VMC: A Tool for Product Variability Analysis

Maurice H. ter Beek, Franco Mazzanti, and Aldi Sulova

Istituto di Scienza e Tecnologie dell'Informazione, ISTI–CNR, Pisa, Italy

Abstract. We present VMC, a tool for the modeling and analysis of variability in product lines. It accepts a product family specified as a modal transition system, possibly with additional variability constraints, after which it can automatically generate all the family's valid products, visualize the family/products as modal/labeled transition systems, and efficiently model check properties expressed in an action- and state-based branching-time temporal logic over products and families alike.

1 Introduction

Product Line Engineering (PLE) is a paradigm for developing a variety of related products from a common product family. Commonality and variability are defined in terms of *features* and managing variability is about identifying variation points in a family design and deciding which combinations are valid products.

In [7], Modal Transition Systems (MTSs) were recognized as a formal method to describe in a compact way the possible operational behavior of all products of a product family. An MTS is a Labeled Transition System (LTS) distinguishing *optional* (may) and *mandatory* (must) transitions. The standard way to derive products (which become LTSs) from an MTS modeling a product family is to include all its (reachable) must transitions and a subset of its (reachable) may transitions; each selection is a product. However, MTSs are incapable of modeling all common variability constraints. The solution chosen in [1,2] is to add a set of constraints to the MTS to define which derivable products are to be considered valid ones. In particular, an appropriate variability and action-based temporal logic to formalize these constraints is defined in [1] and an algorithm to derive only (and possibly all) LTSs describing valid products is defined in [2].

We introduce an experimental tool we developed to implement this solution: the *Variability Model Checker* VMC. We guide the reader through a case study, a family of coffee machines from [1,2] with the following informal requirements:

1. *Initially, a coin must be inserted: either a euro, exclusively for European products, or a dollar, exclusively for Canadian products*;
2. *After inserting a coin, the user has to choose whether (s)he wants sugar, by pressing one of two buttons, after which (s)he may select a beverage*;
3. *The choice of beverage (coffee, tea, cappuccino) varies, but all products must offer coffee while only European products may offer cappuccino*;
4. *Optionally, a ringtone may be rung after delivering a beverage. However, a ringtone must be rung in all products offering cappuccino*;

D. Giannakopoulou and D. Méry (Eds.): FM 2012, LNCS 7436, pp. 450–454, 2012.

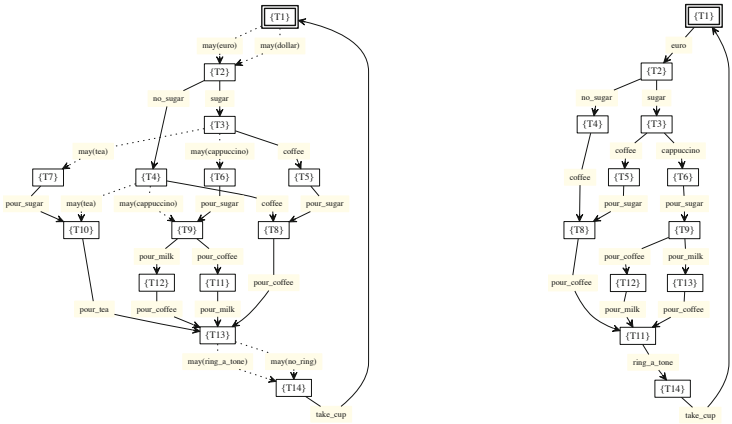

Fig. 1. MTS of coffee machine family (l) and apparently valid European product (r) as generated by VMC; dashed edges labeled may(·) are may transitions, the others must

2 Encoding and Analyzing Product Families in VMC

VMC (`http://fmtlab.isti.cnr.it/vmc/`), beyond interactively exploring an MTS, model checking properties over an MTS, and visualizing the interactive explanations of a verification result, furthermore allows the generation of valid products (according to the given constraints) of an MTS describing a product family and the verification of properties over each valid product.

VMC takes as input the textual encoding of an MTS and a set of constraints of the form ALTernative, EXCludes, REQuires and IFF (a shorthand for bilateral REQs), thus hiding their logic formalization given in [1]. The distinction among may and must transitions is encoded in the resulting LTS by structuring action labels corresponding to may transitions as `may(·)` (i.e., typed actions). We model all valid product behavior of the coffee machine family by the MTS of Fig. 1(l), whose textual representation and associated set of constraints are as follows:

```
T1 = may(euro).T2 + may(dollar).T2
T2 = sugar.T3 + no_sugar.T4
T3 = coffee.T5 + may(cappuccino).T6 + may(tea).T7
T4 = coffee.T8 + may(cappuccino).T9 + may(tea).T10
T5 = pour_sugar.T8
T6 = pour_sugar.T9
T7 = pour_sugar.T10
T8 = pour_coffee.T13
T9 = pour_coffee.T11 + pour_milk.T12
T10 = pour_tea.T13
T11 = pour_milk.T13
T12 = pour_coffee.T13
T13 = may(ring_a_tone).T14
    + may(no_ring).T14
T14 = take_cup.T1
net SYS = T1
Constraints { euro ALT dollar
              dollar EXC cappuccino
              cappuccino REQ ring_a_tone
              ring_a_tone ALT no_ring }
```

The variability logic defined in [1] can be directly encoded in the logic accepted by VMC by considering the typed actions. This latter logic contains the classic box and diamond modal operators $[\,], \langle\,\rangle$, the classic existential and universal state operators E, A (quantifying over paths), and action-based versions of the CTL until operators W, U (resulting also in an action-based version of the 'eventually'

operator F). Using VMC it is thus possible to specify and verify properties which are definitely preserved in all products by checking them over the family MTS:

(1) *The MTS guarantees that if a euro or dollar action occurs, afterwards for all standardly derivable products it is eventually possible to reach action coffee*: $[may(euro)\ or\ may(dollar)]\ E\,[true\ \{not\ may(*)\}\,U\,\{coffee\}\ true]$

This formula prohibits a path leading to action coffee to contain *any* (i.e. $*$) may transition (beyond the initial one). Asked to check it over the MTS of Fig. 1(l), VMC reports it holds. It moreover offers the possibility to explain the result.

3 Generating and Analyzing Valid Products in VMC

VMC implements the algorithm defined in [2] to generate valid products derivable from an MTS when taking into account an associated set of constraints. The latter can also be used to specify (and analyze) specific subsets of the product set. Beyond generating valid products (LTSs), VMC allows browsing them, verifying whether they satisfy a certain property (logic formula) and investigating why a specific valid product does (not) satisfy the verified property. To do so, VMC allows to open for each product a new window with its textual encoding.

Suppose we generate all valid products in VMC and then check for each one:

(2) *If it is possible to obtain a sugared cappuccino, then also an unsugared one*: $(EF\,\langle sugar\rangle\,\langle cappuccino\rangle\,true)\ implies\ EF\,\langle no\_sugar\rangle\,\langle cappuccino\rangle\,true$

Property 2 does not hold for all valid products, revealing ambiguous constraints: the one of Fig. 1(r) satisfies all constraints but offers cappuccino only with sugar. To resolve such ambiguity, we refine the optional actions cappuccino and tea by explicitly distinguishing sugared and unsugared ones and extend the constraints:

```
Constraints {
  euro ALT dollar                              dollar EXC unsugared_cappuccino
  unsugared_cappuccino IFF sugared_cappuccino  unsugared_cappuccino REQ ring_a_tone
  unsugared_tea IFF sugared_tea                ring_a_tone ALT no_ring }
```

This case study `coffeemodel2.txt` is available in VMC as one of the examples. Next we check if all valid European products now offer both types of cappuccino:

$[euro]\ ((EF\,\langle sugared\_cappuccino\rangle\,true)\ and\ EF\,\langle unsugared\_cappuccino\rangle\,true)$

VMC then produces a table of 10 products (no longer containing that of Fig. 1(r)) listing which optional actions they contain and whether they satisfy the formula.

4 VMC and Related Tools

VMC's core contains a command-line version of the model checker and a product generation procedure, both stand-alone executables in Ada (easy to compile for Windows|Linux|Solaris|Mac) wrapped with CGI scripts handled by a web server, facilitating an html-oriented GUI and integration with graph drawing tools. Its development is ongoing, a prototype for academic purposes is freely usable online (`fmtlab.isti.cnr.it/vmc`) and its executables are available upon request.

The current version of VMC is not targeted to very large systems. Its main limitation, however, lies in generating the model from its input language, while its on-the-fly verification engine and advanced explanation techniques are those of the highly optimized family of on-the-fly model checkers developed at ISTI–CNR over the last decades for verifying formulae in an action- and state-based branching-time temporal logic derived from the CTL family of logics, such as FMC [8], UMC [3] and CMC [6]. Their on-the-fly nature means that in general not the whole state space needs to be generated and explored. This feature improves performance and allows to deal with infinite-state systems.

We are aware of two other tools dealing with verification of product families.

MTSA [5] is a prototype, built on top of the LTS Analyser LTSA, for the analysis of MTSs specified in an extension of the process algebra FSP (Finite State Processes). MTSA allows 3-valued FLTL (Fluent LTL) model checking of MTSs by reducing the verification to two FLTL model-checking runs on LTSs.

SNIP [4] is a model checker for PLs modeled as FTSs (Featured Transition Systems) specified in a language based on that of SPIN. Features must be declared in the Text-based Variability Language TVL and are taken into account by SNIP's explicit-state model-checking algorithm for the verification of properties expressed in fLTL (feature LTL) interpreted over FTSs, e.g. to verify a property over only a subset of a family's valid products. Exhaustive model-checking algorithms (continuing the search after a violation is found) moreover allow to verify all products of a family at once and to output all products violating a property. SNIP treats features as first-class citizens, has built-in support for feature diagrams, and it implements model-checking algorithms tailored for PLs.

Acknowledgment. We thank our colleagues Patrizia Asirelli, Alessandro Fantechi and Stefania Gnesi for their contributions to the research that has led to VMC.

References

1. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: A Logical Framework to Deal with Variability. In: Méry, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 43–58. Springer, Heidelberg (2010)
2. Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: Formal Description of Variability in Product Families. In: SPLC 2011, pp. 130–139. IEEE (2011)
3. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: A state/event-based model-checking approach for the analysis of abstract system properties. Sci. Comput. Program. 76(2), 119–135 (2011)
4. Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.-Y.: Model Checking Software Product Lines with SNIP. To appear in Int. J. Softw. Tools Technol. Transfer (2012)
5. D'Ippolito, N., Fischbein, D., Chechik, M., Uchitel, S.: MTSA: The Modal Transition System Analyser. In: ASE 2008, pp. 475–476. IEEE (2008)

6. Fantechi, A., Gnesi, S., Lapadula, A., Mazzanti, F., Pugliese, R., Tiezzi, F.: A Logical Verification Methodology for Service-Oriented Computing. ACM Trans. Softw. Eng. Methodol. 21(3), article 16, 1–46 (2012)
7. Fischbein, D., Uchitel, S., Braberman, V.A.: A foundation for behavioural conformance in software product line architectures. In: ROSATEA 2006, pp. 39–48. ACM (2006)
8. Gnesi, S., Mazzanti, F.: On the Fly Verification of Networks of Automata. In: PDPTA 1999, pp. 1040–1046. CSREA Press (1999)