Featured Team Automata

Maurice H. ter Beek[1], Guillermina Cledou[2], Rolf Hennicker[3], and José Proença[4]

[1] ISTI–CNR, Pisa, Italy, maurice.terbeek@isti.cnr.it
[2] HASLab, INESC TEC & University of Minho, Portugal, mgc@inesctec.pt
[3] Ludwig-Maximilians-Universität München, Munich, Germany
[4] CISTER, ISEP, Polytechnic Institute of Porto, Portugal, pro@isep.ipp.pt

Abstract. We propose featured team automata to support variability in the development and analysis of teams, which are systems of reactive components that communicate according to specified synchronisation types. A featured team automaton concisely describes a family of concrete product models for specific configurations determined by feature selection. We focus on the analysis of communication-safety properties, but doing so product-wise quickly becomes impractical. Therefore, we investigate how to lift notions of receptiveness (no message loss) to the level of family models. We show that featured (weak) receptiveness of featured team automata characterises (weak) receptiveness for all product instantiations. A prototypical tool supports the developed theory.

1 Introduction

Team automata, originally introduced in the context of computer supported cooperative work to model groupware systems [26], are formalised as a theoretical framework to study synchronisation mechanisms in system models [9]. Team automata represent an extension of I/O automata [12]. Their distinguishing feature is the loose nature of synchronisation according to which, in principle, any number of component automata can participate in the synchronised execution of a shared communicating action, either as a sender or as a receiver. Team automata can determine specific synchronisation policies defining when and which actions are executed and by how many components. Synchronisation types classify the policies realisable in team automata (e.g., peer-to-peer or broadcast communication) in terms of ranges for the number of sender and receiver components that can participate in a synchronisation [6]. In extended team automata (ETA) [11], synchronisation type specifications (STS) individually assign a synchronisation type to each communicating action. Such a specification uniquely determines a *team* and gives rise to communication requirements to be satisfied by the team.

For systems composed by components communicating via message exchange, it is desirable to guarantee absence of communication failures, like message loss (typically output not received as input, violating receptiveness) or indefinite waiting (typically for input that never arrives, violating responsiveness). This requires knowledge of the synchronisation policies to establish the compatibility

of communicating components [23,15,30]; for team automata this was first studied for full synchronous products of component automata in [16]. Subsequently, a generic procedure to derive requirements for receptiveness and responsiveness for each synchronisation type was defined, and communication-safety of (extended) team automata was expressed in terms of compliance with such requirements [6,11]. A team automaton is called compliant with a given set of communication requirements if in each reachable state the requirements are met (i.e. the communication is safe). If the required communication cannot occur immediately, but only after some arbitrary other actions have been executed, the team automaton is called weakly compliant (akin to weak compatibility [5,29] or agreement of lazy request actions [3]).

Many of today's software systems are highly configurable, variant-rich systems, developed as a software product line (SPL) with a notion of variability in terms of features that conceptualise pieces of system functionality or aspects that are relevant to the stakeholders [1]. Formal models of SPL behaviour are studied extensively. Such variability-rich behavioural models are often based on the superimposition of multiple product models in a single family model, equipped with feature-based variability such that each product model corresponds to a different configuration. Arguably the best known models are featured transition systems (fTSs) [21,19,20] and modal transition systems [28,27], possibly with variability constraints [2,10], but also I/O automata [30,31], Petri nets [35,34] and contract automata [4,3] have been equipped with variability. An fTS is a labelled transition system (LTS) whose transitions are annotated with feature expressions that are Boolean expressions over features, which condition the presence of transitions in product models, and a feature model, which determines the set of valid product models (configurations) of the family model. The analysis of family models is challenging due to their innate variability, since the number of possible product models may be exponential in the number of features. In particular for larger models, enumerative product-by-product analysis becomes unfeasible; thus, dedicated family-based analysis techniques and tools, which exploit variability in terms of features, have been developed [21,22,38,37,25,14,18,17,13,24].

Motivation fTSs have mostly been studied in the context of families of configurable components. Less attention has been paid on their parallel execution, in particular in the context of systems of reactive, concurrently running components, where interaction is a crucial issue, often realised by message exchange. For this, we need i) to discriminate between senders and receivers and thus between input and output actions in fTSs, and ii) a flexible synchronisation mechanism, not necessarily peer-to-peer, for sets of fTSs, called (featured) systems. In particular, the type of synchronisation should remain variable, depending on selected features (products). Important questions for analysis of such systems concern behavioural compatibility (communication-safety). As mentioned above, compositionality and communication-safety have been studied extensively in the literature for a variety of formal (automata-based) models, but—to the best of our knowledge—not considering variability. Thus, we need a means to define and verify communication-safety for systems of fTSs, ideally performing

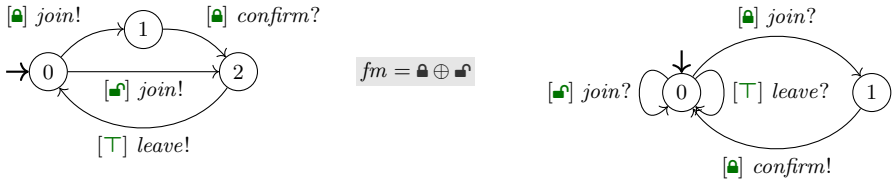
analyses on the level of featured systems such that the respective properties are automatically guaranteed for any product instantiation. In this paper, we focus on the property of (weak) receptiveness.

Running Example We consider a configurable access management system consisting of a server and users who can either login with secure authentication or without (open access). Concrete automata capturing user and server behaviour are specified as family models whose product models correspond to configurations with or without secure authentication.

Fig. 1 shows two fTSs: a family model of user components (Fig. 1a) and a family model of server components (Fig. 1b), as well as a feature model $fm = 🔒 \oplus 🔓$. The feature model expresses an exclusive choice of two features, 🔒 and 🔓, representing access with or without secure authentication, respectively, and defines two valid products (sets of features): {🔒} and {🔓}. The idea is that the server must confirm login access only for secure authentication. Thus, each transition is annotated with a constraint, denoted by a feature expression in square brackets (e.g., [🔒]), to indicate the product(s) that allow this transition.

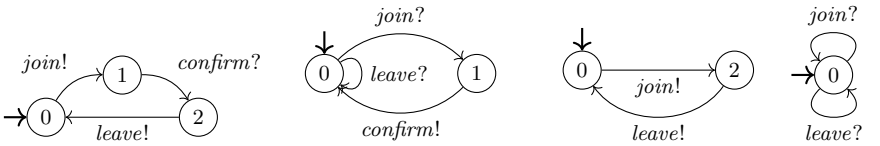
A user starts in the initial state 0, indicated by the incoming arrow, in which only the action *join*! can be executed. Depending on the specific product, this results in a move to state 1 (if feature 🔒 is present) or to state 2 (if 🔓 is present). From state 2, a user can move back to state 0 by executing action *leave*!, in either product, as enabled by the transition constraint [⊤] (denoting truth value true). In state 1, which is only present for the product with secure authentication, the user waits for explicit confirmation of login access from the server.

Figs. 2a and 2c show the LTSs representing the user product models, which result from projecting the user fTS in Fig. 1a onto its set of valid products. Similarly, Figs. 2b and 2d show the LTSs of the server product models, projecting the server fTS onto its two valid products.



(a) 𝕌ser (b) 𝕊erver

Fig. 1: Family models of users 𝕌 and servers 𝕊 and a shared feature model *fm*



(a) $\mathbb{U}|_{🔒}$ (b) $\mathbb{S}|_{🔒}$ (c) $\mathbb{U}|_{🔓}$ (d) $\mathbb{S}|_{🔓}$

Fig. 2: Product models of users and servers (projections of the models in Fig. 1)

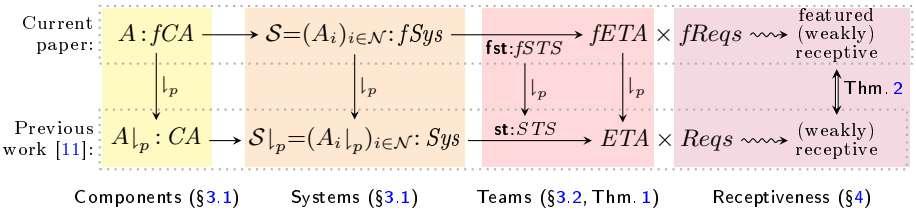


Fig. 3: Overview of this paper, using a valid product p

Contribution Fig. 3 illustrates the contents and contributions of this paper, which we now explain and relate to the literature mentioned above. In particular, we extend [11], by enriching ETA with variability, proposing a new model called *featured* ETA (fETA) to allow the specification of—and reasoning on—a family of ETA parameterised by a set of features. We define *projections* $\downarrow_p$ (for any valid product p) to relate the featured setting of this paper to that without in [11].

First, we extend *component automata* (CA), the building blocks of (extended) team automata, with variability, resulting in fTSs with input and output actions, called *featured* CA (fCA). Basically, CA are LTSs that distinguish between input and output actions (and internal actions, omitted in this paper) and which capture the behaviour of a component. The fTSs in the running example are fCA in which input and output actions are appended by ! and ?, respectively. Multiple CA can run in parallel to form a *system* (*Sys* in Fig. 3) of the CA; we propose a *featured* system (fSys) to consist of fCA instead of CA.

Given a system and a *synchronisation type specification* (STS), it is possible to generate an ETA and derive *receptiveness requirements* (Reqs), and study whether the ETA is (weakly) compliant with all such Reqs, in which case it is called *(weakly) receptive*. An ETA is an LTS that restricts how CA in the system can communicate based on the STS. We propose a *featured* STS (fSTS) to parameterise an STS with variability, giving rise to the aforementioned fETA and *featured* Reqs (fReqs). If the fETA is featured (weakly) compliant with all such fReqs, it is called *featured (weakly) receptive*.

While the extension from CA to fCA (and from systems to featured systems) is rather straightforward, fETA are not simple extensions of ETA: the fSTSs giving rise to fETA are a nontrivial extension of the STSs for ETA, partially due to the variability in synchronisation types. Our first result (Theorem 1) confirms the soundness of our extension. Our main result (Theorem 2) is that featured (weak) receptiveness induces and reflects (weak) receptiveness of product models, i.e. a fETA is featured (weakly) receptive if and only if all ETA obtained by product projections are (weakly) receptive.

Outline Section 2 provides some basic definitions concerning variability. Section 3 lifts the theory of team automata to that of featured team automata, and Section 4 does the same for receptiveness requirements and compliance. We present a prototypical implementation of the developed theory in Section 5, and Section 6 concludes the paper and provides some ideas for future work. The proofs of our results can be found in a companion report [7, Appendix A].

2 Variability

This section provides definitions of the basic notions concerning variability, viz. *features*, *feature expressions*, *feature models*, and fTSs.

A *feature*, ranged over by f, is regarded as a Boolean variable that represents a unit of variability. This paper assumes a finite set of features F. A *product*, ranged over by $p \subseteq F$, is a finite subset of selected features. In the context of SPLs, a product can be interpreted as a configuration used to derive concrete software systems. A *feature expression* ψ over a set of features F, denoted $\psi \in FE(F)$, is a Boolean expression over features with the usual Boolean connectives and constants $\top$ and $\bot$ interpreted by the truth values true and false. A product p satisfies a feature expression ψ, denoted $p \models \psi$, if and only if ψ is evaluated to $\top$ if $\top$ is assigned to every feature in p and $\bot$ to the features not in p. A feature expression ψ is *satisfiable* if there exists a product p such that $p \models \psi$. A *feature model* $fm \in FE(F)$ is a feature expression that determines the set of products for which concrete systems of an SPL can be derived. We use $[\![fm]\!]$ to denote the set of products that satisfy the feature model $fm \in FE(F)$.

Notation. For any product $p \subseteq F$, its view as a feature expression is $\chi_p = \bigwedge_{f \in p} f \wedge \bigwedge_{f \in F \setminus p} \neg f$. p is the unique product with $p \models \chi_p$. A set P of products is characterised by the feature expression $\chi_P = \bigvee_{p \in P} \chi_p$. Clearly, for any product p, $p \in P$ iff $p \models \chi_P$. Note that the conjunctions and disjunctions are finite, since F is finite. Moreover, $\bigwedge_{i \in \emptyset} \psi_i$ stands for $\top$ and $\bigvee_{i \in \emptyset} \psi_i$ stands for $\bot$.

A *featured transition system* (fTS) is a tuple $A = (Q, I, \Sigma, E, F, fm, \gamma)$ such that (Q, I, Σ, E) is an LTS with a finite set of states Q, a set of initial states $I \subseteq Q$, a finite set of actions Σ, and a transition relation $E \subseteq Q \times \Sigma \times Q$. F is a finite set of features, $fm \in FE(F)$ is a feature model and $\gamma : E \to FE(F)$ is a mapping assigning feature expressions to transitions. A product $p \subseteq F$ is *valid* for the feature model fm, if $p \in [\![fm]\!]$. The mapping γ expresses *transition constraints* for the realisation of transitions. A transition $t \in E$ is *realisable* for a valid product p if $p \models \gamma(t)$.

An fTS A can be *projected* to a valid product p by using γ to filter realisable transitions, resulting in the LTS $A{\downarrow_p} = (Q, I, \Sigma, E{\downarrow_p})$, where $E{\downarrow_p} = \{t \in E \mid p \models \gamma(t)\}$. Such a projection is also called product model or configuration. Hereafter, we will generally write projections using superscripts, e.g. A^p to denote $A{\downarrow_p}$.

Notation. Given an LTS or an fTS A, we write $q \xrightarrow{a}_A q'$, or shortly $q \xrightarrow{a} q'$, to denote $(q, a, q') \in E$. For $\Gamma \subseteq \Sigma$, we write $q \xrightarrow{\Gamma}{}^* q'$ if there exist $q \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q'$ for some $n \geq 0$ and $a_1, \ldots, a_n \in \Gamma$. An action a is *enabled* in A at state $q \in Q$, denoted $a\,\mathbf{en}_A@q$, if there exists $q' \in Q$ such that $q \xrightarrow{a} q'$. A state $q \in Q$ is *reachable* if $q_0 \xrightarrow{\Sigma}{}^* q$ for some $q_0 \in I$.

3 Team Automata with Variability

This section proposes to integrate variability in the modelling of teams of reactive components which communicate according to specified synchronisation policies.

For this purpose we define *featured CA*, *featured systems*, and *featured ETA*, and compare them to their featureless counterparts. Throughout this section we will use grey backgrounds to highlight extensions with features.

3.1 Featured Component Automata and Featured Systems

A *featured component automaton* (fCA) is an fTS $A = (Q, I, \Sigma, E, F, fm, \gamma)$ such that $\Sigma = \Sigma^? \uplus \Sigma^!$ consists of disjoint sets $\Sigma^?$ of *input actions* and $\Sigma^!$ of *output actions*. For simplicity, we do not consider internal actions here. For easier readability, input actions will be shown with suffix "?" and output actions with suffix "!". fCA extend *component automata* (CA) [9,11] with features and feature models. The running example in Section 1 contains examples of fCA.

A *featured system* (fSys) is a pair $\mathcal{S} = (\mathcal{N}, (A_i)_{i\in\mathcal{N}})$, where $\mathcal{N}$ is a finite, nonempty set of component names and $(A_i)_{i\in\mathcal{N}}$ is an $\mathcal{N}$-indexed family of fCA $A_i = (Q_i, I_i, \Sigma_i, E_i, F, fm, \gamma_i)$ over a shared set of features F and feature model fm. Composition of feature models is out of the scope of this paper, but note that multiple approaches exist in the literature, e.g., using conjunction or disjunction of feature models [36,19,20].

Featured systems extend *systems of CA* [11] by using fCA instead of CA as system components. An fSys $\mathcal{S} = (\mathcal{N}, (A_i)_{i\in\mathcal{N}})$ induces: the set of system states $Q = \prod_{i\in\mathcal{N}} Q_i$ such that, for any $q \in Q$ and for all $i \in \mathcal{N}$, $q_i \in Q_i$; the set of initial states $I = \prod_{i\in\mathcal{N}} I_i$; the set of system actions $\Sigma = \bigcup_{i\in\mathcal{N}} \Sigma_i$; the set of system labels $\Lambda \subseteq \mathbf{2}^{\mathcal{N}} \times \Sigma \times \mathbf{2}^{\mathcal{N}}$ defined as $\Lambda = \{(S, a, R) \mid \emptyset \neq S \cup R \subseteq \mathcal{N}, \forall_{i\in S} \cdot a \in \Sigma_i^!, \forall_{i\in R} \cdot a \in \Sigma_i^?\}$; and the set of system transitions $E \subseteq Q \times \Lambda \times Q$ defined as $E = \{q \xrightarrow{(S,a,R)} q' \mid \forall_{i\in(S\cup R)} \cdot q_i \xrightarrow{a}_{A_i} q'_i,\ \forall_{j\in\mathcal{N}\setminus(S\cup R)} \cdot q_j = q'_j\}$.

A transition labelled by a *system label* denotes the atomic execution of an action a by a set of components in which a is enabled. More concretely, for a system label $(S, a, R) \in \Lambda$, S represents the set of *senders* and R the set of *receivers* that synchronise on an action $a \in \Sigma$. Since, by definition of system labels, $S \cup R \neq \emptyset$, at least one component participates in any system transition. The transitions of a system capture all possible synchronisations of shared actions of its components, even when only one component participates. Given a system transition $t = q \xrightarrow{(S,a,R)} q'$, we write $t.a$ for a, $t.S$ for S and $t.R$ for R. For ease of presentation, we assume in this paper that systems are closed. This means that any system action $a \in \Sigma$ occurs in (at least) one of its components as an input action and in (at least) one of its components as an output action.

The *projection* of an fSys $\mathcal{S} = (\mathcal{N}, (A_i)_{i\in\mathcal{N}})$ to a product $p \in [\![fm]\!]$ is the system $\mathcal{S}^p = (\mathcal{N}, (A_i^p)_{i\in\mathcal{N}})$.

Example 1. We consider an fSys $\mathcal{S}_@$ with three components, two users and one server following the running example in Section 1. Formally, $\mathcal{S}_@ = (\mathcal{N}, (A_i)_{i\in\mathcal{N}})$, where $\mathcal{N} = \{u_1, u_2, s\}$ are component names, A_{u_1}, A_{u_2} are copies of the fCA $\mathbb{U}$ in Fig. 1a, and A_s is a copy of the fCA $\mathbb{S}$ in Fig. 1b.

The system states are tuples (p, q, r) with user states $p \in Q_{u_1}$ and $q \in Q_{u_2}$, and server state $r \in Q_s$. $\mathcal{S}_@$ has an initial state $(0, 0, 0)$, a total of 18 states ($3 \times$

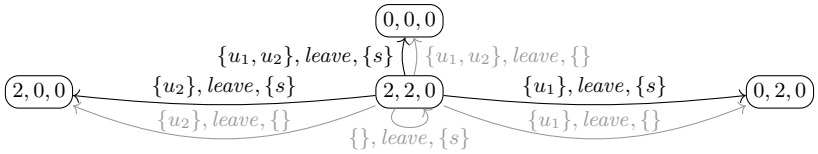


Fig. 4: Some system transitions of $\mathcal{S}_@$

3×2), actions $\Sigma = \{join, leave, confirm\}$, and a total of 142 system transitions. Some of these (with action *leave*) are depicted in Fig. 4; the transitions marked in grey will be discarded based on synchronisation restrictions in the next section.

The projection of $\mathcal{S}_@$ to the valid product $\{🔒\}$, respecting the shared feature model $🔒 \oplus 🔓$, is the system $\mathcal{S}_@^{🔒} = (\mathcal{N}, \{A_{u_1}^{🔒}, A_{u_2}^{🔒}, A_s^{🔒}\})$, such that $A_{u_1}^{🔒}, A_{u_2}^{🔒}$ are copies of $\mathbb{U}|_{🔒}$ in Fig. 2a and $A_s^{🔒}$ is a copy of $\mathbb{S}|_{🔒}$ in Fig. 2b. Similarly, for product $\{🔓\}$, we get the projected system $\mathcal{S}_@^{🔓} = (\mathcal{N}, \{A_{u_1}^{🔓}, A_{u_2}^{🔓}, A_s^{🔓}\})$. ▷

3.2 Featured Team Automata

Featured team automata (fETA) are the key concept to model families of teams. They are constructed over an fSys $\mathcal{S}$ together with a specification of synchronisation types expressing desirable synchronisation constraints. This section first formalises the latter and then fETA as fTSs.

A *synchronisation type* $(s, r) \in \mathsf{Intv} \times \mathsf{Intv}$ is a pair of intervals s and r which determine the number of senders and receivers that can participate in a communication. Each interval is written $[min, max]$, with $min \in \mathbb{N}$ and $max \in \mathbb{N} \cup \{*\}$. We use $*$ to denote 0 or more participants, and write $x \in [n, m]$ if $n \leq x \leq m$ and $x \in [n, *]$ if $x \geq n$. For a system transition t, we define $t \models (s, r)$ if $|t.S| \in s \wedge |t.R| \in r$.

A *featured synchronisation type specification* (fSTS) over an fSys $\mathcal{S}$, is a total function, $\mathbf{fst} : [\![fm]\!] \times \Sigma \to \mathsf{Intv} \times \mathsf{Intv}$, mapping each product $p \in [\![fm]\!]$ and action $a \in \Sigma$ to a synchronisation type. Thus, an fSTS is parameterised by (valid) products and therefore supports variability of synchronisation conditions.

fSTSs are extensions of synchronisation type specifications (STSs) in [11]; an STS $\mathbf{st} : \Sigma \to \mathsf{Intv} \times \mathsf{Intv}$ maps actions to bounds of senders and receivers. For any product $p \in [\![fm]\!]$, an fSTS $\mathbf{fst}$ can be projected to an STS $\mathbf{fst}^p$ such that $\mathbf{fst}^p(a) = \mathbf{fst}(p, a)$ for all $a \in \Sigma$.

Example 2. The definition of $\mathbf{fst}_@$ corresponds to an fSTS for the fSys $\mathcal{S}_@$ in Example 1:

$$\mathbf{fst}_@(p, confirm) = ([1,1],[1,1]) \text{ for } p \in \{\{🔒\}, \{🔓\}\} \quad (1)$$

$$\mathbf{fst}_@(\{🔒\}, a) = ([1,1],[1,1]) \text{ for } a \in \{join, leave\} \quad (2)$$

$$\mathbf{fst}_@(\{🔓\}, a) = ([1,*],[1,1]) \text{ for } a \in \{join, leave\} \quad (3)$$

Intuitively, independently of the selected product, users can receive *confirm*ation from the server in a *one-to-one* fashion (1). If secure authentication 🔒 is required, one user can *join*/*leave* by synchronising exclusively with one server (2). If open access 🔓 is required, multiple users can *join*/*leave* at the same time (3). ▷

Given an fSys $\mathcal{S} = (\mathcal{N}, (A_i)_{i\in\mathcal{N}})$ and an fSTS $\mathbf{fst}$ over $\mathcal{S}$, the *featured team automaton* (fETA) generated by $\mathcal{S}$ and $\mathbf{fst}$, written $\mathbf{fst}[\mathcal{S}]$, is the fTS $(Q, I, \Sigma, E, F, fm, \gamma)$ where Q, I, Σ, E, F, and fm are determined by $\mathcal{S}$. It remains to construct the mapping $\gamma : E \to FE(F)$, which constrains system transitions by feature expressions. The definition of γ is derived from both the transition constraints γ_i of every A_i and from $\mathbf{fst}$. It is motivated by the fact that a system transition $t = q \xrightarrow{(S,a,R)} q' \in E$ should be realisable for those products $p \in [\![fm]\!]$ for which both of the following conditions hold:

1. In each component A_i, with $i \in (S \cup R)$, the local transition $q_i \xrightarrow{a}_{A_i} q_i'$ is realisable for p. This means $p \models \hat{\gamma}(t)$, where $\hat{\gamma}(t) = \bigwedge_{i\in(S\cup R)} \gamma_i(q_i \xrightarrow{a}_{A_i} q_i')$.
2. For any action $a \in \Sigma$, the number of senders $|S|$ and receivers $|R|$ fits the synchronisation type $\mathbf{fst}(p, a)$. This means $p \models \chi_{P(\mathbf{fst},t)}$, where $\chi_{P(\mathbf{fst},t)}$ (cf. Section 2) is the feature expression characterising the set of products $P(\mathbf{fst}, t) = \{p \in [\![fm]\!] \mid t \models \mathbf{fst}(p, t.a)\}$.

In summary, for any $t = q \xrightarrow{(S,a,R)} q' \in E$, we define $\gamma(t) = \hat{\gamma}(t) \wedge \chi_{P(\mathbf{fst},t)}$. Note that, since $P(\mathbf{fst}, t)$ is a subset of $[\![fm]\!]$, it holds $\models \chi_{P(\mathbf{fst},t)} \to fm$ and hence $\models \gamma(t) \to fm$. In cases where $P(\mathbf{fst}, t) = [\![fm]\!]$, $\chi_{P(\mathbf{fst},t)}$ and fm are equivalent and then we will often use $\gamma(t) = \hat{\gamma}(t) \wedge fm$.

Recall that an fTS can be projected to products (as defined in Section 2) and therefore also the fETA $\mathbf{fst}[\mathcal{S}]$ can be projected to a valid product $p \in [\![fm]\!]$ yielding the LTS $\mathbf{fst}[\mathcal{S}]^p$. Thus any fETA $\mathbf{fst}[\mathcal{S}]$ specifies a family of product models.

Example 3. Consider the fSys $\mathcal{S}_@$ and the fSTS $\mathbf{fst}_@$ from Example 2, here and in the following examples simply called $\mathbf{fst}$, as well as the generated fETA $\mathbf{fst}[\mathcal{S}_@]$. There are many system transitions, for instance

$$t_1 = (0,0,0) \xrightarrow{(\{u_1,u_2\},join,\{s\})} (2,2,0) \text{ and}$$
$$t_2 = (0,0,0) \xrightarrow{(\{u_1,u_2\},join,\{s\})} (1,1,1).$$

For t_1, we have $\hat{\gamma}(t_1) = \bigwedge_{i\in\{1,2\}} \gamma_{u_i}(0 \xrightarrow{join}_{A_{u_i}} 2) \wedge \gamma_s(0 \xrightarrow{join}_{A_s} 0) = \text{🔓} \wedge \text{🔓} \wedge \text{🔓}$. Since $\{\text{🔓}\}$ is the only valid product p such that $t_1 \models \mathbf{fst}(p, join) = ([1, *], [1, 1])$—note that only for open access more than one user can join simultaneously—we have $\chi_{P(\mathbf{fst},t_1)} = \text{🔓} \wedge \neg\text{🔒}$ (where $P(\mathbf{fst}, t_1) = \{\{\text{🔓}\}\}$). Thus, in summary, $\gamma(t_1) = (\text{🔓} \wedge \text{🔓} \wedge \text{🔓}) \wedge (\text{🔓} \wedge \neg\text{🔒})$. Hence t_1 can only be realised for open access.

For t_2, we have $\hat{\gamma}(t_2) = \text{🔒} \wedge \text{🔒} \wedge \text{🔒}$ and $\chi_{P(\mathbf{fst},t_2)} = \text{🔓} \wedge \neg\text{🔒}$ as before, since $\{\text{🔓}\}$ is the only product p such that $t_2 \models \mathbf{fst}(p, join)$. Therefore, $\gamma(t_2) = (\text{🔒} \wedge \text{🔒} \wedge \text{🔒}) \wedge (\text{🔓} \wedge \neg\text{🔒})$, which reduces to $\bot$ and thus is not realisable by any product.

Fig. 5 shows the full generated fETA $\mathbf{fst}[\mathcal{S}_@]$, after removing all unreachable states and all non-realisable transitions t, i.e. $\forall_{p\in[\![fm]\!]} \cdot p \not\models \gamma(t)$. For each transition t in Fig. 5 we present $\gamma(t)$ as a conjunction of (a semantics-preserving simplification of) $\hat{\gamma}(t)$ and an underlined $\chi_{P(\mathbf{fst},t)}$ or $fm = \text{🔒} \oplus \text{🔓}$ if $P(\mathbf{fst}, t) = [\![fm]\!]$. The latter is the case in all transitions in which only one user participates. If two users join or leave simultaneously, then $\chi_{P(\mathbf{fst},t)}$ is always $\text{🔓} \wedge \neg\text{🔒}$ as explained above for t_1. (Further reductions are possible for the conjoined $\gamma(t)$.) ▷

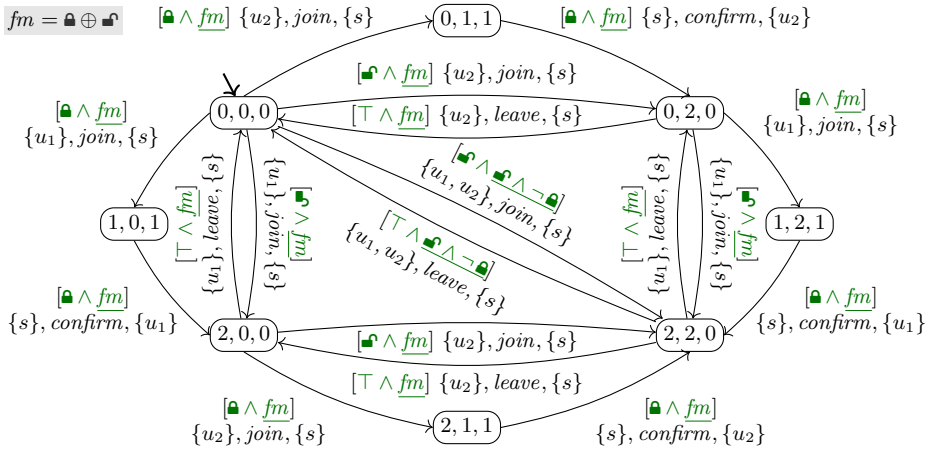


Fig. 5: Generated fETA $\mathbf{fst}_{@}[\mathcal{S}_{@}]$

3.3 fETA versus ETA

fETA are not simple extensions of extended team automata (ETA) introduced in [11]. An ETA is an LTS $\mathbf{st}[\mathcal{S}]$ generated over a system $\mathcal{S}$ of CA by an STS $\mathbf{st}$ that explicitly filters the system transitions that satisfy the synchronisation types determined by $\mathbf{st}$. Concretely, an ETA $\mathbf{st}[\mathcal{S}]$ is the LTS $(Q, I, \Sigma, \mathbf{st}[E])$, where Q, I, Σ, and E are induced by $\mathcal{S}$, and $\mathbf{st}[E] = \{t \in E \mid t \models \mathbf{st}(t.a)\}$.

Observe that an STS thus restricts the set of system transitions of a system $\mathcal{S}$, such that the ETA $\mathbf{st}[\mathcal{S}]$ has only a subset of the transitions of $\mathcal{S}$. Instead, an fSTS and the local transition constraints of the components $\mathcal{A}_i$ impose transition constraints γ on the system transitions of an fSys $\mathcal{S}$ such that the fETA $\mathbf{fst}[\mathcal{S}]$ has all transitions of $\mathcal{S}$, but appropriately constrained such that many of them will not be realisable anymore for concrete products.

The next theorem shows that, for any valid product p, the projection onto p of the fETA $\mathbf{fst}[\mathcal{S}]$, generated over the fSys $\mathcal{S}$ by the fSTS $\mathbf{fst}$, is the same as the ETA over the projected system $\mathcal{S}^p$ generated by the projected STS $\mathbf{fst}^p$. This result justifies the soundness of the definition of a generated fETA, in particular of its transition constraint γ. It also shows that the diagram in Fig. 3 commutes.

Theorem 1. *Let $\mathcal{S}$ be an fSys with feature model fm, let $\mathbf{fst}$ be an fSTS, and let $p \in [\![fm]\!]$ be a valid product. Then:*

$$\mathbf{fst}[\mathcal{S}]^p = \mathbf{fst}^p[\mathcal{S}^p].$$

4 Receptiveness

As explained in Section 3 and formalised in Theorem 1, a fETA $\mathbf{fst}[\mathcal{S}]$ can be projected to a product $p \in [\![fm]\!]$, thus yielding an ETA (i.e. a team) $\mathbf{fst}[\mathcal{S}]^p = \mathbf{fst}^p[\mathcal{S}^p]$.

Any such ETA describes the behaviour of a concrete system $\mathcal{S}^p$ whose components (the team members) are coordinated by the synchronisation type specification $\mathbf{st} = \mathbf{fst}^p$. This section analyses communication-safety of such families of ETA. Our aim is to provide criteria on the level of fETA that guarantee communication-safety properties for all ETA obtained by projection (cf. Section 4.3).

4.1 Receptiveness for ETA

We focus on the property of receptiveness, which has been studied before in the literature [23,15,30], mainly in the context of peer-to-peer communication. An extension to multi-component communications was studied in [16] and in [11], where also a notion of responsiveness not considered here was introduced. The idea of receptiveness is as follows: whenever, in a reachable state q of an ETA $\mathbf{st}[\mathcal{S}]$, a group of components J is (locally) enabled to perform an output action a such that its synchronous execution is in accordance with the synchronisation type $\mathbf{st}(a)$, we get a receptiveness requirement, written as $\mathbf{rcp}(J, a)@q$. The ETA is *compliant* with this requirement if J can find partners in the team which synchronise with the components in J by taking (receiving) a as input. If reception is immediate, we talk about receptiveness; if the other components first perform some intermediate actions before accepting a, we talk about weak receptiveness.

Formally, receptiveness requirements, compliance, and receptiveness are defined as follows and illustrated in Example 4. We assume a given ETA $\mathbf{st}[\mathcal{S}] = (Q, I, \Sigma, \mathbf{st}[E])$ generated by the STS $\mathbf{st}$ over a system $\mathcal{S} = (\mathcal{N}, (A_i)_{i\in\mathcal{N}})$ of CA A_i.

A *receptiveness requirement* (Req) is an expression $\mathbf{rcp}(J, a)@q$, where $q \in Q$ is a reachable state of $\mathbf{st}[\mathcal{S}]$, $a \in \Sigma$ is an action, and $\emptyset \neq J \subseteq \mathcal{N}$ is a set of component names such that $\forall_{j\in J} \cdot a \in \Sigma_j^! \wedge a\ \mathbf{en}_{A_j}@q_j$ and $\mathbf{st}(a) = (s, r) \Rightarrow |J| \in s \wedge 0 \notin r$. The last condition requires that i) the number of components in J fits the number of allowed senders according to the synchronisation type of a, and ii) at least one receiver must exist according to the synchronisation type of a.[5] Hence our subsequent compliance and receptiveness notions, taken from [11] and formalising the informal explanations above, depend strongly on the synchronisation types of actions.

The ETA $\mathbf{st}[\mathcal{S}]$ is *compliant* with a Req $\mathbf{rcp}(J, a)@q$ if the following holds:

$$\exists_{R\neq\emptyset \text{ and } q'\in Q} \cdot q \xrightarrow{(J,a,R)}_{\mathbf{st}[\mathcal{S}]} q'.$$

The ETA $\mathbf{st}[\mathcal{S}]$ is *weakly compliant* with a Req $\mathbf{rcp}(J, a)@q$ if

$$\exists_{R\neq\emptyset \text{ and } \hat{q},q'\in Q} \cdot q \xrightarrow{\Lambda_{\setminus J}}{}^*_{\mathbf{st}[\mathcal{S}]} \hat{q} \xrightarrow{(J,a,R)}_{\mathbf{st}[\mathcal{S}]} q',$$

where $\Lambda_{\setminus J}$ denotes the set of system labels in which no component of J participates. Indeed, only when state $\hat{q}$ is reached, the components of J can actively get rid of their output.

The ETA $\mathbf{st}[\mathcal{S}]$ is *(weakly) receptive* if it is (weakly) compliant with *all* Reqs for $\mathbf{st}[\mathcal{S}]$.

[5] Otherwise, the components in J could simply output a without reception.

Example 4. Let ETA $\mathbf{fst}^{\lock}[\mathcal{S}^{\lock}_{@}]$ be generated by the STS $\mathbf{fst}^{\lock}$ (i.e. the projection of $\mathbf{fst}$ from Example 3 to $\{\lock\}$) over the system $\mathcal{S}^{\lock}_{@} = (\mathcal{N}, \{A^{\lock}_{u_1}, A^{\lock}_{u_2}, A^{\lock}_{s}\})$ of Example 1, with $\mathbf{fst}^{\lock}(join) = \mathbf{fst}^{\lock}(confirm) = \mathbf{fst}^{\lock}(leave) = ([1,1],[1,1])$. In the initial global state $(0,0,0)$ both users are enabled to execute output action *join*, but not simultaneously. Hence, we get two Reqs $\mathbf{rcp}(\{u_i\}, join)@(0,0,0)$, one for each $i \in \{1,2\}$. The ETA $\mathbf{fst}^{\lock}[\mathcal{S}^{\lock}_{@}]$ is compliant with both Reqs because $(0,0,0) \xrightarrow{(\{u_1\},join,\{s\})}_{\mathbf{fst}^{\lock}[\mathcal{S}^{\lock}_{@}]} (1,0,1)$ and $(0,0,0) \xrightarrow{(\{u_2\},join,\{s\})}_{\mathbf{fst}^{\lock}[\mathcal{S}^{\lock}_{@}]} (0,1,1)$. Now assume that user $A^{\lock}_{u_1}$ joins. Then $\mathbf{fst}^{\lock}[\mathcal{S}^{\lock}_{@}]$ ends up in state $(1,0,1)$, where user $A^{\lock}_{u_2}$ may decide to join, i.e. there is a Req $\mathbf{rcp}(\{u_2\}, join)@(1,0,1)$. But the server is not yet ready for $A^{\lock}_{u_2}$ as it first needs to send a confirmation to $A^{\lock}_{u_1}$. Therefore $\mathbf{fst}^{\lock}[\mathcal{S}^{\lock}_{@}]$ is not compliant with $\mathbf{rcp}(\{u_2\}, join)@(1,0,1)$, but it is weakly compliant with this Req. We can show that the ETA $\mathbf{fst}^{\lock}[\mathcal{S}^{\lock}_{@}]$ is either compliant or weakly compliant with any Req and therefore it is weakly receptive.

Next, consider ETA $\mathbf{fst}^{\unlock}[\mathcal{S}^{\unlock}_{@}]$ generated by the STS $\mathbf{fst}^{\unlock}$ over the system $\mathcal{S}^{\unlock}_{@} = (\mathcal{N}, \{A^{\unlock}_{u_1}, A^{\unlock}_{u_2}, A^{\unlock}_{s}\})$ of Example 1 with $\mathbf{fst}^{\unlock}(join) = \mathbf{fst}^{\unlock}(leave) = ([1,*],[1,1])$. In state $(0,0,0)$, both users are enabled to output *join*. Therefore, according to the sending multiplicity $[1,*]$ of $\mathbf{fst}^{\unlock}(join)$, there are three Reqs for that state, among which $\mathbf{rcp}(\{u_1,u_2\}, join)@(0,0,0)$. Note that $\mathbf{fst}^{\unlock}[\mathcal{S}^{\unlock}_{@}]$ is compliant with this Req due to the team transition $(0,0,0) \xrightarrow{(\{u_1,u_2\},join,\{s\})}_{\mathbf{fst}^{\unlock}[\mathcal{S}^{\unlock}_{@}]} (1,1,1)$. In fact, the ETA $\mathbf{fst}^{\unlock}[\mathcal{S}^{\unlock}_{@}]$ is compliant with all Reqs and therefore it is receptive. ▷

4.2 Featured Receptiveness for fETA

We now turn to fETA and discuss how the notions of receptiveness requirements, compliance, and receptiveness can be transferred to the feature level. We assume a given fETA $\mathbf{fst}[\mathcal{S}] = (Q, I, \Sigma, E, F, fm, \gamma)$ generated by the fSTS $\mathbf{fst}$ over an fSys $\mathcal{S} = (\mathcal{N}, (A_i)_{i\in\mathcal{N}})$, with fCA A_i. The crucial difference with the case of ETA is that fETA are based on syntactic specifications modelling families of teams. Hence a Req $\mathbf{rcp}(J,a)@q$ formulated for an ETA cannot be formulated for a fETA as it is. Instead, it must take into account the valid products p of the family for which the requirement is meaningful. For this purpose, we propose to complement $\mathbf{rcp}(J,a)@q$ by a syntactic application condition, resulting in a *featured receptiveness requirement* (fReq), written as $[\mathbf{prod}(J,a,q)]\,\mathbf{rcp}(J,a)@q$. Herein $\mathbf{prod}(J,a,q)$ is a feature expression, which characterises the set of valid products for which the Req $\mathbf{rcp}(J,a)@q$ is applicable for $\mathbf{fst}[\mathcal{S}]^p$. The expression $\mathbf{prod}(J,a,q) = \mathbf{fe}(J,a,q) \wedge \chi_{P(\mathbf{fst},J,a)} \wedge \chi_{P(q)}$ consists of the following parts:

1. $\mathbf{fe}(J,a,q) = \bigwedge_{j\in J} \bigvee \gamma_j(q_j \xrightarrow{a}_{A_j} q'_j)$ combines the feature expressions of all transitions of components $\mathcal{A}_j$ ($j \in J$) with action a and starting in the local state q_j. For any fCA $\mathcal{A}_j$, the disjunction $\bigvee \gamma_j(q_j \xrightarrow{a}_{A_j} q'_j)$ ranges over the feature expressions of all local transitions of A_j starting in q_j and labelled with a. Hence, if there are more such transitions it is sufficient if one of them is realised (in a projection of $\mathcal{A}_j$). Thus $\mathbf{fe}(J,a,q)$ characterises those products p for which outgoing transitions with output a are realisable in the local states q_j of A_j and hence enabled in q_j in the projected component $\mathcal{A}^p_j$.

2. $\chi_{P(\mathbf{fst},J,a)}$ is the feature expression which characterises (cf. Section 2) the set $P(\mathbf{fst},J,a) = \{p \in [\![fm]\!] \mid \mathbf{fst}(p,a){=}(s,r) \Rightarrow |J| \in s \wedge 0 \notin r\}$. This is the set of all products p such that $\mathbf{fst}(p,a)$ allows $|J|$ as number of senders and requires at least one receiver.
3. $\chi_{P(q)}$ is the feature expression which characterises the set $P(q)$ of products for which state q is reachable by transitions of $\mathbf{fst}[\mathcal{S}]$ whose constraints are satisfied by p, i.e. $P(q) = \{p \in [\![fm]\!] \mid \exists_{q_0 \in I} \cdot\ q_0 \xrightarrow{l_1}_{\mathbf{fst}[\mathcal{S}]} q_1 \xrightarrow{l_2} \ldots \xrightarrow{l_n}_{\mathbf{fst}[\mathcal{S}]} q_n = q$ for some $n \geq 0$, and $p \models \gamma(q_{i-1} \xrightarrow{l_i}_{\mathbf{fst}[\mathcal{S}]} q_i)$ for $i = 1,\ldots,n\}$.

In summary, an fReq for $\mathbf{fst}[\mathcal{S}]$ has the form $[\mathbf{prod}(J,a,q)]\,\mathbf{rcp}(J,a)@q$, where $q \in Q$ is a reachable state of $\mathbf{fst}[\mathcal{S}]$, $a \in \Sigma$, $\emptyset \neq J \subseteq \mathcal{N}$ is a set of component names such that $\forall_{j \in J} \cdot a \in \Sigma^!_j \wedge a\ \mathbf{en}_{A_j}@q_j$, and $\mathbf{prod}(J,a,q)$ is a satisfiable feature expression as defined above. Note that $\models \mathbf{prod}(J,a,q) \rightarrow fm$, because $P(\mathbf{fst},J,a)$ in item 2 (and also $P(q)$ in item 3) is a subset of $[\![fm]\!]$.

The following lemma provides a formal relation between Reqs and fReqs.

Lemma 1. *For all products p it holds: $[\mathbf{prod}(J,a,q)]\,\mathbf{rcp}(J,a)@q$ is an fReq for $\mathbf{fst}[\mathcal{S}]$ and $p \models \mathbf{prod}(J,a,q)$ iff $p \in [\![fm]\!]$ and $\mathbf{rcp}(J,a)@q$ is a Req for $\mathbf{fst}[\mathcal{S}]^p$.*

Example 5. Fig. 6 shows an excerpt of the fETA $\mathbf{fst}[\mathcal{S}_@]$ in Fig. 5 depicting the fReqs for states $(0,0,0)$, $(0,1,1)$, and $(0,2,0)$. First note that an output of *join* is enabled at local state 0 in both components A_{u_1} and A_{u_2}. For $\mathbf{rcp}(\{u_1\},join)$ at state $(0,0,0)$ we get $\mathbf{fe}(\{u_1\},join,(0,0,0)) = 🔒\vee🔓$ according to the constraints of both *join* transitions in A_{u_1}. Moreover, $P(\mathbf{fst},\{u_1\},join) = \{\{🔒\},\{🔓\}\} = [\![fm]\!]$ and therefore $\chi_{P(\mathbf{fst},\{u_1\},join)}$ is equivalent to fm. Also $P(0,0,0) = \{\{🔒\},\{🔓\}\}$ since state $(0,0,0)$ is reachable in both products. So $\mathbf{prod}(\{u_1\},join,(0,0,0)) = (🔒 \vee 🔓) \wedge fm \wedge fm$, which reduces to $fm = 🔒 \oplus 🔓$. Thus we get the fReq $[🔒 \oplus 🔓]\,\mathbf{rcp}(\{u_1\},join)$ at $(0,0,0)$. The case of $\{u_2\}$ is analogous.

Considering a possible simultaneous output of *join* by u_1 and u_2 we get $\mathbf{fe}(\{u_1,u_2\},join,(0,0,0)) = (🔒\vee🔓)\vee(🔒\vee🔓)$. And we get $P(\mathbf{fst},\{u_1,u_2\},join) = \{\{🔓\}\}$, since only for the product $\{🔓\}$ a synchronisation of several users is allowed. Therefore $\chi_{P(\mathbf{fst},\{u_1,u_2\},join)} = 🔓 \wedge \neg 🔒$. As above, $\chi_{P(0,0,0)} = fm$. Thus $\mathbf{prod}(\{u_1,u_2\},join,(0,0,0)) = (🔒\vee🔓)\wedge(🔓\wedge\neg🔒)\wedge fm$, which reduces to $🔓\wedge\neg🔒$. Hence we get the fReq $[🔓 \wedge \neg 🔒]\,\mathbf{rcp}(\{u_1,u_2\},join)$ at $(0,0,0)$.

An interesting case is $[🔒 \wedge \neg 🔓]\,\mathbf{rcp}(\{u_1\},join)$ at $(0,1,1)$. Here $\mathbf{fe}(\{u_1\},join,(0,1,1))$ is again $🔒\vee🔓$ and $\chi_{P(\mathbf{fst},\{u_1\},join)}$ is equivalent to fm. However, the state $(0,1,1)$ is only reachable in the product $\{🔒\}$, i.e. $P(0,1,1) = \{\{🔒\}\}$. Therefore, $\chi_{P(0,1,1)} = 🔒\wedge\neg🔓$. In summary, $\mathbf{prod}(\{u_1,u_2\},join,(0,1,1)) = (🔒\wedge🔓)\wedge fm\wedge(🔒\wedge\neg🔓)$, which reduces to $(🔒 \wedge \neg 🔓)$. The other fReqs are computed similarly. ▷

Next, we define featured compliance with an fReq. We use a logical formulation which, as we shall see, captures compliance for the whole family of products.

The fETA $\mathbf{fst}[\mathcal{S}]$ is *featured compliant* with an fReq $[\psi]\,\mathbf{rcp}(J,a)@q$ if for some $n \geq 1$ and for $k = 1,\ldots,n$ there exist transitions $t^k = q \xrightarrow{(J,a,R^k)}_{\mathbf{fst}[\mathcal{S}]} q^k$ with $R^k \neq \emptyset$ such that $\models \psi \rightarrow \bigvee_{k\in\{1,\ldots,n\}} \gamma(t^k)$.

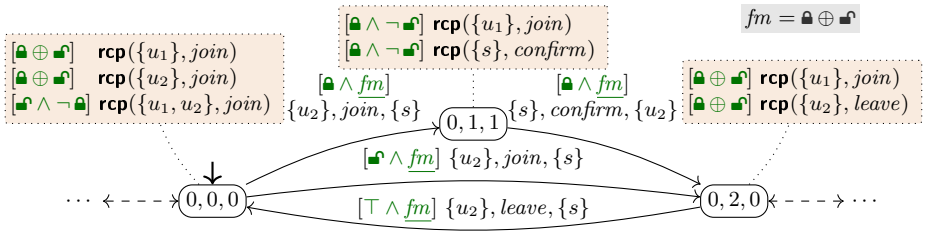


Fig. 6: Part of $\mathbf{fst}[\mathcal{S}_@]$ from Fig. 5 enriched with Reqs

The definition of featured compliance can be unfolded by considering all $p \in [\![fm]\!]$. This shows the relationship to the compliance notion for ETA.

Lemma 2. *Let* $[\psi]\,\mathbf{rcp}(J,a)@q$ *be an fReq for the fETA* $\mathbf{fst}[\mathcal{S}]$. *Then:*

$$\left[\begin{array}{c}\mathbf{fst}[\mathcal{S}] \text{ is featured compliant}\\ \text{with } [\psi]\,\mathbf{rcp}(J,a)@q\end{array}\right] \Leftrightarrow \left[\begin{array}{c}\forall_{p\subseteq F \text{ with } p\models\psi}\ .\quad \exists_{R\neq\emptyset \text{ and } q'\in Q}.\\ q \xrightarrow{(J,a,R)}_{\mathbf{fst}[\mathcal{S}]} q' \text{ and } p \models \gamma(q \xrightarrow{(J,a,R)}_{\mathbf{fst}[\mathcal{S}]} q')\end{array}\right]$$

The next definition generalises featured compliance to featured weak compliance. It is a technical but straightforward extension that transfers the concept of weak receptiveness to the featured level.

The fETA $\mathbf{fst}[\mathcal{S}]$ is *featured weakly compliant* with an fReq $[\psi]\,\mathbf{rcp}(J,a)@q$ if for some $n \geq 1$ and for $k = 1,\ldots,n$ there exist sequences σ^k of transitions

$$\sigma^k = q_0^k \xrightarrow{(S_0^k,a_0^k,R_0^k)}_{\mathbf{fst}[\mathcal{S}]} q_1^k \ \cdots\ q_{m_k}^k \xrightarrow{(S_{m_k}^k,a,R_{m_k}^k)}_{\mathbf{fst}[\mathcal{S}]} q_{m_k+1}^k$$

with $q_0^k = q, m_k \geq 0$, $(S_i^k \cup R_i^k) \cap J = \emptyset$ for $i = 0,\ldots,m_{k-1}$, $R_i^k \neq \emptyset$ for $i = 0,\ldots,m_k$, and $S_{m_k}^k = J$ such that

$$\models \psi \rightarrow \bigvee_{k\in\{1,\ldots,n\}} \bigwedge_{i\in\{0,\ldots,m_k\}} \gamma(q_i^k \xrightarrow{(S_i^k,a_i^k,R_i^k)}_{\mathbf{fst}[\mathcal{S}]} q_{i+1}^k).$$

We remark that Lemma 2 can be extended in a straightforward way to characterise featured weak compliance.

The fETA $\mathbf{fst}[\mathcal{S}]$ is *featured (weakly) receptive* if it is featured (weakly) compliant with all fReqs for $\mathbf{fst}[\mathcal{S}]$.

Example 6. We consider some fReqs for the fETA $\mathbf{fst}[\mathcal{S}_@]$ as depicted in Fig. 6. The first fReq is $[\text{🔒} \oplus \text{🔓}]\,\mathbf{rcp}(\{u_1\}, join)@(0,0,0)$. As we can see in Fig. 5, there are two transitions, say t_1, t_2, in $\mathbf{fst}[\mathcal{S}_@]$ with source state $(0,0,0)$ and label $(\{u_1\}, join, \{s\})$, such that $\gamma(t_1) = \text{🔒} \wedge fm$ and $\gamma(t_2) = \text{🔓} \wedge fm$. Hence, for checking featured compliance with this fReq we have to prove:

$$\models \text{🔒} \oplus \text{🔓} \rightarrow (\text{🔒} \wedge fm) \vee (\text{🔓} \wedge fm).$$

But this is easy, since the conclusion is equivalent to $fm = 🔒 \oplus 🔓$. To achieve this it is essential to have the disjunction of $\gamma(t_1)$ and $\gamma(t_2)$ in the conclusion.

As a second fReq we consider $[🔓 \wedge \neg 🔒]\,\mathbf{rcp}(\{u_1, u_2\}, join)@(0,0,0)$. As we can see in Fig. 5, there is one transition in $\mathbf{fst}[\mathcal{S}_@]$ with source state $(0,0,0)$ and label $(\{u_1, u_2\}, join, \{s\})$, which has the transition constraint $🔓 \wedge \neg 🔒$. Featured compliance with this fReq holds trivially, since

$$\models 🔓 \wedge \neg 🔒 \rightarrow 🔓 \wedge \neg 🔒.$$

As a last example, consider the fReq $[🔒 \wedge \neg 🔓]\,\mathbf{rcp}(\{u_1\}, join)@(0,1,1)$. In state $(0,1,1)$, no transition with action *join* can be performed by the fETA $\mathbf{fst}[\mathcal{S}_@]$. Therefore featured compliance does not hold. However, featured weak compliance holds for the following reasons. We take $n = 1$ (in the definition of featured weak compliance) and select, in Fig. 5, the transition sequence

$$(0,1,1) \xrightarrow{[🔒 \wedge \underline{fm}](\{s\},\, confirm,\, \{u_2\})} (0,2,0) \xrightarrow{[🔒 \wedge \underline{fm}](\{u_1\},\, join,\, \{s\})} (1,2,1).$$

Then, we get the following proof obligation (conjoining the constraints of the two consecutive transitions in the conclusion): $\models (🔒 \wedge \neg 🔓) \rightarrow (🔒 \wedge fm) \wedge (🔒 \wedge fm)$. Obviously, this holds since the conclusion reduces to $🔒 \wedge \neg 🔓$. We can show that the fETA $\mathbf{fst}[\mathcal{S}_@]$ is either featured compliant or featured weakly compliant with any fReq and therefore it is featured weakly receptive. ▷

4.3 From Featured Receptiveness to Receptiveness

This section presents our main result. We show that instead of checking product-wise each member of a family of product configurations for (weak) receptiveness, it is sufficient to verify once featured (weak) receptiveness for the family model. We can even show that this technique is not only sound but also complete in the sense, that if we disprove featured (weak) receptiveness on the family level, then there will be a product for which the projection is not (weakly) receptive.

Theorem 2. *Let S be an fSys with feature model fm, let* **fst** *be an fSTS, and let* $\mathbf{fst}[S]$ *be its generated fETA. Then:*

$$\Big[\mathbf{fst}[\mathcal{S}] \textit{ is featured (weakly) receptive}\Big] \Leftrightarrow \Big[\forall_{p \in [\![fm]\!]} \cdot \mathbf{fst}[\mathcal{S}]^p \textit{ is (weakly) receptive}\Big]$$

Example 7. In Example 6 we showed that the fETA $\mathbf{fst}[\mathcal{S}_@]$ is featured weakly receptive. Therefore, by applying Theorem 2, we know that for both products $\{🔒\}$ and $\{🔓\}$, the ETA $\mathbf{fst}^{🔒}[\mathcal{S}^{🔒}_@]$ and $\mathbf{fst}^{🔓}[\mathcal{S}^{🔓}_@]$ are weakly receptive (a result which we checked product-wise in Example 4). ▷

Note on Complexity Note that an fReq for $\mathbf{fst}[\mathcal{S}]$ necessarily involves a syntactic application condition, which is a feature expression that characterises the set of valid products p for which the featureless Req is applicable for $\mathbf{fst}[\mathcal{S}]^p$. Part of this feature expression is a characterisation $\chi_{P(q)}$ of the set $P(q)$ of products for

which state q is reachable by transitions of $\mathbf{fst}[\mathcal{S}]$ whose constraints are satisfied by p, which requires a reachability check for q. This may seem computationally expensive. However, it has been shown that static analysis of properties of fTSs that concern the reachability of states and transitions in valid products (LTSs) is feasible in reasonable time even for fTSs of considerable size, by reducing the analysis to SAT solving [8]. In fact, while SAT solving is NP-complete, SAT solvers are effectively used for static analysis of feature models with hundreds of thousands of clauses and tens of thousands of variables [33,32]. Finally, we note that the results presented in this section are still sound, but not complete, without the aforementioned characterisation of $P(q)$.

5 Tool Support

We implemented a prototypical tool to specify and analyse fETA. This requires to define an fSys over a set of fCA, a shared feature model, and an fSTS. The tool can be used online and downloaded at https://github.com/arcalab/team-a. The interface is organised by 5 widgets (illustrated in Fig. 7): (1) a text editor to specify a fETA, using a dedicated domain-specific language; (2) an fTS view of the fETA, together with the fReqs generated automatically for each state, similar to Fig. 6; (3) a set of example fETA; (4) a view of each individual fCA, similar to Fig. 1; and (5) some statistics of the various models, including the number of states, transitions, features, and products.

The tool is written in Scala and it uses the Play Framework to generate an interactive website using a client-server architecture. The Scala code is compiled into JavaScript using `Scala.js` to run on the client side, and into JVM binaries that run on the server side. The server side is currently needed to use an off-the-shelf Java library, `Sat4j`, to find all products that satisfy a feature model.

6 Conclusion

We introduced featured team automata to specify and analyse systems of featured component automata and to explore composition and communication-safety. We showed that family-based analysis of receptiveness (no message loss) suffices to study receptiveness of product configurations. We implemented our theory in a prototypical tool.

In the future, we intend to extend our theory to address i) responsiveness, i.e. no indefinite waiting for input, and ii) compositionality, i.e. extend fETA to composition of systems (that behaves well with fSTSs) and investigate conditions under which communication safety is preserved by fETA composition. Moreover, we will further develop the tool and analyse the practical impact of fETA on the basis of larger case studies. This involves a thorough study of the efficiency of featured receptiveness checking compared to product-wise checking of receptiveness. Finally, we aim to implement a family-based analysis algorithm that computes, for a given fETA, the set of all product configurations that yield communication-safe systems.

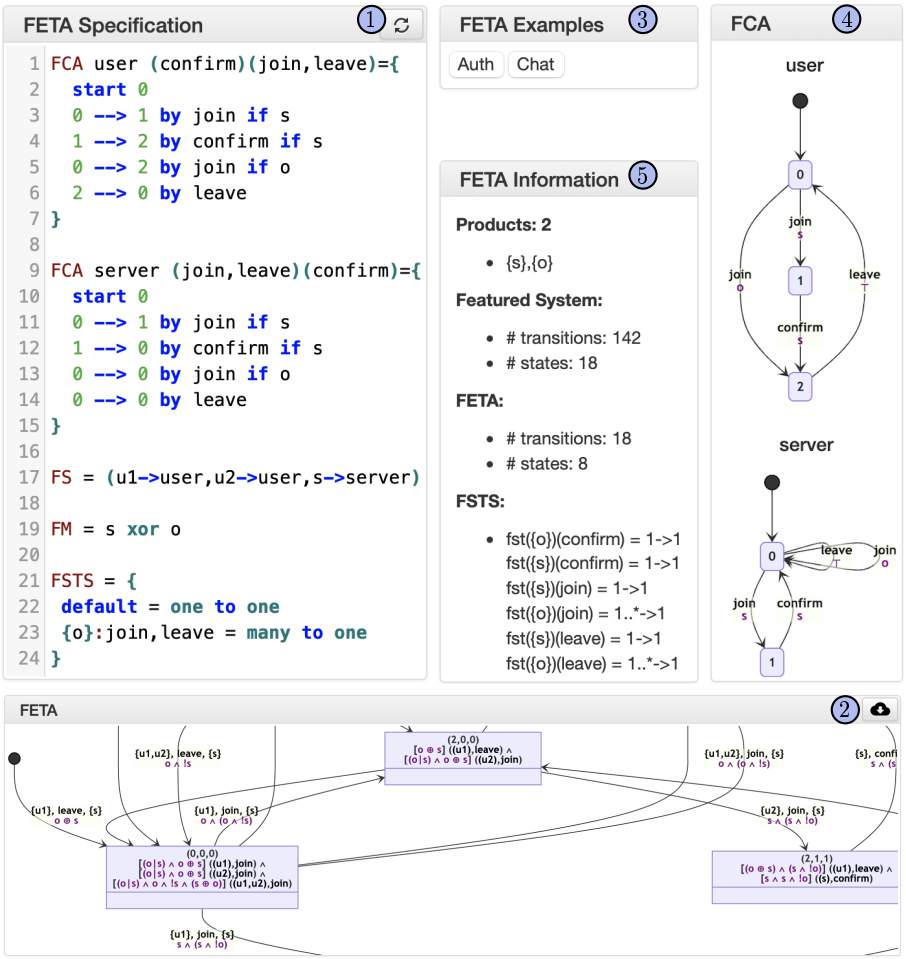


Fig. 7: Screenshots of the widgets in the online tool for fETA

Acknowledgments. Ter Beek received funding from the MIUR PRIN 2017FTXR7S project IT MaTTerS (Methods and Tools for Trustworthy Smart Systems). Cledou and Proença received funding from the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme (project DaVinci, POCI-01-0145-FEDER-029946) and by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia. Proença also received National Funds through FCT/MCTES, within the CISTER Research Unit (UIDP/UIDB/04234/2020); by the Norte Portugal Regional Operational Programme – NORTE 2020 (project REASSURE, NORTE-01-0145-FEDER-028550) under the Portugal 2020 Partnership Agreement, through ERDF the FCT; and European Funds through the ECSEL Joint Undertaking (JU) under grant agreement No 876852 (project VALU3S).

References

1. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer (2013). https://doi.org/10.1007/978-3-642-37521-7
2. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: Formal Description of Variability in Product Families. In: Proceedings of the 15th International Software Product Lines Conference (SPLC). pp. 130–139. IEEE (2011). https://doi.org/10.1109/SPLC.2011.34
3. Basile, D., ter Beek, M.H., Degano, P., Legay, A., Ferrari, G.L., Gnesi, S., Di Giandomenico, F.: Controller synthesis of service contracts with variability. Sci. Comput. Program. **187** (2020). https://doi.org/10.1016/j.scico.2019.102344
4. Basile, D., Di Giandomenico, F., Gnesi, S., Degano, P., Ferrari, G.L.: Specifying Variability in Service Contracts. In: Proceedings of the 11th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS). pp. 20–27. ACM (2017). https://doi.org/10.1145/3023956.3023965
5. Bauer, S.S., Mayer, P., Schroeder, A., Hennicker, R.: On Weak Modal Compatibility, Refinement, and the MIO Workbench. In: Esparza, J., Majumdar, R. (eds.) TACAS. LNCS, vol. 6015, pp. 175–189. Springer (2010). https://doi.org/10.1007/978-3-642-12002-2_15
6. ter Beek, M.H., Carmona, J., Hennicker, R., Kleijn, J.: Communication Requirements for Team Automata. In: Jacquet, J.M., Massink, M. (eds.) COORDINATION. LNCS, vol. 10319, pp. 256–277. Springer (2017). https://doi.org/10.1007/978-3-319-59746-1_14
7. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Featured Team Automata. arXiv:2108.01784 **[cs.FL]** (August 2021), https://arxiv.org/abs/2108.01784
8. ter Beek, M.H., Damiani, F., Lienhardt, M., Mazzanti, F., Paolini, L.: Efficient Static Analysis and Verification of Featured Transition Systems. Empir. Softw. Eng. (2021). https://doi.org/10.1007/s10664-020-09930-8
9. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Synchronizations in Team Automata for Groupware Systems. Comput. Sup. Coop. Work **12**(1), 21–69 (2003). https://doi.org/10.1023/A:1022407907596
10. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints. J. Log. Algebr. Meth. Program. **85**(2), 287–315 (2016). https://doi.org/10.1016/j.jlamp.2015.11.006
11. ter Beek, M.H., Hennicker, R., Kleijn, J.: Compositionality of Safe Communication in Systems of Team Automata. In: Pun, V.K.I., Simão, A., Stolz, V. (eds.) ICTAC. LNCS, vol. 12545, pp. 200–220. Springer (2020). https://doi.org/10.1007/978-3-030-64276-1_11
12. ter Beek, M.H., Kleijn, J.: Modularity for teams of I/O automata. Inf. Process. Lett. **95**(5), 487–495 (2005). https://doi.org/10.1016/j.ipl.2005.05.012
13. ter Beek, M.H., van Loo, S., de Vink, E.P., Willemse, T.A.: Family-Based SPL Model Checking Using Parity Games with Variability. In: Wehrheim, H., Cabot, J. (eds.) FASE. LNCS, vol. 12076, pp. 245–265. Springer (2020). https://doi.org/10.1007/978-3-030-45234-6_12
14. ter Beek, M.H., de Vink, E.P., Willemse, T.A.C.: Family-Based Model Checking with mCRL2. In: Huisman, M., Rubin, J. (eds.) FASE. LNCS, vol. 10202, pp. 387–405. Springer (2017). https://doi.org/10.1007/978-3-662-54494-5_23

15. Carmona, J., Cortadella, J.: Input/Output Compatibility of Reactive Systems. In: Aagaard, M.D., O'Leary, J.W. (eds.) FMCAD. LNCS, vol. 2517, pp. 360–377. Springer (2002). https://doi.org/10.1007/3-540-36126-X_22
16. Carmona, J., Kleijn, J.: Compatibility in a multi-component environment. Theor. Comput. Sci. **484**, 1–15 (2013). https://doi.org/10.1016/j.tcs.2013.03.006
17. Ceska, M., Jansen, N., Junges, S., Katoen, J.: Shepherding Hordes of Markov Chains. In: Vojnar, T., Zhang, L. (eds.) TACAS. LNCS, vol. 11428, pp. 172–190. Springer (2019). https://doi.org/10.1007/978-3-030-17465-1_10
18. Chrszon, P., Dubslaff, C., Klüppelholz, S., Baier, C.: ProFeat: feature-oriented engineering for family-based probabilistic model checking. Form. Asp. Comp. **30**(1), 45–75 (2018). https://doi.org/10.1007/s00165-017-0432-4
19. Classen, A.: Modelling and Model Checking Variability-Intensive Systems. Ph.D. thesis, University of Namur (2011), http://hdl.handle.net/2078.2/90863
20. Classen, A., Cordy, M., Schobbens, P.Y., Heymans, P., Legay, A., Raskin, J.F.: Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. IEEE Trans. Softw. Eng. **39**(8), 1069–1089 (2013). https://doi.org/10.1109/TSE.2012.86
21. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A., Raskin, J.F.: 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). pp. 335–344. ACM (2010). https://doi.org/10.1145/1806799.1806850
22. Damiani, F., Schaefer, I.: Family-based analysis of type safety of delta-oriented software product lines. In: Margaria, T., Steffen, B. (eds.) ISoLA. LNCS, vol. 7609, pp. 193–207. Springer (2012). https://doi.org/10.1007/978-3-642-34026-0_15
23. de Alfaro, L., Henzinger, T.A.: Interface Automata. In: Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE). pp. 109–120. ACM (2001). https://doi.org/10.1145/503209.503226
24. Dimovski, A.S.: CTL* family-based model checking using variability abstractions and modal transition systems. Int. J. Softw. Tools Technol. Transf. **22**(1), 35–55 (2020). https://doi.org/10.1007/s10009-019-00528-0
25. Dimovski, A.S., Al-Sibahi, A.S., Brabrand, C., Wąsowski, A.: Efficient family-based model checking via variability abstractions. Int. J. Softw. Tools Technol. Transf. **19**(5), 585–603 (2017). https://doi.org/10.1007/s10009-016-0425-2
26. Ellis, C.A.: Team Automata for Groupware Systems. In: Proceedings of the 1st International ACM SIGGROUP Conference on Supporting Group Work (GROUP). pp. 415–424. ACM (1997). https://doi.org/10.1145/266838.267363
27. Fantechi, A., Gnesi, S.: A Behavioural Model for Product Families. In: Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE). pp. 521–524. ACM (2007). https://doi.org/10.1145/1287624.1287700
28. Fischbein, D., Uchitel, S., Braberman, V.A.: A Foundation for Behavioural Conformance in Software Product Line Architectures. In: Proceedings of the ISSTA Workshop on Role of Software Architecture for Testing and Analysis (ROSATEA). pp. 39–48. ACM (2006). https://doi.org/10.1145/1147249.1147254
29. Hennicker, R., Bidoit, M.: Compatibility Properties of Synchronously and Asynchronously Communicating Components. Log. Meth. Comp. Sci. **14**(1), 1–31 (2018). https://doi.org/10.23638/LMCS-14(1:1)2018

30. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: De Nicola, R. (ed.) ESOP. LNCS, vol. 4421, pp. 64–79. Springer (2007). https://doi.org/10.1007/978-3-540-71316-6_6
31. Lauenroth, K., Pohl, K., Töhning, S.: Model Checking of Domain Artifacts in Product Line Engineering. In: Proceedings of the 24th International Conference on Automated Software Engineering (ASE). pp. 269–280. IEEE (2009). https://doi.org/10.1109/ASE.2009.16
32. Liang, J.H., Ganesh, V., Czarnecki, K., Raman, V.: SAT-based Analysis of Large Real-world Feature Models is Easy. In: Proceedings of the 19th International Software Product Line Conference (SPLC). pp. 91–100. ACM (2015). https://doi.org/10.1145/2791060.2791070
33. Mendonça, M., Wąsowski, A., Czarnecki, K.: SAT-based Analysis of Feature Models is Easy. In: Proceedings of the 13th International Software Product Line Conference (SPLC). pp. 231–240. ACM (2009)
34. Muschevici, R., Proença, J., Clarke, D.: Feature Nets: behavioural modelling of software product lines. Softw. Sys. Model. **15**(4), 1181–1206 (2016). https://doi.org/10.1007/s10270-015-0475-z
35. Muschevici, R., Proença, J., Clarke, D.: Modular Modelling of Software Product Lines with Feature Nets. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM. LNCS, vol. 7041, pp. 318–333. Springer (2011). https://doi.org/10.1007/978-3-642-24690-6_22
36. Schobbens, P., Heymans, P., Trigaux, J.C., Bontemps, Y.: Feature Diagrams: A Survey and a Formal Semantics. In: Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE). pp. 136–145. IEEE (2006). https://doi.org/10.1109/RE.2006.23
37. Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G.: A Classification and Survey of Analysis Strategies for Software Product Lines. ACM Comput. Surv. **47**(1), 6 (2014). https://doi.org/10.1145/2580950
38. Thüm, T., Schaefer, I., Hentschel, M., Apel, S.: Family-based deductive verification of software product lines. In: Proceedings of the 11th International Conference on Generative Programming and Component Engineering (GPCE). pp. 11–20. ACM (2012). https://doi.org/10.1145/2371401.2371404