ARTICLE IN PRESS
JLAMP:69

Journal of Logical and Algebraic Methods in Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Incremental model checking of delta-oriented software product lines [☆]

Malte Lochau [a,*], Stephan Mennicke [b], Hauke Baller [b], Lars Ribbeck [b]

[a] TU Darmstadt, Real-Time Systems Lab, Germany
[b] TU Braunschweig, Institute for Programming and Reactive Systems, Germany

A R T I C L E   I N F O

Article history:
Received 4 September 2014
Received in revised form 27 August 2015
Accepted 15 September 2015
Available online xxxx

Keywords:
Variability modeling
Operational semantics
Model checking

A B S T R A C T

We propose DeltaCCS, a delta-oriented extension to Milner's process calculus CCS to formalize behavioral variability in software product line specifications in a modular way. In DeltaCCS, predefined change directives are applied to core process semantics by overriding the CCS term rewriting rule in a determined way. On this basis, behavioral properties expressed in the Modal μ-Calculus are verifiable for entire product-line specifications both product-by-product as well as in a family-based manner as usual. To overcome potential scalability limitations of those existing strategies, we propose a novel approach for incremental model checking of product lines. Therefore, variability-aware congruence notions and a respective normal form for DeltaCCS specifications allow for a rigorous local reasoning on the preservation of behavioral properties after varying CCS specifications. We present a prototypical DeltaCCS model checker implementation based on MAUDE and provide evaluation results obtained from various experiments concerning efficiency trade-offs compared to existing approaches.

1. Introduction

Modern software-intensive systems tend to exhibit more and more diversity, e.g., by means of an ever-growing number of configuration options for tailoring those systems to specific customers' needs. To cope with the additional complexity introduced by the inherent *variability* in software system implementations nowadays, software product-line engineering has been proposed as a comprehensive methodology for efficiently developing families of similar software *variants* upon a common core product [1]. A software product line, therefore, explicitly captures the commonality and variability among the different members of a product family by means of their supported *features*. Each feature denotes a configuration option, i.e., a variable product characteristic within the problem domain, being relevant for some customer/user. In addition, each feature corresponds to composable engineering artifacts within the solution space which allows for an automated assembling of respective product variant implementations according to a feature selection made by the customer. This concept enables a fine-grained *reuse* of shared feature artifacts among the different variants throughout all development phases and levels of

[☆] This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution and under DFG Grant GO-671/6-2.
[*] Corresponding author.
E-mail addresses: malte.lochau@es.tu-darmstadt.de (M. Lochau), mennicke@ips.cs.tu-bs.de (S. Mennicke), baller@ips.cs.tu-bs.de (H. Baller), l.ribbeck@tu-bs.de (L. Ribbeck).

http://dx.doi.org/10.1016/j.jlamp.2015.09.004
2352-2208/© 2015 Published by Elsevier Inc.

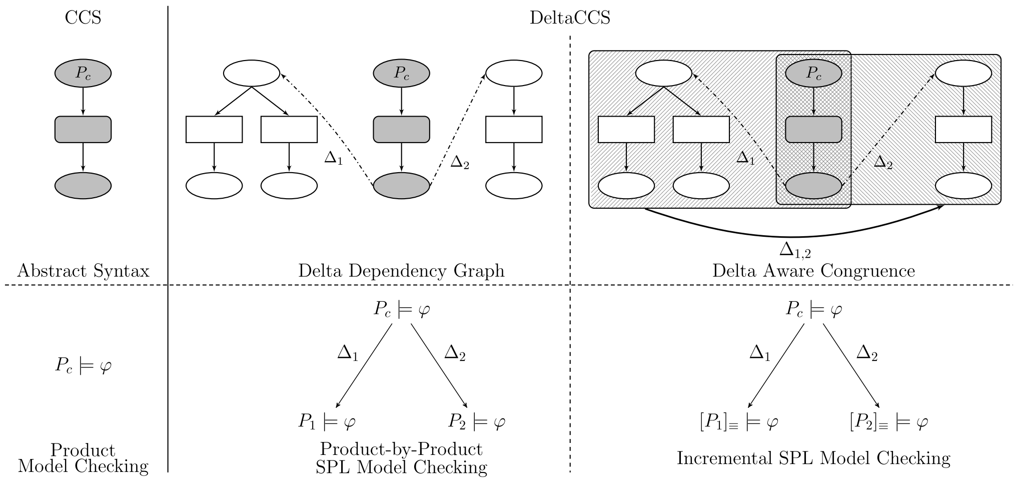

Fig. 1. High-level concept of delta-oriented product-line model checking.

abstraction. Thus, adding explicit *variability* capabilities to existing modeling and programming languages constitutes a key concept of product-line engineering [2].

Since its initial proposal, software product-line technology has found its way into industrial practice in various application domains, including mission-critical and even safety-critical systems.[1] However, for software product-lines to become fully established in practice, not only the development principles, but also the methods for quality assurance such as testing [3] and formal verification [4] have to be adapted in order to become applicable to families of software products, rather than being performed product-by-product which contradicts SPL philosophy. In particular, to also benefit from the concepts of systematic artifact reuse and managed variability during quality assurance, the additional reasoning required, e.g., to verify correctness properties for all members of a product family has to be conducted in a *variability-aware* manner [5]. To this end, various promising approaches have been proposed in the literature to define core calculi that gather the essence of variability in a formal way [6–16]. However, our recently gained experiences in applying those techniques to industrial case studies [17,18] from the automation engineering domain have shown that existing approaches may potentially show at least one of the following two deficiencies.

- They require a so-called 150% specification, superimposing all possible variants of the product line into one specification [2]. Thereupon, variability is emulated by adapting existing and/or adding new language constructs, e.g., selection/projection of variability parts [9], (guarded) choice among variable parts [13,11,15], and modal refinement of variable parts [12,14,7]. Although many advances have been made in this field, modeling and analyzing 150% product-line representations potentially become intractable for large-scale product lines due to the additional computational overhead, e.g., caused by handling variability annotations.
- They concentrate on phenomena arising from structural/syntactical variability [6,2,11,8], whereas the behavioral impact of those variations is out of scope. Thus, concepts for a systematic propagation of behavioral properties established for one variant also to other variants are missing.

To tackle these open issues, we apply a novel calculus for behavioral variability, called DeltaCCS introduced in a previous paper [19] to serve as a basis for modular specification and incremental verification of temporal behavioral properties for entire software product lines. DeltaCCS extends Milner's process calculus CCS [20] by a modular variability concept that adopts the principles of delta modeling [8] by separating a core process definition from change directives thereon, so-called deltas, that alter the term rewriting semantics of the core process in a determined way.

The overall concept of our delta-oriented product-line model checking framework is illustrated in Fig. 1. Our proposed representation of variability consists of collections $\Delta_1, \Delta_2, \ldots$ of CCS deltas. When applied to the abstract syntax representation of a core process term P_c, a CCS delta alters designated sub-processes of P_c in a determined way to yield arbitrarily fine-grained behavioral changes. We make use of a dependency graph representation of CCS terms augmented with CCS deltas to provide a rigorous definition and efficient detection mechanism of potential conflicts between pairs Δ_1, Δ_2 of CCS Deltas at the syntactic level [8]. At the semantic level, behavioral variability in DeltaCCS is not emulated by an a priori *resolution* of variation points as done in 150% specifications [13,15,11], but rather by *changing* the rewriting of process terms on-the-fly by overriding the CCS recursion rule. This variability mechanism enables a precise propagation of arbitrary structural variations of core processes onto the semantic level. Thereupon, we define delta-aware congruence $[P]_\equiv$ and a

[1] Cf. SPLC Hall of Fame, http://splc.net/fame.html.