Abstract—A software product line is a set of program variants, typically generated from a common code base. Feature models describe variability in product lines by documenting features and their valid combinations. In product-line engineering, we need to reason about variability and program variants for many different tasks. For example, given a feature model, we might want to determine the number of all valid feature combinations or compute specific feature combinations for testing. However, we found that contemporary reasoning approaches can only reason about feature combinations, not about program variants, because they do not take abstract features into account. Abstract features are features used to structure a feature model that, however, do not have any impact at implementation level. Using existing feature-model reasoning mechanisms for program variants leads to incorrect results. Hence, although abstract features represent domain decisions that do not affect the generation of a program variant. We raise awareness of the problem of abstract features for different kinds of analyses on feature models. We argue that, in order to reason about program variants, abstract features should be made explicit in feature models. We present a technique based on propositional formulas that enables to reason about program variants rather than feature combinations. In practice, our technique can save effort that is caused by considering the same program variant multiple times, for example, in product-line testing.

Keywords—Software product lines, program families, feature modeling, feature model, automated analyses.

I. INTRODUCTION

A software product line is a set of software-intensive systems (program variants) that share common code artifacts [1]. The overall idea is to efficiently develop similar programs simultaneously, by systematically reusing development artifacts. Program variants are distinguished in terms of features, which are prominent or distinctive user-visible aspects, qualities, or characteristics of a software system [2]. Two variants may have several features in common and differ in other features. In particular, we focus on product-line engineering methods, in which program variants can be generated from a common implementation by specifying a selection of features. This includes implementation techniques, such as conditional compilation [3], plug-ins and frameworks, or advanced programming-language mechanisms with aspects [4], feature modules [5], [6], or delta modules [7].

In general, not all combinations of features are useful and result in meaningful program variants. For instance, there might be mutually exclusive features or features that require other features. A variability model specifies all valid feature combinations and thus the program variants that can be generated. A common form of variability models are feature models [2], [8], [9]. Feature models can have several representations, e.g., a propositional formula in which each feature belongs to a variable and the formula evaluates to true for all valid combinations [10].

We noticed that not all features in typical feature models are used to distinguish program variants. Some are only used to structure the model and selecting or eliminating them does not make any difference in the generated variant code. We denote such features as abstract features. Due to abstract features, the set of feature combinations and the set of program variants are not equivalent. Multiple valid feature combinations result in the same generated program variant.

There are numerous approaches to reason about valid feature combinations in feature models [9]. However, there are also many interesting questions, for which we want to reason about program variants. For example, for combinatoric testing [11], type checking [12]–[15], or verification [16], we want to select certain sets of program variants or need to reason about the relationship of features in program variants. Similarly, for reasoning about non-functional properties of program variants [17], abstract features are not relevant, but can increase measurement effort significantly. Finally, deleting an abstract feature from the feature model does not change possible program variants and may hence be considered as feature-model refactoring [18]. Contemporary automated analyses of feature models can be used to reason about valid combinations of features, but not to reason about program variants. When using automated reasoning nonetheless, the results are at least inaccurate or inefficient.

As a consequence, we distinguish two semantics: (a) the semantics of feature models as known from literature [19], describing the valid combinations of features and (b) the semantics of program families describing distinct program variants of a product line. According to our experience, both are needed and complementary to each other. However, in order to not redevelop all reasoning mechanisms again for the program-families semantics, we provide a mechanism to translate the program-families semantics in a way that existing reasoning mechanisms for the feature-model seman-