Coherent branching feature bisimulation

Tessa Belder
TU/e, Eindhoven
The Netherlands

Maurice H. ter Beek
ISTI–CNR, Pisa
Italy

Erik P. de Vink*
TU/e, Eindhoven & CWI, Amsterdam
The Netherlands

Progress in the behavioral analysis of software product lines at the family level benefits from further development of the underlying semantical theory. Here, we propose a behavioral equivalence for feature transition systems (FTS) generalizing branching bisimulation for labeled transition systems (LTS). We prove that branching feature bisimulation for an FTS of a family of products coincides with branching bisimulation for the LTS projection of each the individual products. For a restricted notion of coherent branching feature bisimulation we furthermore present a minimization algorithm and show its correctness. Although the minimization problem for coherent branching feature bisimulation is shown to be intractable, application of the algorithm in the setting of a small case study results in a significant speed-up of model checking of behavioral properties.

1 Introduction

Notions of behavioral equivalence, like bisimulation, play an important role in the analysis of large systems in general and thus of (software) product lines in particular. Abstractions based on behavioral equivalences compress, via abstraction operations and minimization algorithms, a model's state space prior to verification. Subsequently, verification can be done in less time, using less memory.

Compared to single system verification, SPLE adds variability as yet another dimension to the complexity of behavioral analysis. In general, the number of possible products of a product line is exponential in the number of features. This calls for dedicated modeling and analysis techniques that allow to specify and reason about an entire product line at once. In this paper we consider the model of feature transition systems [5, 6], which facilitates efficient family-based verification. Dedicated techniques generally use variability knowledge about valid feature configurations to deduce results for products from a family model, as opposed to enumerative product-based verification, in which every product is examined individually. For example, in [7] behavioral pre-orders of FTS are given with respect to specific products to define abstractions based on simulation quotients that preserve LTL properties. We refer to [19] for an overview of verification strategies in SPLE and the trade-off of product-based vs. family-based analysis.

In [3,4] we applied tailored property preserving reductions to a product line modeled with mCRL2 [8] and we verified by means of model checking a number of behavioral properties of the product line. The mCRL2 toolset provides specific support for reduction modulo branching bisimulation [14]. This led us to investigate a feature-oriented notion of branching bisimulation inspired by the research reported in [7] (which focuses on a notion of simulation). In this paper, we propose a definition of what is coined branching feature bisimulation, extending the definition in [14], and we seek to adapt the efficient algorithm of [15] to compute, given an FTS, a minimal FTS that is branching feature bisimilar.

In our pursuit to transfer the results of [7] to the case of branching bisimulation, a number of issues arises due to the presence of feature expressions, though. One such issue for FTS is that minimization in the number of states is not the same as minimization in the number of transitions, a situation that does not occur with LTS. Our effort here is to reduce in the number of states. In order to make our minimization

*Corresponding author, email evink@win.tue.nl.

J. Atlee and S. Gnesi (Eds.): FMSPLE 2015
EPTCS 182, 2015, pp. 14–30, doi:10.4204/EPTCS.182.2

algorithm work, we restrict to so-called coherent rather than arbitrary branching feature bisimulation relations. We will prove that our algorithm reduces an FTS $\mathcal{S}$ to a minimal FTS $\mathcal{S}_{min}$ for which there exists a coherent branching feature bisimulation relation for $\mathcal{S}$ and $\mathcal{S}_{min}$. Moreover, no smaller FTS $\mathcal{S}'$ exists which is also coherent branching feature bisimilar to $\mathcal{S}$. However, as we will argue by a reduction of graph coloring, the minimization problem is NP-complete for coherent branching feature bisimulations (and we suspect this is the case for branching feature bisimulation as well). Still, as an evaluation of the approach for a relatively small toy example illustrates, overall a substantial reduction in computation time is achieved for bisimulation-enhanced family-based analysis as compared to enumerative product-based analysis. In particular, for properties involving a limited number of features, verification time using the family FTS is only a third to a quarter of the time needed to verify all product LTS.

Behavioral equivalences also form the basis of conformance notions as used for model elaboration by iterative refinement of partial behavioral models. In SPLE, this allows to relate fully configured product behavior to family models with optional behavior reflecting product variability. Examples are approaches based on process algebra [20] and on modal transition systems (MTS) [1, 2, 10]. In [20], a so-called variant process algebra is introduced, which allows to model family behavior that subsumes the behavior of all possible product variants. Special-purpose bisimulation relations then allow to compare variants among each other and against the family. In SPLE, MTS are one of the models used to specify family behavior encompassing all possible product behavior, represented by those LTS that are implementations of the MTS (obtained by refinement of admissible behavior). In [10], weak and strong refinement for MTS as defined in [16] (based on weak and strong bisimulation) are shown to be inadequate for applications in SPLE (mainly due to the lack of support for unobservable actions and for preserving branching behavior, respectively) and a novel notion of refinement is introduced preserving the branching structure. It moreover preserves properties expressed in 3-valued weak μ-calculus. However, its definition is not operational and algorithms for conformance checking conformance are thus infeasable.

The paper outline is as follows. Building on definitions and an algorithm for branching bisimulation of LTS reviewed in Section 2, we introduce in Section 3 the notion of branching feature bisimulation and show its soundness for branching bisimulation with respect to all products. The algorithm for minimizing modulo coherent branching feature bisimulation is given in Section 4, which also provides an NP-completeness proof for the minimization problem. A validation of the approach, based on a toy example of a product line of coffee/soup vending machines is reported in Section 5. Finally, Section 6 briefly wraps up with concluding remarks and future work.

2 Branching bisimulation for labeled transition systems

Strong bisimulation is a cornerstone of the theory of LTS [17], but is often too fine a behavioral equivalence for verification purposes. Application of its minimization algorithm typically reduces the system under verification only in a limited way. Having this in mind, various weaker notions have been studied in the literature [11, 12]. In the context of model checking, branching bisimulation as proposed for LTS by Van Glabbeek & Weijland enjoys a number of appealing properties [13]. We recall and illustrate its definition, and discuss the outline of a minimization algorithm that returns the smallest LTS that is branching bisimilar to a given one. To this end, we fix an alphabet of actions $\mathcal{A}$, distinguish a symbol $\tau \notin \mathcal{A}$, referred to as the silent action, and let $\mathcal{A}_\tau = \mathcal{A} \cup \{\tau\}$.

Definition 1. *A labeled transition system is a triple $\mathcal{S} = (S, \rightarrow, s_*)$ with set of states S, transition relation $\rightarrow \subseteq S \times \mathcal{A}_\tau \times S$, and initial state $s_* \in S$.*

(a) For $s,s' \in S$, we write $s \Rightarrow s'$ if $\exists n \exists s_0 \cdots s_n \colon s_0 = s \wedge \big(\forall i, 1 \leqslant i \leqslant n \colon s_{i-1} \xrightarrow{\tau} s_i\big) \wedge s_n = s'$.

(b) A symmetric relation $R \subseteq S \times S$ is called a branching bisimulation relation if $\forall s,s',t \in S$, $\alpha \in \mathcal{A}_\tau$ such that $R(s,t)$ and $s \xrightarrow{\alpha} s'$, it holds that $R(s,\hat{t})$, $R(s',t')$ and $t \Rightarrow \hat{t} \xrightarrow{(\alpha)} t'$ for some $\hat{t},t' \in S$.

(c) Two states s,t of $\mathcal{S}$ are called branching bisimilar if $R(s,t)$ for some branching bisimulation relation R. Notation $s \simeq_b t$.

Note the notation $\hat{t} \xrightarrow{(\alpha)} t'$ used in part (b) of this definition. Following [14], we have $\hat{t} \xrightarrow{(\alpha)} t'$ if either $\hat{t} \xrightarrow{\alpha} t'$ or $\alpha = \tau$ and $\hat{t} = t'$, an elegant trick to allow the transition $s \xrightarrow{\tau} s'$ to be matched by $t = \hat{t} = t'$, i.e. by no transition for t in case $R(s',t)$.

In Figure 1 at the left-hand side, s_0 and t_0 are not branching bisimilar: Clearly state s_1 is not branching bisimilar to state t_0 since s_1 has no b-transition. But then, the transition $t_0 \xrightarrow{a} t_2$ cannot be matched by the transition sequence $s_0 \Rightarrow s_1 \xrightarrow{a} s_2$ because the intermediate state s_1 cannot be related to state t_0, as specifically required by the definition. However, for u_0 and v_0 at the right-hand side, the transition $v_0 \xrightarrow{a} v_1$ can be matched by $u_0 \Rightarrow u_1 \xrightarrow{a} u_4$, since in this case v_0 and u_1 are branching bisimilar. It is noted that u_0 and v_0, but also s_0 and t_0, are weakly bisimilar in the sense of Milner [17].

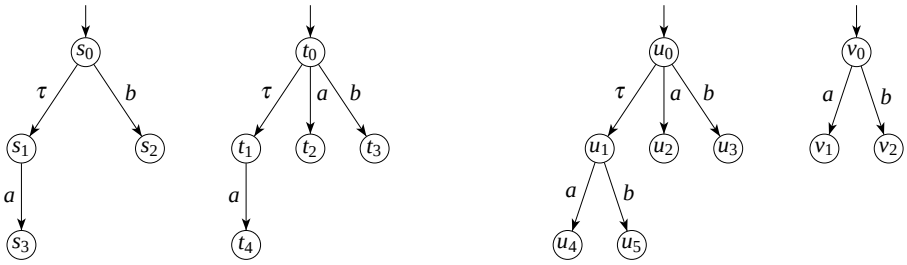

Figure 1: Two non-branching bisimilar states and two branching bisimilar states

An efficient minimization algorithm for branching bisimulation is due to Groote & Vaandrager [15], based on the partition refinement algorithm of Paige & Tarjan [18]. It involves the notions of a partition of the set of states, and of a splitter: Consider a finite LTS $\mathcal{S} = (S, \rightarrow, s_*)$ over the action set $\mathcal{A}_\tau$.

- A partition of $\mathcal{S}$ is a collection $\mathcal{B} = \{ B_i \mid i \in I \}$ of subsets of S that disjointly covers S, i.e. $\bigcup_{i \in I} B_i = S$, and $B_i \cap B_j = \varnothing$ if $i \neq j$, for all $i,j \in I$. The elements of a partition are referred to as blocks.
- For a partition $\mathcal{B}$, blocks $B,B' \in \mathcal{B}$, and $\alpha \in \mathcal{A}_\tau$ we let $pos_\alpha(B,B') = \{ s \in B \mid \exists \hat{s} \in B \exists s' \in B' \colon s \Rightarrow \hat{s} \xrightarrow{\alpha} s' \}$, and $neg_\alpha(B,B') = \{ s \in B \mid \forall \hat{s} \in B \forall s' \in B' \colon (s \not\Rightarrow \hat{s}) \vee (\hat{s} \xnrightarrow{\alpha} s') \}$.
- For blocks B,B' of a partition $\mathcal{B}$, the block B' is called a splitter of B for an action $\alpha \in \mathcal{A}_\tau$ if both $pos_\alpha(B,B') \neq \varnothing$ and $neg_\alpha(B,B') \neq \varnothing$.

A simplified version of the algorithm of [15] for minimization modulo branching bisimulation starts with the trivial partition $\mathcal{B} = \{S\}$ and iterates

while splitter B' of block $B \in \mathcal{B}$ for $\alpha \in \mathcal{A}_\tau$ exists **do** $\mathcal{B} := (\mathcal{B} \backslash \{B\}) \cup \{pos_\alpha(B,B'), neg_\alpha(B,B')\}$ **end**

Thus, starting from the trivial partition $\{S\}$, having the complete set of states S as a single block, we keep refining the partition based on a splitter. Clearly, the algorithm terminates for a finite LTS in at most $|S|$ many steps. We refer to [15] for a proof of the following result.

Theorem 2. *Assume $\mathcal{B}_{min}$ is the partition obtained upon termination after applying the algorithm to the LTS $\mathcal{S} = (S, \rightarrow, s_*)$. Define the LTS $\mathcal{S}_{min} = (\mathcal{B}_{min}, \rightarrow_{min}, B_*)$ by letting $B \xrightarrow{\alpha}_{min} B'$ if there exist $s \in B$, $s' \in B'$ such that $s \xrightarrow{\alpha} s'$ for $B, B' \in \mathcal{B}$, $\alpha \in \mathcal{A}_\tau$ with $B \neq B'$ or $\alpha \neq \tau$, and by choosing B_* such that $s_* \in B_*$. Then $\mathcal{S}_{min}$ is the smallest LTS that is branching bisimilar to $\mathcal{S}$.* □

In the simplified algorithm sketched above, major part of the computation is spent on unfolding of the relation $\Rightarrow$. The algorithm of [15] reduces this by eliminating τ-cycles and by keeping track, per block, of so-called bottom states. The complexity of the Groote & Vaandrager algorithm is $O(m \log m + m \cdot n)$, with n the number of states and m the number of transitions. Typically, for an LTS $m \ll n^2$. It is known that branching bisimulation preserves the fragment of the modal μ-calculus consisting of CTL* minus the next operator [9]. Therefore, exploiting this fact in practical situations, significant reduction of the state space and corresponding speed-up of subsequent verification can be obtained by applying hiding of action followed by the minimization algorithm for branching bisimulation.

In the sequel of this paper, we seek to apply the idea of branching bisimulation (i.e. allowing silent moves through bisimulation equivalent states but through no other) and its minimization techniques to the setting of FTS, where not only actions but also feature expressions decorate the transitions.

3 Branching bisimulation for feature transition systems

We fix a finite non-empty set $\mathcal{F}$ of features, a subset $\mathcal{P} \subseteq \mathbf{2}^{\mathcal{F}}$ of products, and again a set $\mathcal{A}_\tau$ including the silent action τ. We let $\mathbb{B}(\mathcal{F})$ denote the set of boolean expressions over $\mathcal{F}$. We refer to elements of $\mathbb{B}(\mathcal{F})$ as feature expressions. For a product $P \in \mathcal{P}$, we use $\chi(P)$ to denote its characteristic formula. The notion of a feature transition system (FTS) was proposed in [6].

Definition 3. *A feature transition system (FTS) $\mathcal{S}$ is a triple $\mathcal{S} = (S, \theta, s_*)$, with S the set of states, $\theta : S \times \mathcal{A}_\tau \times S \to \mathbb{B}(\mathcal{F})$ the transition constraint function, and $s_* \in S$ the initial state.*

For states $s, s' \in S$, an action $\alpha \in \mathcal{A}_\tau$ and a satisfiable feature expression $\psi \in \mathbb{B}(\mathcal{F})$, we write $s \xrightarrow{\alpha|\psi} s'$ if $\theta(s, \alpha, s') = \psi$. We say that a product $P \in \mathcal{P}$ satisfies a feature expression $\varphi \in \mathbb{B}(\mathcal{F})$ if φ is valid when the boolean variables corresponding to the features of P are assigned the value *true* and those not in P the value *false*, denoted by $P \models \varphi$. The equivalence relation $\sim_{\mathcal{P}}$ on $\mathbb{B}(\mathcal{F})$ is given by $\varphi \sim_{\mathcal{P}} \psi$ iff $\forall P \in \mathcal{P}$: $P \models \varphi \Leftrightarrow P \models \psi$. We let $\widehat{\mathbb{B}}(\mathcal{F}) = \mathbb{B}(\mathcal{F})/\sim_{\mathcal{P}}$. For an FTS $\mathcal{S} = (S, \theta, s_*)$, we define the reachability function $\rho : S \to \mathbb{B}(\mathcal{F})$ for $\mathcal{S}$ to be such that

$$\forall P \in \mathcal{P}\colon P \models \rho(s) \text{ iff } \exists n \exists s_0 \cdots s_n \exists \alpha_1 \cdots \alpha_n \exists \psi_1 \cdots \psi_n : \\ s_0 = s_* \wedge (\forall i, 1 \leqslant i \leqslant n : s_{i-1} \xrightarrow{\alpha_i|\psi_i} s_i \wedge P \models \psi_i) \wedge s_n = s$$

for all $s \in S$. We note that, for the ease of presentation in this paper, the definition of an FTS above is slightly more abstract compared to the original definition given in [6].

Next, we introduce a notion of branching feature bisimulation for FTS, generalizing the notion of branching bisimulation given by Definition 1 for LTS.

Definition 4. *Let $\mathcal{S} = (S, \theta, s_*)$ and $\mathcal{S}' = (S', \theta', s'_*)$ be two FTS.*

(a) For $s, s' \in S$, and satisfiable $\eta \in \mathbb{B}(\mathcal{F})$, we write $s \stackrel{\eta}{\Rightarrow} s'$ if $\exists n \exists s_0, \ldots, s_n \exists \eta_1, \ldots, \eta_n$: $s = s_0 \wedge \forall i, 1 \leqslant i \leqslant n$: $s_{i-1} \xrightarrow{\tau|\eta_i} s_i \wedge s' = s_n \wedge \eta = \bigwedge_{1 \leqslant i \leqslant n} \eta_i$. Furthermore, we write $s \xrightarrow{(\alpha|\psi)} s'$ in case $s \xrightarrow{\alpha|\psi} s'$ or $\alpha = \tau \wedge s = s' \wedge \psi = true$.

(b) A symmetric relation $R \subseteq S \times \widehat{\mathbb{B}}(\mathcal{F}) \times S$ is called a branching feature bisimulation relation for $\mathcal{S}$ if for $s,t \in S$, $\alpha \in \mathcal{A}_\tau$ such that $R(s,\hat{\varphi},t)$ the so-called transfer condition holds:

$$s \xrightarrow{\alpha|\psi} s' \quad \textit{implies} \quad \exists n\, \exists \hat{t}_1,\ldots,\hat{t}_n\, \exists t'_1,\ldots,t'_n\, \exists \eta_1,\ldots,\eta_n\, \exists \psi_1,\ldots,\psi_n\, \exists \varphi_1,\ldots,\varphi_n\, \exists \varphi'_1,\ldots,\varphi'_n\colon$$
$$\forall i, 1 \leqslant i \leqslant n\colon t \stackrel{\eta_i}{\Rightarrow} \hat{t}_i \xrightarrow{(\alpha|\psi_i)} t'_i \wedge R(s,\hat{\varphi}_i,\hat{t}_i) \wedge R(s',\hat{\varphi}'_i,t'_i) \;\; \textit{and}$$
$$\forall P \in \mathcal{P}\colon\; P \models \varphi \wedge \psi \;\Rightarrow\; P \models \textstyle\bigvee_{1\leqslant i\leqslant n} \eta_i \wedge \psi_i \wedge \varphi_i \wedge \varphi'_i$$

(c) Two states $s,t \in S$ are called branching feature bisimilar with respect to $\mathcal{S}$ if $R(s,\widehat{true},t)$ for some branching feature bisimulation R for $\mathcal{S}$. Notation $s \simeq_{bf} t$.

(d) A branching feature bisimulation relation R for $\mathcal{S}$ and $\mathcal{S}'$ is called coherent if $R(s,\hat{\varphi},s')$ implies $\rho(s) \Rightarrow \varphi$, for all $s \in S$, $\varphi \in \mathbb{B}(\mathcal{F})$, and $s' \in S'$. Notation $\mathcal{S} \simeq_{cbf} \mathcal{S}'$.

The specific subset of coherent branching feature bisimulations will be used as a yardstick of comparison in the minimization algorithm discussed in Section 4. Intuitively, the feature expression $\rho(s)$ captures all products that can reach state s. Coherency requires that φ does not exclude part of these products. So the 'products of s' are not split by φ, but treated as a coherent set of products.

Figure 2 depicts the general situation for the transfer condition where a transition $s \xrightarrow{\alpha|\psi} s'$ is matched by n transition sequences from t in total, viz. $t \stackrel{\eta_1}{\Rightarrow} \hat{t}_1 \xrightarrow{(\alpha|\psi_1)} t'_1$ to $t \stackrel{\eta_n}{\Rightarrow} \hat{t}_n \xrightarrow{(\alpha|\psi_n)} t'_n$. Moreover, for a product P for which state s admits the transition labelled α, i.e. a product satisfying the constraint φ derived from R as well as the feature expression ψ derived from the transition, it is required that state t provides a related transition sequence labeled α for this product as well. Thus, for some i, $1 \leqslant i \leqslant n$, P meets η_i and ψ_i, thus can move from t to $\hat{t}_i$ and t'_i, while P is included by the constraint φ_i for the relation on s and $\hat{t}_i$ and by the constraint φ'_i on s' and t'_i.

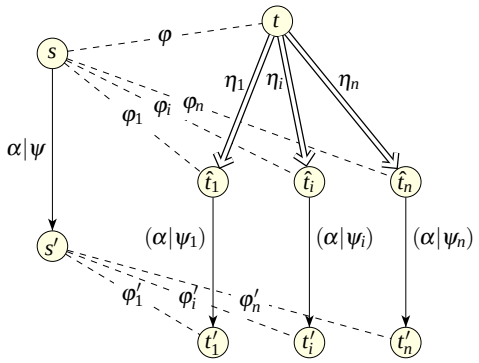

Figure 2: Transfer diagram for branching feature bisimilarity

Figure 3 below shows an example of two FTS (without τ-moves) at the left-hand side. At first sight the relation $R = \{\, (s_0,\widehat{true},t_0),\ (s_1,\hat{\varphi}_1,t_1),\ (s_2,\hat{\varphi}_2,t_1),\ (s_3,\widehat{true},t_2) \,\}$ may look like a branching feature bisimulation. However, a closer inspection of the transition $t_1 \xrightarrow{b|(\varphi_1\wedge\psi_1)\vee(\varphi_2\wedge\psi_2)} t_2$ reveals that this means that we need the formulas $\varphi_i \wedge ((\varphi_1 \wedge \psi_1) \vee (\varphi_2 \wedge \psi_2)) \Rightarrow \psi_i \wedge \mathit{true}$ to hold for $i = 1,2$. However, this only holds when $\varphi_1 \wedge \varphi_2 \Rightarrow (\psi_1 \Leftrightarrow \psi_2)$; in that case R is indeed a branching feature bisimulation. Reversely, if a product meets $\varphi_1 \wedge \varphi_2 \wedge \psi_1 \wedge \neg\,\psi_2$, there will be a transition for t_1 for that product, but not for s_2 as shown by the two LTS at the right-hand side of Figure 3. It is clear that with a transition from state s_0 to state s_2 but without a transition between states s_2 and s_3, on the one hand, and with a path from

t_0 to t_2, on the other hand, the underlying LTS for the two FTS (and therefore the FTS themselves as we shall see) cannot be bisimilar.

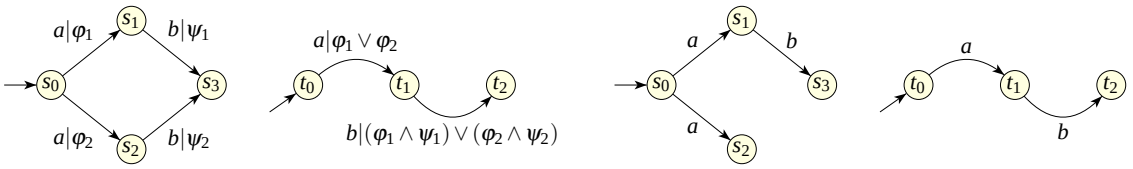

Figure 3: Bisimilar FTS assuming $\varphi_1 \wedge \varphi_2 \Rightarrow (\psi_1 \Leftrightarrow \psi_2)$ and non-bisimilar LTS

For branching feature bisimulation we have a strict correspondence with branching bisimulation for all products using the notion of a projection of an FTS. The projection results in an LTS.

Definition 5. *Given an FTS $\mathcal{S} = (S, \theta, s_*)$ and a product $P \in \mathcal{P}$, the projection $\mathcal{S}_P$ of $\mathcal{S}$ for the product P is the LTS $\mathcal{S}_P = (S, \rightarrow_P, s_*)$, where $s \xrightarrow{\alpha}_P s'$ if some $\psi \in \mathbb{B}(\mathcal{F})$ exists such that $s \xrightarrow{\alpha|\psi} s'$ and $P \models \psi$, for $s, s' \in S$ and $\alpha \in \mathcal{A}_\tau$.*

We use $s \simeq_P t$ to denote that s and t are branching bisimilar states for the projected LTS $\mathcal{S}_P$.

Theorem 6. *Let $\mathcal{S}$ be an FTS with states s and t. It holds that $s \simeq_{bf} t$ iff $s \simeq_P t$ for all $P \in \mathcal{P}$.*

Proof. Suppose $R \subseteq S \times \widehat{\mathbb{B}}(\mathcal{F}) \times S$ is a branching feature bisimulation relation with $R(s, \widehat{true}, t)$. Pick $P \in \mathcal{P}$. Define $R_P = \{ (s', t') \mid \exists \varphi \colon R(s', \hat{\varphi}, t') \wedge P \models \varphi \}$. We claim that R_P is a branching bisimulation relation with $R_P(s,t)$. Clearly R_P is symmetric and $R_P(s,t)$, since $R(s, \widehat{true}, t)$ and $P \models true$. In order to verify the transfer condition for R_P, suppose $R_P(s', t')$ and $s' \xrightarrow{\alpha}_P s''$. Pick, with appeal to the definitions of R_P and $\mathcal{S}_P$, feature expressions φ, ψ such that (i) $R(s', \hat{\varphi}, t')$ and $P \models \varphi$, and (ii) $s' \xrightarrow{\alpha|\psi} s''$ and $P \models \psi$. Since R is a branching feature bisimulation, we can find $\hat{t}_i$, t_i, η_i, ψ_i, φ_i and φ_i', for $i = 1, \ldots, n$, such that

$$t' \overset{\eta_i}{\Rightarrow} \hat{t}_i \xrightarrow{(\alpha|\psi_i)} t_i' \wedge R(s', \hat{\varphi}_i, \hat{t}_i), R(s'', \hat{\varphi}_i', t_i') \quad \text{and} \quad P \models \bigvee_{1 \leqslant i \leqslant n} \eta_i \wedge \psi_i \wedge \varphi_i \wedge \varphi_i'$$

for $i = 1, \ldots, n$. Choose i such that $P \models \eta_i \wedge \psi_i \wedge \varphi_i \wedge \varphi_i'$. Since $t' \overset{\eta_i}{\Rightarrow} \hat{t}_i \xrightarrow{(\alpha|\psi_i)} t_i'$, $P \models \eta_i \wedge \psi_i \wedge \varphi_i$, and $R(s'', \hat{\varphi}_i', t_i')$, we have by definition of $\mathcal{S}_P$ and R_P that $t' \Rightarrow \hat{t} \xrightarrow{(\alpha)}_P t_i''$ and $R_P(s'', t_i')$. Thus, R_P satisfies the transfer condition, as was to be shown.

To prove the reverse implication, pick for each $P \in \mathcal{P}$, a branching bisimulation relation R_P such that $R_P(s,t)$. Define $R \subseteq S \times \mathbb{B}(\mathcal{F}) \times S$ by $R = \{ (s', \hat{\varphi}, t') \mid \forall P \in \mathcal{P} \colon P \models \varphi \Leftrightarrow R_P(s', t') \}$. We verify that R is a branching feature bisimulation. Clearly, $R(s, \widehat{true}, t)$. In order to check the transfer condition for R, suppose $R(s', \hat{\varphi}, t')$ and $s' \xrightarrow{\alpha|\psi} s''$. Then it holds, for all $P \in \mathcal{P}$ with $P \models \varphi$, that $R_P(s', t')$. Moreover, for all $P \in \mathcal{P}$ with $P \models \psi$, we have $s' \xrightarrow{\alpha}_P s''$. Thus, for all $P \in \mathcal{P}$ with $P \models \varphi \wedge \psi$, we can pick $\hat{t}_P, t_P'$ and η_P, ψ_P such that $P \models \eta_P \wedge \psi_P$, $t' \overset{\eta_P}{\Rightarrow} \hat{t}_P \xrightarrow{(\alpha|\psi_P)} t_P'$ and $R_P(s', \hat{t}_P)$ and $R_P(s'', t_P')$.

Suppose $\{ P \in \mathcal{P} \mid P \models \varphi \wedge \psi \} = \{ P_1, \ldots, P_k \}$. Also, for $i = 1, \ldots, k$, let $\hat{t}_i, t_i'$ and η_i, ψ_i be shorthand for $\hat{t}_{P_i}, t_{P_i}'$ and η_{P_i}, ψ_{P_i}, respectively. Since $P_i \models \chi(P_i)$, $R_{P_i}(s', \hat{t}_i)$ and $R_{P_i}(s'', t_i')$, it holds that $R(s', \hat{\varphi}_i, \hat{t}_i)$ and $R(s'', \hat{\varphi}_i', t_i')$ for $\varphi_i, \varphi_i' \in \mathbb{B}(\mathcal{F})$ such that $\widehat{\chi(P_i)} \Rightarrow \varphi_i$ and $\widehat{\chi(P_i)} \Rightarrow \varphi_i'$. We conclude that, for $i = 1, \ldots, k$, it holds that $t' \overset{\eta_i}{\Rightarrow} \hat{t}_i \xrightarrow{(\alpha|\psi_i)} t_i''$, $R(s', \hat{\varphi}_i, \hat{t}_i)$ and $R(s'', \hat{\varphi}_i', t_i')$ while $P \models \varphi \wedge \psi \Rightarrow P \models \bigvee_{1 \leqslant i \leqslant n} \eta_i \wedge \psi_i \wedge \varphi \wedge \varphi'$, which verifies the transfer condition for R. □

The theorem asserts the soundness of branching feature bisimulation for FTS with respect to branching bisimulation for the projected LTS for all products. In the sequel, we propose an algorithm for minimization of an FTS modulo branching feature bisimulation and compare, in a case study, verification of properties against the minimized FTS to verification of properties against the minimized product LTS.

4 Minimization modulo coherent branching feature bisimulation

When minimizing an FTS $\mathcal{S}$ we look for an FTS $\mathcal{S}'$ satisfying $\mathcal{S} \simeq_{bf} \mathcal{S}'$ and such that it is the smallest in 'size'. For branching bisimulation for LTS it is the case that a branching bisimilar LTS with the minimal number of states also has the minimal number of transitions (after removal of τ-loops). Algorithms for branching bisimulation reduction make use of this fact by looking for the unique LTS with the minimal number of states. Unfortunately, this is not true for branching feature bisimulation, as is demonstrated in Figure 4: The FTS $\mathcal{T}$ and $\mathcal{U}$ are both branching feature bisimilar to FTS $\mathcal{S}$, and both have the minimal number of states. However, $\mathcal{U}$ has twice as many transitions as $\mathcal{T}$.

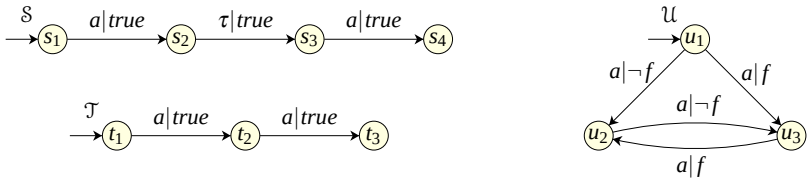

Figure 4: Three branching feature bisimilar FTS

We see that the property of feature bisimulation that allows to merge multiple transitions with the same label and different feature expressions into a single transition now hinders us, since it also allows to split transitions. To avoid this problem we restrict to *coherent* bisimulations (cf. Definition 4d). Thus, we require that states of $\mathcal{S}$ can only be related to states of the reduced $\mathcal{S}'$ for (supersets of) their reachability set. Unfortunately, this recipe does not guarantee that a minimal FTS is found, as Figure 5 below shows, but among all coherent branching feature bisimilar FTS our algorithm is able to find the smallest one, see Theorem 12.

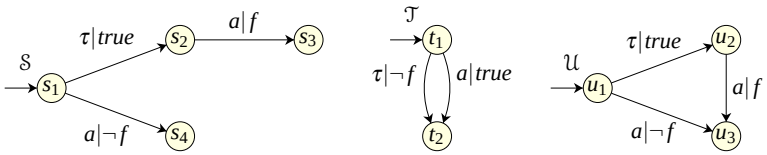

Figure 5: Minimal branching feature bisimilar vs. minimal coherent branching feature bisimilar

In Figure 5, FTS $\mathcal{T}$ is branching feature bisimilar to FTS $\mathcal{S}$, and has the minimal number of states and transitions. However, when restricting to coherent branching feature bisimulation relations, FTS $\mathcal{U}$ is the smallest FTS that can be obtained from $\mathcal{S}$ such that $\mathcal{S} \simeq_{cbf} \mathcal{U}$. Note that the relation R with $R(s_2, \hat{f}, t_1)$ and $R(s_2, \neg\hat{f}, t_2)$ is not coherent, since $\rho(s_2) = true$ does not imply f nor $\neg f$. We will adapt the reduction algorithm described in Section 2 for minimization modulo coherent branching feature bisimulation.

Before describing the algorithm, we first show that the problem of coherent branching feature bisimulation minimization is NP-hard by reducing the chromatic number problem to it: given a graph, what is the minimum number of colors to color the nodes such that adjacent nodes have different colors? To verify the construction, we need an auxiliary result.

Lemma 7. *Let $\mathcal{S} = (S, \theta, s_0)$ be an FTS with states s and t. If $R(s, \rho(s) \wedge \rho(t), t)$ for a branching feature bisimulation relation R, then $\mathcal{S} \simeq_{cbf} \mathcal{S}'$ with states s and t related to a single state of $\mathcal{S}'$.*

Proof. Let $\mathcal{S}' = (S', \theta', s'_*)$ with $S' = (S \setminus \{s,t\}) \cup \{r\}$ for some $r \notin S$, with $\theta'(u,a,v) = \theta(u,a,v)$ for $u,v \in S'$, $u,v \neq r$ and $\theta'(u,a,r) = \theta(u,a,s) \vee \theta(u,a,t)$ for $u \neq r$, $\theta'(r,a,v) = \theta(s,a,v) \vee \theta(t,a,v)$, for $v \neq r$, and $\theta'(r,a,r) = \bigvee_{q,w \in \{s,t\}} \theta(q,a,w)$, and finally with $s'_* = s_*$ if $s_* \neq s,t$, and $s'_* = r$ otherwise. Using that R is a branching feature bisimulation with $R(s, \rho(s) \wedge \rho(t), t)$, one constructs a coherent branching feature bisimulation R' such that $R'(s, \rho(s), r)$ and $R'(t, \rho(t), r)$. □

Next we set the stage for a reduction of graph coloring to coherent branching feature bisimulation minimization. Consider an undirected graph $\mathcal{G} = (V,E)$ with nodes in V and edges in E. Let $\mathcal{A} = \{a\}$, $\mathcal{F} = \{ f_v \mid v \in V \}$ and $\mathcal{P} = \{ P_v \mid v \in V \}$. The FTS $\mathcal{S}_G = (S_G, \theta_G, s_1)$ of $\mathcal{G}$ is such that $S_G = \{s_1, s_2\} \cup \{ s_v \mid v \in V \}$ for distinct states s_1 and s_2, $\theta(s_1, a, v) = \bigvee_{u \in V} \{ f_u \mid (u,v) \in \mathcal{G} \} \vee f_v$ for all $v \in V$, and $\theta(v, a, s_2) = f_v$, and finally such that $\theta(s,a,s') = \mathit{false}$ in all other cases.

Theorem 8. *Let $\mathcal{S}'_G$ be the minimal FTS that is coherent branching feature bisimilar to the FTS $\mathcal{S}_G$ given above. Then the number of states in $\mathcal{S}'_G$ is equal to the chromatic number of $\mathcal{G}$ plus 2.*

Proof. Let Γ be a set of colors. Suppose $\gamma : V \to \Gamma$ is a coloring of $\mathcal{G}$ using all colors. Then the FTS $(\{s_1, s_2\} \cup \Gamma, \theta_\gamma, s_1)$, where $\theta_\gamma(s_1, a, C) = \bigvee_{\gamma(u)=C} \theta(s_1, a, s_u)$, $\theta_\gamma(C, a, s_2) = \bigvee_{\gamma(u)=C} f_u$ is coherent branching feature bisimilar to $\mathcal{S}_G$ via the relation R such that $R(s_i, \widehat{true}, s_i)$ for $i = 1, 2$, and $R(s_u, \rho(s_u), \gamma(u))$.

Reversely, an FTS $\mathcal{S}'$ that is coherent branching feature bisimilar to $\mathcal{S}_G$ can only identify states s_u, s_v for $u,v \in V$. Hence such an FTS induces a coloring for $\mathcal{G}$: Pick for each state s_v a single $s' \in S'$ such that $R(s_v, \varphi, s')$ for a coherent branching feature bisimulation R relating $\mathcal{S}$ and $\mathcal{S}'$. If states s_u and s_v correspond to the same state of $\mathcal{S}'$, there can be no edge between u and v in $\mathcal{G}$. For if (u,v) is an edge in $\mathcal{G}$, we have $s_1 \xrightarrow{a} u \xrightarrow{a} s_2$ and $s_1 \xrightarrow{a} v \not\to$ in the projection of $\mathcal{S}_G$ for the product p_u, but $s_1 \xrightarrow{a} u \not\to$ and $s_1 \xrightarrow{a} v \xrightarrow{a} s_2$ in the projection of $\mathcal{S}_G$ for the product p_v.

It follows that the FTS $\mathcal{S}'_G$ that is minimal coherent branching feature bisimilar to $\mathcal{S}_G$ corresponds to a minimal coloring of $\mathcal{G}$. Moreover, the number of states different from the images of s_1 and s_2 corresponds to the number of colors needed. □

Note how, in the proof above, the coherence condition 'if $R(s, \varphi, s')$ then $\rho(s) \Rightarrow \varphi$' enforces that for the minimal FTS $\mathcal{S}'_G$ the products that can reach s in $\mathcal{S}_G$ are not split over multiple states in $\mathcal{S}'_G$. From the theorem we obtain the following result.

Corollary 9. *Constructing a minimal coherent branching feature bisimilar FTS is NP-complete.* □

Before we provide an algorithm for minimization of an FTS modulo coherent branching feature bisimulation, we slightly generalize the notion of a partition as used in Section 2, to allow a state to belong to separate groups of products.

A collection $\mathcal{B} = \{ B_i \mid i \in I \}$ of non-empty subsets of a set S is called a *semi-partition* of S if (i) $\bigcup_{i \in I} B_i = S$, and (ii) for $j \neq i : B_j \setminus B_i \neq \varnothing$. Thus, $\mathcal{B}$ covers S and no B_j is strictly contained in a B_i. Also, for a semi-partition its elements are referred to as *blocks*. We say that a semi-partition $\mathcal{B}'$ is a refinement of a semi-partition $\mathcal{B}$ if every block of $\mathcal{B}'$ is a subset of a block of $\mathcal{B}$. Likewise, we say that $\mathcal{B}$ is coarser than $\mathcal{B}'$. A semi-partition $\mathcal{B}$ of S induces a relation $\sim_{\mathcal{B}}$ on S (not necessarily an equivalence relation), where two elements of S are related iff they are included in the same block of $\mathcal{B}$.

Given an FTS $\mathcal{S} = (S, \theta, s_*)$, we first do some preprocessing. We eliminate unreachable states and strengthen the transition constraint with the reachability condition for its source state:

$$S := \{\, s \in S \mid \rho(s) \not\sim_{\mathcal{P}} \mathit{false} \,\} \quad \text{and} \quad \theta(s,\alpha,s') := \theta(s,\alpha,s') \wedge \rho(s)$$

We define the set $\mathcal{A}_f$ of so-called featured labels by $\mathcal{A}_f = \{\, (\alpha,\psi) \mid \exists s,t\, \exists \alpha\colon\ \theta(s,\alpha,t) = \psi \wedge \psi \not\sim_{\mathcal{P}} \mathit{false} \,\}$. For a semi-partition $\mathcal{B}$ of S, $B, B' \in \mathcal{B}$ and featured label $(\alpha,\psi) \in \mathcal{A}_f$ we let

$$\begin{aligned} \mathit{non\text{-}neg}_{(\alpha,\psi)}(B,B') \;=\; \{\, s \in B \mid \forall P \in \mathcal{P}, P \models \rho(s) \wedge \psi\colon\ \exists n\, \exists s_0,\ldots,s_n \in B\, \exists s' \in B'\, \exists \psi_1,\ldots,\psi_n,\psi'\colon \\ s_0 = s \wedge (\forall i, 1 \leqslant i \leqslant n\colon\ s_{i-1} \xrightarrow{\tau|\psi_i} s_i \wedge P \models \psi_i) \wedge s_n \xrightarrow{(\alpha|\psi')} s' \wedge P \models \psi' \,\}, \end{aligned}$$

and define its subset $\mathit{pos}_{(\alpha,\psi)}(B,B')$ to include all $s \in \mathit{non\text{-}neg}_{(\alpha,\psi)}(B,B')$ for which $\psi \Rightarrow \rho(s)$ and $s_n \xrightarrow{\alpha|\psi'} s'$ for $s_n \in B$, $s' \in B'$ as above. Moreover, we define $\mathit{neg}_{(\alpha,\psi)}(B,B') = B \setminus \mathit{non\text{-}neg}_{(\alpha,\psi)}(B,B')$. We know for sure that two states s and t of a block B are behaviorally different, if $s \in \mathit{pos}_{(\alpha,\psi)}(B,B')$ and $t \in \mathit{neg}_{(\alpha,\psi)}(B,B')$. Therefore, we say that B' is a *splitter* of B with respect to (α,ψ) if $B \neq B'$ or $\alpha \neq \tau$, and $\mathit{pos}_{(\alpha,\psi)}(B,B'), \mathit{neg}_{(\alpha,\psi)}(B,B') \neq \varnothing$ (meaning there is at least one state in the *pos*-set that must do an actual τ-step to reach B'). If $\mathcal{B}$ is a semi-partition of S and B' is a splitter of B with respect to (α,ψ), then the semi-partition $\mathcal{B}'$ is obtained from $\mathcal{B}$ by replacing block B by $B_1 = \mathit{non\text{-}neg}_{(\alpha,\psi)}(B,B')$ and $B_2 = B \setminus \mathit{pos}_{(\alpha,\psi)}(B,B')$. However, in the case that B_1 or B_2 is a subset of another block in the partition (apart from B), it is not added to ensure that $\mathcal{B}'$ is a semi-partition.

The minimization algorithm starts from the trivial semi-partition $\{S\}$, and keeps refining the semi-partition until no splitters are left. This results in the coarsest semi-partition, but still a block may be covered completely by other blocks. Therefore, as post-processing, we remove as many blocks as possible from the semi-partition, while preserving the semi-partition properties, to find the smallest semi-partition (e.g. using an algorithm for the minimum set cover problem).

$\mathcal{B} := \{S\}$;
while a splitter B' for a block B with respect to a featured label (α,ψ) exists **do**
  $\mathcal{B} := \mathcal{B} \setminus \{B\}$;
  if $\mathit{non\text{-}neg}_{(\alpha,\psi)}(B,B') \subseteq B''$ for no $B'' \in \mathcal{B}$ **then** $\mathcal{B} := \mathcal{B} \cup \{\mathit{non\text{-}neg}_{(\alpha,\psi)}(B,B')\}$ **end**;
  if $B \setminus \mathit{pos}_{(\alpha,\psi)}(B,B') \subseteq B''$ for no $B'' \in \mathcal{B}$ **then** $\mathcal{B} := \mathcal{B} \cup \{B \setminus \mathit{pos}_{(\alpha,\psi)}(B,B')\}$ **end**
$\mathcal{B}_{\mathit{min}} :=$ smallest subset of $\mathcal{B}$ covering S;

It is easy to see that the algorithm terminates: Note that after each iteration at least two states have been permanently split from each other. Since there are less than $|S|^2$ possible pairs of states in S, termination will occur in at most $|S|^2$ iterations. In the theorem below, we call a semi-partition $\mathcal{C}$ a *stable* partition with respect to a block B' if for no block B and for no featured label (α,ψ), B' is a splitter of B with respect to (α,ψ). The semi-partition $\mathcal{C}$ is itself called stable if $\mathcal{C}$ is stable with respect to all its blocks.

Lemma 10. *For an FTS $\mathcal{S} = (S, \theta, s_*)$, $\mathcal{B}_{\mathit{min}}$ obtained from the algorithm is the smallest stable semi-partition refining $\{S\}$.*

Proof. We show by induction on the number of iterations of the algorithm that each stable partition refines the current semi-partition $\mathcal{B}$. Let $\mathcal{C}$ be a stable semi-partition. Clearly the statement holds initially, each semi-partition refines $\{S\}$. Suppose $\mathcal{C}$ refines semi-partition $\mathcal{B}$ obtained after a number of iterations and suppose a splitter B' of a block B exists with respect to a featured label (α,ψ). It suffices to show

that any block C of $\mathcal{C}$ is included in a block of $\mathcal{B}'$, the semi-partition obtained by splitting B. Pick a block of $\mathcal{B}$ containing C. If this block is different from B, we are done. So, suppose $C \subseteq B$. We have to show that either $C \subseteq \textit{non-neg}_{(\alpha,\psi)}(B,B')$ or $C \subseteq B \setminus \textit{pos}_{(\alpha,\psi)}(B,B')$.

Suppose $s,t \in C$ with $s \in \textit{pos}_{(\alpha,\psi)}(B,B')$ and $t \in \textit{neg}_{(\alpha,\psi)}(B,B')$. We derive a contradiction. Pick a product $P \in \mathcal{P}$ such that $P \models \psi$. Such a product exists by definition of $\mathcal{A}_f$. Choose $s_0,\ldots,s_n \in B$, $s' \in B'$, $\psi_1,\ldots,\psi_n,\psi' \in \mathbb{B}(\mathcal{F})$ such that $s_0 = s$, $s_{i-1} \xrightarrow{\tau|\psi_i} s_i$ for $1 \leqslant i \leqslant n$, $s_n \xrightarrow{(\alpha|\psi')} s'$, and moreover $P \models \psi_i$, for $1 \leqslant i \leqslant n$, and $P \models \psi'$. Let $C_0,\ldots,C_n,C'$ be the blocks of $\mathcal{C}$ such that $s_i \in C_i$ and $s' \in C'$. Note that $C_i \subseteq B$, for $0 \leqslant i \leqslant n$, and $C' \subseteq B'$. Using the fact that $\mathcal{C}$ is stable we can construct a sequence $t_0,\ldots,t_m \in B$, $t' \in B'$, $\varphi_1,\ldots,\varphi_m,\varphi' \in \mathbb{B}(\mathcal{F})$ such that $t_0 = t$, $t_{i-1} \xrightarrow{\tau|\varphi_i} t_i$ for $1 \leqslant i \leqslant m$, $t_n \xrightarrow{(\alpha|\varphi')} t'$, and moreover $P \models \varphi_i$ for $1 \leqslant i \leqslant m$, and $P \models \varphi'$. This contradicts $t \in \textit{neg}_{(\alpha,\psi)}(B,B')$, and proves the induction step. Finally, we observe that $\mathcal{B}_{min}$ itself is a stable semi-partition that refines $\{S\}$. □

Lemma 11. *Let $\mathcal{S} = (S,\theta,s_*)$ be an FTS, and $\mathcal{S}' = (S',\theta',s'_*)$ be an FTS such that $\mathcal{S} \simeq_{cbf} \mathcal{S}'$ by a relation R. Then R defines a stable semi-partition $\mathcal{C}$ of S such that $s \sim_{\mathcal{C}} t$ iff $\exists r \in S' \colon R(s,\rho(s),r) \wedge R(t,\rho(t),r)$.*

Proof. We have to show that $\mathcal{C}$ is stable indeed. Suppose that there are blocks B,B' in $\mathcal{C}$ such that B' is a splitter of B with respect to a featured label (α,ψ). This means there are states s and t in B such that $s \in \textit{pos}_{(\alpha,\psi)}(B,B')$ and $t \in \textit{neg}_{(\alpha,\psi)}(B,B')$. We pick $P \in \mathcal{P}$ such that $P \models \rho(s) \wedge \rho(t) \wedge \psi$. By definition of the *pos*-set there exist $s_0,\ldots,s_n \in B$, $s' \in B'$, $\psi_1,\ldots,\psi_n,\psi' \in \mathbb{B}(\mathcal{F})$ such that $s_0 = s$, $s_{i-1} \xrightarrow{\tau|\psi_i} s_i$ for $1 \leqslant i \leqslant n$, $s_n \xrightarrow{(\alpha|\psi')} s'$, and moreover $P \models \psi_i$, for $1 \leqslant i \leqslant n$, and $P \models \psi'$. Since $s_n \in B$ we have, by construction of $\mathcal{C}$, both $R(s_n,\rho(s_n),r)$ and $R(t,\rho(t),r)$ for suitable $r \in \mathcal{S}'$. Therefore, there exists a feature bisimulation relation R' on $\mathcal{S}$ such that $R'(s_n,\rho(s_n) \wedge \rho(t),t)$. Using the transfer condition of this relation we can construct a sequence $t_0,\ldots,t_m \in B$, $t' \in B'$, $\varphi_1,\ldots,\varphi_m,\varphi' \in \mathbb{B}(\mathcal{F})$ such that $t_0 = t$, $t_{i-1} \xrightarrow{\tau|\varphi_i} t_i$ for $1 \leqslant i \leqslant m$, $t_n \xrightarrow{(\alpha|\varphi')} t'$, and moreover $P \models \varphi_i$ for $1 \leqslant i \leqslant m$, and $P \models \varphi'$. This contradicts $t \in \textit{neg}_{(\alpha,\psi)}(B,B')$, and proves that $\mathcal{C}$ is stable. □

We are now in a position to prove the correctness of the minimization algorithm.

Theorem 12. *Assume that $\mathcal{B}$ is the partition obtained upon termination after applying the algorithm to the FTS $\mathcal{S} = (S,\theta,s_*)$. Define the FTS $\mathcal{S}_{min} = (\mathcal{B},\theta_{min},B_*)$ by letting (i) $\theta_{min}(B,\alpha,B') = \bigvee\{\, \theta(s,a,s') \mid s \in B, s' \in B' \,\}$ with $B \neq B'$ or $\alpha \neq \tau$, and (ii) by choosing B_* such that $s_* \in B_*$. Then $\mathcal{S}_{min}$ is the smallest FTS that is coherent branching feature bisimilar to $\mathcal{S}$.*

Proof. By Lemma 10 we have that $\mathcal{B}_{min}$ is the smallest stable semi-partition refining $\{S\}$. It suffices to show, using Lemma 7, that a coherent branching feature bisimulation for $\mathcal{S}$ and $\mathcal{S}_{min}$ exists. Since, by Lemma 11 we have that every coherent branching feature bisimulation relation from $\mathcal{S}$ to an FTS $\mathcal{S}'$ induces a stable semi-partition on $\{S\}$, implying that $\mathcal{S}_{min}$ is indeed minimal. □

Thus, given an FTS $\mathcal{S}$, we continue to refine the trivial semi-partition until no more splitter can be found. Splitting a block is done cautiously: (i) it must eliminate a splitter and (ii) it must yield a semi-partition again. The final semi-partition that is reached induces an FTS $\mathcal{S}_{min}$ that is the smallest FTS that is coherent branching feature bisimilar to $\mathcal{S}$. The next section reports on a small case study using this approach.

5 Experimental evaluation

We extended the example SPL of a coffee vending machine described in [1–4] with a soup component running in parallel. The complete SPL consists of 18 features and 118 products and the FTS modeling it contains 182 states and 691 transitions. The details of this SPL can be found in Appendix A. Basically, each product contains the well-known beverage component and optionally a soup component, and allows the insertion of either euros or dollars (returned upon a cancel) in either of its components. The user chooses a beverage (sugared or not) among those offered (at least coffee, cappuccino only for euros) or else a type of soup (at least one among chicken, tomato, pea). The user must place a cup to get soup. A cup detector is optional (mandatory for dollars). When present, soup is only poured if a cup was placed, else soup may be spilled. Placing a cup may need to be repeated if not detected. A soup order may be canceled until a cup is detected. Optionally, a shared ringtone may ring after delivery (mandatory for cappuccino), after which the user takes a cup (with a drink or soup) and can again insert money in either component. Concrete features have an associated cost (zero for abstract features) and the total cost of a product, summing the costs of the features it includes, does not exceed the fixed upper bound of 35.

We used the mCRL2 toolset to verify the 12 properties listed in Appendix A against this SPL, both product-by-product and by using the FTS-based family approach described in [3, 4], and both with and without branching (feature) bisimulation minimization. For the approach with bisimulation we applied branching feature bisimulation to the FTS, resulting in a reduced FTS, which we projected to obtain the reduced LTS for each product. The results are shown in Table 1. For the product-by-product approaches, generating the projections for all products is included in the computation time, and so is the time for bisimulation reduction in case of the approaches with bisimulation. To even out effects caused by other processes running whilst performing the experiments, all computation times are averaged over 5 runs.

Regarding the product-by-product approach, performing bisimulation reduction for the product LTS reduces the computation time by about 8%. For property 2 (*The SPL is deadlock-free*), the computation time with bisimulation is significantly larger than for other properties. In this case abstraction does not reduce the LTS. A similar observation holds for properties 1 (*If a coffee is ordered, it is eventually poured*), 5a (*If a beverage is ordered, then eventually it is canceled or a cup is taken*) and 5b (*If soup is ordered, then eventually it is canceled, a cup is taken or the customer has bad luck*), which are false, but deemed true after applying bisimulation reduction. They state that something eventually happens, which is not true in reality since the two components are running in parallel, thus abstraction creates infinite loops that allow postponing that something indefinitely. Applying bisimulation reduction causes these loops to be abstracted from completely, making the properties true for the reduced system. However, standard tricks, like the explicit signaling of the end of a cycle, could be applied to alleviate this problem.

Now consider the FTS-based family approach. Without applying bisimulation reduction, the total computation time increases by almost 50% with respect to the product-by-product approach. Hence, for this SPL, FTS-based verification with mCRL2 is not beneficial compared to regular enumerative verification. However, if we apply bisimulation reduction, then the FTS-based computation times decrease by $>$70%. Only property 2 still needs more computation time than in the product-based approach (again because abstraction is not beneficial for the verification). Note that in case less actions are involved in a property, it is possible to abstract from larger parts of the FTS, implying faster verification. This effect was much less in the product-by-product approach. Hence, the more local a property, the more beneficial it is to perform FTS-based family verification in combination with branching feature bisimulation reduction using mCRL2. Obviously, this observation needs to be confirmed by experimenting with different SPL, but based on this example the techniques proposed in this paper look rather promising.

PROPERTIES	PRODUCT-BY-PRODUCT				FTS-BASED FAMILY APPROACH			
	WITHOUT BISIMULATION		WITH BISIMULATION		WITHOUT BISIMULATION		WITH BISIMULATION	
	TIME (s)	RESULT	TIME (s)	RESULT	TIME (s)	RESULT	TIME (s)	RESULT
1	42.04	FALSE	38.18	TRUE	52.96	FALSE	13.60	TRUE
2	41.78	TRUE	41.65	TRUE	53.86	TRUE	53.69	TRUE
3a	42.32	TRUE	37.76	TRUE	70.57	TRUE	7.70	TRUE
3b	42.01	TRUE	37.78	TRUE	59.96	TRUE	7.98	TRUE
4a	40.62	TRUE	38.00	TRUE	24.18	TRUE	8.65	TRUE
4b	40.20	TRUE	37.88	TRUE	20.78	TRUE	10.68	TRUE
5a	42.38	FALSE	38.51	TRUE	66.08	FALSE	18.59	TRUE
5b	42.34	FALSE	38.09	TRUE	69.95	FALSE	14.92	TRUE
6	43.63	TRUE	39.17	TRUE	105.35	TRUE	29.72	TRUE
7a	42.45	TRUE	38.19	TRUE	71.07	TRUE	13.84	TRUE
7b	42.35	TRUE	38.04	TRUE	79.05	TRUE	9.48	TRUE
8	42.82	TRUE	39.09	TRUE	80.69	TRUE	20.47	TRUE
TOT	504.94		462.34		754.50		209.32	

Table 1: Experimental evaluation results (time in seconds)

6 Concluding remarks

We have defined a novel notion of branching feature bisimilarity for FTS and an algorithm to minimize an FTS modulo coherent branching feature bisimulation. This complements and formalizes part of the feature-oriented modular verification approach of SPL with mCRL2 that we outlined in [3, 4]. An initial application of the minimization algorithm to a simplistic SPL promises significant verification speed-ups.

It remains to establish the subset of the modal μ-calculus that is preserved by (coherent) branching feature bisimulation, i.e. what properties are respected by our reduction technique. It is known that branching bisimulation preserves modal μ-formula without the next operator [9]. Theorem 6 may be used to lift the result to branching feature bisimulation, if the property $\mathcal{S} \models \varphi$ iff $\mathcal{S}_P \models \varphi$ is to hold. We leave this to future work. It would also be interesting to see whether the minimization algorithm's complexity can be reduced, possibly by lifting some optimizations from the Groote & Vaandrager algorithm for LTS to our FTS setting, or split multiple blocks based on a single splitter.

Finally, we plan to evaluate our modular verification approach on a more realistic SPL. By expanding the SPL of a coffee vending machine to examples growing in size, we may see if the exponential blow-up forecast by the NP-completeness result of Theorem 8 can be traced, in particular to observe at what point reduction time outweighs the gain of family-based verification. As noted by one of the reviewers, family-based verification approaches perform better on larger models (both in terms of states and variability), whereas reduction techniques are difficult to apply on real, industrial models. We hope that the idea, sketched in [3], to exploit the inherent modular structure of SPL to guide the abstraction, will prove fruitful in finding balance in this trade-off and help to come up with automated support to reduce a system given a property. For this it is useful to reconstruct the experiments reported in [7] and to compare the performance gain. Also a study of the relationship of the preorder proposed in [7] to the equivalences put forward here, is an interesting topic of research that may increase our understanding of the interplay between variability and internal behaviour.

Acknowledgements

Maurice ter Beek was supported by the EU FP7-ICT FET-Proactive project QUANTICOL (600708) and by the Italian MIUR project CINA (PRIN 2010LHT4KM).

References

[1] P. Asirelli, M.H. ter Beek, A. Fantechi & S. Gnesi (2011): *Formal description of variability in product families*. In E.S. de Almeida, T. Kishi, C. Schwanninger, I. John & K. Schmid, editors: *SPLC'11*, IEEE, pp. 130–139, doi:10.1109/SPLC.2011.34.

[2] P. Asirelli, M.H. ter Beek, A. Fantechi & S. Gnesi (2012): *A compositional framework to derive product line behavioural descriptions*. In T. Margaria & B. Steffen, editors: *ISoLA'12, LNCS* 7609, Springer, pp. 146–161, doi:10.1007/978-3-642-34026-0_12.

[3] M.H. ter Beek & E.P. de Vink (2014): *Towards modular verification of software product lines with mCRL2*. In T. Margaria & B. Steffen, editors: *FMSPLE track at ISoLA'14, LNCS* 8802, Springer, pp. 368–385, doi:10.1007/978-3-662-45234-9_26.

[4] M.H. ter Beek & E.P. de Vink (2014): *Using mCRL2 for the analysis of software product lines*. In S. Gnesi & N. Plat, editors: *FormaliSE workshop at ICSE'14*, IEEE, pp. 31–37, doi:10.1145/2593489.2593493.

[5] A. Classen, M. Cordy, P. Heymans, P.-Y. Schobbens, A. Legay & J.-F. Raskin (2013): *Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking*. *IEEE Trans. Software Eng.* 39, pp. 1069–1089, doi:10.1109/TSE.2012.86.

[6] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay & J.-F. Raskin (2010): *Model checking lots of systems: Efficient verification of temporal properties in software product lines*. In J. Kramer, J. Bishop, P.T. Devanbu & S. Uchitel, editors: *ICSE'10*, ACM, pp. 335–344, doi:10.1145/1806799.1806850.

[7] M. Cordy, A. Classen, G. Perrouin, P.-Y. Schobbens, P. Heymans & A. Legay (2012): *Simulation-based abstractions for software product-line model checking*. In M. Glinz, G.C. Murphy & M. Pezzè, editors: *ICSE'12*, IEEE, pp. 672–682, doi:10.1109/ICSE.2012.6227150.

[8] S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, W. Wesselink & T.A.C. Willemse (2013): *An overview of the mCRL2 toolset and its recent advances*. In N. Piterman & S.A. Smolka, editors: *TACAS'13, LNCS* 7795, Springer, pp. 199–213, doi:10.1007/978-3-642-36742-7_15.

[9] R. De Nicola & F.W. Vaandrager (1995): *Three logics for branching bisimulation*. *J. ACM* 42(2), pp. 458–487, doi:10.1145/201019.201032.

[10] D. Fischbein, S. Uchitel & V.A. Braberman (2006): *A foundation for behavioural conformance in software product line architectures*. In R.M. Hierons & H. Muccini, editors: *ROSATEA workshop at ISSTA'06*, ACM, pp. 39–48, doi:10.1145/1147249.1147254.

[11] R.J. van Glabbeek (1990): *The linear time – branching time spectrum (extended abstract)*. In J.C.M. Baeten & J.W. Klop, editors: *CONCUR'90, LNCS* 458, Springer, pp. 278–297, doi:10.1007/BFb0039066.

[12] R.J. van Glabbeek (1993): *The linear time – branching time spectrum II: The semantics of sequential systems with silent moves (extended abstract)*. In E. Best, editor: *CONCUR'93, LNCS* 715, Springer, pp. 66–81, doi:10.1007/3-540-57208-2_6.

[13] R.J. van Glabbeek & W.P. Weijland (1989): *Branching time and abstraction in bisimulation semantics (extended abstract)*. In G.X. Ritter, editor: *IFIP Congress'89*, North-Holland, pp. 613–618. Available at http://theory.stanford.edu/~rvg/abstracts.html#11.

[14] R.J. van Glabbeek & W.P. Weijland (1996): *Branching time and abstraction in bisimulation semantics*. *J. ACM* 43(3), pp. 555–600, doi:10.1145/233551.233556.

[15] J.F. Groote & F.W. Vaandrager (1990): *An efficient algorithm for branching bisimulation and stuttering equivalence*. In M. Paterson, editor: *ICALP'90, LNCS* 443, Springer, pp. 626–638, doi:10.1007/BFb0032063.

[16] K.G. Larsen & B. Thomsen (1988): *A modal process logic*. In: *LICS'88*, IEEE, pp. 203–210, doi:10.1109/LICS.1988.5119.

[17] R. Milner (1989): *Communication and Concurrency*. Prentice Hall.

[18] R. Paige & R.E. Tarjan (1987): *Three partition refinement algorithms*. *SIAM J. Comput.* 16(6), pp. 973–989, doi:10.1137/0216062.

[19] T. Thüm, S. Apel, C. Kästner, I. Schaefer & G. Saake (2014): *A classification and survey of analysis strategies for software product lines*. *ACM Comput. Surv.* 47(1):6, doi:10.1145/2580950.

[20] M. Tribastone (2014): *Behavioral relations in a process algebra for variants*. In S. Gnesi, A. Fantechi, P. Heymans, J. Rubin & K. Czarnecki, editors: *SPLC'14*, ACM, pp. 82–91, doi:10.1145/2648511.2648520.

A Example SPL

Here we provide the details of the example SPL used for the experiments described in Section 5. It is an extension of the coffee vending machine described in [1–4] with a soup component running in parallel with the usual beverage component. It has the following list of functional requirements:

- Each product contains a beverage component. Optionally, also a soup component is present.
- Initially, either a euro must be inserted, exclusively for European products, or a dollar must be inserted, exclusively for Canadian products. The money can be inserted in either of the components.
- Optionally, money inserted in a component can be retrieved via a cancel button, after which money can be inserted in this component anew.
- If money was inserted in the beverage component, the user has to choose whether (s)he wants sugar, by pressing one of two buttons, after which (s)he can select a beverage.
- The choice of beverage (coffee, tea, cappuccino) varies, but coffee must be offered by all products whereas cappuccino may be offered solely by European products.
- Optionally, a ringtone may be rung after delivering a beverage. However, a ringtone must be rung by all products offering cappuccino.
- After the beverage is taken, money can be inserted again in the beverage component.
- If money was inserted in the soup component, the user has to choose a type of soup (chicken, tomato, pea). The types of soup offered vary, but at least one type must be offered by all products with a soup component.
- The soup component does not contain cups to serve the soup in. Hence, the user has to place a cup to pour the soup in. Optionally, a cup detector may be present in the soup component. It is required that all Canadian products with a soup component are equipped with a cup detector.
- If cup detection is present, the chosen type of soup will only be delivered after a cup has been detected by the soup component. However, the cup detector may fail to detect an already placed cup, after which the user will have to place it again. If a cancel option is available, the user may cancel the order as long as no cup has been detected.
- If cup detection is not present, the soup will be delivered immediately after a type of soup was chosen, regardless of whether a cup was placed. If no cup was placed there will be no soup to take.
- Optionally, a ringtone (shared with the beverage component) may be rung after delivering soup.
- If a cup was present, money can be inserted again in the soup component after the soup is taken.

These yield the attributed feature model in Figure 6 and the behavioral models in Figures 7 and 8.

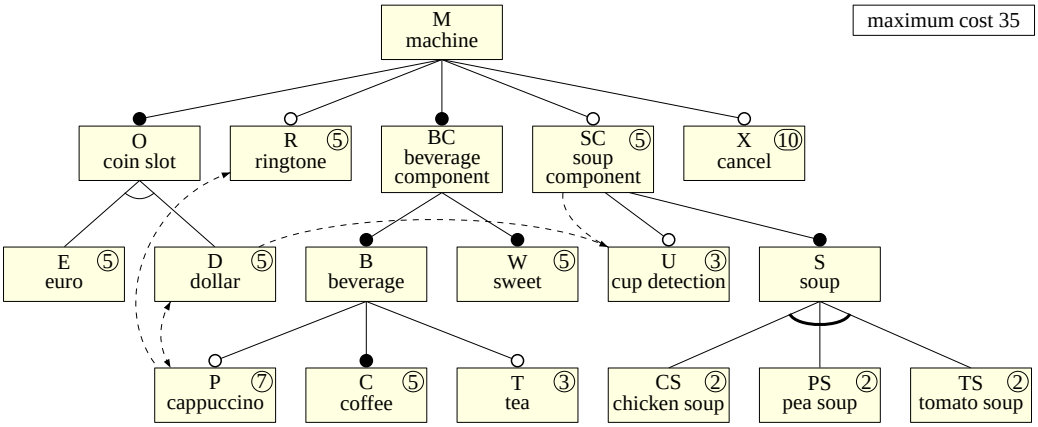

Figure 6: Feature model of family of coffee vending machines

In the attributed feature model, mandatory (core) features are marked by a closed bullet, optional features by an open one. Exactly one of the features *E* and *D* is selected, while at least one of the features *CS*, *PS* and *TS* is selected. As to cross-tree constraints, features *P* and *D* exclude each other, feature *P* requires feature *R*, and the simultaneous selection of features *D* and *SC* requires feature *U*. The value of the cost attribute of the concrete features is put inside a small circle (i.e. $cost(X) = 10$). Finally, as an additional constraint, we require that the total costs of all selected features does not exceed the threshold 35.

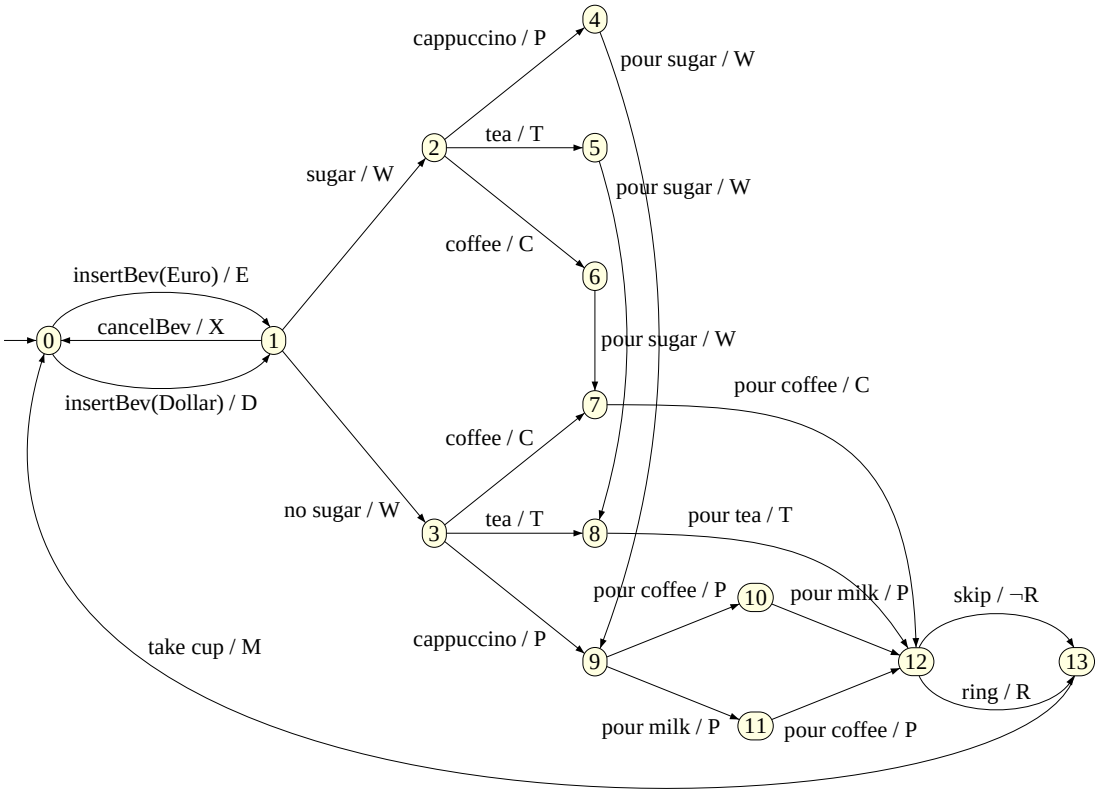

Figure 7: FTS of beverage component

The FTS of the beverage component contains 14 states and 23 transitions and that of the soup component contains 13 states and 28 transitions, for a total of 182 states and 691 transitions in parallel composition.

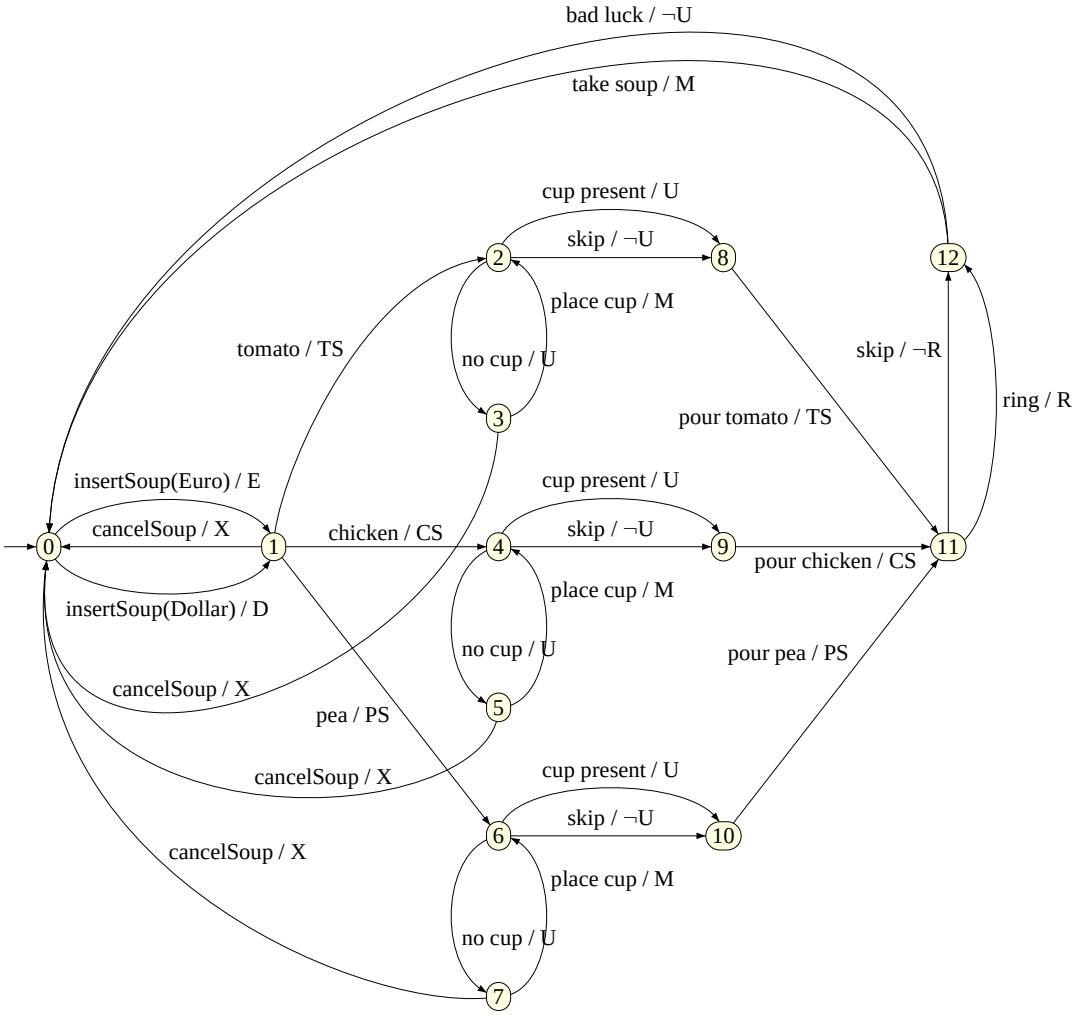

Figure 8: FTS of soup component

As reported in Section 5, we used the mCRL2 toolset to verify 12 properties against this SPL. These properties are listed next, together with their formalization in the mCRL2 variant of the modal μ-calculus.

1. If a coffee is ordered, then eventually coffee is poured: $[\mathit{true}*.\mathit{coffee}](\mathrm{mu}\,X.[!\mathit{pour\_coffee}]X)$
2. The SPL is deadlock-free: $[\mathit{true}*]\langle\mathit{true}\rangle\mathit{true}$

3a. A machine that accepts Euros does not accept Dollars:
$[\mathit{true}*.(\mathit{insertBev}(\mathit{Euro})\,||\,\mathit{insertSoup}(\mathit{Euro})).\mathit{true}*.(\mathit{insertBev}(\mathit{Dollar})\,||\,\mathit{insertSoup}(\mathit{Dollar}))]\mathit{false}$

3b. A machine that accepts Dollars does not accept Euros:
$[\mathit{true}*.(\mathit{insertBev}(\mathit{Dollar})\,||\,\mathit{insertSoup}(\mathit{Dollar})).\mathit{true}*.(\mathit{insertBev}(\mathit{Euro})\,||\,\mathit{insertSoup}(\mathit{Euro}))]\mathit{false}$

4a. A cup can only be taken out of the beverage component after a beverage was ordered:
$[(!\mathit{coffee}\ \&\&\ !\mathit{tea}\ \&\&\ !\mathit{cappuccino})*.\mathit{take\_cup}]\ \mathit{false}$

4b. A cup can only be taken out of the soup component after soup was ordered:
$[(!\mathit{tomato}\ \&\&\ !\mathit{chicken}\ \&\&\ !\mathit{pea})*.\mathit{take\_soup}]\ \mathit{false}$

5a. If a beverage is ordered, then eventually the beverage is canceled or a cup is taken out of the beverage component: $[\mathit{true}*.(\mathit{coffee} \mid\mid \mathit{tea} \mid\mid \mathit{cappuccino})](\mathrm{mu}\,X.[(!\mathit{cancelBev}\ \&\&\ !\mathit{take\_cup})]X)$

5b. If soup is ordered, then eventually the soup is canceled, a cup is taken out of the soup component or the customer has bad luck:
$[\mathit{true}*.(\mathit{tomato} \mid\mid \mathit{chicken} \mid\mid \mathit{pea})](\mathrm{mu}\,X.[(!\mathit{cancelSoup}\ \&\&\ !\mathit{take\_soup}\ \&\&\ !\mathit{bad\_luck})]X)$

6. If the machine has a soup component, then a beverage can be ordered without inserting more money after soup was ordered: $[\mathit{true}*.(\mathit{insertSoup}(\mathit{Euro}) \mid\mid \mathit{insertSoup}(\mathit{Dollar}))]\langle \mathit{true}*.(\mathit{tomato} \mid\mid \mathit{chicken} \mid\mid \mathit{pea}).(!\,\mathit{insertBev}(\mathit{Euro})\ \&\&\ !\,\mathit{insertBev}(\mathit{Dollar}))*.(\mathit{coffee} \mid\mid \mathit{tea} \mid\mid \mathit{cappuccino})\rangle\,\mathit{true}$

7a. A beverage cannot be ordered without inserting more money if a previous beverage order is still pending: $[\mathit{true}*.(\mathit{coffee} \mid\mid \mathit{tea} \mid\mid \mathit{cappuccino}).(!\mathit{insertBev}(\mathit{Dollar})\ \&\&\ !\mathit{insertBev}(\mathit{Euro}))*.(\mathit{coffee} \mid\mid \mathit{tea} \mid\mid \mathit{cappuccino})]\ \mathit{false}$

7b. Soup cannot be ordered without inserting more money if a soup order is pending: $[\mathit{true}*.(\mathit{tomato} \mid\mid \mathit{chicken} \mid\mid \mathit{pea}).(!\,\mathit{insertSoup}(\mathit{Dollar})\ \&\&\ !\,\mathit{insertSoup}(\mathit{Euro}))*.(\mathit{tomato} \mid\mid \mathit{chicken} \mid\mid \mathit{pea})]\ \mathit{false}$

8. In a machine with cup detection, soup can only be poured after detecting a cup: $[\mathit{true}*.\mathit{cup\_present}]$ $[\mathit{true}*.(\mathit{take\_soup} \mid\mid \mathit{bad\_luck}).(!\,\mathit{cup\_present})*.(\mathit{pour\_tomato} \mid\mid \mathit{pour\_chicken} \mid\mid \mathit{pour\_pea})]\ \mathit{false}$