$) that $\{e\}$ is a configuration of $\mathcal{E}$ for pr which is a trace of $\mathcal{E}$ for pr by the definition of traces (Def. 4.33).

Induction step: Assume that for all traces $tr \in \text{Tr}(\text{FES2FTS}(\mathcal{E}), pr)$ of length n, it holds that $tr \in \text{Tr}(\mathcal{E}, pr)$. Let $tr' \in \text{Tr}(\text{FES2FTS}(\mathcal{E}), pr)$ be a trace of length $n+1$ with $tr' = \langle e_1 \cdot \ldots \cdot e_n \cdot e \rangle$ for $e_i, e \in E$ with $i \in \{1, \ldots, n\}$, and let $s \in S$ with $s_0 \xrightarrow{tr'} s$. It holds $\text{eval}_{pr}(t) = tt$ for all transitions t in tr'. By the definition of traces of an FTS for a product (Def. 2.24), the sequence $tr = \langle e_1 \cdot \ldots \cdot e_n \rangle$ is a trace of FES2FTS($\mathcal{E}$) for pr of length n. From the induction hypothesis we obtain that $tr \in \text{Tr}(\mathcal{E}, pr)$. We have to show that $tr' \in \text{Tr}(\mathcal{E}, pr)$.

Since tr is a trace in $\mathcal{E}$ for pr, it is enough to show (0′) $\lambda(e) \in pr$ and $pr \models \nu(e)$, (1′) $(e_i, e) \notin \#$ for all $i \in \{1, \ldots, n\}$ and (2′) $\forall X \subseteq E'$ with $X \mapsto e$ it holds $X \cap \{e_1, \ldots, e_n\} \neq \varnothing$ (see Def. 4.33). By Lemma 5.13 it holds that $s = \text{events}(tr')$, and by Line 1 of Alg. 4 it holds that $s \in \text{Config}(\mathcal{E})$. Thus, the definition of a configuration of an FES (Def. 4.31) implies (1′) and (2′). Since tr' is a trace in FES2FTS($\mathcal{E}$) for pr, it holds that $pr \models \mu((s', e, s)) = \phi_s \wedge \phi_{s'}$, for $(s, e, s') \in \delta$, implying $pr \models \phi_s$ and $pr \models \phi_{s'}$, see Def. 4.36. By Lemma 4.38, this implies $s \in \text{Config}(\mathcal{E}, pr)$ and thus (0′) holds.

□

In the proof of Alg. 4, we need a lemma that determines, for a state of a translated FES via Alg. 4, the corresponding configuration of the FES.

Lemma 5.13 (On Alg. 4). *Let $\mathcal{E}$ be an FES and* FES2FTS($\mathcal{E}$) *its translation via Alg. 4. Let $s \in S$ and $tr \in \text{Tr}(\text{FES2FTS}(\mathcal{E}))$, with $s_0 \xrightarrow{tr} s$. Then $s = \text{events}(tr)$.*

Proof. The proof is the same as the proof of Lemma 5.6, using a trace with respect to a product of $\mathcal{E}$. □

As in Lemma 5.10, we prove that the behavioral products of an FES and its translation via Alg. 4 have the same events.

Lemma 5.14 (FES2FTS($\mathcal{E}$) preserves events of behavioral products). *Let $\mathcal{E}$ be an FES, let* FES2FTS($\mathcal{E}$) *be its translation via Alg. 4, and let $pr \in \text{products}(\mathcal{E}) = \text{products}(\text{FES2FTS}(\mathcal{E}))$ be a product. Then* behPr($\mathcal{E}, pr$) *and* behPr(FES2FTS($\mathcal{E}$), pr) *have the same events.*

Proof. Let $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ be an FES, and let FES2FTS($\mathcal{E}$) $= (\mathcal{M}, \mathcal{T}, \mu)$ with $\mathcal{T} = (S, E, s_0, \delta)$ be its translation via Alg. 4. Let pr be a product of $\mathcal{E}$ (which is also a product of FES2FTS($\mathcal{E}$)), let

Algorithm 5 LinFTS2BFM: from linear FTS to BFM

Input: linear FTS $\mathcal{F}$
Output: BFM $\mathcal{B}$
1: $\mathcal{E} :=$ LinFTS2FES($\mathcal{F}$) ▷ Get trace equivalent FES
2: $\mathcal{B} :=$ bfm($\mathcal{E}$) ▷ Get isomorphic BFM
3: **return** $\mathcal{B}$

Algorithm 6 bfm: BFM view of FES (see Def. 4.26)

Input: FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ with $\mathcal{M} = (f, \{\mathcal{G}_i\}_{i\in I^0}, \mathrm{FC})$ and $\mathcal{S} = (E, \#, \mapsto)$
Output: BFM $\mathcal{B} = (f' : \mathcal{S}', \{C_i\}_{i\in I^0}, \mathrm{FC}', \mathrm{EC}, \mathrm{PC})$
1: $f' := f,\ \mathrm{FC}' := \mathrm{FC},\ \mathrm{EC} = \mathrm{interFtEc}_f(\mathcal{E}),\ \mathrm{PC} := \mathrm{productC}_f(\mathcal{E})$ ▷ Extract knowledge related to f
2: $\mathcal{S}' := (\mathrm{events}_f(\mathcal{E}), \mathrm{confl}(\mathrm{intraFtEc}_f(\mathcal{E})), \mathrm{cause}(\mathrm{intraFtEc}_f(\mathcal{E})))$ ▷ Extract BES of f
3: **switch** $\mathcal{G}_i$ **do** ▷ Do the procedure for every child of f
4: **case mnd**($\hat{\mathcal{M}}$)
5: $C_i :=$ **mnd**(bfm($\hat{\mathcal{M}}$)) ▷ Call algorithm on $\hat{\mathcal{M}}$ and include as mandatory sub-FM group
6: **case opt**($\hat{\mathcal{M}}$)
7: $C_i :=$ **opt**(bfm($\hat{\mathcal{M}}$)) ▷ Call algorithm on $\hat{\mathcal{M}}$ and include as optional sub-FM group
8: **case or** $(\hat{\mathcal{M}}_j)_{j\in I^2}$
9: $C_i :=$ **or** $\left(\mathrm{bfm}(\hat{\mathcal{M}}_j)_{j\in I^2}\right)$ ▷ Call algorithm on all $\hat{\mathcal{M}}_j$ and include as or sub-FM group
10: **case xor** $(\hat{\mathcal{M}}_j)_{j\in I^2}$
11: $C_i :=$ **xor** $\left(\mathrm{bfm}(\hat{\mathcal{M}}_j)_{j\in I^2}\right)$ ▷ Call algorithm on all $\hat{\mathcal{M}}_j$, and include as xor sub-FM group
12: **return** $(f' : \mathcal{S}', \{C_i\}_{i\in I^0}, \mathrm{FC}', \mathrm{EC}, \mathrm{PC})$

behPr($\mathcal{E}, pr$) = $(E', \#, \mapsto)$, and let behPr(FES2FTS($\mathcal{E}$), pr) = (S', E'', s_0, δ'). We want to show that $E' = E''$

"$\subseteq$" Let $e \in$ events(behPr($\mathcal{E}, pr$)). Then there exists a trace tr' in behPr($\mathcal{E}, pr$) such that $e \in$ events(tr') and e is the last element of tr'. Let tr be the trace tr' without e. By Lemma 4.35, the sets $C' =$ events(tr') and $C = C'\backslash\{e\}$ are configurations of $\mathcal{E}$ for pr. By Lemma 4.38 it holds that $\mathrm{eval}_{pr}(\phi_C) = tt = \mathrm{eval}_{pr}(\phi_{C'})$, where ϕ_C and $\phi_{C'}$ are the feature expressions associated with C and C', respectively. This implies $\mathrm{eval}_{pr}(\phi_C \wedge \phi_{C'}) = tt$.

By Lines 2 and 3 of Alg. 4, there are states $s = C$ and $s' = C'$ in S such that $(s, e, s) \in \delta$. Since $\mu((s, e, s')) = \phi_C \wedge \phi_{C'}$ and $\mathrm{eval}_{pr}(\phi_C \wedge \phi_{C'}) = tt$, it holds that $e \in$ behPr(FES2FTS($\mathcal{E}$), pr).

"$\supseteq$" Let $e \in$ behPr(FES2FTS($\mathcal{E}$), pr), and let $s, s' \in S$ such that $(s, e, s') \in \delta'$. Then $\mathrm{eval}_{pr}(\mu((s, e, s')) = tt$, and by Line 1 of Alg. 4 it holds that $\mu((s, e, s')) = \phi_s \wedge \phi_{s'}$, where ϕ_s and $\phi_{s'}$ are the feature expressions associated with the configurations of $\mathcal{E}$ that s and s', respectively, represent. This implies $\mathrm{eval}_{pr}(\phi_s) = tt$ and $\mathrm{eval}_{pr}(\phi_{s'}) = tt$. Lemma 4.38 then implies the lemma. □

5.5 From Linear FTSs to BFMs by Preserving Traces

Algorithm 5 defines a translation from a linear FTS to a BFM, as proven in Prop. 5.15. The algorithm works as follows. First, the linear FTS is translated to a trace-equivalent FES using Alg. 3, see Line 1. This FES is then translated to its isomorphic BFM using Alg. 6, see Line 2, which is the algorithmic version of Def. 4.26. Getting the BFM view of an FES consists of splitting the knowledge aggregated in the FES to knowledge associated with single features. To determine which knowledge to associate to which feature, the event location knowledge (mostly written as λ) of an FES can be used as defined in Not. 4.9. The structure of the isomorphic BFM is the same as the structure of

Algorithm 7 BFM2FTS: from BFM to FTS

Input: BFM $\mathcal{B}$
Output: FTS $\mathcal{F}$
1: $\mathcal{E} := \text{fes}(\mathcal{B})$ ▷ Get isomorphic FES
2: $\mathcal{F} := \text{FES2FTS}(\mathcal{E})$ ▷ Get trace equivalent FTS
3: **return** $\mathcal{F}$

Algorithm 8 fes: FES view of BFM (see Def. 4.24)

Input: BFM $\mathcal{B}$
Output: FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$
1: $\mathcal{E} := (\text{fm}(\mathcal{B}), \text{bes}(\mathcal{B}), \text{eLocKnow}(\mathcal{B}), \text{prodKnow}(\mathcal{B}))$ ▷ Use Algorithms 9 to 12
2: **return** $\mathcal{E}$

the FM of the FES, see Lines 3–11. For every feature f of the FM, we extract the cross-tree feature constraints of f, the inter-feature event constraints of f, the product constraints of f, and the BES modeling the behavior of f which consists of the events of f and the intra-feature event constraints of f, see Lines 1 and 2.

We first prove that Alg. 5 translates a linear FTS to a BFM.

Proposition 5.15 (LinFTS2BFM($\mathcal{F}$) is a BFM). *Let $\mathcal{F}$ be a linear FTS. Then* LinFTS2BFM($\mathcal{F}$) *is a BFM.*

Proof. In Prop. 5.8 we proved that Alg. 3 produces an FES, and Alg. 6 produces a BFM by its definition. Therefore, for a linear FTS $\mathcal{F}$, the translation LinFTS2BFM($\mathcal{F}$) = bfm(LinFTS2FES($\mathcal{F}$)) is a BFM. □

Now we can prove that a linear FTS and its translation via Alg. 5 are trace equivalent.

Proposition 5.16 (Trace equivalence of $\mathcal{F}$ and LinFTS2BFM($\mathcal{F}$)). *Let $\mathcal{F}$ be a linear FTS. Then* LinFTS2BFM($\mathcal{F}$) *and $\mathcal{F}$ are trace equivalent.*

Proof. By Prop. 5.9, $\mathcal{F}$ is trace equivalent to LinFTS2FES($\mathcal{F}$), and by Prop. 4.28 it holds that bfm(LinFTS2FES($\mathcal{F}$)) and LinFTS2FES($\mathcal{F}$) are isomorphic. Therefore, it holds that LinFTS2BFM($\mathcal{F}$) = bfm(LinFTS2FES($\mathcal{F}$)) and $\mathcal{F}$ are trace equivalent. □

5.6 From BFMs to FTSs by Preserving Traces

Algorithm 7 defines a translation from a BFM to a (mostly non-linear) FTS, as proven in Prop. 5.17. The algorithm works as follows. First, we use the isomorphism defined in Def. 4.24 to translate a BFM to its isomorphic FES view, see Line 1, which we then translate via Alg. 4 to an FTS, see Line 2. To define the FES view of a BFM, see Alg. 8, we provide algorithms for extracting the FM, the BES, the event location knowledge, and the product knowledge of a BFM in Algorithms 9 to 12, respectively. These algorithms reflect the definitions of these mappings provided in Definitions 4.15, 4.18, 4.20 and 4.22. To extract the FM of a BFM, see Alg. 9, it is enough to delete all elements of the BFM that are not needed for the FM. The BES of a BFM can be extracted by collecting all events, conflicts and causalities associated with all features of the BFM, see Alg. 10. The event location knowledge of a BFM consists of all pairs of events and features such that the event is associated with the feature, see Alg. 11, and the product knowledge consists of all product constraints associated with all features of the BFM, see Alg. 12.

We first prove that Alg. 7 translates a BFM to an FTS.

Proposition 5.17 (BFM2FTS($\mathcal{B}$) is an FTS). *Let $\mathcal{B}$ be a BFM. Then* BFM2FTS($\mathcal{B}$) *is an FTS.*

Algorithm 9 fm: extract FM from a BFM (see Def. 4.15)

Input: BFM $\mathcal{B} = (f : \mathcal{S}, \{C_i\}_{i \in I^0}, \mathrm{FC}, \mathrm{EC}, \mathrm{PC})$
Output: FM $\mathcal{M} = (f', \{\mathcal{G}_i\}_{i \in I^0}, \mathrm{FC}')$

$f' := f$, $\mathrm{FC}' := \mathrm{FC}$
switch C_i **do** ▷ Do the procedure for every child of f
  case $\mathbf{mnd}(\hat{\mathcal{B}})$
    $\mathcal{G}_i := \mathbf{mnd}(\mathrm{fm}(\hat{\mathcal{B}}))$ ▷ Call algorithm on $\hat{\mathcal{B}}$ and include as mandatory sub-BFM group
  case $\mathbf{opt}(\hat{\mathcal{B}})$
    $\mathcal{G}_i := \mathbf{opt}(\mathrm{fm}(\hat{\mathcal{B}}))$ ▷ Call algorithm on $\hat{\mathcal{B}}$ and include as optional sub-BFM group
  case or $(\mathcal{B}_j)_{j \in I^2}$
    $\mathcal{G}_i := \mathbf{or}\left(\mathrm{fm}(\mathcal{B}_j)_{j \in I^2}\right)$ ▷ Call algorithm on all $\hat{\mathcal{B}}_j$ and include as or sub-BFM group
  case xor $(\mathcal{B}_j)_{j \in I^2}$
    $\mathcal{G}_i := \mathbf{xor}\left(\mathrm{fm}(\mathcal{B}_j)_{j \in I^2}\right)$ ▷ Call algorithm on all $\hat{\mathcal{B}}_j$ and include as xor sub-BFM group
return $(f', \{\mathcal{G}_i\}_{i \in I^0}, \mathrm{FC}')$

Algorithm 10 bes: extract BES from a BFM (see Def. 4.18)

Input: BFM $\mathcal{B} = (f : \mathcal{S}, \{C_i\}_{i \in I^0}, \mathrm{FC}, \mathrm{EC}, \mathrm{PC})$ with $\mathcal{S} = (E, \#, \mapsto)$
Output: BES $\mathcal{S}' = (E', \#', \mapsto')$

$E' := E$, $\#' := \# \cup \mathrm{confl}(\mathrm{EC})$, $\mapsto' := \mapsto \cup \mathrm{cause}(\mathrm{EC})$
switch C_i **do** ▷ Do the procedure for every child of f
  case $\mathrm{Uop}(\hat{\mathcal{B}})$
    $(\hat{E}, \hat{\#}, \hat{\mapsto}) := \mathrm{bes}(\hat{\mathcal{B}})$ ▷ Get BES of sub-BFM
    $E' := E' \cup \hat{E}$, $\#' := \#' \cup \hat{\#}$, $\mapsto' := \mapsto' \cup \hat{\mapsto}$
  case $\mathrm{Op}(\mathcal{B}_i)_{i \in I^2}$
    for $i \in I^2$ **do**
      $(\hat{E}_i, \hat{\#}_i, \hat{\mapsto}_i) := \mathrm{bes}(\mathcal{B}_i)$ ▷ Get BES of sub-BFM
      $E' := E' \cup \hat{E}_i$, $\#' := \#' \cup \hat{\#}_i$, $\mapsto' := \mapsto' \cup \hat{\mapsto}_i$
return $(E', \#', \mapsto')$

Algorithm 11 eLocKnow: extract event location knowledge from a BFM (see Def. 4.20)

Input: BFM $\mathcal{B} = (f : \mathcal{S}, \{C_i\}_{i \in I^0}, \mathrm{FC}, \mathrm{EC}, \mathrm{PC})$ with $\mathcal{S} = (E, \#, \mapsto)$ over features N
Output: $\lambda \subseteq E \times N$

$\lambda := \{(e, f) \mid e \in E\}$
switch C_i **do** ▷ Do the procedure for every child of f
  case $\mathrm{Uop}(\hat{\mathcal{B}})$
    $\lambda := \lambda \cup \mathrm{eLocKnow}(\hat{\mathcal{B}})$ ▷ Get event location knowledge of sub-BFM
  case $\mathrm{Op}(\mathcal{B}_i)_{i \in I^2}$
    $\lambda := \lambda \bigcup_{i \in I^2} \mathrm{eLocKnow}(\mathcal{B}_i)$ ▷ Get event location knowledge of sub-BFMs
return λ

Proof. By definition, the FES view $\mathrm{fes}(\mathcal{B})$ of a BFM $\mathcal{B}$ is an FES. In Prop. 5.11 we proved that Alg. 4 produces an FTS. Therefore, $\mathrm{BFM2FTS}(\mathcal{B}) = \mathrm{FES2FTS}(\mathrm{fes}(\mathcal{B}))$ is an FTS. □

Now we can prove that a BFM and its translation via Alg. 7 are trace equivalent.

Algorithm 12 prodKnow: extract product (configuration) knowledge from a BFM (see Def. 4.22)

Input: BFM $\mathcal{B} = (f : \mathcal{S}, \{C_i\}_{i \in I^0}, \mathrm{FC}, \mathrm{EC}, \mathrm{PC})$ with $\mathcal{S} = (E, \#, \mapsto)$ over features N
Output: $\nu \subseteq E \times \mathrm{FExp}(N)$

```
1: ν := PC
2: switch C_i do                                  ▷ Do the procedure for every child of f
3:     case Uop(B̂)
4:         ν := ν ∪ prodKnow(B̂)                   ▷ Get product knowledge of sub-BFM
5:     case Op(B_i)_{i∈I²}
6:         ν := ν ⋃_{i∈I²} prodKnow(B_i)          ▷ Get product knowledge of sub-BFMs
7: return ν
```

Proposition 5.18 (Trace equivalence of $\mathcal{B}$ and BFM2FTS($\mathcal{B}$)). *Let $\mathcal{B}$ be a BFM. Then* BFM2FTS($\mathcal{B}$) *and $\mathcal{B}$ are trace equivalent.*

Proof. By Prop. 4.28, a BFM $\mathcal{B}$ and its FES view fes($\mathcal{B}$) are isomorphic, and by Prop. 5.12 it holds that fes($\mathcal{B}$) and FES2FTS(fes($\mathcal{B}$)) are trace equivalent. Therefore, BFM2FTS($\mathcal{B}$) = FES2FTS(fes($\mathcal{B}$)) and $\mathcal{B}$ are trace equivalent. □

6 EVALUATION

In this section, we lay out the methodology, report on the carried out evaluation (both in terms of theoretical results and empirical evaluation), and reflect on the answers to the following overarching research question:

Do BFMs lead to a more compact representation of feature behavior than the state-of-the-art approaches?

FTSs are the most expressive and succinct family-based behavioral model [23, 104]. As witnessed by many benchmarks (also used as subject systems in this section), they are the most widely used model in the literature. Thus, we use them as the baseline model for evaluating the expressiveness and succinctness of BFMs.

We answer our overarching research question in terms of the following sub-questions:

RQ1 In theory, how do the expressiveness and succinctness of BFMs and FTSs compare?
  RQ1.1 Are BFMs as expressive as linear FTSs?
  RQ1.2 How do the sizes of trace-equivalent linear FTSs and BFMs compare?
RQ2 In practice, how does the size of a BFM compare to the size of an FTS?

6.1 Methodology

In this section, we describe the methodology used to answer our research questions. To explain our methodology and define our metrics, we need some theoretical concepts. We start by defining parallel compositions of (linear) FTSs in Sect. 6.1.1, then define the metrics we use to compare (parallel compositions of) FTSs and BFMs in Sect. 6.1.2. We then present the subject systems used in our evaluation in Sect. 6.1.3 and the details of our implementation in Sect. 6.1.4. Finally, Sect. 6.1.5 highlights several practical considerations; namely, points where the implementation extends the presented algorithms for optimization or generalization purposes.

6.1.1 Parallel Compositions of FTSs.

In RQ2.2, we compare BFMs to parallel compositions of linear FTSs. We define the parallel composition of FTSs according to Classen [34].

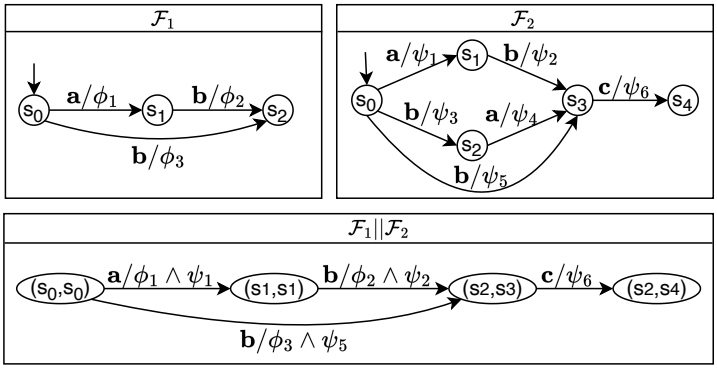

Fig. 9. An example of a parallel composition of FTSs

Definition 6.1 (Parallel composition of FTSs). Let $\mathcal{F}_i = (\mathcal{M}, \mathcal{T}_i, \mu_i)$ be FTSs with $\mathcal{T}_i = (S_i, E_i, s_{0,i}, \delta_i)$ for $i \in \{1, \ldots, n\}$. Then the *parallel composition* $\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_n$ is the FTS $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$ where $\mathcal{T} = (S, E, s_0, \delta)$ with

- $S = S_1 \times \cdots \times S_n$,
- $E = \bigcup_{i=1,\ldots,n} E_i$,
- $s_0 = (s_{0,1}, \ldots, s_{0,n})$,
- $\forall (s_1, \ldots, s_n), (s'_1, \ldots, s'_n) \in S, \forall e \in E. ((s_1, \ldots, s_n), e, (s'_1, \ldots, s'_n)) \in \delta$ if and only if both of the following conditions hold
  (i) $\forall i \in \{1, \ldots, n\}. e \in E_i \Rightarrow (s_i, e, s'_i) \in \delta_i$,
  (ii) $\forall i \in \{1, \ldots, n\}. e \notin E_i \Rightarrow s_i = s'_i$.
- $\forall (s_1, \ldots, s_n), (s'_1, \ldots, s'_n) \in S, \forall e \in E. \mu((s_1, \ldots, s_n), e, (s'_1, \ldots, s'_n)) = \bigwedge_{i \in I} \phi_i$ where $I = \{1 \leq i \leq n \mid e \in E_i\}$ and $\mu_i(s_i, e, s'_i) = \phi_i$ for $i \in I$.

The transition relation δ and the product knowledge μ reflect that the FTSs $\mathcal{F}_i$, $i \in \{1, \ldots, n\}$, synchronize on events they have in common and interleave for the other events: There exists a transition from a state $(s_1, \ldots, s_n) \in S$ to a state $(s'_1, \ldots, s'_n) \in S$ with event $e \in E$ if and only if the following three conditions hold:

(1) for every $\mathcal{F}_i$ with $e \in E_i$ it holds $(s_i, e, s'_i) \in \delta_i$ and $\mu((s_i, e, s'_i)) = \phi_i$,
(2) for every $\mathcal{F}_i$ with $e \notin E_i$, the state is not changed, and
(3) the feature expression ϕ associated to the transition is the conjunction of all feature expressions ϕ_i.

Note that by defining a composition operator on FMs as, e.g., in [3], it is also possible to compose FTSs with distinct FMs.

Example 6.2 (Parallel composition of FTSs). Figure 9 shows the FTSs $\mathcal{F}_1$ and $\mathcal{F}_2$ as well as their parallel composition. Note that the trace $\langle b \cdot a \cdot c \rangle$ of $\mathcal{F}_2$ is not a trace of $\mathcal{F}_1 \parallel \mathcal{F}_2$ because $\mathcal{F}_1$ also contains the events a and b but not the trace $\langle b \cdot a \rangle$.

Algorithms 1, 3 and 5 are only designed for linear (F)TSs. However, with minor adaptations, they can also be applied to parallel compositions of linear (F)TSs. We describe these adaptations in Sect. 6.1.4, as part of our implementation details.

6.1.2 *Evaluation Metric.*

In our research questions, we assess the expressiveness and succinctness of BFMs with respect to FTSs according to the following definition.

Definition 6.3 (Expressiveness and succinctness). Let M_1 and M_2 be modeling languages.

- M_1 is at *least as expressive* as M_2, if there exists a translation scheme $||\_|| : M_1 \to M_2$ such that for every model in $m \in M_1$, its translation $||m|| \in M_2$ has the same semantics (set of traces).
- M_1 is *at least as succinct* as M_2, if the translation scheme $||\_|| : M_1 \to M_2$ always maps a model $m \in M_1$ to $||m|| \in M_2$ such that $||m||$ has a polynomial size with respect to m.
- M_1 is *less succinct* than M_2 if there is no such polynomial translation scheme.

To assess the succinctness of BFMs with respect to FTSs, we compare their respective sizes, defined by the following metric (further discussed in Sect. 6.4):

Definition 6.4 (Size of FMs, TSs, FTSs, BESs, FESs, and BFMs).

- The *size* $|\mathcal{M}|$ of an FM $\mathcal{M}$ is defined as $|\mathcal{M}| = |\{f \mid f \in \text{features}(\mathcal{M})\}|$.
- The *size* $|\mathcal{T}|$ of a TS $\mathcal{T} = (S, E, s_0, \delta)$ is defined as $|\mathcal{T}| = |S| + |E| + |\{(s, e, s') \in \delta\}|$.
- The *size* $|\mathcal{F}|$ of an FTS $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$ is defined as $|\mathcal{F}| = |\mathcal{M}| + |\mathcal{T}|$.
- The *size* $|\mathcal{S}|$ of a BES $\mathcal{S} = (E, \#, \mapsto)$ is defined as $|\mathcal{S}| = |E| + |\{(e, e') \in \#\}| + |\{(X, e) \in \mapsto\}|$.
- The *size* $|\mathcal{E}|$ of an FES $\mathcal{E} = (\mathcal{M}, \mathcal{S}, \lambda, \nu)$ is defined as $|\mathcal{E}| = |\mathcal{M}| + |\mathcal{S}|$.
- The *size* $|\mathcal{B}|$ of a BFM $\mathcal{B}$ is defined in terms of its FES-view: $|\mathcal{B}| = |\,\text{fes}(\mathcal{B})|$.

Remark 6.5 (On sizes of FTSs and BFMs/FESs). Without loss of generality, we may assume that trace-equivalent FTSs and BFMs/FESs have the same FM; this is the case for Algorithms 3 and 4. Therefore, to compare the sizes of an FTS and an BFM/FES, it is enough to compare the sizes of the underlying TS and BES, respectively.

6.1.3 Subject Systems.

To compare FTSs and BFMs in practice (RQ2), we build an evaluation benchmark with 43 subject systems, curated from well-known case studies of FTSs from the literature [15, 33, 46]. Our benchmark contains all FTSs from the literature that could be linearized in a reasonable time. In addition, for systems where component FTSs were available, the benchmark set also contains all possible combinations of their parallel compositions, leading to the largest evaluation benchmark of FTSs in the literature. In the evaluation benchmark, we curated the FTS specifications as follows. In each case study, we first linearized the FTS by renaming events to avoid duplicates and by removing self-loops as well as transitions returning to the initial state. For parallel compositions, this linearization was applied to each individual component before constructing the flattened compositional FTS. Below, we describe the subject systems in our benchmark.

Cleaning Robot. We again consider the example of the cleaning robot SPL, introduced in Example 1.1, with FTS presented in Example 2.21.

Card Payment Terminal. The Card Payment Terminal [46] models the Eurocard-Mastercard-Visa (EMV) protocol. It defines different card payment schemes for direct debit and/or credit transactions. The terminal supports optional connectivity and offers multiple ways of reading a card: via chip, magnetic stripe, or near-field communication (NFC). Before proceeding with a payment, the cardholder must always authenticate. Depending on the selected product variant, this is done either by entering a PIN or by providing a signature.

Vending Machines. We evaluate a collection of Vending Machines (VMs) and all their possible combinations through parallel composition. We use two versions of parallel composition: one with

only interleaving, and one with synchronization over a subset of selected actions. Each machine type dispenses a different category of beverages. The *Coffee VM* [15] offers coffee, cappuccino, and tea. The *Soda VM* [33, 46] also provides tea, but replaces coffee-based drinks with sodas; it additionally includes an optional feature that allows dispensing free drinks. The *Soup VM* [15] offers three types of soup: chicken, pea, and tomato. All VMs support payments in euros or dollars and include an optional feature to cancel an order. Note that while the Soup and Coffee VMs have previously been composed by ter Beek et al. [15], we additionally compose these two with the Soda VM.

Mine Pump System. The *Mine Pump system* [33, 46] models the regulation mechanism of a mine shaft and is composed of five components. At the heart of the system lies the *Controller (C)*, which acts as the central orchestrator. It manages communications among components and determines the actions to take based on incoming signals. An alternative version of the Controller [15] includes its composition with the *State (S)* component which logs the system state at any given time. The remaining three components model actuators interacting with the environment. The *Water (W)* component monitors the water level in the shaft and signals the *Controller* when intervention is required (e.g., to raise or lower the water level). In response, the Controller activates the *Pump (P)*, which adjusts the water level accordingly. The *Methane detector (M)* continuously reports the methane concentration to the Controller to ensure it remains within safe limits, thereby preventing potential explosions. In this case study, we systematically evaluate all possible compositions of the Controller with the other components. We enforce the presence of the Controller in every combination because it is the only component that exhibits internal variability. As before, we use two versions of parallel composition: one with only interleaving and one with synchronization over a subset of selected actions.

6.1.4 *Implementation.*

We implemented our six translation algorithms (Algorithms 1 to 5 and 7) in Java. The full implementation, along with all experimental data, is available in our replication package.[4] Our code builds upon VIBeS,[5] a library for manipulating FTSs, and integrates the *Universal Variability Language (UVL)* [22, 92] to ensure compatibility with modern feature modeling tools.

To bridge the differences between UVL and VIBeS, particularly in how they represent feature models, we introduced a unified structure that combines the advantages of both approaches. Specifically, we adopt a UVL-based FM restricted to Boolean features, and, using VIBeS, couple it with a solver to evaluate feature constraints. In VIBeS, the solver is used, for example, to determine whether a given feature expression is valid with respect to the FM, or to compute the set of products that satisfy a set of feature constraints. VIBeS supports two types of solvers: SAT4J[6] and JavaBDD.[7] In our implementation, we use JavaBDD.[8] As in VIBeS, we encode feature expressions using Ben Podgursky's `jbool_expressions` library.[9] Our unified representation preserves the intuitive tree-like structure of feature models, lost in VIBeS's CNF-based encoding, while supporting efficient

[4] https://github.com/sfortz/BehavioralFeatureModel
[5] https://github.com/xdevroey/vibes
[6] https://sat4j.org/
[7] https://github.com/com-github-javabdd/com.github.javabdd
[8] While both solvers are theoretically interchangeable, since they should yield the same results, in practice, we rely on JavaBDD because the SAT4J interface is currently incompatible with UVL FMs. Binary Decision Diagrams (BDDs) usually offer more efficient reasoning than traditional SAT solvers for most types of analysis, though it requires a costly compilation step. Fortunately, recent advances in scalable compilation [55, 65, 93] have made BDD use more practical.
[9] https://github.com/bpodgursky/jbool_expressions

constraint reasoning. We support import and export of models in XML format. For TSs and FTSs, we also support the *Graphviz* DOT format,[10] which facilitates their visualization.

Our BFMs are built on a new `BehavioralFeature` class, which extends the base `Feature` class. A `BehavioralFeatureModel` is implemented as a specialization of `FeatureModel<BehavioralFeature>`, conforming to the `FeaturedEventStructure` interface.

6.1.5 *Practical Considerations.*

Our implementation faithfully follows the translation algorithms presented in this paper, with a few practical adaptations to improve scalability and support more general input models, as discussed in this section.

Relaxing Linearity for Parallel Compositions. In our implementation of Alg. 3, we relaxed the original linearity assumption and now allow the input FTS to be the result of a *parallel composition of linear FTSs*. This modification mainly affects the algorithm in two ways:

- In Line 5, we additionally check that $\neg\,\text{reachable}(e, e')$ holds, i.e., the event e' is not reachable from e. This is necessary because the composition of two linear FTSs could lead to interleaving, where the resulting FTS has a trace that contains e before e' and another that contains e' before e, implying that none of them causes the other.
- We introduce an additional step to split bundle sets containing non-conflicting events after Line 6. Linearity of (F)TSs ensures that two events always appear in the same order in every trace of the (F)TS. Therefore, for linear (F)TSs, the construction in Line 5 produces bundle sets in which all events are in conflict. However, composing two or more linear FTSs may lead to interleaving of events such that the bundle sets defined in Line 5 contain non-conflicting events.
  Therefore, the bundle sets must be split into maximal subsets containing only mutually conflicting events. This non-trivial problem corresponds to finding all maximal cliques in an undirected graph where nodes represent events in the bundle and edges represent conflicts between them. A clique is a subset of nodes in which every pair is connected by an edge, and a maximal clique is a clique that cannot be extended by including any additional adjacent node [27]. This classical graph-theoretical problem is well-studied and can be efficiently solved using the *Bron–Kerbosch algorithm* [27], which we also employ in our implementation.

Handling Multiple Feature Expressions. In a parallel composition of linear FTSs, an event might be associated with several transitions, each of them associated with a different feature expression. Therefore, we have to change Lines 4 and 5 of Alg. 3 to define the event location knowledge and product knowledge of an event e using the disjunction of all feature expressions associated with transitions labeled with e. That is, we use $\bigvee_{(s,e,s')\in\delta} \mu((s, e, s'))$ instead of $\mu((s, e, s'))$.

Conflict Optimization via Bicliques. We optimize the representation of conflicts by moving from an explicit pairwise encoding to a more compact *biclique representation* [5, 29, 83]. A conflict (A, B) denotes that every event in the set A is in conflict with every event in the set B. For example, a TS including the transitions $s \xrightarrow{a} s_a \xrightarrow{b} s_b \xrightarrow{c} s_c$ and the transitions $s' \xrightarrow{d} s_d \xrightarrow{e} s_e \xrightarrow{f} s_f$ would yield $3^2 = 9$ conflict pairs in the standard encoding. In our biclique format, they are captured by a single conflict: $\{a, b, c\}\#\{d, e, f\}$. This representation is semantically equivalent yet can be significantly more compact. It reduces the total number of conflicts but increases the size of each (as a pair of sets rather than pairs of individual events). To reflect this change, Tbl. 1 in Sect. 6.3 reports

[10] https://graphviz.org/

- the total number of conflicts, defined as the number of bicliques, and
- the maximum conflict size, defined as $|A| + |B|$ for a conflict $A\#B$.

Transforming the pairwise conflict representation into a biclique-based representation involves solving the *minimum biclique edge cover problem* [83], which is NP-complete. To perform this transformation, we construct a conflict graph capturing all conflicts present in the BFM. This graph can quickly scale to over 100 nodes and hundreds or even thousands of edges. Then, we partition this graph into bicliques, adopting a greedy heuristic to cover all edges. Since the problem is NP-complete and due to the size of our problem instances, such heuristic is the most practical solution.

6.2 RQ1: In Theory, How do BFMs and FTSs Compare in Terms of Expressiveness and Succinctness?

We answer this research question using results from Sect. 5.

6.2.1 *RQ1.1: Are BFMs as Expressive as Linear FTSs?*

To show that BFMs are at least as expressive as linear FTSs, we prove in Prop. 5.16 that Alg. 5 can translate every linear FTS to a trace equivalent BFM.

Answer to RQ1.1: Are BFMs as Expressive as Linear FTSs?

BFMs are as expressive as linear FTSs because we can translate every linear FTS to a BFM.

6.2.2 *RQ1.2: How do the sizes of trace-equivalent linear FTSs and BFMs compare?*

To show that BFMs are at least as succinct as FTSs, we first show in the following proposition that Alg. 1 translates a linear TS to a BES that has at most quadratic size with respect to the source TS.

Proposition 6.6 (Upper bound for the output of LinTS2BES). *Let $\mathcal{T} = (S, E, s_0, \delta)$ be a linear TS, and let $\mathcal{S} = (E, \#, \mapsto)$ be its translation via Alg. 1. Then the size of $\mathcal{S}$ is in $O(|\mathcal{T}|^2)$.*

Proof. The fact that $\mathcal{T}$ is linear implies that $|S| = |\{(s, e, s') \in \delta| + 1$ and $|E| = |\{(s, e, s') \in \delta|$. Therefore, $|\mathcal{T}| = |E| + |S| + |\{(s, e, s') \in \delta| = 3|E| + 1$. We want to find upper bounds for $|\{(e, e') \in \#\}|$ and $|\{(X, e) \in \mapsto\}|$.

The conflict relation $\#$ is a symmetric relation containing pairs of events. If $E = \{e_1, \ldots, e_n\}$, then e_1 can be combined with $e_2, \ldots, e_n$, i.e., with $|E| - 1$ elements, e_2 can be combined with $e_3, \ldots, e_n$, i.e., $|E| - 2$ elements, and so on. Thus, an upper bound for $|\{(e, e') \in \#\}|$ is $\sum_{i=1}^{|E|-1} i$. This upper bound can be reached if $\mathcal{T}$ is a "star", where each transition in δ is of the form (s_0, e, s) for $e \in E$ and $s \in S$. Then Alg. 1 gives that $\#$ contains all pairs of events because no event is reachable from another event.

The causality relation $\mapsto$ can contain at most $|E|$ elements, one for each event in E. This upper bound can almost be reached if $\mathcal{T}$ is a "row" of transitions, i.e., $\mathcal{T}$ has exactly one trace. Then Alg. 1 gives that $\mapsto$ contains a pair (X_e, e) for every $e \in E$ apart from the event associated to the first transition, where X_e contains only the event associated to the transition directly before the one e is associated to.

Considering both upperbounds, we have $|\mathcal{S}| = |E| + \sum_{i=1}^{|E|-1} i + |E| = |E| + \sum_{i=1}^{|E|} i$, implying $|\mathcal{S}| \in O(|\mathcal{T}|^2)$. □

Proposition 6.7 (Upper bound for the output of BES2TS). *Let $\mathcal{S} = (E, \#, \mapsto)$ be a BES, and let $\mathcal{T} = (S, E, s_0, \delta)$ be its translation via Alg. 2. Then the size of $\mathcal{T}$ is in $O(2^{|\mathcal{S}|})$.*

Proof. We want to find upper bounds for S and $|\{(s, e, s') \in \delta\}|$.

The set of states S is defined as the set of all configurations of $\mathcal{S}$ in Line 1 of Alg. 2. The set of configurations of $\mathcal{S}$ is a subset of $\mathcal{P}(E)$. Thus, an upper bound for the size of S is $|\mathcal{P}(E)| = 2^{|E|}$. If this upper bound is reached, there exist no conflicts and causalities between the events of $\mathcal{S}$. Thus, in this case we have $|\mathcal{S}| = |E|$.

The upper bound for the number of transitions can be reached with the maximum number of states because there is only a transition between two states, if they differ in exactly one event (see Lines 2 and 3 of Alg. 2). If $S = \mathcal{P}(E)$, then for every $s \in S$ there exist $|E| - |s|$ outgoing transitions from s. Namely, for every $e \in E \backslash \{e' \in s\}$ there exists a transition $(s, e, s \cup \{e\})$. There are $|E| - i + 1$ states with size $i \neq 0$ and one state with size 0. Therefore, an upper bound for $|\{(s, e, s') \in \delta\}|$ is $|E| + \sum_{i=1}^{|E|}(|E| - i)(|E| - i + 1) = 1/3 * |E|(|E|^2 - 1) + |E|$.

Considering both upper bounds, we have $|\mathcal{T}| = |E| + |S| + |\{(s, e, s') \in \delta\}| = |E| + 2^{|E|} + 1/3 * |E|(|E|^2 - 1) + |E|$, implying $|\mathcal{T}| \in O(2^{|S|})$. □

Proposition 6.8 (Reaching upper bound for the output of BES2TS). *For every natural number n there exists a BES $\mathcal{S}$ with $|\mathcal{S}| = n$ such that for all TSs $\mathcal{T}$ with $\mathrm{Tr}(\mathcal{T}) = \mathrm{Tr}(\mathcal{S})$ it holds that $|\mathcal{T}| \in \Theta(2^{|\mathcal{S}|})$.*

Proof. Let n be a natural number, and let $\mathcal{S} = (E, \emptyset, \emptyset)$ with $E = \{e_1, \ldots, e_n\}$, i.e., $|\mathcal{S}| = n$. Then $\mathrm{Tr}(\mathcal{S})$ contains for every subset E' of E all permutations of events in E'. Let $\mathcal{T} = (S, E, s_0, \delta)$ be a TS with $\mathrm{Tr}(\mathcal{T}) = \mathrm{Tr}(\mathcal{S})$. Then $\mathcal{T}$ has to contain at least $2^{|\mathcal{S}|}$ states: To ensure that the traces of $\mathcal{T}$ contain each event at most once, each state of the TS must distinguish which events have led to this state. Thus, for every subset of E, there exists at least one state in $\mathcal{T}$. This implies $|S| \geq |\mathcal{P}(E)| = 2^{|E|} = 2^{|\mathcal{S}|}$ and thus $|\mathcal{T}| \in \Omega(2^{|\mathcal{S}|})$. In combination with Prop. 6.7 this implies $|\mathcal{T}| \in \Theta(2^{|\mathcal{S}|})$ □

In conclusion, given a linear TS $\mathcal{T}$, we can construct a BES $\mathcal{S}$ with $|\mathcal{S}| \in O(|\mathcal{T}|^2)$, see Prop. 6.6. On the other hand, for every n there exists a BES such that for all trace-equivalent TSs $\mathcal{T}$ it holds $|\mathcal{T}| \in \Theta(2^{|\mathcal{S}|})$. By Rem. 6.5, these results can be translated to (linear) FTSs and BFMs: Given a linear FTS $\mathcal{F} = (\mathcal{M}, \mathcal{T}, \mu)$, we can construct a BFM/FES $\mathcal{X}$ with $|\mathcal{X}| \in O(|\mathcal{M}| + |\mathcal{T}|^2)$. Thus, BFMs/FESs are at least as succinct as linear FTSs. On the other hand, for every n there exists an FES $\mathcal{E} = (\mathcal{F}, \mathcal{S}, \lambda, \nu)$ with $|\mathcal{E}| = n$ (and equivalently a BFM $\mathcal{B}$ with $\mathrm{fes}(\mathcal{B}) = (\mathcal{F}, \mathcal{S}, \lambda, \nu)$ and $|\mathcal{B}| = n$) such that for all trace-equivalent FTSs $\mathcal{F}$ it holds $|\mathcal{F}| \in \Theta(|\mathcal{M}| + 2^{|\mathcal{S}|})$. This shows that FTSs are less succinct than BFMs.

Answer to RQ1.2: How do the sizes of trace-equivalent linear FTSs and BFMs compare?

BFMs are at least as succinct as linear FTSs. FTSs are less succinct than BFMs, i.e., for certain BFMs any translation scheme leads to an exponential blow-up in size.

6.3 RQ2: In Practice, How Does the Size of a BFM Compare to the Size of an FTS?

We address this research question by applying our implementation of Alg. 5 to the subject systems introduced in Sect. 6.1.3. Our implementation, along with deviations from the original algorithm, is discussed in Sects. 6.1.4 and 6.1.5. As described before, we consider single systems and parallel compositions where we distinguish between compositions that synchronize events and those that do not. In the absence of synchronization, all events interleave.

For each system, we report statistics for both the input FTS and the resulting BFM in Tbl. 1. To compute their sizes, we use the sizes defined in Def. 6.4, where we exclude the sizes of FMs since

Table 1. Evaluation results of applying Algorithm 5 on different case studies. For each system, we report metrics from the input FTS $\mathcal{F}$—number of events ($|E|$), states ($|S|$), and transitions ($|\delta|$), along with their total ($|\mathcal{F}|$)—and the resulting BFM $\mathcal{B}$—number of events ($|E|$), conflicts ($|\#|$, along with $|\#_{max}|$, the size of the largest conflict), and causalities ($|\mapsto|$), along with their total (($|\mathcal{B}|$). Execution time reflects the duration of the translation process.

System	FTS Metrics |E|	|S|	$\|\delta\|$	$\|\mathcal{F}\|$	BFM Metrics |E|	|#| ($\|\#_{max}\|$)	$\|\mapsto\|$	$\|\mathcal{B}\|$	Exec. Time
Systems without composition									
Cleaning Robot	10	6	10	26	10	6 (8)	6	22	1.388 ms
Card Payment Terminal	17	13	17	47	17	8 (14)	16	41	6.616 ms
Coffee Vending Machine	23	16	23	62	23	12 (21)	21	56	5.526 ms
Soda Vending Machine	13	10	13	36	13	5 (10)	11	29	2.529 ms
Soup Vending Machine	25	14	25	64	25	13 (23)	23	61	13.314 ms
Mine Pump Controller (C)	41	26	41	108	41	37 (39)	40	118	18.576 ms
Vending Machine system in parallel composition (no synchronization)									
Coffee ‖ Soda	36	150	425	611	36	18 (21)	39	93	23.541 s
Coffee ‖ Soup	48	210	697	955	48	40 (21)	60	148	1.484 min
Soda ‖ Soup	38	140	432	610	38	32 (19)	56	126	22.391 s
Coffee ‖ Soda ‖ Soup	61	2100	9700	11861	61	45 (21)	57	163	46.966 h
Vending Machine system in parallel composition with synchronization over actions									
Coffee ‖ Soda	31	77	160	268	31	17 (28)	33	81	1.396 s
Coffee ‖ Soup	42	140	347	529	42	39 (34)	45	126	8.671 s
Soda ‖ Soup	35	100	233	368	35	31 (29)	38	104	3.536 s
Coffee ‖ Soda ‖ Soup	50	727	2294	3071	50	47 (46)	63	160	37.133 min
Mine Pump system in parallel composition (no synchronization)									
C ‖ M	43	78	175	296	43	37 (39)	41	121	1.176 s
C ‖ P	43	78	175	296	43	37 (39)	41	121	1.130 s
C ‖ S	93	68	93	254	93	74 (78)	89	256	165.690 ms
C ‖ W	45	104	268	417	45	38 (39)	42	125	4.328 s
C ‖ M ‖ W	47	312	1012	1371	47	38 (39)	43	128	2.678 min
C ‖ P ‖ M	45	234	681	960	45	37 (39)	42	124	50.854 s
C ‖ P ‖ W	47	312	1012	1371	47	38 (39)	43	128	2.819 min
C ‖ S ‖ M	95	204	415	714	95	74 (78)	90	259	13.884 s
C ‖ S ‖ P	93	204	551	848	93	74 (74)	81	248	31.862 s
C ‖ S ‖ W	99	272	1188	1559	99	78 (78)	89	266	5.573 min
C ‖ P ‖ M ‖ W	49	936	3660	4645	49	38 (39)	44	131	1.794 h
C ‖ S ‖ M ‖ W	101	816	4108	5025	101	78 (78)	90	269	3.504 h
C ‖ S ‖ P ‖ M	95	612	2061	2768	95	74 (74)	82	251	19.370 min
C ‖ S ‖ P ‖ W	99	816	4652	5567	99	78 (74)	81	258	4.547 h
C ‖ S ‖ P ‖ M ‖ W	101	2448	15588	18137	101	78 (74)	82	261	163.586 h
Mine Pump system in parallel composition with synchronization over actions									
C ‖ M	33	66	137	236	33	30 (29)	31	94	642.395 ms
C ‖ P	37	27	38	102	37	35 (35)	36	108	16.094 ms
C ‖ W	38	68	130	236	38	40 (35)	36	114	509.015 ms
C ‖ M ‖ W	40	204	526	770	40	40 (35)	37	117	19.659 s
C ‖ P ‖ M	39	81	168	288	39	35 (35)	37	111	908.914 ms
C ‖ P ‖ W	34	60	110	204	34	31 (31)	32	97	260.106 ms
C ‖ S ‖ M	66	150	292	508	66	53 (57)	61	180	4.111 s
C ‖ S ‖ P	71	55	71	197	71	56 (59)	67	194	58.699 ms
C ‖ S ‖ W	90	152	276	518	90	78 (74)	85	253	3.754 s
C ‖ P ‖ M ‖ W	36	180	450	666	36	31 (31)	33	100	12.822 s
C ‖ S ‖ M ‖ W	92	456	1132	1680	92	78 (74)	86	256	3.181 min
C ‖ S ‖ P ‖ M	73	165	323	561	73	56 (59)	68	197	5.360 s
C ‖ S ‖ P ‖ W	68	112	198	378	68	53 (60)	63	184	1.217 s
C ‖ S ‖ P ‖ M ‖ W	70	336	818	1224	70	53 (60)	64	187	1.188 min

they are the same for FTSs and their trace-equivalent BFMs. More concretely, for each FTS, we include the number of events, the number of states, and the number of transitions along with their sum. For each BFM, we include the number of events, the number of conflicts (each represented as

a biclique (A, B)), the number of causality relations, and their total count. We also include the size of the largest conflict (calculated as $|A| + |B|$) for reference, even if it is not part of our global BFM metric. The last column indicates the total execution time for the translation algorithm.

Whereas RQ1 was limited to strictly linear FTS inputs, our implementation generalizes the LinFTS2BFM translation by lifting this restriction. Specifically, we support FTSs that are either linear or the result of parallel compositions of linear systems. This generalization highlights a key strength of BFMs: in systems with significant interleaving, BFMs offer a far more compact representation than FTSs. This is reflected in the size reduction from the FTS ($|\mathcal{F}|$) to the corresponding BFM ($|\mathcal{B}|$). This advantage becomes particularly apparent in large systems. For instance, we observe a size reduction of up to 98.56% in the largest case study, $C||S||P||M||W$ (non-synchronized). The BFM size never exceeds 300, whereas the corresponding FTS typically involves hundreds or even thousands of elements, even in medium-sized systems. Nevertheless, we also observe meaningful size reductions in smaller systems such as the *Cleaning Robot* and the *Card Payment Terminal*. Overall, we observe a median size reduction of 70.02%, with a standard deviation of 32.66%.

To statistically assess the size reduction between the FTS and BFM models, we performed a paired t-test [66, 75] that compares the distributions of model sizes before and after the translation with the algorithm LinFTStoBFM.

- **Null hypothesis (H_0):** The translation algorithm LinFTStoBFM returns models of equal size.
- **Alternative hypothesis (H_1):** The translation algorithm LinFTStoBFM returns models of different sizes.

A t-statistic close to 0 indicates no difference in model sizes, a negative value indicates smaller FTS models, while a positive value indicates smaller BFM models. In our case, the test yielded a t-statistic of 2.9493 and a p-value of 0.0026. Since the p-value is less than the significance threshold $\alpha = 0.05$, we can safely reject H_0. The positive t-statistic confirms that LinFTStoBFM produces statistically significantly more succinct BFM models, as expected.

The number of events remains the same across both models, as expected. The more informative comparison is between the number of FTS transitions (δ) and BFM causalities ($|\mapsto|$). In most cases, BFMs contain fewer causalities than the number of transitions in the FTS. However, we identify three notable exceptions across variants of the Mine Pump system: C, $C||S$ (without synchronization), and $C||P$ (synchronized). In these cases, the BFM is slightly larger than the FTS. This can be explained as follows: the translation retains nearly the same number of causalities as the number of FTS transitions, but BFMs additionally require explicit representation of conflicts. When there is no significant gain in representing transitions as causalities, the overhead from encoding conflicts can lead to a small increase in size.

In conclusion, we affirm that in nearly all cases, in practice, BFMs are strictly more succinct than the flattened FTSs. For analysis tasks on FTSs (such as model checking [36, 39], testing [48, 52], or model translation as done here), the dominant cost typically lies in graph traversal, which depends on the number of states and transitions. For BFMs, we can expect similar tasks to depend primarily on the number of causalities, with conflicts playing only a minor role (graph traversal versus table lookup). Therefore, even in the few cases where BFMs are not smaller than FTSs, we expect that the BFM structure will offer a practical advantage.

Answer to RQ2: In Practice, How Does the Size of a BFM Compare to the Size of an FTS?

In practice, we observe that BFMs are significantly more succinct than FTSs in nearly all cases (40 out of 43). The size reductions are particularly notable in large, highly interleaved systems, where BFMs compress the representation by up to 98.56%. A paired t-test further confirmed that our FTS to BFM translation yields models that are on average significantly more succinct.

6.4 Threats to Validity

Theoretical Evaluation (RQ1). Some FTSs in the literature are not linear FTS, i.e., they have traces in which events occur more than once. However, BFMs/FESs as we define them in this paper, are not able to reflect these traces. One way of overcoming this limitation, which is already present in the literature on (bundle) event structures [61], is distinguishing between actions and events. In this case, an event structure contains a set of events E, a set of actions A (also called the *alphabet* of E), and a labelling function $l : E \to A$ which labels every event with an action. An action then represents something the system can do, while an event, labeled with an action a, represents an occurrence of a in a system run. Thus, several events might represent the occurrence of the same action, and an action might not be represented in a system run at all if there exists no event labeled with this action. Furthermore, conflicts and causalities are defined over (sets of) events, not actions, so events labeled with the same action might have different constraints on their occurrence in a trace. In this way, we can model traces where the same behavior (now in form of actions) may occur more than once.

This allows us to translate (non-linear) acyclic FTSs to BFMs/FESs: What we call "events" in this paper is mostly called "actions" in the FTS literature [40], so let us also call it "actions" now. Then the set of actions of an acylic FTS corresponds to the set of actions of an FES. For each occurrence of an action in the acyclic FTS we can include a new event in the set of events of the FES and label it with the corresponding action. This is reminiscent of labeling actions with fresh tags in reversible process algebras [45, 84]. The definition of conflicts and causalities can be done in the same way as in Alg. 3.

Using FESs where we distinguish between actions and events will also still allow us to go from an FES to an (acyclic) FTS using a modified version of Alg. 4: The set of events of the FTS will correspond to the set of actions of the FES, and the labeling of the transitions can be done similarly as in Alg. 4 where we use the action with which the event of the FES is labeled instead of the event itself.

Empirical Evaluation (RQ2). Concerning the empirical evaluation, we identified several threats to validity on which we will shortly elaborate. We also discuss the measures we have taken (or we plan to take in future research) to address them.

First, there is a threat regarding the representativeness of the subject systems. Some of the FTSs used in our evaluation are artificially constructed by researchers to showcase the benefits of FTSs [33] and therefore do not pose any threat with respect to bias towards our proposed approach. We selected these systems because they are widely used and recognized in the literature. However, they do not necessarily reflect realistic industrial models. This issue is also symptomatic of a broader challenge in the community: the lack of diverse and representative datasets [95]. Despite this limitation, our study makes a step forward by evaluating on larger FTSs than those typically used for similar tasks. Also the phenomena that lead to significant reduction in model size (interleaving and synchronization) are common among different models. While our subject

systems are not perfect and there is scope for future investigation with real-world models, they demonstrate the scalability of our approach with respect to some common modeling phenomena.

Another risk is the presence of faults in algorithms and code. For algorithms, we mitigated this risk by producing formal proofs of their correctness. For implementation, we mitigated the risk of bugs by performing extensive testing and implementing the algorithms in a modular way which allowed us to also test them modularly.

As FTSs and BFMs are non-isomorphic representations, there is also a risk that our choice of metric for comparing them is biased towards one or the other. To make this transparent, we report all structural elements of the FTSs and BFMs in Tbl. 1. The aggregated metrics $|\mathcal{F}|$ and $|\mathcal{B}|$ are computed by summing up all these elements. However, one could argue that certain elements do not have direct counterparts in the two representations. For instance, FTS states have no clear analogue in BFMs. Conversely, BFM conflicts raise a similar concern. Additionally, they are treated with the same weight as causalities in the aggregated sum, despite typically being less costly in practice. Operations such as checking whether two events are in conflict, or retrieving all events that are in conflict with a given one, can be performed very efficiently. Excluding states or conflicts from the total metric would be unjustified, as they remain essential to the structure and semantics of each model. Since both FTSs and BFMs encode information about which transition or event can be taken in which product, we excluded the product knowledge in FTSs and the product constraints in BFMs from the comparison metric, as these represent equivalent information expressed differently.

We evaluated two variants of parallel composition, with and without event synchronization. Had we only considered interleaving (i.e., without synchronization), our approach would appear disproportionately advantageous relative to FTSs. Moreover, in realistic scenarios, it is unlikely that one would compose large systems without introducing at least some degree of synchronization, further motivating the inclusion of both cases in our evaluation.

7 RELATED WORK

In this section, we first review the related work regarding compositionality of structural and behavioral models of SPLs in Sects. 7.1 and 7.2, respectively. Subsequently, we focus on unifying structural and behavioral modeling of SPLs in Sect. 7.3. Finally, we provide an overview of the available body of work concerning the analysis of behavioral models of SPLs in Sect. 7.4.

7.1 Compositionality of Structural Models of SPLs

The composition of FMs has been studied in the literature [2–4], with the aim of scaling up the analysis by separating different concerns into different FMs. In this approach, composition is realized through an insert and a merge operator. The insert operator either inserts newly created elements, or (a part of) a so-called aspect, into a base FM. The merge operator merges two FMs that share several features. In our approach, we also take a compositional view of FMs: we construct an FM incrementally in a bottom-up fashion, and we also allow to insert single features or new FMs into an existing FM. Other approaches to editing and composing features follow a delta-oriented approach [32, 44, 79] in which new structures are added, removed, or modified in a core feature structure. We improve upon the state of the art, since previous work considers neither feature constraints nor behavior in the compositional approach.

7.2 Compositionality of Behavioral Models of SPLs

There is a plethora of behavioral models originating from foundational models of computation (e.g., finite automata [90]) and concurrency theory (e.g., labeled transition systems, process algebras [1], and event structures [81, 105]). Some models have been extended to capture behavioral variability (cf. a formal comparison of a number these extensions [14, 23]); notable examples are

FTSs [35, 40], featured finite state machines (FFSMs) [59], and variants of modal transition systems [8, 16, 72, 73] and of featured process calculi [56, 60, 62, 74, 102]. In our research, we started by considering composition operators of FMs on FTSs and FFSMs, both of which are equipped with a notion of parallel composition [9]. However, their composition mechanisms do not allow for associating behavior to features and to capture the intricate interactions among different features in a composition (e.g., reminiscent of conjunctive or disjunctive operators as used in FMs). In other words, one has to repeat substantial pieces of behavior across different features. Particularly, when there are cross-tree causal dependencies (e.g., a part of a feature behavior happening before and another part appearing after the behavior of another feature), the behavior of each feature has to be copied in both specifications to allow for synchronization between them. The composition operators provided by SPL process calculi [56, 60, 62, 74, 102] offer more flexibility, yet there is no known technique to date to provide a behavioral specification of individual features. Event structures [80, 105] provided us with the flexibility required to capture cross-tree constraints among the behavior of different features and hence, we opted for unifying FMs and event structures into BFMs. Among the variants of event structures, we chose *bundle event structures* [71], because they provide us with the ability to express the same types of compositions as those provided by structural models such as FMs (e.g., conjunctive and disjunctive compositions). This is essential for many practical applications, including our case studies.

7.3 Unifying Structural and Behavioral Modeling of SPLs

We are aware of only few approaches aimed at unifying structural and behavioral variability.

Supervisory Control Theory (SCT) [86] is a model-based approach to automatically synthesize controller from specification. It produces correct-by-construction controller, typically in the form of automata. SCT has been applied to SPLs [18]; there, it was shown how the CIF 3 toolset [103] can deal with structural as well as behavioral variability. The approach manages both product structural configuration, and product behavior synthesis. To achieve this it uses behavioral requirements (in the form of admissible scenarios given by state invariants, event orderings, and guards on events akin to FTSs). Also, the SCT approach [11] unifies structural and behavioral variability of service product lines. It guarantees that the synthesized orchestrations respect both an FM and contract-based service product line behavior (modeled as featured modal contract automata). The approach has also been implemented in the FMCAT tool [13] Both approaches were extended to cater for dynamic feature reconfigurations [12, 98]. These approaches use the flattened representation of FMs in propositional logic, whereas our approach is inherently compositional and paves the way for compositional synthesis of components.

Clafer with behavior [68] is a general-purpose modeling language for both structural and behavioral variability. It unifies FMs with classes, meta-models, and behavior by providing feature modeling by means of a special-purpose constraint language akin to Alloy [67]. Behavioral variability is introduced by means of hierarchical UML state diagrams and automata (in a superimposed fashion akin to FTSs). Additional behavioral constraints have the form of temporal logic patterns enabling (bounded) model checking. Clafer offers first-class support for architectural modeling, but no support for mapping features to behavior, for compositional behavior, or controller synthesis. The former two gaps are filled by the present work, enabling a compositional approach to behavior analysis and synthesis.

7.4 Analysis of Behavioral Models of SPLs

SPLs are concerned with massively reused critical software components. Thus, it is important to demonstrate their correct structural and behavioral (re)configuration. The automated analysis of structural variability models has more than three decades of history. Examples of such analyses

include detecting dead or false optional *features* or edits in FMs [21, 100, 101]. However, the automated analysis of behavioral variability models has a shorter history. There are only few available results, mostly concerning family-based model checking [16, 17, 19, 20, 31, 39, 54, 63, 73]. More recently, some attention has been paid to ambiguity detection in behavioral models, including the identification of dead or false optional *transitions* and deadlocks in FTSs [15]. In addition, a number of family-based (model-based) testing approaches have been proposed. These techniques address automated test case generation in the context of variability. Some of these works [48–51, 53] explore techniques for test case selection and generation based on coverage criteria, sampling strategies, or similarity and dissimilarity metrics. Others focus on mutation analysis, aiming to support test generation or evaluate the fault-detection ability of existing test suites [47, 52].

Other analysis approaches operate directly on the source code of SPLs, often obtained by adapting existing tools to deal with variability. In this case, structural analysis is related to static (program) analysis [42, 82], which includes the detection of bugs in the code (e.g., using a variable before initialization) but also the identification of code that is redundant or unreachable, whereas behavioral analysis includes software model-checking tools such as ProMoVer [91] with variability annotations [88], the SPLverifier [7] tool chain for Java and CPAchecker [24] for C code.

We are not aware of any other related work that unifies the structural and behavioral analysis in a compositional manner (since the underlying models for such analyses has not been available). Our integrated proposal of BFMs enables the analysis of both structural and behavioral anomalies.

8 DISCUSSION

In the following, we include a short discussion on translating FTSs in Sect. 8.1 and on composing and decomposing them in Sect. 8.2. Furthermore, we provide a discussion about how BFMs can be modified to include traces where events can appear more than once in Sect. 8.3.

8.1 On the Complexity of Translating FTSs

Our implementation for translating parallel compositions of linear FTSs to BFMs, see Sect. 6.1.4, requires iterating over all pairs of FTS transitions, resulting in an overall complexity of $O(t^2)$, where t denotes the number of transitions. This is reflected in the execution times of the translation, see Tbl. 1. It takes a few milliseconds for small models but nearly a week for the largest analyzed system (with 15588 transitions). There are several ways to optimize the translation algorithms, which was not the goal of this paper. For Alg. 5, efficiency could be improved by pre-selecting candidate transition pairs instead of considering all pairs, thereby avoiding unnecessary iterations. In Alg. 7, the naive configuration generator is the most computationally expensive step and the primary target for optimization. Another avenue would be to parallelize suitable parts of the algorithms.

Model translation—and, more broadly, model construction—is inherently expensive. Although there is growing work on automatically learning FTSs [43, 58, 94, 96], most existing benchmarks, such as [95], focus on relatively small FTSs. In [15], FTS benchmarks with up to 11236 states are analyzed. Our work provides further empirical evidence of both the feasibility and the computational challenges involved in handling such large models.

8.2 On Composing and Decomposing FTSs

To the best of our knowledge, only one composition operator for FTSs has been presented in the literature. This is exactly the parallel composition of FTSs presented in Def. 6.1. In the literature, the parallel composition is usually defined between two FTSs, where the FTSs additionally contain atomic propositions and a labeling function for labeling states with a set of atomic propositions [35, Def. 9].

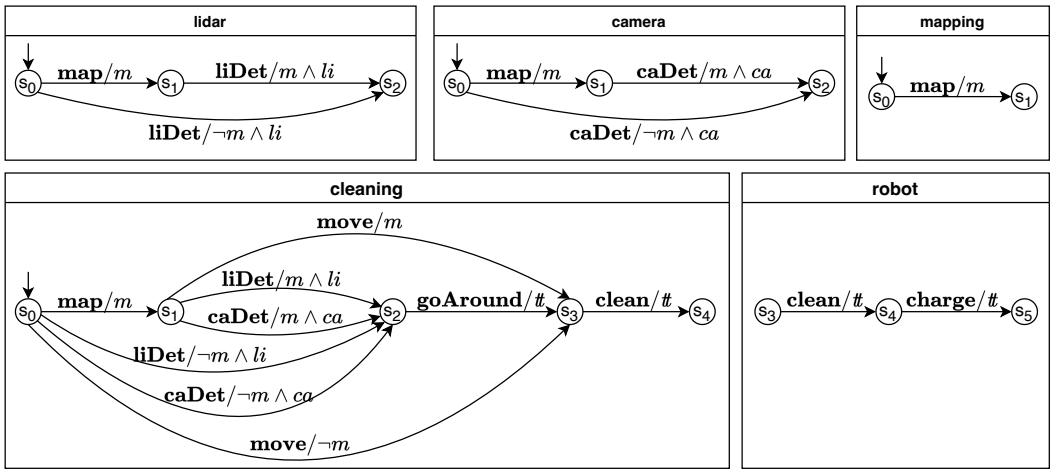

Fig. 10. A decomposed version of the FTS for the cleaning robot SPL (see Example 2.21) into an FTS for each feature.

However, to the best of our knowledge, there is no work on decomposing an FTS into FTSs for each feature, reflecting the behavior of each particular feature. We provide evidence, using the cleaning robot SPL, that this can lead to a significant blowup in the sizes of the feature FTSs because behavior has to be shared between FTSs.

Example 8.1 (Decomposing the FTS of the cleaning robot SPL). Consider the FTS $\mathcal{F}_r$ of the cleaning robot as defined in Example 2.21 and shown in Fig. 3. Then we assign events (and thus behavior) to the features of the feature model as in the BFM of the cleaning robot, see Example 4.5. That is, the FTSs $\mathcal{F}_i$, $i \in \{r, m, c, o, li, ca\}$, for the features *robot*, *mapping*, *cleaning*, *obstacleDetection*, *lidar*, and *camera*, respectively, have to contain at least the sets of events $\{charge\}$, $\{map\}$, $\{move, goAround, clean\}$, $\emptyset$, $\{liDet\}$, and $\{caDet\}$, respectively.

However, the parallel composition of the $\mathcal{F}_i$, $i \in \{r, m, c, o, li, ca\}$, has to be $\mathcal{F}_r$. This means that some $\mathcal{F}_i$ have to share behavior, i.e., include the same transitions, for a correct synchronization: Assume that $\mathcal{F}_m$ contains only a transition labeled with *map*. Then the FTSs $\mathcal{F}_{ca}$ and $\mathcal{F}_{li}$ have to include the event *map* in addition to *caDet* and *liDet*, respectively, because the occurrence of these events depends on the event *map*. Now the FTS $\mathcal{F}_c$ has to ensure that the event *move* has to happen after the event *map* if the feature *mapping* is included, and that either *liDet* or *caDet* happens before the event *goAround*. Lastly, the FTS $\mathcal{F}_r$ only has to ensure that the event *clean* happens before the event *charge*. A graphical representation of the resulting FTSs can be seen in Fig. 10.

Note that other decompositions into FTSs related to the features of the feature model are of course possible, but they are likely to require shared behavior as well.

As in Sect. 6.3, we can compare the size of the "flattened" FTS of the cleaning robot SPL, as used throughout the paper, and its BFM/FES, where we use the sum of all events, conflicts and causalities for the BFM/FES. The result is reported in the last row of Tbl. 2. However, we can also compare the sizes of the models modeling the behavior of single features, i.e., the sizes of the FTSs in Fig. 10 and sub-BFMs in Example 4.5. For the sub-BFMs, we only need to consider the BESs modelling the behavior of the root feature of the considered sub-BFM, as this represents the behavior of the feature. The results are shown in Tbl. 2, where $|\mathcal{F}|$ and $|\mathcal{B}|$ are the sum of the considered elements (events, states and transitions for FTSs and events, conflicts and causalities for BFMs), and the total

$\sum$ is the sum of all sums for the single-feature models. It can be seen that, even though the sizes of the "flattened" models are similar, the sizes of the models per feature are mostly significantly bigger for FTSs. Thus, when comparing the sum of the sizes of the models per feature, shown in *Tot.*, the sum of all FTS models is 2.7 times higher than the sum of all BFM/FES models.

Table 2. Sizes of "flattened" and compositional FTSs $\mathcal{F}$ and BFMs/FESs $\mathcal{B}$ for the cleaning robot SPL

	FTS					BFM				
System	$\|\mathbf{E}\|$	$\|\mathbf{S}\|$	$\|\delta\|$	$\|\mathcal{F}\|$	$\sum$	$\|\mathbf{E}\|$	$\|\#\|$	$\|\mapsto\|$	$\|\mathcal{B}\|$	$\sum$
lidar	2	3	3	8		1	0	0	1	
camera	2	3	3	8		1	0	0	1	
mapping	1	2	1	4	46	1	0	0	1	17
cleaning	5	5	9	19		3	1	1	5	
robot	2	3	2	7		1	2	6	9	
"flattened"	7	6	10	23		7	3	7	17	

The decomposition of the FTS of the cleaning robot SPL illustrated in Example 8.1 shows that, since each event can be associated with multiple features, decomposing an FTS intrinsically requires to duplicate transitions among the component FTSs to ensure a correct synchronization. This redundancy, which entangles the behavior of different features, makes it harder to reuse the decomposed FTSs for other SPLs and exacerbates the state explosion problem related to the analyses of behavioral models.

8.3 On the Location of Events in BFMs

BFMs enable us to determine which behavior is associated with which feature, and how the behavior of different features interact. This is not possible in FTSs, where the behavior of all features is modeled at the same time. This also has an effect on our translation algorithm from an FTS to an FES: Every event is associated with the root feature of the feature model, even if it might be possible to use domain knowledge about which feature exhibits which behavior to associate events to other features. Since FTSs do not contain information about which behavior is associated with which feature, it is impossible to automatically determine which feature an event should be associated with. The translation from FTS to FES/BFM will therefore lead to an FES/BFM where all behavior is associated with the root feature. This also means we lose information about the location of behavior when going from an FES/BFM to an FTS.

9 CONCLUSION AND FUTURE WORK

To conclude the paper, we provide a summary and reflections as well as ideas for future work.

9.1 Summary and Reflections

In this paper, we propose Behavioral Feature Models as a compositional, unifying model for both structural and behavioral variability in SPLs. Our proposal fills two significant gaps in the literature: (i) the lack of compositionality in behavioral variability; and (ii) the isolated development of structural and behavioral variability. We fill this gap by introducing the concept of bundle event structures in FMs where we extend an FM by attaching a bundle event structure to every feature and introducing constraints among the events of bundle event structures associated to different features (which we call a BFM). In fact, we show that this is the same as extending bundle event structures to cater for variability by mapping each event to a feature and a feature expression over

features of an FM (which we call a featured event structure (FES)). Thus, BFMs introduce behavioral variability in a structural model, while featured event structures introduce structural variability in a behavioral model, but we show that the result is isomorphic.

Throughout the paper, we use the running example of a cleaning robot SPL to illustrate the idea of our approach and show its applicability. We provide algorithms to translate (parallel compositions of) linear FTSs to BFMs and BFMs to FTSs and show their trace equivalence. Furthermore, we provide an evaluation of our approach consisting of both a theoretical an empirical evaluation, where the empirical evaluation shows that BFMs can reduce the size of FTSs by up to 98.56%. The empirical evaluation was conducted on a large set of FTS benchmarks.

9.2 Future Work

BFMs enable a wealth of future developments which we plan to address in our ongoing research. As already outlined in other sections, we would like to address the limitation of only considering (parallel compositions of) linear FTSs for the algorithm from FTSs to BFMs by distinguishing between actions and events in FESs and characterizing the class of FTSs that are trace equivalent to BFMs/FESs such that we can provide algorithms to go back and forth between them.

To extend BFMs with more composition operators [9] currently not available in FMs, we would like to identify and characterize typical composition patterns in SPLs. Ideally, these composition patterns, translated into composition operators in BFMs, would compose both structural and behavioral aspects of SPLs.

We would like to explore how the compositional structure of BFMs can be leveraged for compositional analysis [76, 77]. This would allow (i) analysis results to be reused among different SPLs, and (ii) focusing the analysis effort on the effect of changes in the case of dynamic SPLs. Synthesizing feature behavior from global or local properties of the system [25, 30, 57] as well as compositional model learning [64, 70, 78, 87] are other promising areas of future research.

Finally, we plan to study how to verify the correctness of feature behavior and BFMs, based on some of the well-established family-based analysis techniques mentioned earlier in Sects. 2 and 7.

ACKNOWLEDGMENTS

This work was supported by the European Union's Horizon 2020 Framework Programme through the MSCA network REMARO (Grant Agreement No 956200), by the Italian project NODES (which has received funding from the MUR – M4C2 1.5 of PNRR with grant agreement no. ECS00000036) and by the CNR project "Formal Methods in Software Engineering 2.0", CUP B53C24000720005; by the EPSRC project on Verified Simulation for Large Quantum Systems (VSL-Q), grant reference EP/Y005244/1; by the EPSRC project on Robust and Reliable Quantum Computing (RoaRQ), Investigation 009 Model-based monitoring and calibration of quantum computations (ModeMCQ) grant reference EP/W032635/1; by InnovateUK QAssure; and by the ITEA/InnovateUK projects GENIUS and ITEA/InnovateUK GreenCode.

During the preparation of this work, the authors used ChatGPT-4 (OpenAI) to provide ideas for how to prove Prop. 6.8, suggest algorithms and design patterns from the literature for addressing specific problems in the implementation, provide guidance on structuring the project's README, adopt standard software engineering practices for testing, and assist in formatting output data. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

REFERENCES

[1] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jirí Srba. 2007. *Reactive Systems: Modelling, Specification and Verification.* Cambridge University Press, UK. https://doi.org/10.1017/CBO9780511814105

[2] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2010. Comparing Approaches to Implement Feature Model Composition. In *Proceedings of the 6th European Conference, on Modelling Foundations and Applications (ECMFA'10) (Lecture Notes in Computer Science, Vol. 6138)*, Thomas Kühne, Bran Selic, Marie-Pierre Gervais, and François Terrier (Eds.). Springer, Germany, 3–19. https://doi.org/10.1007/978-3-642-13595-8_3

[3] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2010. Composing Feature Models. In *Proceedings of the 2nd International Conference on Software Language Engineering (SLE'09) (Lecture Notes in Computer Science, Vol. 5969)*, Mark van den Brand, Dragan Gasevic, and Jeff Gray (Eds.). Springer, Germany, 62–81. https://doi.org/10.1007/978-3-642-12107-4_6

[4] Mathieu Acher, Benoît Combemale, Philippe Collet, Olivier Barais, Philippe Lahire, and Robert B. France. 2013. Composing Your Compositions of Variability Models. In *Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems (MODELS'13) (Lecture Notes in Computer Science, Vol. 8107)*, Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter J. Clarke (Eds.). Springer, Germany, 352–369. https://doi.org/10.1007/978-3-642-41533-3_22

[5] Jérôme Amilhastre, Marie-Catherine Vilarem, and Philippe Janssen. 1998. Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. *Discrete applied mathematics* 86, 2–3 (1998), 125–144. https://doi.org/10.1016/S0166-218X(98)00039-0

[6] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, Germany. https://doi.org/10.1007/978-3-642-37521-7

[7] Sven Apel, Hendrik Speidel, Philipp Wendler, Alexander von Rhein, and Dirk Beyer. 2011. Detection of feature interactions using feature-aware verification. In *Proceedings of the 26th International Conference on Automated Software Engineering (ASE'11)*. IEEE, USA, 372–375. https://doi.org/10.1109/ASE.2011.6100075

[8] Patrizia Asirelli, Maurice H. ter Beek, Alessandro Fantechi, and Stefania Gnesi. 2011. Formal Description of Variability in Product Families. In *Proceedings of the 15th International Software Product Lines Conference (SPLC'11)*. IEEE, USA, 130–139. https://doi.org/10.1109/SPLC.2011.34

[9] J. C. M. Baeten, T. Basten, and M. A. Reniers. 2010. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge Tracts in Theoretical Computer Science, Vol. 50. Cambridge University Press, UK.

[10] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press, USA.

[11] Davide Basile, Maurice H. ter Beek, Pierpaolo Degano, Axel Legay, Gian Luigi Ferrari, Stefania Gnesi, and Felicita Di Giandomenico. 2020. Controller synthesis of service contracts with variability. *Science of Computer Programming* 187 (2020), 102344. https://doi.org/10.1016/j.scico.2019.102344

[12] Davide Basile, Maurice H. ter Beek, Felicita Di Giandomenico, and Stefania Gnesi. 2017. Orchestration of Dynamic Service Product Lines with Featured Modal Contract Automata. In *Proceedings of the 21st International Systems and Software Product Line Conference (SPLC'17)*, Vol. 2. ACM, USA, 117–122. https://doi.org/10.1145/3109729.3109741

[13] Davide Basile, Maurice H. ter Beek, and Stefania Gnesi. 2018. Modelling and Analysis with Featured Modal Contract Automata. In *Proceedings of the 22nd International Systems and Software Product Line Conference (SPLC'18)*, Vol. 2. ACM, USA, 11–16. https://doi.org/10.1145/3236405.3236408

[14] Maurice H. ter Beek, Ferruccio Damiani, Stefania Gnesi, Franco Mazzanti, and Luca Paolini. 2019. On the Expressiveness of Modal Transition Systems with Variability Constraints. *Science of Computer Programming* 169 (2019), 1–17. https://doi.org/10.1016/j.scico.2018.09.006

[15] Maurice H. ter Beek, Ferruccio Damiani, Michael Lienhardt, Franco Mazzanti, and Luca Paolini. 2022. Efficient static analysis and verification of featured transition systems. *Empirical Software Engineering* 22, 1 (2022), 10:1–10:43. https://doi.org/10.1007/s10664-020-09930-8

[16] Maurice H. ter Beek, Alessandro Fantechi, Stefania Gnesi, and Franco Mazzanti. 2016. Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints. *Journal of Logical and Algebraic Methods in Programming* 85, 2 (2016), 287–315. https://doi.org/10.1016/j.jlamp.2015.11.006

[17] Maurice H. ter Beek, Axel Legay, Alberto Lluch Lafuente, and Andrea Vandin. 2020. A Framework for Quantitative Modeling and Analysis of Highly (Re)configurable Systems. *IEEE Transactions on Software Engineering* 46, 3 (2020), 321–345. https://doi.org/10.1109/TSE.2018.2853726

[18] Maurice H. ter Beek, Michel A. Reniers, and Erik P. de Vink. 2016. Supervisory Controller Synthesis for Product Lines Using CIF 3. In *Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques (ISoLA'16) (Lecture Notes in Computer Science, Vol. 9952)*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer, Germany, 856–873. https://doi.org/10.1007/978-3-319-47166-2_59

[19] Maurice H. ter Beek and Erik P. de Vink. 2014. Towards Modular Verification of Software Product Lines with mCRL2. In *Proceedings of the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'14) (Lecture Notes in Computer Science, Vol. 8802)*, T. Margaria and B. Steffen (Eds.). Springer, Germany, 368–385. https://doi.org/10.1007/978-3-662-45234-9_26

[20] Maurice H. ter Beek, Erik P. de Vink, and Tim A. C. Willemse. 2017. Family-Based Model Checking with mCRL2. In *Proceedings of the 20th International Conference on Fundamental Approaches to Software Engineering (FASE'17) (Lecture Notes in Computer Science, Vol. 10202)*, Marieke Huisman and Julia Rubin (Eds.). Springer, Germany, 387–405. https://doi.org/10.1007/978-3-662-54494-5_23

[21] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: a Literature Review. *Information Systems* 35, 6 (2010), 615–636. https://doi.org/10.1016/j.is.2010.01.001

[22] David Benavides, Chico Sundermann, Kevin Feichtinger, José A Galindo, Rick Rabiser, and Thomas Thüm. 2025. UVL: Feature modelling with the Universal Variability Language. *Journal of Systems and Software* 225 (2025), 112326. https://doi.org/10.1016/j.jss.2024.112326

[23] Harsh Beohar, Mahsa Varshosaz, and Mohammad Reza Mousavi. 2016. Basic behavioral models for software product lines: Expressiveness and testing pre-orders. *Science of Computer Programming* 123 (2016), 42–60. https://doi.org/10.1016/J.SCICO.2015.06.005

[24] Dirk Beyer and M. Erkan Keremoglu. 2011. CPAchecker: A Tool for Configurable Software Verification. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11) (Lecture Notes in Computer Science, Vol. 6806)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer, Germany, 184–190. https://doi.org/10.1007/978-3-642-22110-1_16

[25] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. 2012. Synthesis of Reactive(1) designs. *J. Comput. System Sci.* 78, 3 (2012), 911–938. https://doi.org/10.1016/j.jcss.2011.08.007

[26] Gérard Boudol and Ilaria Castellani. 1991. *Flow models of distributed computations: event structures and nets.* Technical Report RR-1482. INRIA.

[27] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: Finding All Cliques of an Undirected Graph. *Commun. ACM* 16, 9 (1973), 575–577. https://doi.org/10.1145/362342.362367

[28] Rafael Capilla, Jan Bosch, and Kyo Chul Kang (Eds.). 2013. *Systems and Software Variability Management: Concepts, Tools and Experiences.* Springer, Germany. https://doi.org/10.1007/978-3-642-36583-6

[29] Sunil Chandran, Davis Issac, and Andreas Karrenbauer. 2017. On the Parameterized Complexity of Biclique Cover and Partition. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC'16) (LIPIcs, Vol. 63)*, Jiong Guo and Danny Hermelin (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 11:1–11:13. https://doi.org/10.4230/LIPICS.IPEC.2016.11

[30] Wonhyuk Choi, Bernd Finkbeiner, Ruzica Piskac, and Mark Santolucito. 2022. Can Reactive Synthesis and Syntax-Guided Synthesis Be Friends?. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI'22).* ACM, USA, 229–243. https://doi.org/10.1145/3519939.3523429

[31] Philipp Chrszon, Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. 2018. ProFeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Aspects of Comping* 30, 1 (2018), 45–75. https://doi.org/10.1007/s00165-017-0432-4

[32] Dave Clarke, Michiel Helvensteijn, and Ina Schaefer. 2015. Abstract delta modelling. *Mathematical Structures in Computer Science* 25, 3 (2015), 482–527. https://doi.org/10.1017/S0960129512000941

[33] Andreas Classen. 2010. *Modelling with FTS: a Collection of Illustrative Examples.* Technical Report P-CS-TR SPLMC-00000001. University of Namur. https://researchportal.unamur.be/files/1051983/69416.pdf

[34] Andreas Classen. 2011. *Modelling and Model Checking Variability-Intensive Systems.* Ph. D. Dissertation. University of Namur.

[35] Andreas Classen, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, and Jean-François Raskin. 2013. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Transactions on Software Engineering* 39, 8 (2013), 1069–1089. https://doi.org/10.1109/TSE.2012.86

[36] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François Raskin. 2010. 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, USA, 335–344. https://doi.org/10.1145/1806799.1806850

[37] Andreas Classen, Patrick Heymans, and Pierre-Yves Schobbens. 2008. What's in a Feature: A Requirements Engineering Perspective. In *Proceedings of the 11th International Conference on Fundamental Approaches to Software Engineering (FASE 2008)*, José Luiz Fiadeiro and Paola Inverardi (Eds.). Lecture Notes in Computer Science, Vol. 4961. Springer, Berlin, Heidelberg, 16–30. https://doi.org/10.1007/978-3-540-78743-3_2

[38] Paul C. Clements and Linda M. Northrop. 2002. *Software Product Lines: Practices and Patterns.* Addison-Wesley, USA.

[39] Maxime Cordy, Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, and Axel Legay. 2013. ProVeLines: a product line of verifiers for software product lines. In *Proceedings of the 17th International Software Product Line Conference co-located workshops (SPLC'13).* ACM, USA, 141–146. https://doi.org/10.1145/2499777.2499781

[40] Maxime Cordy, Xavier Devroey, Axel Legay, Gilles Perrouin, Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, and Jean-François Raskin. 2019. A Decade of Featured Transition Systems. In *From Software Engineering to Formal Methods and Tools, and Back*, Maurice H. ter Beek, Alessandro Fantechi, and Laura Semini (Eds.). Lecture Notes in Computer Science, Vol. 11865. Springer, Germany, 285–312. https://doi.org/10.1007/978-3-030-30985-5_18

[41] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *Proceedings of the 6th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'12)*. ACM, USA, 173–182. https://doi.org/10.1145/2110147.2110167

[42] Krzysztof Czarnecki and Andrzej Wąsowski. 2007. Feature Diagrams and Logics: There and Back Again. In *Proceedings of the 11th International Conference on Software Product Lines (SPLC'07)*. IEEE, USA, 23–34. https://doi.org/10.1109/SPLINE.2007.24

[43] Carlos Diego Nascimento Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simao. 2021. Learning by sampling: learning behavioral family models from software product lines. *Empirical Software Engineering* 26, 4 (2021), 1–46. https://doi.org/10.1007/s10664-020-09912-w

[44] Ferruccio Damiani, Luca Padovani, Ina Schaefer, and Christoph Seidl. 2018. A core calculus for dynamic delta-oriented programming. *Acta Informatica* 55, 4 (2018), 269–307. https://doi.org/10.1007/S00236-017-0293-6

[45] Vincent Danos and Jean Krivine. 2004. Reversible Communicating Systems. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04) (Lecture Notes in Computer Science, Vol. 3170)*, Philippa Gardner and Nobuko Yoshida (Eds.). Springer, Germany, 292–307. https://doi.org/10.1007/978-3-540-28644-8_19

[46] Xavier Devroey. 2020. VIBeS Case Studies: Featured Transition Systems and Feature Models. Dataset, Zenodo. https://doi.org/10.5281/zenodo.4105900

[47] Xavier Devroey, Gilles Perrouin, Maxime Cordy, Mike Papadakis, Axel Legay, and Pierre-Yves Schobbens. 2014. A variability perspective of mutation analysis. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 841–844. https://doi.org/10.1145/2635868.2666610

[48] Xavier Devroey, Gilles Perrouin, Maxime Cordy, Hamza Samih, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. 2017. Statistical prioritization for software product line testing: an experience report. *Software & Systems Modeling* 16 (2017), 153–171. https://doi.org/10.1007/s10270-015-0479-8

[49] Xavier Devroey, Gilles Perrouin, Axel Legay, Maxime Cordy, Pierre-Yves Schobbens, and Patrick Heymans. 2014. Coverage Criteria for Behavioural Testing of Software Product Lines. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 336–350.

[50] Xavier Devroey, Gilles Perrouin, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. 2015. Covering SPL Behaviour with Sampled Configurations: An Initial Assessment. In *Proceedings of the Ninth International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '15, Hildesheim, Germany, January 21-23, 2015*, Klaus Schmid, Øystein Haugen, and Johannes Müller (Eds.). ACM, New York, NY, USA, 59. https://doi.org/10.1145/2701319.2701325

[51] Xavier Devroey, Gilles Perrouin, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. 2016. Search-based Similarity-driven Behavioural SPL Testing. In *Proceedings of the 10th International Workshop on Variability Modelling of Software-Intensive Systems* (Salvador, Brazil) *(VaMoS '16)*. Association for Computing Machinery, New York, NY, USA, 89–96. https://doi.org/10.1145/2866614.2866627

[52] Xavier Devroey, Gilles Perrouin, Mike Papadakis, Axel Legay, Pierre-Yves Schobbens, and Patrick Heymans. 2016. Featured Model-based Mutation Analysis. In *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*. ACM, USA, 655–666. https://doi.org/10.1145/2884781.2884821

[53] Xavier Devroey, Gilles Perrouin, and Pierre-Yves Schobbens. 2014. Abstract test case generation for behavioural testing of software product lines. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2* (Florence, Italy) *(SPLC '14)*. Association for Computing Machinery, New York, NY, USA, 86–93. https://doi.org/10.1145/2647908.2655971

[54] Aleksandar S. Dimovski, Ahmad S. Al-Sibahi, Claus Brabrand, and Andrzej Wąsowski. 2015. Family-Based Model Checking using Off-the-Shelf Model Checkers. In *Proceedings of the 19th International Software Product Line Conference (SPLC'15)*, Douglas C. Schmidt (Ed.). ACM, USA, 397. https://doi.org/10.1145/2791060.2791119

[55] Clemens Dubslaff, Nils Husung, and Nikolai Käfer. 2024. Configuring BDD Compilation Techniques for Feature Models. In *Proceedings of the 28th ACM International Systems and Software Product Line Conference - Volume A, SPLC 2024, Dommeldange, Luxembourg, September 2-6, 2024*, Maxime Cordy, Daniel Strüber, Mónica Pinto, Iris Groher, Deepak Dhungana, Jacob Krüger, Juliana Alves Pereira, Mathieu Acher, Thomas Thüm, Maurice H. ter Beek, Jessie Galasso-Carbonnel, Paolo Arcaini, Mohammad Reza Mousavi, Xhevahire Tërnava, José A. Galindo, Tao Yue, Lidia Fuentes, and José Miguel Horcas (Eds.). ACM, New York, NY, USA, 209–216. https://doi.org/10.1145/3646548.3676538

[56] Martin Erwig and Eric Walkingshaw. 2011. The Choice Calculus: A Representation for Software Variation. *ACM Transactions on Software Engineering and Methodology* 21, 1 (2011), 6:1–6:27. https://doi.org/10.1145/2063239.2063245

[57] Bernd Finkbeiner, Gideon Geier, and Noemi Passing. 2023. Specification decomposition for reactive synthesis. *Innovations in Systems and Software Engineering* 19, 4 (2023), 339–357. https://doi.org/10.1007/s11334-022-00462-6

[58] Sophie Fortz. 2023. Variability-aware Behavioural Learning. In *Proceedings of the 27th ACM International Systems and Software Product Line Conference (SPLC'23)*, Vol. 2. ACM, USA, 11–15. https://doi.org/10.1145/3579028.3609007

[59] Vanderson Hafemann Fragal, Adenilso Simão, and Mohammad Reza Mousavi. 2019. Hierarchical featured state machines. *Science of Computer Programming* 171 (2019), 67–88. https://doi.org/10.1016/J.SCICO.2018.10.001

[60] Fatemeh Ghassemi and Mohammad Reza Mousavi. 2016. Product line process theory. *Journal of Logical and Algebraic Methods in Programming* 85, 1 (2016), 200–226. https://doi.org/10.1016/J.JLAMP.2015.09.008

[61] Rob J. van Glabbeek and Frits W. Vaandrager. 2003. Bundle Event Structures and CCSP. In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03) (Lecture Notes in Computer Science, Vol. 2761)*, Roberto M. Amadio and Denis Lugiez (Eds.). Springer, Germany, 57–71. https://doi.org/10.1007/978-3-540-45187-7_4

[62] Alexander Gruler, Martin Leucker, and Kathrin D. Scheidemann. 2008. Calculating and Modeling Common Parts of Software Product Lines. In *Proceedings of the 12th International Software Product Lines Conference (SPLC'08)*. IEEE, USA, 203–212. https://doi.org/10.1109/SPLC.2008.22

[63] Alexander Gruler, Martin Leucker, and Kathrin D. Scheidemann. 2008. Modeling and Model Checking Software Product Lines. In *Proceedings of the 10th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'08) (Lecture Notes in Computer Science, Vol. 5051)*, Gilles Barthe and Frank S. de Boer (Eds.). Springer, Germany, 113–131. https://doi.org/10.1007/978-3-540-68863-1_8

[64] Leo Henry, Thomas Neele, Mohammad Reza Mousavi, and Matteo Sammartino. 2025. Compositional Active Learning of Synchronous Systems through Automated Alphabet Refinement. arXiv:2504.16624 [cs.LG]. https://doi.org/10.48550/ARXIV.2504.16624 arXiv:2504.16624 [cs.LG]

[65] Tobias Heß, Sean Niklas Semmler, Chico Sundermann, Jacobo Torán, and Thomas Thüm. 2024. Towards Deterministic Compilation of Binary Decision Diagrams From Feature Models. In *Proceedings of the 28th ACM International Systems and Software Product Line Conference* (Dommeldange, Luxembourg) *(SPLC '24)*. Association for Computing Machinery, New York, NY, USA, 136–147. https://doi.org/10.1145/3646548.3672598

[66] Henry Hsu and Peter A Lachenbruch. 2014. Paired t test. *Wiley StatsRef: statistics reference online* (2014). https://doi.org/10.1002/9781118445112.stat05929

[67] Daniel Jackson. 2006. *Software Abstractions: Logic, Language, and Analysis.* MIT, USA.

[68] Paulius Juodisius, Atrisha Sarkar, Raghava Rao Mukkamala, Michał Antkiewicz, Krzysztof Czarnecki, and Andrzej Wąsowski. 2019. Clafer: Lightweight Modeling of Structure, Behaviour, and Variability. *The Art, Science, and Engineering of Programming* 3, 1 (2019), 657–681. https://doi.org/10.22152/programming-journal.org/2019/3/2

[69] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report CMU/SEI-90-TR-021. Carnegie Mellon University. https://www.sei.cmu.edu/library/feature-oriented-domain-analysis-foda-feasibility-study/

[70] Faezeh Labbaf, Jan Friso Groote, Hossein Hojjat, and Mohammad Reza Mousavi. 2023. Compositional Learning for Interleaving Parallel Automata. In *Proceedings of the 26th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'23) (Lecture Notes in Computer Science, Vol. 13992)*, Orna Kupferman and Pawel Sobocinski (Eds.). Springer, Germany, 413–435. https://doi.org/10.1007/978-3-031-30829-1_20

[71] Rom Langerak. 1992. Bundle event structures: a non-interleaving semantics for LOTOS. In *Proceedings of the 5th IFIP TC6/WG6.1 International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'92) (IFIP Transactions, Vol. C-10)*, Michel Diaz and Roland Groz (Eds.). North-Holland, The Netherlands, 331–346.

[72] Kim G. Larsen, Ulrik Nyman, and Andrzej Wąsowski. 2007. Modal I/O Automata for Interface and Product Line Theories. In *Proceedings of the 16th European Symposium on Programming (ESOP'07) (Lecture Notes in Computer Science, Vol. 4421)*, Rocco De Nicola (Ed.). Springer, Germany, 64–79. https://doi.org/10.1007/978-3-540-71316-6_6

[73] Kim Lauenroth, Klaus Pohl, and Simon Töhning. 2009. Model Checking of Domain Artifacts in Product Line Engineering. In *Proceedings of the 24th International Conference on Automated Software Engineering (ASE'09)*. IEEE, USA, 269–280. https://doi.org/10.1109/ASE.2009.16

[74] Malte Lochau, Stephan Mennicke, Hauke Baller, and Lars Ribbeck. 2014. DeltaCCS: A Core Calculus for Behavioral Change. In *Proceedings of the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Technologies for Mastering Change (ISoLA'14) (Lecture Notes in Computer Science, Vol. 8802)*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer, Germany, 320–335. https://doi.org/10.1007/978-3-662-45234-9_23

[75] XU Manfei, Drew Fralick, Julia Z Zheng, Bokai Wang, M TU Xin, and FENG Changyong. 2017. The differences and similarities between two-sample t-test and paired t-test. *Shanghai archives of psychiatry* 29, 3 (2017), 184. https://doi.org/10.11919/j.issn.1002-0829.217070

[76] Ken L. McMillan. 2000. A methodology for hardware verification using compositional model checking. *Science of Computer Programming* 37, 1 (2000), 279–309. https://doi.org/10.1016/S0167-6423(99)00030-1

[77] Kedar S. Namjoshi and Richard J. Trefler. 2016. Parameterized Compositional Model Checking. In *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16) (Lecture Notes in Computer Science, Vol. 9636)*, Marsha Chechik and Jean-François Raskin (Eds.). Springer, Germany, 589–606. https://doi.org/10.1007/978-3-662-49674-9_39

[78] Thomas Neele and Matteo Sammartino. 2023. Compositional Automata Learning of Synchronous Systems. In *Proceedings of the 26th International Conference on Fundamental Approaches to Software Engineering (FASE'23) (Lecture Notes in Computer Science, Vol. 13991)*, Leen Lambers and Sebastián Uchitel (Eds.). Springer, Germany, 47–66. https://doi.org/10.1007/978-3-031-30826-0_3

[79] Michael Nieke, Adrian Hoff, Ina Schaefer, and Christoph Seidl. 2022. Experiences with Constructing and Evolving aSoftware Product Line with Delta-Oriented Programming. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS'22)*. ACM, USA, 11:1–11:9. https://doi.org/10.1145/3510466.3511271

[80] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. 1979. Petri Nets, Event Structures and Domains. In *Proceedings of the International Symposium on Semantics of Concurrent Computation (Lecture Notes in Computer Science, Vol. 70)*, Gilles Kahn (Ed.). Springer, Germany, 266–284. https://doi.org/10.1007/BFB0022474

[81] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. 1981. Petri Nets, Event Structures and Domains, Part I. *Theoretical Computer Science* 13 (1981), 85–108. https://doi.org/10.1016/0304-3975(81)90112-2

[82] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. 2005. *Principles of Program Analysis*. Springer, Germany. https://doi.org/10.1007/978-3-662-03811-6

[83] René Peeters. 2003. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics* 131, 3 (2003), 651–654. https://doi.org/10.1016/S0166-218X(03)00333-0

[84] Iain Phillips and Irek Ulidowski. 2007. Reversing algebraic process calculi. *Journal of Logic and Algebraic Programming* 73, 1–2 (2007), 70–96. https://doi.org/10.1016/J.JLAP.2006.11.002

[85] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, Germany. https://doi.org/10.1007/3-540-28901-1

[86] Peter J. Ramadge and Walter M. Wonham. 1987. Supervisory Control of a Class of Discrete Event Processes. *SIAM Journal on Control and Optimization* 25, 1 (1987), 206–230. https://doi.org/10.1137/0325013

[87] Mahboubeh Samadi, Aryan Bastany, and Hossein Hojjat. 2025. Compositional Learning for Synchronous Parallel Automata. In *Proceedings of the 28th International Conference on Fundamental Approaches to Software Engineering (FASE'25) (Lecture Notes in Computer Science, Vol. 15693)*, Artur Boronat and Gordon Fraser (Eds.). Springer, Germany, 101–121. https://doi.org/10.1007/978-3-031-90900-9_6

[88] Ina Schaefer, Dilian Gurov, and Siavash Soleimanifard. 2012. Compositional Algorithmic Verification of Software Product Lines. In *Revised Papers of the 9th International Symposium on Formal Methods for Components and Objects (FMCO'10) (Lecture Notes in Computer Science, Vol. 6957)*, Bernhard K. Aichernig, Frank S. de Boer, and Marcello M. Bonsangue (Eds.). Springer, Germany, 184–203. https://doi.org/10.1007/978-3-642-25271-6_10

[89] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE'06)*. IEEE, USA, 136–145. https://doi.org/10.1109/RE.2006.23

[90] Michael Sipser. 2012. *Introduction to the Theory of Computation*. Cengage, USA.

[91] Siavash Soleimanifard, Dilian Gurov, and Marieke Huisman. 2011. ProMoVer: Modular Verification of Temporal Safety Properties. In *Proceedings of the 9th International Conference on Software Engineering and Formal Methods (SEFM'11) (Lecture Notes in Computer Science, Vol. 7041)*, Gilles Barthe, Alberto Pardo, and Gerardo Schneider (Eds.). Springer, Germany, 366–381. https://doi.org/10.1007/978-3-642-24690-6_25

[92] Chico Sundermann, Kevin Feichtinger, Dominik Engelhardt, Rick Rabiser, and Thomas Thüm. 2021. Yet Another Textual Variability Language? A Community Effort Towards a Unified Language. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference (SPLC'21)*. ACM, USA, 136–147. https://doi.org/10.1145/3461001.3471145

[93] Chico Sundermann, Jacob Loth, and Thomas Thüm. 2024. Efficient Slicing of Feature Models via Projected d-DNNF Compilation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 1332–1344. https://doi.org/10.1145/3691620.3695594

[94] Shaghayegh Tavassoli, Carlos Diego N. Damasceno, Ramtin Khosravi, and Mohammad Reza Mousavi. 2022. Adaptive Behavioral Model Learning for Software Product Lines. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume A* (Graz, Austria) *(SPLC '22)*. Association for Computing Machinery, New York, NY, USA, 142–153. https://doi.org/10.1145/3546932.3546991

[95] Shaghayegh Tavassoli, Carlos Diego N. Damasceno, Mohammad Reza Mousavi, and Ramtin Khosravi. 2022. A Benchmark for Active Learning of Variability-Intensive Systems. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference (SPLC'22)*. ACM, USA, 245–249. https://doi.org/10.1145/3546932.3547014

[96] Shaghayegh Tavassoli and Ramtin Khosravi. 2025. Efficient construction of family-based behavioral models from adaptively learned models. *Software and Systems Modeling* 24, 1 (2025), 225–251. https://doi.org/10.1007/s10270-024-01199-5

[97] Paul Temple and Gilles Perrouin. 2023. Explicit or Implicit? On Feature Engineering for ML-based Variability-intensive Systems. In *Proceedings of the 17th International Working Conference on Variability Modelling of Software-Intensive Systems* (Odense, Denmark) *(VaMoS '23)*. Association for Computing Machinery, New York, NY, USA, 91–93. https://doi.org/10.1145/3571788.3571804

[98] Sander Thuijsman and Michel Reniers. 2024. Supervisory Control for Dynamic Feature Configuration in Product Lines. *ACM Transactions on Embedded Computing Systems* 23, 5 (2024), 71:1–71:25. https://doi.org/10.1145/3579644

[99] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *Comput. Surveys* 47, 1 (2014), 6:1–6:45. https://doi.org/10.1145/2580950

[100] Thomas Thüm, Don S. Batory, and Christian Kästner. 2009. Reasoning about Edits to Feature Models. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*. IEEE, USA, 254–264. https://doi.org/10.1109/ICSE.2009.5070526

[101] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming* 79 (2014), 70–85. https://doi.org/10.1016/j.scico.2012.06.002

[102] Mirco Tribastone. 2014. Behavioral Relations in a Process Algebra for Variants. In *Proceedings of the 18th International Software Product Line Conference (SPLC'14)*. ACM, USA, 82–91. https://doi.org/10.1145/2648511.2648520

[103] Dirk A. van Beek, Wan J. Fokkink, Dennis Hendriks, Albert T. Hofkamp, Jasen Markovski, Joanna M. van de Mortel-Fronczak, and Michel A. Reniers. 2014. CIF 3: Model-Based Engineering of Supervisory Controllers. In *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14) (Lecture Notes in Computer Science, Vol. 8413)*, Erika Ábrahám and Klaus Havelund (Eds.). Springer, Germany, 575–580. https://doi.org/10.1007/978-3-642-54862-8_48

[104] Mahsa Varshosaz, Harsh Beohar, and Mohammad Reza Mousavi. 2018. Basic behavioral models for software product lines: Revisited. *Science of Computer Programming* 168 (2018), 171–185. https://doi.org/10.1016/J.SCICO.2018.09.001

[105] Glynn Winskel. 1986. Event Structures. In *Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties (ACPN'86) (Lecture Notes in Computer Science, Vol. 255)*, Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg (Eds.). Springer, Germany, 325–392. https://doi.org/10.1007/3-540-17906-2_31