Model Checking Partial Software Product Line Designs

Yufeng Shi, Ou Wei, Yu Zhou
Department of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing,China
{shiyufeng,owei,zhouyu}@nuaa.edu.cn

ABSTRACT

Software product line (SPL) maximizes commonality between software products to reduce cost and improve productivity, where each product is represented by a selection of features that corresponds to particular customer requirements. SPL has been widely applied in critical systems such as communications, automobile, and aerospace, and ensuring correctness of the system is thus of great importance. In this paper, we consider model checking partial software product line designs, i.e., the incomplete designs in the early stage of software development, where the design decisions for a feature may be unknown. This enables detecting design errors earlier, reducing the cost of later development of final products. To this end, we first propose bilattice-based feature transitions systems (BFTSs) for modeling partial software product line designs, which support description of uncertainty and preserve features as a first class notion. We then express system behavioral properties using ACTL formulas and define its semantics over BFTSs. Finally, to leverage the power of existing model checking engine for verification, we provide the procedures that translate BFTSs and ACTL formulas to the inputs of the symbolic model checker χChek. We implement our approach and illustrate its effectiveness on a benchmark from literature.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Model checking*

General Terms

Theory, Verification

Keywords

Model Checking, Software Product Line, Partial Model

InnoSWDev'14, November 16, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3226-2/14/11...$15.00
http://dx.doi.org/10.1145/2666581.2666589

1. INTRODUCTION

Software product line maximizes commonality between software products to reduce cost and improve productivity. Each product is represented by a selection of features that correspond particular customer requirements. Recently, software product lines have been widely applied in industries such as communications, automobile, and aerospace [24], where correctness of the systems are of great importance to the success of the industry. In this paper, we consider model checking software product lines with uncertainty, i.e., the incomplete design models in the early stage of software product line development, where the design decisions for a feature may be unknown. By conducting verification on early design models, it allows for detecting errors, particularly the ones associated with different features, earlier in the development life cycle, and thus reduces the cost of fixing the problems; the model checking results also help to resolve design uncertainties that influence later implementation choices of the systems [20].

Model checking [6] is an automatic verification technique for checking behavioral system properties via exhaustively searching all possible contained behaviors, where a system is typically modeled as a transition system and properties of the system are specified using temporal logics. In order to perform model checking on incomplete software product lines designs, we need appropriate formalisms for system modeling and property specification, and efficient model checking approach.

Multi-valued models, based on multi-valued logics, have been proposed for partial product line modeling and verification [13, 4], where the values in such a logic capture the degree of uncertainty, used to specify system behaviors and represent model checking results. These models, however, as pointed in [8], express product features as part of the behavioral models, rather than a first-class notion, which makes it hard to explicitly capture relations between products and feature selections. In [8, 7], Classen et al. propose *feature transition systems* (FTSs) for modeling behaviors of product line systems, which recognize feature as the unit of difference and explicitly relate system behaviors to their originating features at the level of individual transitions. An FTS, however, is complete, i.e, the relation between features and transitions are boolean — a transition is either associated with a set of features, or not; therefore, it does not support modeling designs that are not yet finished.

In this paper, we propose bilattice-based feature transitions systems (BFTSs) – a combination of multi-valued and feature-based product line models – for model checking incomplete software product line designs. The transitions in a BFTS are labeled by the values from a world-based bilattice $\mathcal{B}$ [14] that is a multi-valued logic built from the set W of SPL features. Each value in $\mathcal{B}$ is a pair $\langle U, V\rangle$, where U and V are subsets of W, and U and W are not necessarily complement each other. A transition labeled by $\langle U, V\rangle$ means that it is required by the features in U and refuted by the ones in V, and the design decisions of this transition on the rest of the features in W are unknown. We assume that $U \cap V = \emptyset$, i.e., design decisions for each feature are always consistent. BFTSs, therefore, support modeling incomplete software product line designs while preserving features as a first class notion. For the system properties of interest, we express them using Action Computation Tree Logic (ACTL) [10], which are interpreted over BFTSs based on a multi-valued semantics. The value of an ACTL formula φ on a BFTS is also a value $\langle U, V\rangle$ from the bilattice $\mathcal{B}$, which means that φ is satisfied by the products with features selected only from U, refuted by the products containing features from V, and unknown for the rest. Such a result provides verification information of an incomplete design, which are valuable for later implementation of final products.

Furthermore, instead of implementing an ad hoc model checker for verification of ACTL formulas over BFTSs, we leverage the power of existing model checking engine χChek [5] for this. χChek is a symbolic model checker that is optimized for multi-valued model checking, which takes as inputs a multi-valued logic, a multi-valued Kripke structure, and a CTL formula, and returns a value that denotes the degree of the truth of the formula on the structure. In this context, we provide the translation from a BFTS to a multi-valued Kripke structure and the one from an ACTL formula to a CTL formula. Finally, as a proof of concept, we implement the procedure and illustrate our approach with a benchmark from software product line literature [12, 2].

Structure of the paper. Section 2 reviews the necessary background. Section 3 introduces the BFTS notion for modelling SPLs, its projection on products, and the relationship with complete product models. Section 4 defines the semantics of ACTL formulas on BFTSs. Section 5 provides the translations for BFTSs and ACTL formulas, and reports the results of model checking an SPL benchmark using χChek. Section 6 discusses related work, and Section 7 concludes the paper.

2. PRELIMINARIES

In this section, we review basic concepts and background of our work, including feature diagrams and world-based bilattices.

2.1 Feature Diagram

Products of an SPL share substantial commonalities while show variabilities for particular needs. Feature diagram is a common approach for variability modeling.

Definition 1. [1] A *feature diagram (FD)* is a tree over the feature set $\mathcal{F}$, where each edge is defined by exactly one feature constraint: *mandatory*, *optional*, *alternative*, or *or*, where:

$$\begin{aligned} mandatory(p, f) &\equiv f \Leftrightarrow p \\ optional(p, f) &\equiv f \Rightarrow p \\ alternative(p, f_1, ...f_n) &\equiv ((f_1 \vee ... \vee f_n) \Leftrightarrow p) \wedge \\ &\quad \bigwedge_{i<j} \neg(f_i \wedge f_j) \\ or(p, f_1, ...f_n) &\equiv (f_1 \wedge ... \wedge f_n) \Leftrightarrow p \end{aligned}$$

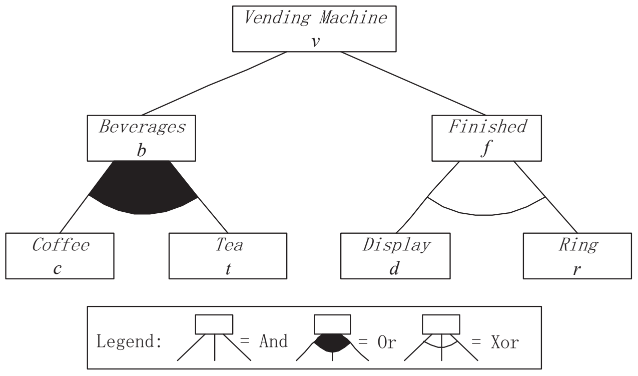


Figure 1: FD of a beverage vending machine SPL.

Consider the example of a product line of vending machine adapted from [12]. The FD of this product line is shown in Figure 1. The beverage vending machine has a mandatory feature *Finished* that offers alternative ways of reminding customers the beverage is finished, either by displaying "done" or ringing a tone. And there is a mandatory feature *Beverages* that allows the machine be able to offer at least one type of drink between coffee and tea.

A valid product P of a product line is a selection of features that satisfies the constraints defined in the FD [1]. For the example of vending machine, the semantics of the FD would be the following set of six products (using the short feature names):

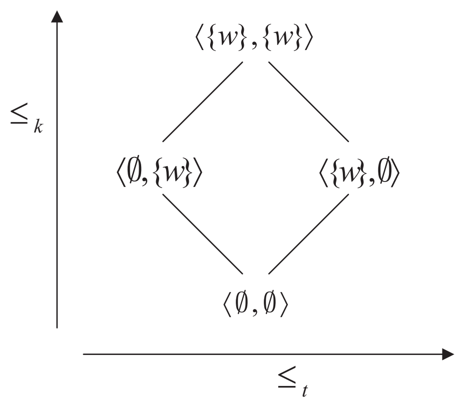
$$\{\{v, b, f, c, d\}, \{v, b, f, c, r\}, \{v, b, f, t, d\}, \{v, b, f, t, r\}, \\ \{v, b, f, c, t, d\}, \{v, b, f, c, t, r\}\}.$$

2.2 World-based Bilattices

Our work is based on the multi-valued logic defined by world-based bilattices [14]. Bilattices are a natural generalization of classical two-valued logic. Informally, a bilattice is a space of generalized truth values with two lattices ordering: one ordering, $\leq_t$, records the degree of truth, where the bottom is denoted by $false$ and the top – by $true$, and the other ordering, $\leq_i$, records the degree of information or knowledge. Thus, if $x \leq_i y$, then y gives at least as much as information as x.

Definition 2. [14] A *world-based bilattice* is a structure $\mathcal{B}_W = \langle B_W, \leq_t, \leq_i, \neg\rangle$ such that: (i) $B_W \triangleq \mathcal{P}(W) \times \mathcal{P}(W)$, where W is a set of worlds and $\mathcal{P}(W)$ is the power set of W; (ii) $\neg$ is a negation and (iii) $\leq_t$ and $\leq_i$ are truth and information orderings, respectively, where

$$\begin{aligned} \langle U, V\rangle \leq_i \langle S, T\rangle &\triangleq U \subseteq S \wedge T \subseteq V \\ \langle U, V\rangle \leq_t \langle S, T\rangle &\triangleq U \subseteq S \wedge V \subseteq T \\ \neg\langle U, V\rangle &\triangleq \langle V, U\rangle \end{aligned}$$



$\langle\{w\},\{w\}\rangle$
$\langle\emptyset,\{w\}\rangle$
$\langle\{w\},\emptyset\rangle$
$\langle\emptyset,\emptyset\rangle$
$\leq_k$
$\leq_t$

Figure 2: A world-based bilattice $\mathcal{B}_w$ over $\{w\}$.

For example, Figure 2 shows a world-based bilattice $\mathcal{B}_w$ over $\{w\}$, where $\langle\{w\},\emptyset\rangle$ represents *true* – maximal degree of truth, $\langle\emptyset,\{w\}\rangle$ represents *false* — minimal degree of falsity, $\langle\emptyset,\emptyset\rangle$ represents *unknown* – minimal degree of information, and $\langle\{w\},\{w\}\rangle$ represents *inconsistency* – maximal degree of information. For convenience, for each pair $\langle U,V\rangle$, we introduce projections π_t and π_f defined as $\pi_t(\langle U,V\rangle) \triangleq U$ and $\pi_f(\langle U,V\rangle) \triangleq V$, respectively.

Particularly, the meet and join operations for $\leq_i$ are denoted by $\otimes$ and $\oplus$ while the meet and join operations for $\leq_t$ are denoted by $\wedge$ and $\vee$, which are defined as follows:

THEOREM 1. *[14] Given a world-based bilattice $\mathcal{B}_W = \langle B_W, \leq_t, \leq_i, \neg\rangle$ such that $B_W \triangleq P(W) \times P(W)$. Then, for any $\langle U,T\rangle, \langle S,T\rangle \in \mathcal{B}_W$,*

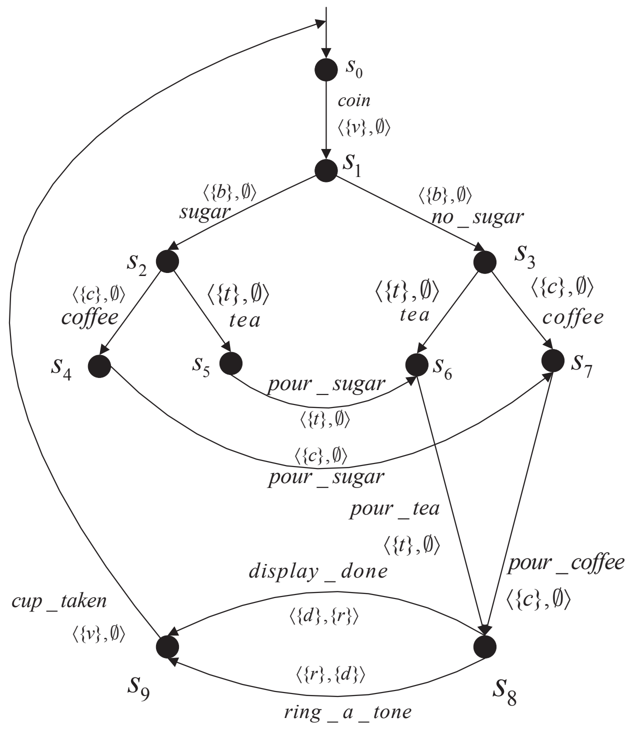
$$\begin{array}{rcl}\langle U,V\rangle \wedge \langle S,T\rangle & \triangleq & \langle U\cap S, V\cup T\rangle \\ \langle U,V\rangle \vee \langle S,T\rangle & \triangleq & \langle U\cup S, V\cap T\rangle \\ \langle U,V\rangle \otimes \langle S,T\rangle & \triangleq & \langle U\cap S, V\cap T\rangle \\ \langle U,V\rangle \oplus \langle S,T\rangle & \triangleq & \langle U\cup S, V\cup T\rangle\end{array}$$

3. PARTIAL MODELING OF SOFTWARE PRODUCT LINE

In this section, we introduce Bilattice-based Feature Transition Systems (BFTSs) that support modeling product line behaviors with uncertainty. We also define the projection of a BFTS over a selection of features, and its relations with the behavioral model of a product.

3.1 Bilattice-based Featured Transition System

Typically, the behaviors of a system are modeled using state-transition system. A feature represents a system function for particular needs; it is associated with certain behaviors of the system, i.e., the transitions of the system model. Due to incomplete information in the early design phase, there exist three types of relations between features and transitions: *require*, *forbid*, and *unknown*. Therefore, given a set W of features, we can associate each transition (s,a,s') a value $\langle U,V\rangle$ from the world-based lattice $\mathcal{B}_W$. Intuitively, if a feature $f \in U$, (s,a,s') is required by f; if $f \in V$, (s,a,s') is forbidden by f; if $f \notin U \cup V$, (s,a,s') may be required by f. We assume that $U \cap V = \emptyset$; that is, the designs of each feature in consistent. We formally define *bilattice-based featured transition system* as follows.



s_0
coin
$\langle\{v\},\emptyset\rangle$
s_1
$\langle\{b\},\emptyset\rangle$
sugar
$\langle\{b\},\emptyset\rangle$
no_sugar
s_2
s_3
$\langle\{c\},\emptyset\rangle$
coffee
$\langle\{t\},\emptyset\rangle$
tea
$\langle\{t\},\emptyset\rangle$
tea
$\langle\{c\},\emptyset\rangle$
coffee
s_4
s_5
s_6
s_7
pour_sugar
$\langle\{t\},\emptyset\rangle$
$\langle\{c\},\emptyset\rangle$
pour_sugar
pour_tea
$\langle\{t\},\emptyset\rangle$
pour_coffee
$\langle\{c\},\emptyset\rangle$
display_done
$\langle\{d\},\{r\}\rangle$
cup_taken
$\langle\{v\},\emptyset\rangle$
$\langle\{r\},\{d\}\rangle$
ring_a_tone
s_9
s_8

Figure 3: BFTS of the vending machine SPL.

Definition 3. A *Bilattice-based Featured Transition Systems (BFTS)* is a 6-tuple $M = (S, Act, \rightarrow, s_0, \mathcal{B}_W, R)$, where:

- S is a finite set of states,
- Act is a set of actions,
- $\rightarrow \subseteq S \times Act \times S$ is a set of transitions,
- s_0 is the initial state,
- $\mathcal{B}_W = \langle \mathcal{P}(W) \times \mathcal{P}(W), \leq_t, \leq_i, \neg\rangle$ is a world-based bilattice,
- $R : S \times Act \times S \rightarrow \mathcal{B}_W$ is a mapping of transitions to truth values in $\mathcal{B}_W$.

Each transition (s,a,s') in M is labeled with a truth value in $\mathcal{B}_W$. $R : S \times Act \times S \rightarrow \mathcal{B}_W$ is a total function assigning a value from $\mathcal{B}_W$ to each transition between any two states with a certain action. The value of (s,a,s') in M is thus referred to $R(s,a,s')$. If the value of the transition (s,a,s') is $\langle U,V\rangle$, this means (s,a,s') is required by features in U but forbidden by those in V, and its relation with the rest of the features are unknown. In particular, if $R(s,a,s') = \langle\emptyset, W\rangle$, the transition (s,a,s') does not exist in M, i.e., $(s,a,s') \notin \rightarrow$.

The BFTS depicted in Figure 3, in which the truth value of a transition is shown next to its action label, is an attempt to model all possible behaviors conceived for SPL of beverage vending machine. The transition $(s_0, coin, s_1)$, for instance, has the truth value $\langle\{v\},\emptyset\rangle$, meaning that it is required by the root feature, *VendingMachine* and other features (e.g., the feature *Beverages*) are uncertain with it.

The truth values specified in pairs such that $\langle U, V \rangle$ offer an intuitive way to express cases in which two or more features are mutually exclusive, i.e., those cannot be present simultaneously in products. If they are mutually exclusive, there must exist some behaviors where they disagree. The feature d and the feature r, for instance, are mutually exclusive as depicted in Figure 1. Thus in the behavioral model of the SPL, there exist some transition they disagree on, e.g., the transition $(s_8, display\_done, s_9)$ which has the truth value $\langle \{d\}, \{r\} \rangle$.

3.2 Projection and Product Model

Given a partial SPL design represented by a BFTS M, it contains the information of all the products that may not complete. For each product P, i.e., a selection of features, we can derive a partial model of P from M via projection.

Definition 4. Given a BFTS $M = (S, Act, \rightarrow, s_0, \mathcal{B}_W, R)$ that specifies an SPL and a selection of features P. A BFTS $M' = (S', Act, \rightarrow', s_0, \mathcal{B}_P, R')$ is the *projection* of M over P, noted $M_{|P}$, where:

1. $\rightarrow' = \{(s, a, s') | (s, a, s') \in \rightarrow \cdot \pi_f(R(s, a, s')) \cap P = \emptyset\}$,
2. $S' = \{s | \exists s' \in S \exists a \in Act \cdot ((s, a, s') \in \rightarrow' \vee (s', a, s) \in \rightarrow')\}$,
3. $Act' = \{a | \exists s \in S \exists s' \in S \cdot (s, a, s') \in \rightarrow'\}$,
4. $$R'(s, a, s') = \begin{cases} \langle P, \emptyset \rangle, & \text{if } P \subseteq \pi_t(R(s, a, s')), \\ \langle \emptyset, P \rangle, & \text{if } \pi_f(R(s, a, s')) \cap P \neq \emptyset, \\ \langle \emptyset, \emptyset \rangle, & \text{otherwise.} \end{cases}$$

The projection M' is indeed a 3-valued model, since features in products may contain *unknown* relations. If all features definitely *require* or *forbid* transitions, M' is a 2-valued model. In our beverage vending machine example, the projection on the set $\{v, b, f, c, r\}$ has the behaviors specified in the 3-valued BFTS in Figure 4. We notice that with the selection $\{v, b, f, c, r\}$, existence of some transitions in the corresponding product $P = \{v, b, f, c, r\}$ is not definitely determined because of unsolved uncertainty, e.g., the transition (s_2, tea, s_5).

Given the projection over the selection of features, the behavioral model of the product over the same selection of features can be derived by resolving all the *unknown* relations in the projection; then product's behavioral models only contain transitions which are definitely required.

Definition 5. [12] Let $M_{|P} = (S_1, Act_1, \rightarrow_1, s_{0_1}, \mathcal{B}_P, R_1)$ be the projection of a BFTS M over a product P. The *product model* of P is a 2-valued BFTS $M(P) = (S_2, Act_2, \rightarrow_2, s_{0_2}, \mathcal{B}_P, R_2)$ such that there exists a simulation $H \subseteq S_2 \times S_1$, where $(s_{0_2}, s_{0_1}) \in H$ and for any $(s_2, s_1) \in H$, the following conditions hold:

- $\forall s_1' \in S_1 \cdot \forall a \in Act_1 \cdot P \subseteq \pi_t(R_1(s_1, a, s_1')) \Rightarrow \exists s_2' \in S_2 \cdot P \subseteq \pi_t(R_2(s_2, a, s_2')) \wedge (s_2', s_1') \in H$
- $\forall s_2' \in S_2 \cdot \forall a \in Act_2 \cdot P \not\subseteq \pi_f R_2(s_2, a, s_2') \Rightarrow \exists t_1 \in S_1 \cdot P \not\subseteq \pi_f(R_1(s_1, a, s_1')) \wedge (s_2', s_1') \in H$

Figures 5 and 6 show two possible implementations of the product $p = \{v, b, f, c, r\}$ according to two different resolutions on uncertainty: the former indicates that unknown transitions are not required by features finally; the latter indicates that they are required.

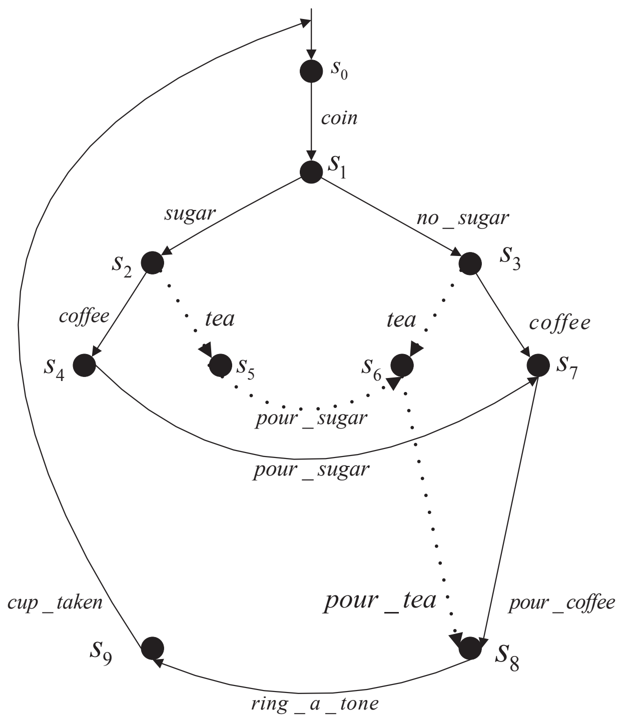


Figure 4: Projection of the vending machine model over the features $\{v, b, f, c, r\}$ (solid arcs are transitions with the value $\langle P, \emptyset \rangle$ and dashed arcs – with $\langle \emptyset, \emptyset \rangle$).

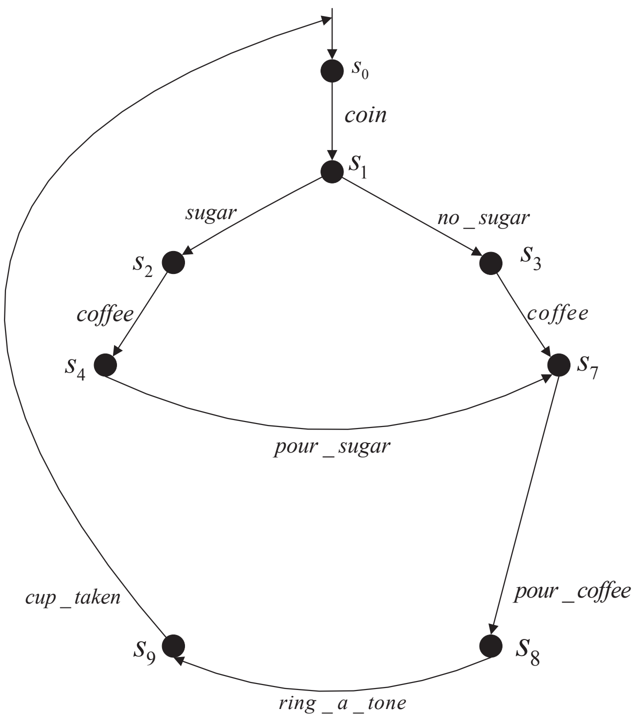


Figure 5: An implementation of the vending machine product defined by $P = \{v, b, f, c, r\}$.

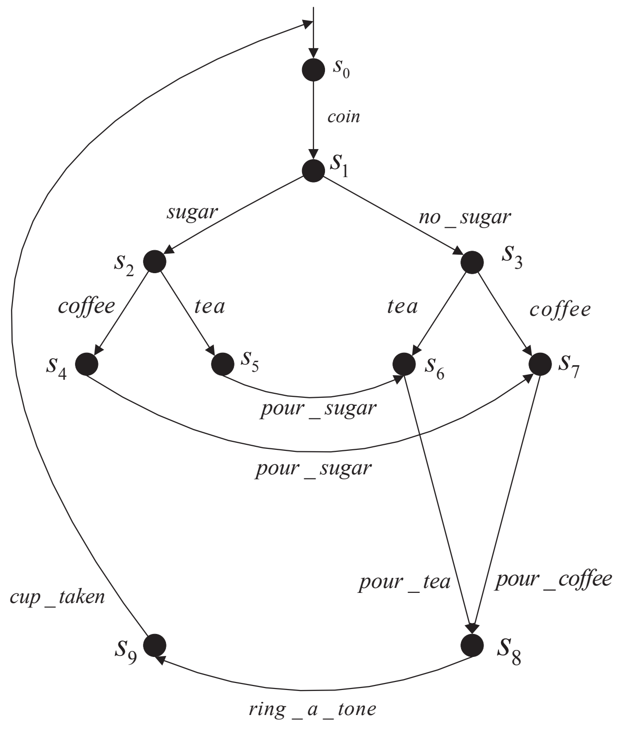


Figure 6: An alternative implementation of the vending machine product defined by $P = \{v, b, f, c, r\}$.

4. SEMANTICS OF ACTL ON BFTS

Our study of model checking SPLs focuses on verifying *Action Computation Tree Logic* (ACTL) [10] properties for the SPL system. As an action-based temporal logic, ACTL is suitable to express properties of reactive systems whose behaviors are characterized by the actions they perform.

Definition 6. The syntax of ACTL logic is given by the following grammar:

$$\varphi ::= true|false|\neg\varphi|\varphi \wedge \varphi'|\varphi \vee \varphi'|(\varphi \rightarrow \varphi')\varphi|\langle a\rangle\varphi|$$
$$[a]\varphi|AF\varphi|AG\varphi|EF\varphi|EG\varphi|A[\varphi U \varphi']|E[\varphi U \varphi']|$$

with $a \in Act$ is an action.

Intuitively, given an ACTL formula $\langle a\rangle\varphi$ (resp. $[a]\varphi$), it requires some (resp. every) next state arrived through the action a to satisfy φ. The symbols $\wedge$,$\vee$,$\rightarrow$ and $\neg$ have their usual meanings. As in CTL, E (resp. A) means at least one path satisfies (resp. all paths satisfy) the following path operators: U – the *until* operator such that the formula $A[\varphi U \varphi']$ (resp. $E[\varphi U \varphi']$) means that for every (resp. every) computation path there exists an initial prefix of the path where φ' holds at the last state and φ holds at all other states, F – the *future* operator such that the formula $AF\varphi$ (resp. $EF\varphi$) means that for every (resp. some) computation path φ holds at some future state, and G – the *global* operator such that the formula $AG\varphi$ (resp. $EG\varphi$) means that for every (resp. some) computation path φ holds at all future states. An example property for our vending machine is

$$AG[sugar]AF\langle pour\_sugar\rangle true$$

which reads that "whenever customers choose any drink with sugar, then the system will pour sugar in drinks."

The classical semantic of ACTL operators is introduced by De Nicola in [10] on (boolean) Labelled Transition Systems. In the following, we define a multi-valued semantics of ACTL over BFTSs.

Definition 7. Let $M = (S, Act, \rightarrow, s_0, \mathcal{B}_W, R)$ be a BFTS and φ be an ACTL formula. The semantics of φ, denoted by $\|\varphi\|^M$ ($\|\varphi\|$ when M is clear from context) which returns the value of φ for each state $s \in S$, is a mapping $S \rightarrow B_W$ and defined inductively on the structure of the formula:

$$\begin{array}{rcl}
\|true\|^M & = & \lambda s.\langle W, \emptyset\rangle \\
\|false\|^M & = & \lambda s.\langle \emptyset, W\rangle \\
\|\neg\varphi\|^M & = & \lambda s.\neg\|\varphi\|^M(s) \\
\|\varphi \wedge \varphi'\|^M & = & \lambda s.\|\varphi\|^M(s) \wedge \|\varphi'\|^M(s) \\
\|\varphi \vee \varphi'\|^M & = & \lambda s.\|\varphi\|^M(s) \vee \|\varphi'\|^M(s) \\
\|\langle a\rangle\varphi\|^M & = & pre_a[R](\|\varphi\|^M) \\
\|[a]\varphi\|^M & = & \tilde{pre}_a[R](\|\varphi\|^M) \\
\|AF\varphi\|^M & = & \mu Q.\|\varphi\|^M \vee (\bigwedge_{a\in Act} \tilde{pre}_a[R](Q)) \\
\|EF\varphi\|^M & = & \mu Q.\|\varphi\|^M \vee (\bigvee_{a\in Act} pre_a[R](Q)) \\
\|AG\varphi\|^M & = & \upsilon Q.\|\varphi\|^M \wedge (\bigwedge_{a\in Act} \tilde{pre}_a[R](Q)) \\
\|EG\varphi\|^M & = & \upsilon Q.\|\varphi\|^M \wedge (\bigvee_{a\in Act} pre_a[R](Q)) \\
\|A[\varphi \cup \varphi']\|^M & = & \mu Q.\|\varphi'\|^M \vee (\|\varphi\|^M \wedge (\bigwedge_{a\in Act} \tilde{pre}_a[R](Q)) \\
\|E[\varphi \cup \varphi']\|^M & = & \mu Q.\|\varphi'\|^M \vee (\|\varphi\|^M \wedge (\bigwedge_{a\in Act} pre_a[R](Q))
\end{array}$$

where the μ and υ operators are used to express least and greatest fixpoints, respectively; and for a transition relation $R : S \times Act \times S \rightarrow \mathcal{B}_W$, we define the *preimage* of $Q : S \rightarrow \mathcal{B}_W$ w.r.t. R and a given action a, $pre_a[R] : [S \rightarrow \mathcal{B}_W] \rightarrow [S \rightarrow \mathcal{B}_W]$ as

$$pre_a[R](Q) \triangleq \lambda s. \bigvee_{s'\in S} (R(s, a, s') \wedge Q(s'))$$

and its dual is $\tilde{pre}$:

$$\tilde{pre}_a[R](Q) \triangleq \neg pre_a[R](\neg Q)$$

For example, a property of the beverage vending machine SPL shown in Figure 3 is that when the system is at the state s_8, it can ring a tone and then customers take the cup, expressed as the formula

$$\langle ring\_a\_tone\rangle\langle cup\_taken\rangle true$$

According to Definition 7, the value of $\|\varphi\|(s_8)$ is computed as following:

$$\begin{array}{l}
\|\varphi\|(s_8) \\
=R(s_8, ring\_a\_tone, s_9) \wedge \|\langle cup\_taken\rangle true\|(s_9) \\
=R(s_8, ring\_a\_tone, s_9) \wedge R(s_9, cup\_taken, s_0) \\
=\langle\{r\}, \{d\}\rangle \wedge \langle\{v\}, \emptyset\rangle \\
=\langle\emptyset, \{d\}\rangle
\end{array}$$

Based on the relation between a partial SPL model M and the final implementation of a final product P (Definitions 4 and 5), we have the following result. For any ACTL formula φ, if the value of φ on M is $\langle U, V\rangle$, then if $P \cap V \neq \emptyset$, then φ is false on $M(P)$; if $P \subseteq U$, then φ is true on $M(P)$; otherwise, the value of φ is unknown on $M(P)$.

For the previous example, $\|\varphi\|(s_8) = \langle\emptyset, \{d\}\rangle$, which means that the property does not hold on s_8 on any product that contains the feature d. On the other hand, for the formula $\psi = AF(\langle tea\rangle true \vee \langle coffee\rangle true \rightarrow \langle cup\_taken\rangle true)$ and the product $P = \{v, b, f, c, , r\}$, $\|\psi\|(s_0) = \langle\{v, b, c, t, f, r, d\}, \emptyset\rangle$ and $P \subseteq \{v, b, c, t, f, r, d\}$, therefore ψ is true on the two implementations of P shown in Figures 5 and 6.

5. MODEL CHECKING WITH χCHEK

To experiment our approach, we leverage the existing multi-valued symbolic model checker χChek [5] for model checking BFTSs. χChek is implemented in Java, and provides support for multi-valued model checking. Given a CTL property and a system modeled in χKripke structure – a multi-valued Kripke structure [3], χChek checks the model and returns a value that denotes the degree of the truth of the formula on the structure. BFTSs and ACTL can not be directly checked by χChek. Therefore, we provide the translation from a BFTS to a multi-valued Kripke structure and the one from an ACTL formula to a CTL formula. We implement the procedure and report the model checking results on the vending machine example.

Translations of BFTSs and ACTL. Action-based transition systems and Kripke structures are two types of formalisms that have been used for modeling reactive and concurrent systems. In [11], two transformation functions are introduced for Kripke structures and Labelled Transition Systems; Godefroid [15] proves the same expressiveness between 3-valued Kripke structures and modal transition systems. Inspired by these results, we define the translation from a BFTS to a χKripke structure.

Definition 8. [3] A *χKripke Structure* is a 7-tuple $M = (S, AP, \rightarrow, S_0, \mathcal{B}_W, R, L)$ where:

1. S is a finite set of states,
2. AP is a set of atomic propositions,
3. $\rightarrow \subseteq S \times S$ is a set of transitions,
4. $S_0 \subseteq S$ is the non-empty set of initial states,
5. $\mathcal{B}_W$ is a world-based bilattice,
6. R:$S \times S \rightarrow \mathcal{B}_W$ is a mapping of transitions to truth values,
7. L:$S \times AP \rightarrow \mathcal{B}_W$ is a total function that maps a state and an atomic proposition to truth values.

A χKripke Structure is based on the same logic as a BFTS: the degrees of truth are given in a world-based bilattice and truth values of transitions are pairs such that $\langle U, V\rangle$.

Definition 9. The semantics of a CTL formula φ on a χKripke Structure $M = (S, AP, \rightarrow, S_0, \mathcal{B}_W, R, L)$, written $\|\varphi\|^M$($\|\varphi\|$ when M is clear from context), is defined inductively as follows:

$$\begin{aligned}
\|true\|^M &= \lambda s.\langle W, \emptyset\rangle \\
\|false\|^M &= \lambda s.\langle\emptyset, W\rangle \\
\|\neg\varphi\|^M &= \lambda s.\neg\|\varphi\|^M(s) \\
\|\varphi \wedge \varphi'\|^M &= \lambda s.\|\varphi\|^M(s) \wedge \|\varphi'\|^M(s) \\
\|EX\varphi\|^M &= pre[R](\|\varphi\|^M) \\
\|AX\varphi\|^M &= \tilde{pre}[R](\|\varphi\|^M) \\
\|AF\varphi\|^M &= \mu Q.\|\varphi\|^M \vee \tilde{pre}[R](Q) \\
\|EF\varphi\|^M &= \mu Q.\|\varphi\|^M \vee pre[R](Q) \\
\|AG\varphi\|^M &= \upsilon Q.\|\varphi\|^M \wedge \tilde{pre}[R](Q) \\
\|EG\varphi\|^M &= \upsilon Q.\|\varphi\|^M \wedge pre[R](Q) \\
\|A[\varphi \cup \varphi']\|^M &= \mu Q.\|\varphi'\|^M \vee (\|\varphi\|^M \wedge \tilde{pre}[R](Q)) \\
\|E[\varphi \cup \varphi']\|^M &= \mu Q.\|\varphi'\|^M \vee (\|\varphi\|^M \wedge pre[R](Q))
\end{aligned}$$

where the μ and υ operators are used to express least and greatest fixpoints, respectively; and for a transition relation $R : S \times S \rightarrow \mathcal{B}_W$, we define the *preimage* of $Q : S \rightarrow \mathcal{B}_W$ w.r.t. R , $pre[R] : [S \rightarrow \mathcal{B}_W] \rightarrow [S \rightarrow \mathcal{B}_W]$ as

$$pre[R](Q) \triangleq \lambda s. \bigvee_{s' \in S} (R(s, s') \wedge Q(s'))$$

and its dual is $\tilde{pre}$:

$$\tilde{pre}[R](Q) \triangleq \neg pre[R](\neg Q)$$

The translation from a BFTS to a χKripke structure is then defined as follows.

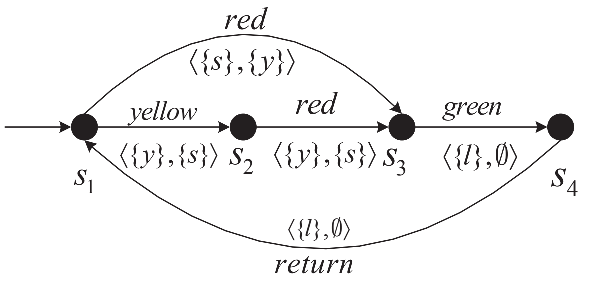
Definition 10. For any BFTS $M = (S, Act, \rightarrow, s_0, \mathcal{B}_W, R)$, we define an equivalent χKripke structure $M' = (S', AP, \rightarrow'$ $, S'_0, \mathcal{B}_W, R', L)$ such that

1. $S' = S \times Act$,
2. $AP = Act$,
3. $S'_0 = s_0 \times Act$,
4. $\rightarrow' = \{((s, a), (s', a')) | (s, a', s') \in \rightarrow\}$,
5. $\forall(s, a), (s', a') \in S' : R'((s, a), (s', a')) = R(s, a', s')$,
6. $\forall(s, a) \in S' : \forall p \in AP :$

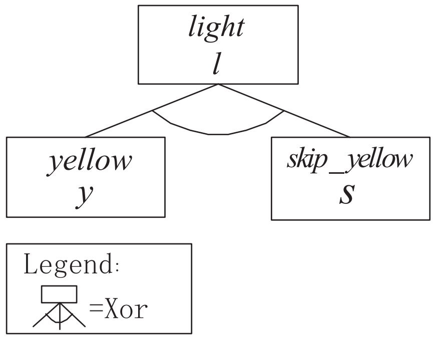
$$L((s, a), p) = \begin{cases} \langle W, \emptyset\rangle, & \text{if } p = a \ , \\ \langle\emptyset, W\rangle, & \text{else.} \end{cases}$$

Given a BFTS with $|S|$ states and an action set Act, the number of states in the translated χKripke structure M' defined by definition is at most $|S| \cdot |Act|$; that is, each state of M can be copied at most $|Act|$ times in M'. For a fixed action set Act, the number of states and transitions in M' is nevertheless linear in the number of states and transitions, respectively, in M. Particularly, if $\mathcal{B}_W$ in M is defined such that $|W| = 1$, the above translation then becomes translation from modal transition systems to Kripke modal transition systems in [15].

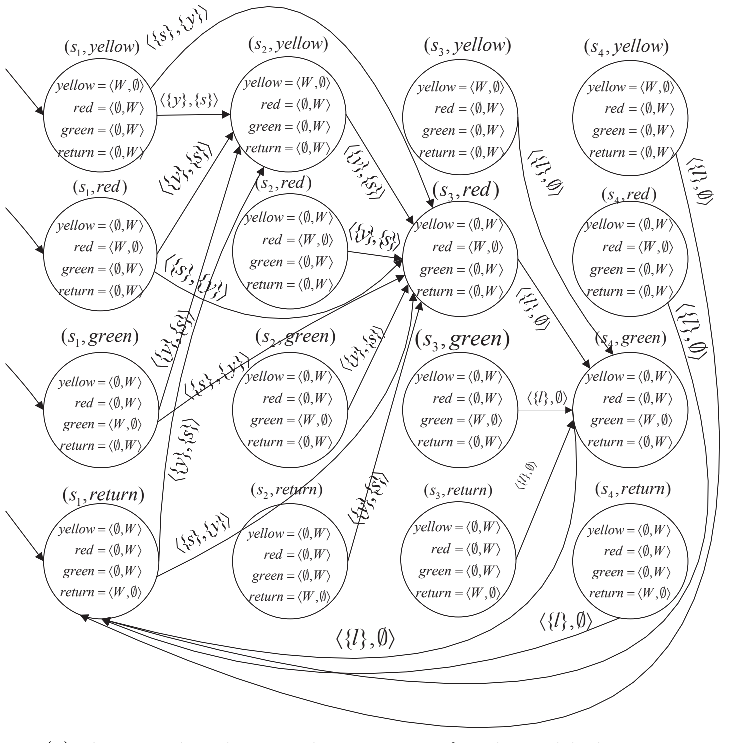
We illustrate the translation using the example of a red light system. Required by the root feature *light*, the red lights can switch with red and green. A variability of the system is that whether the light will show the yellow when switching between red and green (required by the feature *skip_yellow*) or not (required by the *yellow*). The BFTS and FD for the red light system are depicted in Figures 7(a) and 7(b). Applying the translation in Definition 10, we get



(a) BFTS for the red light system



(b) FD for the red light system



(c) the translated $\mathcal{X}$Kripke structure for the red light system

Figure 7: Modeling of the red light system.

the $\mathcal{X}$Kripke structure in Figure 7(c), which has 4×4 states with 4 labelled atomic propositions and 4×5 transitions.

Furthermore, the following definition gives the translation from ACTL to CTL [10]

Definition 11. [10] The mapping $ks' : ACTL \rightarrow CTL$ is inductively defined by:

$$\begin{aligned}
ks'(true) &= true,\\
ks'(false) &= false,\\
ks'(\neg\varphi) &= \neg ks'(\varphi),\\
ks'(\varphi \wedge \varphi') &= ks'(\varphi) \wedge ks'(\varphi'),\\
ks'(\varphi \vee \varphi') &= ks'(\varphi) \vee ks'(\varphi'),\\
ks'([a]\varphi) &= AX(a \rightarrow ks'(\varphi)),\\
ks'(\langle a\rangle\varphi) &= EX(a \wedge ks'(\varphi)),\\
ks'(AF\varphi) &= \mu Q.ks'(\varphi) \vee (\bigwedge_{a\in Act} ks'([a]Q)),\\
ks'(EF\varphi) &= \mu Q.ks'(\varphi) \vee (\bigvee_{a\in Act} ks'(\langle a\rangle Q)),\\
ks'(AG\varphi) &= \upsilon Q.ks'(\varphi) \wedge (\bigwedge_{a\in Act} ks'([a]Q)),\\
ks'(EG\varphi) &= \upsilon Q.ks'(\varphi) \wedge (\bigvee_{a\in Act} ks'(\langle a\rangle Q)),\\
ks'(A(\varphi U\varphi')) &= \mu Q.ks'(\varphi') \vee (ks'(\varphi) \wedge \bigwedge_{a\in Act} ks'([a]Q)),\\
ks'(E(\varphi U\varphi')) &= \mu Q.ks'(\varphi') \vee (ks'(\varphi) \wedge \bigwedge_{a\in Act} ks'(\langle a\rangle Q)).
\end{aligned}$$

Table 1 presents some properties for the beverage machine example expressed in ACTL formulas and their corresponding CTL formulas translated through ks'. The ACTL formula $\langle display\_done\rangle true$, for instance, is translated to the CTL formula $EXdisplay$.

Model Checking Results with $\mathcal{X}$Chek. We implement the above procedures and use $\mathcal{X}$Chek to prove properties of the vending machine system, as described in Table 1.

For example, for the property $\neg AF\langle ring\_a\_tone\rangle true$ that checks whether the tone will ring after the drink is delivered. The equivalent CTL formula is $\neg AFring\_a\_tone$. The result of model checking this formula using $\mathcal{X}$Chek is shown in Figure 8. Since the value is $\langle\emptyset, \{r\}\rangle$, the property fails on any product that contains the feature r.

On the other hand, for the property $AF(\langle sugar\rangle true \rightarrow \langle pour\_sugar\rangle$ that ensures that if a customer wants sugar, then the request will be satisfied eventually. The equivalent CTL formula is $AF(sugar \rightarrow pour\_sugar)$. The result is $\langle\{v, b, c, t, f, r, d\}, \emptyset\rangle$, which means the property holds on any product built from the features $\{v, b, c, t, f, r, d\}$.

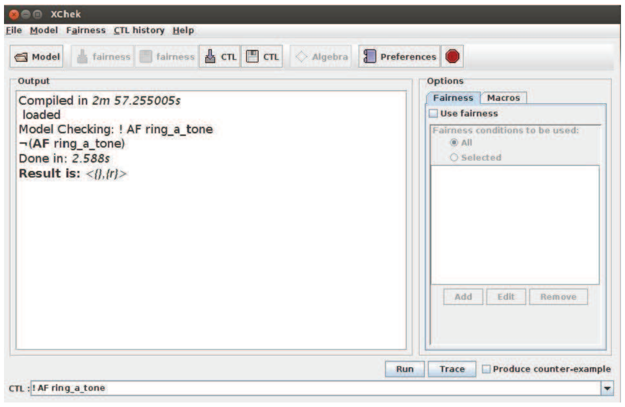
6. RELATED WORK

Modeling Variability for SPLs. A number of models for SPL have been proposed in the literature, which can be categorized into two classes. On the one hand, there are approaches that organize and structure the whole SPL process in terms of features. In this way, it is easy to trace the requirements of stakeholders to the software artifacts that provide the corresponding functionality [1]. On the other hand, there exist formal modeling formalisms for description of SPL behaviors, which can be used for the verification of temporal properties as what we do.

Most feature-oriented approaches are based on feature diagrams. Kang et al. introduce the notion of FD in [16]. Since then, various FD variants have been devised to make up for lack of precision and expressiveness [22]. Feature are

Table 1: Verification on The Vending Machine SPL.

ACTL Formula φ	CTL Formula φ'	Results
$AF(\langle tea\rangle true \vee \langle coffee\rangle true \rightarrow \langle cup\_taken\rangle true)$	$AF(tea \vee coffee \rightarrow cup\_taken)$	$\langle\{v,b,c,t,f,r,d\},\emptyset\rangle$
$AF(\langle sugar\rangle true \rightarrow \langle pour\_sugar\rangle true)$	$AF(sugar \rightarrow pour\_sugar)$	$\langle\{v,b,c,t,f,r,d\},\emptyset\rangle$
$\neg AF\langle display\_done\rangle true$	$\neg AF display\_done$	$\langle\emptyset,\{d\}\rangle$
$\neg AF\langle ring\_a\_tone\rangle true$	$\neg AF ring\_a\_tone$	$\langle\emptyset,\{r\}\rangle$
$\neg AF(\langle sugar\rangle true \vee \langle no\_sugar\rangle true)$	$\neg AF(sugar \vee no\_sugar)$	$\langle\emptyset,\{v,b,c,t,f,r,d\}\rangle$



```
Compiled in 2m 57.255005s
 loaded
Model Checking: ! AF ring_a_tone
¬(AF ring_a_tone)
Done in: 2.588s
Result is: <{},{r}>
```

[ ] Use fairness
Fairness conditions to be used: [✓ All] Selected
[ ] Produce counter-example
CTL : ! AF ring_a_tone

Figure 8: Model checking result of the property $\neg AF\langle ring\_a\_tone\rangle true$.

used to distinguish products for a SPL. Ideally, a software manufacture can generate a software product based on the feature selection from customers. In this case, the authors leverage viewpoints to develop a particular SPL model according to stakeholders' personal views [18] and propose a model for viewpoints integration [19]. These approaches, however, focus on defining features existing in an SPL, not behaviors.

Modal Transition Systems are one of the most popular formal frameworks for modeling SPL behaviors [13]. In an MTS, transitions may be possible or necessary. Based on MTSs, there are proposals such as [12] to extend them to overcome the low distinguishing power of MTSs when modeling variation points. However, they ignore relations between features and behaviors; therefore, invalid products may derived from their approaches. In [8], Clssen et al. propose feature transition systems (FTSs) to relate features and behaviors. Every transition is labeled by a feature set. A model checing tool SNIP [7] has been developed to provide support for verifying FTS against LTL properties. But FTSs do not support modeling uncertainties; thus, their approach is not applicable for verifying partial product line designs.

Formal Verification for SPLs. Recently, a number of researchers have observed the need for verification of SPLs. In [17], Larsen defines a behavioral variability model for product line development based on modal I/O automata, aimed at verifying the error fee combinability of interfaces. Therefore, they do not focus on verifying product specific functional properties. In [9], the modeling checking approach is based on UML. By mapping feature models to concise representations of variability in different kinds of other models a product can be derived from the model of the SPL. However, the verification does not refer to temporal logic properties. Asirellis et al. formalize properties in MHML logic, and consider managing variabilities in products derivation [2]; they also develop a tool VMC [23] for model checking with MTSs. However, they don't consider relation between features and actions, which is typically required by software product line designers.

Multi-Valued Model Checking. Multi-valued models are widely used for reasoning about systems with incomplete information. In [4], Chechik et al. propose quasi-boolean multi-valued logics for reasoning about systems with uncertainty and inconsistency. A symbolic model checker χchek is also implemented to support multi-valued model checking [5]. Salay et al. propose an approach for expressing unknown information using partial models [21] and apply it to management of requirements uncertainty in [20]. But they have not been directly applied to model checking software product lines.

7. CONCLUSION

In this paper, we consider model checking partial software product line designs, i.e., the incomplete designs in the early stage of software development, where the design decisions for a feature may be unknown. This enables detecting design errors earlier, reducing the cost of later development of final products. We first propose bilattice-based feature transitions systems (BFTSs) for modeling partial software product line designs, which support description of uncertainty and preserve features as a first class notion. We then express system behavioral properties using ACTL formulas and define its semantics over BFTSs. Finally, we provide the procedures that translate BFTSs and ACTL formulas to the inputs of the existing symbolic model checker χChek. We implement our approach and illustrate its effectiveness on a benchmark from literature. A major limitation of our work is lack of high-level modeling language for description of software product line designs; we would like to address this problem in future by extending fPromela [7] – the input language of SNIP – for specifying incomplete designs of features, and provide model checking approach for the extended language.

8. ACKNOWLEDGMENT

We thank Mingyu Huang for his help with the experiments. This work has been financially supported by the National Natural Science Foundation of China under Grant No.61170043 and Grant No. 61202002, and the National Basic Research Program of China under Grant No.2014CB744904.

9. REFERENCES

[1] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-Oriented Software Product Lines*. Springer, 2013.

[2] P. Asirelli, M. H. Ter Beek, S. Gnesi, and A. Fantechi. Formal description of variability in product families. In *Software Product Line Conference*, volume 11, pages 130–139, 2011.

[3] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. Multi-valued symbolic model-checking. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(4):371–408, 2003.

[4] M. Chechik, S. Easterbrook, and V. Petrovykh. Model-checking over multi-valued logics. In *FME 2001: Formal Methods for Increasing Software Productivity*, pages 72–98. Springer, 2001.

[5] M. Chechik, A. Gurfinkel, and B. Devereux. χchek: a multi-valued model-checker. In *Computer Aided Verification*, pages 505–509. Springer, 2002.

[6] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[7] A. Classen, M. Cordy, P. Heymans, A. Legay, and P.-Y. Schobbens. Model checking software product lines with snip. *International Journal on Software Tools for Technology Transfer*, 14(5):589–612, 2012.

[8] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, and J.-F. Raskin. Model checking lots of systems: efficient verification of temporal properties in software product lines. In *ICSE*, pages 335–344. ACM, 2010.

[9] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *Generative Programming and Component Engineering*, pages 422–437. Springer, 2005.

[10] R. De Nicola and F. Vaandrager. Action versus state based logics for transition systems. In *Semantics of Systems of Concurrent Processes*, pages 407–419. Springer, 1990.

[11] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *Journal of the ACM (JACM)*, 42(2):458–487, 1995.

[12] A. Fantechi and S. Gnesi. Formal modeling for product families engineering. In *Software Product Line Conference*, pages 193–202. IEEE, 2008.

[13] D. Fischbein, S. Uchitel, and V. Braberman. A foundation for behavioural conformance in software product line architectures. In *Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, pages 39–48. ACM, 2006.

[14] M. L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational intelligence*, 4(3):265–316, 1988.

[15] P. Godefroid and R. Jagadeesan. On the expressiveness of 3-valued models. In *Verification, Model Checking, and Abstract Interpretation*, pages 206–222. Springer, 2003.

[16] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.

[17] K. G. Larsen, U. Nyman, and A. Wasowski. Modal i/o automata for interface and product line theories. In *Programming Languages and Systems*, pages 64–79. Springer, 2007.

[18] M. Mannion, J. Savolainen, and T. Asikainen. Viewpoint-oriented variability modeling. In *Computer Software and Applications Conference*, volume 1, pages 67–72. IEEE, 2009.

[19] N. Niu, J. Savolainen, and Y. Yu. Variability modeling for product line viewpoints integration. In *Computer Software and Applications Conference*, pages 337–346. IEEE, 2010.

[20] R. Salay, M. Chechik, J. Horkoff, and A. Di Sandro. Managing requirements uncertainty with partial models. *Requirements Engineering*, 18(2):107–128, 2013.

[21] R. Salay, M. Famelis, and M. Chechik. Language independent refinement using partial modeling. In *Fundamental Approaches to Software Engineering*, pages 224–239. Springer, 2012.

[22] P. Schobbens, P. Heymans, and J.-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *Requirements Engineering*, pages 139–148. IEEE, 2006.

[23] M. H. ter Beek, F. Mazzanti, and A. Sulova. Vmc: A tool for product variability analysis. In *FM 2012: Formal Methods*, pages 450–454. Springer, 2012.

[24] D. M. Weiss. The product line hall of fame. In *Software Product Line Conference*, page 395, 2008.