Requirements Eng (2016) 21:185–208
DOI 10.1007/s00766-014-0216-9

ORIGINAL ARTICLE

Attribute-based variability in feature models

Ahmet Serkan Karataş · Halit Oğuztüzün

Received: 17 February 2014 / Accepted: 27 October 2014 / Published online: 6 November 2014

Abstract Extended feature models enable the expression of complex cross-tree constraints involving feature attributes. The inclusion of attributes in cross-tree relations not only enriches the constraints, but also engenders an extended type of variability that involves attributes. In this article, we elaborate on the effects of this new variability type on feature models. We start by analyzing the nature of the variability involving attributes and extend the definitions of the configuration and the product to suit the emerging requirements. Next, we propose classifications for the features, configurations, and products to identify and formalize the ramifications that arise due to the new type of variability. Then, we provide a semantic foundation grounded on constraint satisfaction for our proposal. We introduce an ordering relation between configurations and show that the set of all the configurations represented by a feature model forms a semilattice. This is followed by a demonstration of how the feature model analyses will be affected using illustrative examples selected from existing and novel analysis operations. Finally, we summarize our experiences, gained from a commercial research and development project that employs an extended feature model.

Keywords Software product lines · Extended feature models · Variability management · Variability involving attributes

A. S. Karataş (✉) · H. Oğuztüzün
Department of Computer Engineering, Middle East Technical University, Ankara, Turkey
e-mail: karatas@ceng.metu.edu.tr

H. Oğuztüzün
e-mail: oguztuzun@ceng.metu.edu.tr

1 Introduction

Over the years, the idea of building platforms to develop product families has been realized successfully in many areas of engineering. The benefits offered by mass customization such as a reduced development cost, enhanced quality of the products, decreased maintenance cost and simplified project management have resulted in a growing tendency toward the use of product lines [40]. Within recent decades, as software products and software-intensive systems became more complex, software researchers and practitioners inspired by the success stories initiated the adoption of product lines within the realm of software engineering. Since then, Software Product Lines (SPLs) have attracted increasing attention in both academic and industrial communities.

SPLs offer effective strategies to develop families of products. The members of a product family share common concepts that connect them to the family and include varying properties that make them distinguishable products. Thus, capturing commonality and variability in SPLs is a key issue, and the success of a product family is highly dependent on effective variability management [8]. Feature modeling [26] has proven to be an effective activity for modeling and managing commonality and variability in product families. Since the introduction of feature models, important extensions have been devised in response to growing expectations, and consequently, feature models have become more expressive.

One of the major extensions devised for feature models is the introduction of feature attributes. These provide extra information about the features in terms of measurable characteristics and can figure in complex cross-tree relations. The inclusion of attributes in cross-tree relations enriches the constraints, but introduces complications that are addressed in this article.

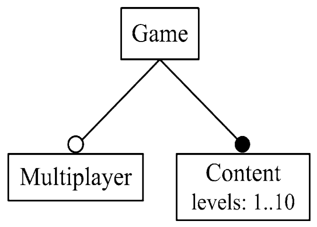


Fig. 1 The feature model for our hypothetical computer game family

A recent proposal [31] enabled the automated analyses of extended feature models that can include complex cross-tree relations involving attributes. This work also pointed to some of the complications that will arise. For instance, with the inclusion of attributes in cross-tree relations, a new variability type concerning attributes and their values has arisen. With this extended variability, traditional configuration and product specifications referring only to the sets of features to be included or excluded are no longer able to specify a product or a configuration precisely as they do not refer to the attributes and their values.

As a hypothetical scenario, consider a software company that designs computer games. The company wants to market different versions of a game, in which some versions will include multiplayer capability and others will not, and also the levels included in a game package may differ. Thus, the company builds the extremely simplified extended feature model depicted in Fig. 1.

First, it is assumed that only some of the levels, say 1 through 4, are suitable for multi-play. Clearly, an actual product derived from this model must obey all of the requirements. Thus, the customer should not encounter a product that is designed for multi-play that includes the levels 5 through 10. However, the existing product definition (as given in [6]) considers only the features to be included. Therefore, it is not possible to include information regarding the values to be assigned to the attributes of the included features. Obviously, this is not desirable since it can affect the analyses performed on the model and the products in an erroneous way. For instance, a product that contains the features *Game*, *Multiplayer*, and *Content* appears valid and marketable regarding the requirements imposed by the model when we consider only the features. However, if the value of the attribute *levels* is 6, this will be a faulty product that promises multi-play, but the level included is not suitable for multiple players.

In this article, we address this type of issue and elaborate on the consequences. To highlight our motivations, we also briefly present a commercial research and development (R&D) project involving an extended feature model for a mobile unit to be used for localization purposes in a wireless sensor network (WSN). We discuss the effects of the ideas proposed in this study through our experiences gained from the R&D project in order to provide an insight for the practitioners that would work with extended feature models including complex constraints that involve attributes.

The contributions of this article are summarized as follows. We extend the definitions for configuration and product specifications; this allows the feature models to meet the novel requirements that emerge due to the use of the extended variability. Using an example from an industrial R&D project on the localization software family for a WSN and a series of simpler additional examples, we elaborate on the ramifications that originate from these extensions. We delineate classifications for configurations and products to categorize configurations/products with common and different properties. We provide semantics that are grounded in constraint satisfaction, for the extended definitions and the new categories. We introduce an ordering relation between the configurations and products that allows engineers to benefit from the rich literature on partial orders and lattices. We discuss how existing analysis operations are affected by the extended definitions and classifications using selected illustrative operations and present reformulations and revised definitions for some of these operations. Finally, we propose a number of new analysis operations that the practitioners can make use of when working with such extended models.

The remainder of this article is organized as follows. Section 2 presents a brief background to basic, cardinality-based and extended feature models and the analysis efforts. In Sect. 3, we briefly present the R&D project. In Sect. 4, we discuss a new variability type that has emerged due to the inclusion of attributes in cross-tree relations and extend the definitions for the configuration and product to meet the emerging requirements. In Sect. 5, we present classifications for features, configurations, and products, establish a semantic foundation for our proposal, and propose an ordering relation between the configurations and products. In Sect. 6, we discuss how the existing analysis operations are affected by offering selected illustrative operation examples. In Sect. 7, we present our experience gained from the commercial R&D project to offer practitioners insight into putting the new concepts into practice. In Sect. 8, we discuss related work. In Sect. 9, we present a discussion on an alternative approach and the challenges ahead, and Sect. 10 contains our conclusions.

2 Background

2.1 Feature models

Since the introduction of feature models by Kang et al. [26] as part of Feature Oriented Domain Analysis, these models

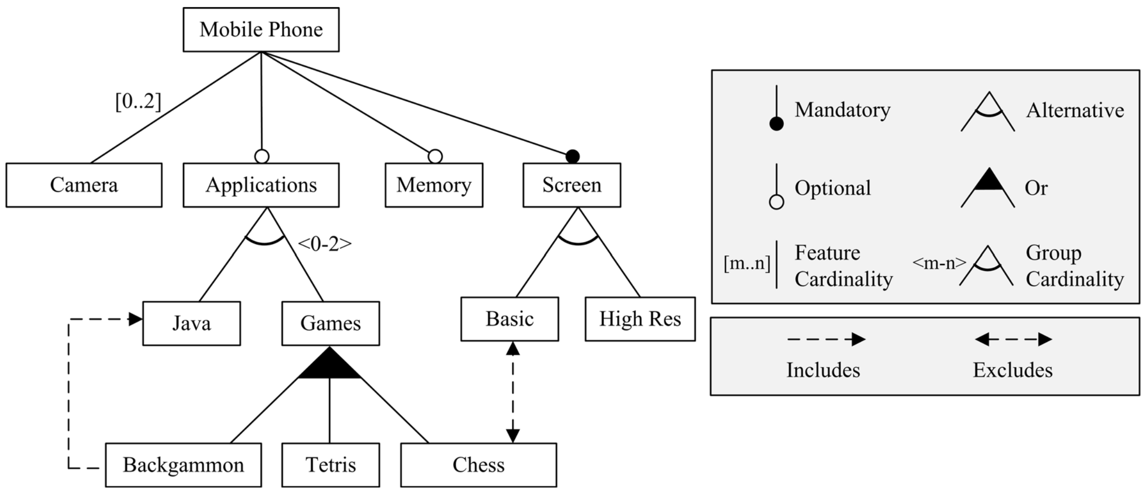


Fig. 2 A feature model for a mobile phone family

have been widely used in SPLs. We refer the reader to [43] for a literature survey on feature models; here we briefly discuss the basic, cardinality-based, and extended feature models.

A feature is a distinguishable characteristic of a concept (e.g., system, component and context) that is relevant to some stakeholder of the concept [44]. A feature model is a hierarchically arranged set of features showing the relations defining the composition rules among these features, relations defining the cross-tree constraints, additional information such as the issues/decisions that record various trade-offs, and the rationale and justification for the feature selection [26]. Feature models are often represented using feature diagrams. For instance, Fig. 2 shows a feature diagram of a feature model for a mobile phone family.

There are two types of relations in feature models: decomposition relations defining the relation between a parent feature and its child features, and cross-tree relations that specify the constraints among arbitrary features.

If a feature is not included in a product, then none of its children can be included. If a feature is included in a product, then the decomposition relations determine whether its children will be included. If there is a *mandatory* relation between a feature and its parent, then the child feature is included in every product that includes the parent feature. For instance, the feature *Screen* will be included in every product that includes the feature *Mobile Phone*. If there is an *optional* relation between a feature and its parent, then the child feature may or may not be included in a product that includes the parent feature. For instance, some mobile phones that include the feature *Mobile Phone* may include the feature *Applications,* whereas other phones may not. When there is an *alternative* relation between a feature and a group of its children, then only one of the child features from the group will be incorporated into a product that includes the parent feature. For instance, either the feature *Basic* or the feature *High Res* (but not both) must be included in every product that includes the feature *Screen*. Finally, if there is an *or* relation between a feature and a group of its children, then a non-empty subset of the child features from the group must be incorporated into a product that includes the parent feature. For instance, when the feature *Games* are included, then one, two, or all three of the features *Backgammon*, *Tetris,* and *Chess* must also be included.

One of the major extensions to feature models has been the introduction of cardinality-based decomposition relations. Multiplicities such as *m–n* have been proposed for use as group cardinalities by Riebisch et al. [41], and by Czarnecki et al. [15] as feature and group cardinalities in decomposition relations. If there is a *feature cardinality* relation in the form *[m..n]* between a feature and its parent, then at least *m*, at most *n* clones of the child feature will be included in every product that includes the parent feature. For instance, zero, one, or two clones of the feature *Camera* will be included in every product that includes the feature *Mobile Phone*. When there is a *group cardinality* relation in the form <*m–n*> between a feature and a group of its children, then at least *m*, at most *n* of the child features from the group must be incorporated into a product that includes the parent feature. For instance, when the feature *Applications* is included, then none, one, or two of the features *Java* and *Games* must also be included.

If a feature *X requires* another feature *Y*, a product that includes *X* must also include *Y*. For instance, every product that includes *Backgammon* must also include *Java*. If a feature *X excludes* another feature *Y*, inclusion of *X* in a product implies the exclusion of *Y* from that product and vice versa. For instance, no product can include both the *Basic* and *Chess* features.

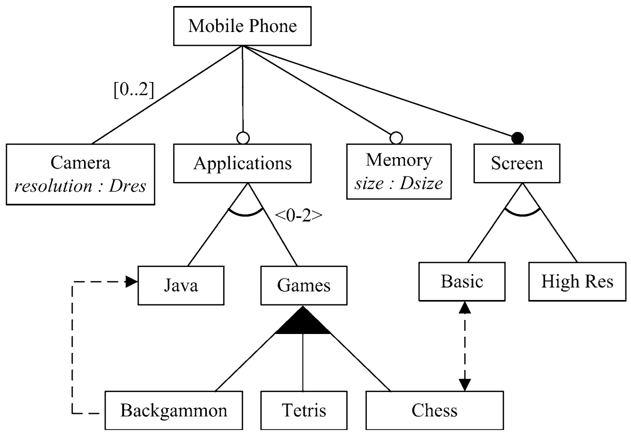


Fig. 3 An extended feature model for a mobile phone family

2.2 Extended feature models

The introduction of feature attributes is an important extension to feature models [13]. An attribute of a feature is any characteristic of the feature that can be measured. Every feature attribute belongs to a domain, the space of possible values from which the attribute takes its values [7]. For instance, the example diagram given in Fig. 3 represents an extended model since it has features with attributes. The feature *Memory* has an attribute named *size* (in gigabytes) with the domain *Dsize*, where we assume *Dsize* = {4, 8, 16, 32}, and the feature *Camera* has an attribute named *resolution* (in megapixels) with the domain *Dres*, where we assume *Dres* = {5, 8, 13}. Generally, the domain of an attribute can be a finite or an infinite set (e.g., a real interval); however, the current work is restricted to finite sets as domains.

Extended feature models enable cross-tree constraints involving attributes to be expressed. For instance, the model given in Fig. 3 may include a cross-tree constraint such as *Chess* requires *Memory.size* $\geq$8. These constraints can also be combined using propositional logic connectives to build more complex relations such as (*Chess* requires *Memory.size* ≥ 8) $\wedge$ (if $Camera_1.resolution > 4$ then $Camera_1$ requires *HighRes*). We refer the reader to [31] for a context-free grammar that precisely formalizes the syntax for relations in extended feature models.

2.3 Analysis of feature models

As feature modeling plays a key role in SPL engineering, it is an important but challenging task to analyze the feature models and reveal their characteristics. In the literature, a number of analysis operations have been reported that can extract information from feature models. Here we shall briefly discuss some of these operations; for a more detailed discussion of the definitions and practical uses of the analysis operations on feature models, we refer the reader to [6].

Besides feature models, the typical inputs to the analysis operations are configurations and products, which are defined as follows [6]:

Configuration: Given a feature model with a set of features *F*, a *configuration* is a 2-tuple of the form *(I, E)* such that *I* and *E* are two disjoint subsets of *F*, where *I* is the set of features to be included, and *E* is the set of features to be excluded. A configuration is called a *full configuration* If $I \cup E = F$, and a *partial configuration* otherwise.

Product: A *product* is a full configuration. For the sake of brevity, a product can be specified by only the set of included features.

A number of analysis operations can be used to extract information about the products represented by a given model. For instance, the analysis operation *valid product* can be used to answer the question whether a given product is valid with respect to a given model. Another analysis operation, namely *all products*, can be used to answer the question “which products are represented by this model?” This operation can be used to identify products that can be derived from the product line although they were not considered in the initial scope. There are other analysis operations to answer the questions related to the products represented by a given model, such as the number of products represented by this model and which products represented by this model include a given configuration.

Some analysis operations can be used to extract information about the features included in a model. For instance, the analysis operation *core features* finds the set of features that are included in all products represented by a model. This operation can be used to identify the features that should be supported by the core architecture. The analysis operation *variant features* can be used to find the set of features that are not included in all products. It is also possible to find the set of *dead features* (i.e., features that do not appear in any of the products), *conditionally dead features* (i.e., features that become dead under certain circumstances), or *false optional features* (i.e., features that are not modeled as mandatory despite being included in all the products). These operations can be used to detect the anomalies in the product line.

It is also possible to extract information about the model itself or its relations with other models using analysis operations. For instance, the analysis operation *void feature model* can be used to check if a feature model is void (i.e., represents no products) or not. Some metrics such as *homogeneity* (that indicates the degree to which a model is homogeneous), or *variability factor* (i.e., the ratio between the number of actual products represented and the number of potential products that can be represented) can be computed using analysis operations. A more homogeneous model would include a few features that are unique to products, and this information can be used to determine how closely the products are related to the family. Similarly, the variability factor can be used to understand the extent of the flexibility of the product line. It is also possible to discover whether two models are related (e.g., if one is a generalization of another) by using appropriate analysis operations; this relationship can be useful in understanding how a feature model has evolved over time.

In industrial applications, feature models can rapidly grow large; therefore, performing the analyses manually is a tedious and error-prone task. Hence, automated support for the analyses is extremely desirable. A number of proposals reported in the literature have provided for automated analyses by mapping feature models to a variety of other formalisms such as propositional logic, descriptive logic, and constraint programming [6].

A recent study by Karataş et al. [31] established the foundations for the constraint-programming-based analyses of the extended feature models that include complex cross-tree constraints involving attributes. The authors provided a mapping from such extended feature models to constraint logic programming over finite domains [CLP(FD)], which enables the use of CLP(FD) solvers for feature model analyses. This proposal allows the user to benefit from many powerful techniques and tools offered by the CLP community. The same authors proposed the use of global constraints provided by CLP systems in feature model analyses in [28]. However, from this enhancement, new issues emerged due to the inclusion of attributes in cross-tree constraints. An extended type of variability, *variability involving attributes*, discussed in Sect. 4, is responsible for these issues, and the effects of this new variability type will be elaborated in the following sections.

3 A software product line for a localization family in wireless sensor networks

WSNs are becoming more popular as low-cost sensors, microcontrollers, and radio frequency transceivers enter the markets. Localization in WSNs is a key technique that has many areas of application such as location service, monitoring, tracking, rescue, coverage, and routing [12]. There are various localization techniques reported in the literature. Cheng et al. [12] classify these techniques into two main categories. The first category includes techniques that adopt a centralized approach to compute the location of a target/source node and includes methods for inferring the location of a single target or the locations of multiple targets in WSNs or wireless binary sensor networks. The second category consists of the methods to be used when the node is to compute its self-location information. This category is further divided into two subcategories: range-free and range-based localization.

Range-free localization methods compute the location of a node without measuring the absolute distances between the unknown node and the beacon nodes, the nodes whose locations are known by the unknown node [12]. For instance, the *hop-count-based localization* method uses the average hop distances in the computations instead of the actual distances. Alternatively, the *pattern-matching localization* method relies on the usage of pattern-matching algorithms and a radio map, which is a database of radio signals received at selected locations, where the current observed radio signals are matched to the recorded values in the map in order to determine the location of the node.

Range-based localization techniques rely on the actual values (i.e., distances or angles) measured between the unknown node and the beacon nodes to infer the location of the unknown node. Cheng et al. [12] list four methods for range-based localization: *Received Signal Strength Indicator* (RSSI), *Time of Arrival* (ToA), *Time Difference of Arrival* (TDoA), and *Angle of Arrival* (AoA). The RSSI method uses the strength of the received radio signal to determine the distance between the unknown node and the beacon node. As radio signals decay on their voyage through the air, this method estimates the distance between the source and the receiver by measuring the decay in the signal. The ToA method records the signal propagation time to estimate the distance between the transmitter and the receiver; this includes calculations that use the speed of the signal. The TDoA method uses two different signals with different propagation speeds (e.g., radio frequency and ultrasound) in order to infer the distance using the delay between the arrival times of the two signals. The AoA method calculates the angle at which signals are received and uses simple geometrical relations to infer the position of the unknown node with respect to the beacon nodes.

When using a range-based localization technique, once a sufficient number of distances or angles between the unknown node and the beacon nodes are measured, the unknown node can compute its location information using different algorithms. For instance, when the distances are available, the *min–max algorithm*, the *maximum likelihood algorithm*, or the *trilateration algorithm* can be used [20], and when the angles are available, the *triangulation algorithm* can be executed [39].

KaVIS is a commercial R&D project funded by the Republic of Turkey, Ministry of Science, Industry and Technology (with grant number 603.TGSD.2012). KaVIS aims to use the location information of a mobile unit in a WSN in order to provide the user with context sensitive data.

KaVIS is planned to employ a range-based localization technique to infer the location of a mobile unit. However, there is no *"best range-based localization technique"* as each approach has a number of advantages and disadvantages when compared to the other range-based methods. For instance, RSSI is an inexpensive approach since it can work on low-cost chips, which come with ready-to-use features, and unlike some of the other techniques, it requires no additional hardware [12, 37, 50]. Moreover, it requires relatively low-energy input [12]. However, RSSI measurements highly suffer from noise and are very sensitive to environmental effects [50]. Consequently, RSSI provides less accurate results when compared to the other methods [39]. Similarly, ToA approaches are also relatively cheap as only low-cost chips with ready-to-use features are required [50]. On the other hand, it is very difficult to measure the time of the flight of the signal when the distance to be measured is extremely small in comparison with the propagation speed; thus, ToA methods produce highly noisy measurements and are over-sensitive to environmental effects [50]. The TDoA technique achieves high ranging accuracy at the expense of more energy consumption and the requirement of extra hardware that increases the cost of the solution [12]. The AoA technique achieves very high accuracy and precision even with a small number of beacons [39]. However, it is an expensive solution since an antenna array or multiple receivers are required by the nodes [50].

The algorithms used in the presence of distance measurements also have advantages and disadvantages compared with their alternatives [20]. For instance, the min–max algorithm is the most efficient algorithm with respect to computational complexity. However, the computational cost increases when the number of beacons to be considered increases. The trilateration algorithm also has a relatively low cost which is almost fixed since it uses only three beacons in its computations. However, the trilateration algorithm performs root operations and many divisions, which can significantly decrease the efficiency of the localization process if the microcontroller does not employ a mathematical coprocessor. Both min–max and trilateration algorithms produce significantly more accurate results than the maximum likelihood algorithm when there are only a few beacons to be considered. However, studies have shown that the maximum likelihood algorithm outperforms the other two when using six or more beacons.

Thus, it is not possible to tailor a single localization system that will achieve the best fit for different scenarios. For instance, when the budget is the most important issue, it is more attractive to build a localization system that uses an RSSI- or a ToA-based technique. When high localization accuracy is required, it is more desirable to employ a TDoA- or AoA-based system. Due to these considerations, KaVIS is planned to utilize a localization family that will be able to derive different products for different requirements.

The feature diagram that represents a simplified version of the feature model for the mobile unit family in KaVIS, MU, is shown in Fig. 4.

This model is an extended feature model since some of the features have attributes. The feature *Ultrasonic Receiver* has an attribute named *frequency* with the domain {22, 25, 40} in kHz, and the feature *Temperature Sensor* has an attribute named *accuracy* with the domain {0.5, 1.0, 1.5} in degrees Celsius.

4 Variability involving attributes

Considering the product P1 = {Mobile Unit, Method, TDoA, Algorithm, Min–Max, Hardware, RF Transceiver, Ultrasonic Receiver, Power Generator, RS232, MCU} derived from the feature model MU, given in Fig. 4; since the conventional product definition is built on specifying the included features, P1 does not give any information regarding the attributes. For instance, the feature *Ultrasonic Receiver* has an attribute *frequency*; however, P1 does not provide any information about the value(s) that can be assigned to this attribute in P1. Thus, it is possible to interpret this product specification as:

P1 = {…, Ultrasonic Receiver, …} where Ultrasonic Receiver.frequency ∈ {22, 25, 40}

This interpretation does not have any variability in terms of features; it is precise in terms of which features are included in the product. However, the same precision is not present for attributes; the value to be assigned to the attribute *frequency* has not been determined yet. Thus, there is no variability in terms of the features; however, there is still variability in terms of the values that will be assigned to the attributes of the included features. If we proceed to remove this variability, we can derive three different products from the product P1 as follows:

- $P1_1$ = {…,Ultrasonic Receiver,…} where Ultrasonic Receiver.frequency = 22
- $P1_2$ = {…,Ultrasonic Receiver,…} where Ultrasonic Receiver.frequency = 25
- $P1_3$ = {…,Ultrasonic Receiver,…} where Ultrasonic Receiver.frequency = 40

Although these three products include exactly the same set of features, they differ in the value assigned to the attribute *Ultrasonic Receiver.frequency*; therefore, they can be considered different products. Hence, removing the variability in terms of features would not be enough to achieve a fully specialized product, as the features

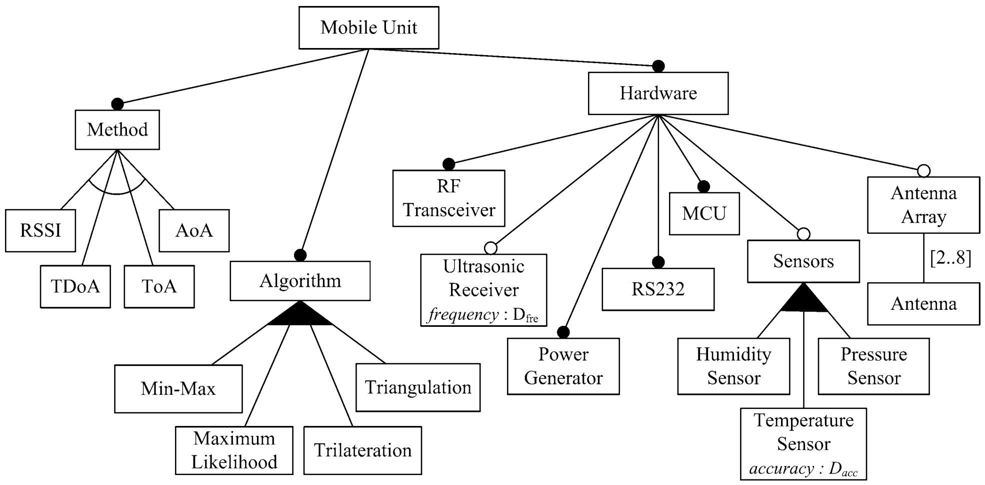


Fig. 4 Feature model MU for the KaVIS mobile unit family

themselves contain an extended type of variability, *variability involving attributes*.

As this example suggests, it is necessary to include information in the product definition regarding the values that can be assigned to the attributes that belong to the included features. However, including this information raises the question of how such sets of values will be determined. In the previous example, two cases were given: using the domain specified in the feature model for the attribute under question (i.e., {22, 25, 40}), or using a singleton that includes a value from the attribute's specified domain (e.g., {22}). However, the example below reveals that the choices are not limited to these two cases.

Assume that there is a product P2 derived from MU such that P2 = {Mobile Unit, Method, RSSI, Algorithm, Trilateration, Hardware, RF Transceiver, Power Generator, RS232, MCU, Sensors, Temperature Sensor}. In this product the feature *Temperature Sensor* has an attribute named *accuracy* with the specified domain {0.5, 1.0, 1.5}. Assume that there is a cross-tree constraint such that "*RSSI* requires the measurement errors of the *Temperature Sensor* to be at most 1.0 °C" (i.e., *RSSI* requires *Temperature Sensor.accuracy* $\leq$1.0). A simple analysis reveals that P2 will fail this constraint if 1.5 is assigned to *Temperature Sensor.accuracy*. Fortunately, it is possible to include this information in P2 as follows:

P2 = {…, Temperature Sensor, …} where Temperature Sensor.accuracy $\in$ {0.5, 1.0}

The first example presented in this section shows that information regarding the values that may be assigned to the attributes of the included features must be included in a product. The second example shows that, whenever possible, it is desirable to provide restrictions on the values that may be assigned to the attributes of the included features. Thus, the existent definitions for the *configuration* and *product* must be revised. However, first, the background must be prepared to provide the necessary information regarding attributes in the configurations and products.

Below we provide a set of values for each attribute of the included features in the configuration and product definitions. For example, let *a* be an attribute of a feature that is included in a product, *D* its domain, and *A* the set of values from which *a* is allowed to take a value. The cases we examined show that *A* can be equal to *D*, *A* can be a subset of *D*, or *A* can be a singleton that includes a member of *D*, inclusively. *A* cannot be empty, otherwise the value of *a* would be undefined in the product. Moreover, in order to be consistent with the feature model, *A* cannot include a value that *D* does not include. Based on these considerations we introduce the notion of an admissible domain.

Definition 1 *(A-domain)*: Given a feature model *M*, let *a* be an attribute of a feature in *M*, and *D* the domain of *a*, where *D* is a non-empty finite set. An *Admissible Domain* of *a* is a non-empty subset of *D*. □

For instance, the sets {0.5}, {1.0}, {0.5, 1.0}, {0.5, 1.5}, {0.5, 1.0, 1.5} are all valid A-domains for the attribute *Temperature Sensor.accuracy*, whereas {} and {0.5, 2.0} are not. Now the configuration and product definitions can be extended to suit the new requirements.

Definition 2 *(Configuration, Product)*: Given a feature model with a set of features *F*, a *configuration* is a triple of the form (*I, E,* Δ), where *(i) I* and *E* are two disjoint subsets of *F* (the set of features included and the set of features excluded, respectively), *(ii)* Δ is a set of pairs such that the first element (an attribute) $\in$ the second element (an A-domain) in the pairs, where every attribute that belongs to a feature in *I* appears exactly in one pair. A full configuration is called a *product* and can be represented by the pair (*I,* Δ). □

For instance, the products P1, P1$_1$, and P2 are represented as:

P1 = ({…,Ultrasonic Receiver,…}, {Ultrasonic Receiver.frequency ∈ {22, 25, 40}})
P1$_1$ = ({…,Ultrasonic Receiver,…}, {Ultrasonic Receiver.frequency = 22}})
P2 = ({…, Temperature Sensor,…}, {Temperature Sensor.accuracy ∈ {0.5, 1.0}})

The reader will notice that we used the notation *Ultrasonic Receiver.frequency = 22* instead of *Ultrasonic Receiver.frequency ∈ {22}* in the product P1$_1$. From now on we will use *attribute = value* as a shorthand for *attribute ∈ {value}* when the A-domain of an attribute is a singleton.

Since the A-domain of an attribute can be any nonempty subset of its specified domain this definition allows information to be provided about the restrictions on the values that the attribute can take. The specified domain of an attribute is totally acceptable as its A-domain in any configuration or product, which establishes full backward compatibility with the existing configuration and product definitions as provided in [6].

5 Full specialization versus partial specialization

Since the new variability type introduced in the previous section is defined at the level of attributes, some features may contain such variability, whereas others may not. For instance, consider the product P1. The feature *Ultrasonic Receiver* contains variability since any value from the A-domain {22, 25, 40} can be assigned to its attribute *frequency*; thus, it is possible to further specialize this feature. However, the features *MCU* or *RS232* cannot be further specialized as they do not contain variability involving attributes. Hence, it is possible to classify the features in a configuration as *fully specialized features* and *partially specialized features*.

Definition 3 *(F-feature)*: A feature is called a *fully specialized feature* in a configuration *C* if it is an included feature in *C* (i.e., a member of the set of included features) and has no variability involving attributes (i.e., it either has no attributes or all of its attributes have singletons as their A-domains). □

Definition 4 *(P-feature)*: A feature is called a *partially specialized feature* in a configuration *C* if it is an included feature in *C* (i.e., a member of the set of included features) and it has unresolved variability involving attributes (i.e., at least one of its attributes has two or more values in its A-domain). □

A feature is classified as an F-feature or P-feature with respect to a configuration. For instance, a feature X may be an F-feature in one configuration and a P-feature in another (e.g., *Ultrasonic Receiver* is an F-feature in the product P1$_1$, whereas it is a P-feature in the product P1).

The classifications for configurations and products are as follows.

Definition 5 *(F-configuration, F-product)*: An *F-configuration* is a configuration (*I*, *E*, Δ) such that all the features in *I* are F-features. A full F-configuration is called an *F-product*. □

Definition 6 *(P-configuration, P-product)*: A *P-configuration* is a configuration (*I*, *E*, Δ) such that *I* contains at least one P-feature. A full P-configuration is called a *P-product*. □

For instance, product P1 is a P-product since it includes a P-feature, namely *Ultrasonic Receiver*, whereas products P1$_1$, P1$_2$, and P1$_3$ are all F-products since they include only F-features.

A configuration where $I = \varnothing$ is considered to be an F-configuration. From their definition, P-configurations (and P-products) have unresolved variability involving attributes, whereas F-configurations (and F-products) do not.

As discussed in the previous section, the extended definitions we provide for the configuration and product establish a full backward compatibility with the existing definitions. However, it is worth noting that the extended definition for the product causes a slight shift in the understanding of products in the sense that the existing product definition (i.e., that given in [6]) imposes on a product to bear no variability, whereas the extended definition allows some products (i.e., P-products) to bear some type of variability (i.e., variability involving attributes). From this point of view, the F-products are the true counterparts of the products in the sense of [6]. On the other hand, P-products also obey all the requirements imposed by the existing definition as they also do not bear any variability in terms of features. Hence, we preferred to follow the existing literature and use the term "product" for all the byproducts of the extended definition (i.e., F-product, P-product, and the two subtypes of P-product presented in Sect. 5.3).

5.1 Semantics

The extended definitions provided for *product* and *configuration* are based on a syntactic notion. However, well-formedness in syntactical sense does not necessarily imply validity in the semantic domain, where the semantics are grounded in constraint satisfaction. Since the analysis of

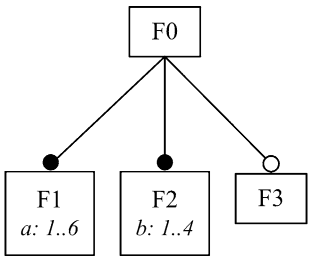


Fig. 5 An extended feature model, *EM*

feature models relies on the semantics of the models it is necessary to establish the foundations for the semantics. Therefore, in this section, we discuss the semantic validity of two key entities in the feature model analysis, namely, product and configuration.

Definition 7 *(Valid F-product)*: Let *M* be a feature model and *P* an F-product. *P* is semantically *valid* with respect to *M*, denoted by $M \mapsto P$, if and only if *P* satisfies all the constraints imposed in *M*. If $M \mapsto P$, then we can state that *P* is represented by *M*. □

Example Consider the example extended feature model, *EM*, given in Fig. 5.

Assume that we have the following cross-tree relations in this model:

- (Ctr1) If *F1.a* ≥ 3, then *F1* requires *F2.b* ≥ 3
- (Ctr2) If *F1.a* < 3, then *F1* requires *F2.b* < 3
- (Ctr3) If *F1.a* ≥ 5, then *F1* requires *F3*
- (Ctr4) If *F1.a* < 5, then *F1* excludes *F3*

The F-products P3 = ({F0, F1, F2}, {F1.a = 2, F2.b = 1}) and P4 = ({F0, F1, F2, F3}, {F1.a = 5, F2.b = 3}) are valid with respect to *EM*, since they satisfy all the constraints imposed by the relations in *EM*. On the other hand, P5 = ({F0, F1}, {F1.a = 2}) is not valid, since it fails the constraint imposed by the mandatory relation between F0 and F2. Similarly, P6 = ({F0, F1, F2}, {F1.a = 2, F2.b = 4}) is not valid, since it fails the constraint imposed by Ctr2. □

We provide semantics for the P-products using the semantic foundation established for the F-products.

Definition 8 *(Valid P-product)*: Let *M* be a feature model, *P* a P-product, and *S* the set of all F-products that can be derived from *P* by assigning values to the attributes from their respective A-domains. *P* is semantically *valid* with respect to *M*, denoted by $M \mapsto P$, if and only if at least one F-product in *S* is valid with respect to *M*. If $M \mapsto P$, then we can state that *P* is represented by *M*. □

Example Reconsidering the example extended feature model *EM*, assume that we have the P-product P7 = ({F0, F1, F2}, {F1.a = 1, F2.b ∊ 2..4}). Three F-products can be derived from P7:

- $P7_1$ = ({F0, F1, F2}, {F1.a = 1, F2.b = 2})
- $P7_2$ = ({F0, F1, F2}, {F1.a = 1, F2.b = 3})
- $P7_3$ = ({F0, F1, F2}, {F1.a = 1, F2.b = 4})

Although $P7_2$ and $P7_3$ are not valid, since they both fail Ctr2, the P-product P7 is considered to be valid since $P7_1$ is a valid F-product.

On the other hand, the P-product P8 = ({F0, F1, F2}, {F1.a ∊ 1..2, F2.b ∊ 3..4}) is not valid, since none of the F-products, which can be derived from P8 and are listed below, are valid with respect to *EM*.

- $P8_1$ = ({F0, F1, F2}, {F1.a = 1, F2.b = 3})
- $P8_2$ = ({F0, F1, F2}, {F1.a = 1, F2.b = 4})
- $P8_3$ = ({F0, F1, F2}, {F1.a = 2, F2.b = 3})
- $P8_4$ = ({F0, F1, F2}, {F1.a = 2, F2.b = 4}) □

The definition and the semantics of P-products allow different representations for the same set of products. For instance, consider the following three P-products represented by *EM*:

- P9 = ({F0, F1, F2}, {F1.a = 1, F2.b ∊ 1..2})
- P10 = ({F0, F1, F2}, {F1.a = 1, F2.b ∊ 1..3})
- P11 = ({F0, F1, F2}, {F1.a = 1, F2.b ∊ 1..4})

Although these three P-products are not the same, since their Δ components are different, when we derive all valid F-products from each one of them we obtain the same set of F-products: {P12, P13} where P12 = ({F0, F1, F2}, {F1.a = 1, F2.b = 1}) and P13 = ({F0, F1, F2}, {F1.a = 1, F2.b = 2}). Hence, there is a notion of equivalence emerging from this observation, which we formalize as follows:

Definition 9 *(Product equivalence)*: Let *M* be a feature model, *P* and *P′* two products represented by *M*, and *S* and *S′* the sets of all valid F-products that can be derived from *P* and *P′*, respectively. The products *P* and *P′* are considered to be *equivalent* if and only if *S* = *S′*. □

This formalization allows different types of products to be equivalent. For instance, the F-product ({F0, F1, F2}, {F1.a = 1, F2.b = 2}) is equivalent to the P-product ({F0, F1, F2}, {F1.a = 1, F2.b ∊ 2..4}). Two different P-products can be equivalent (e.g., P9 and P10); however, two F-products are equivalent if and only if they are same.

We build the semantics for partial configurations on the foundations established for the products.

Definition 10 *(Valid partial F-configuration)*: Let *M* be a feature model, *C* = (I, E, Δ) a partial F-configuration, and *S* = {P_1 = (I_1, $Δ_1$), …, P_n = (I_n, $Δ_n$)} the set of all F-products that can be derived from C such that the following three conditions hold:

- $I \subseteq I_i$,
- $E \cap I_i = \varnothing$,
- $\Delta \subseteq \Delta_i$

for $i = 1, \ldots, n$. C is semantically *valid* with respect to M, denoted by $M \mapsto C$, if and only if at least one F-product in S is valid with respect to M. □

Example Consider the extended feature model *EM*, and the partial F-configuration C1 = ({F0, F1}, {F3}, {F1.a = 1}). There are five F-products that can be derived from C, P_1 = ({F0, F1}, {F1.a = 1}) and P_2 = ({F0, F1, F2}, {F1.a = 1, F2.b = 1}) through P_5 = ({F0, F1, F2}, {F1.a = 1, F2.b = 4}). Although some of these F-products are not valid (e.g., P_1), we have at least one valid F-product (e.g., P_2); therefore, we state that $EM \mapsto$ C1.

Next, consider the partial F-configuration C2 = ({F1}, {F3}, {F1.a = 6}). The list below contains all F-products that can be derived from C2:

- ({F1}, {F1.a = 6})
- ({F0, F1}, {F1.a = 6})
- ({F1, F2}, {F1.a = 6, F2.b = 1}) through ({F1, F2}, {F1.a = 6, F2.b = 4})
- ({F0, F1, F2}, {F1.a = 6, F2.b = 1}) through ({F0, F1, F2}, {F1.a = 6, F2.b = 4})

Although there are ten F-products that can be derived from C2, none of them are valid (i.e., the first product fails the mandatory relation between F0 and F1, the next five products fail the mandatory relation between F0 and F2, and the remainders fail Ctr3); therefore, we state that C2 is not valid with respect to *EM*. □

Definition 11 *(Valid partial P-configuration)*: Let M be a feature model, C a partial P-configuration, and S the set of all partial F-configurations that can be derived from C by assigning values to the attributes from their respective A-domains. C is semantically *valid* with respect to M, denoted by $M \mapsto C$, if and only if at least one F-configuration in S is valid with respect to M. □

Example Consider the extended feature model, *EM*, and the partial P-configuration C3 = ({F0, F1, F2}, { }, {F1.a ∊ 1..2, F2.b ∊ 2..3}). The set of all partial F-configurations that can be derived from C3 by assigning values to the attributes from their respective A-domains is {$C3_1$, $C3_2$, $C3_3$, $C3_4$} where:

- $C3_1$ = ({F0, F1, F2}, { }, {F1.a = 1, F2.b = 2})
- $C3_2$ = ({F0, F1, F2}, { }, {F1.a = 1, F2.b = 3})
- $C3_3$ = ({F0, F1, F2}, { }, {F1.a = 2, F2.b = 2})
- $C3_4$ = ({F0, F1, F2}, { }, {F1.a = 2, F2.b = 3})

The configurations $C3_1$ and $C3_3$ are valid with respect to *EM*, whereas $C3_2$ and $C3_4$ are not. Since we have at least one valid partial F-configuration, C3 is a valid partial P-configuration with respect to *EM*; thus, $EM \mapsto$ C3.

Next, consider the partial P-configuration C4 = ({F1, F3}, { }, {F1.a ∊ 2..4}). The set of all F-configurations that can be derived from C4 is {$C4_1$, $C4_2$, $C4_3$} where:

- $C4_1$ = ({F1, F3}, { }, {F1.a = 2})
- $C4_2$ = ({F1, F3}, { }, {F1.a = 3})
- $C4_3$ = ({F1, F3}, { }, {F1.a = 4})

None of these F-configurations is valid; hence, C4 is not a valid P-configuration with respect to *EM*. □

5.2 An ordering relation

In the literature there are reports of various efforts to derive a product from a feature model in a stepwise manner. For instance, Czarnecki et al. [15] proposed successive specializations over configurations, which they refer to as staged configuration that eliminates some configuration choices. They start with a feature model and continue the specialization in stages until they reach a full configuration. In another attempt, Stoiber et al. [47] begin with a feature model, which employs the full variability for the family it represents, and perform a stepwise variability binding process to derive a single product represented by the model while facilitating automated tool support. Bagheri et al. [2] propose an interactive staged configuration approach and tool support to enhance the quality and ease the configuration process in the application engineering phase.

In this section we introduce an ordering relation between configurations to decide which configuration is more general (i.e., has more variability) and which is more specialized, in order to support the formal basis for product derivation. We start with a definition of the relation for the configurations that are defined as in [6] in order to cover the existing approaches, and then extend the definition to suit the extended configuration definition presented in Sect. 4.

Definition 12 *(Configuration ordering)*: Let $C_1 = (I_1, E_1)$ and $C_2 = (I_2, E_2)$ be two configurations represented by a feature model. C_1 is less or equally specialized with respect to C_2, denoted by $C_1 \preceq C_2$, if and only if the following two conditions hold:

- $I_1 \subseteq I_2$,
- $E_1 \subseteq E_2$ □

Given a set S, the binary relation $\preceq$ on S is a *partial order* (or *order*) if and only if for all $x, y, z \in S$, $\preceq$ is reflexive (i.e., $x \preceq x$), antisymmetric (i.e., $x \preceq y$ and $y \preceq x$ imply $x = y$), and transitive (i.e., $x \preceq y$ and $y \preceq z$ imply $x \preceq z$) [16]. Next, we show that the configuration ordering is a partial order on the set of all valid configurations represented by a feature model (see Appendix 1 for the proof of Proposition 1).

Proposition 1 Let M be a feature model, and S the set of all valid configurations with respect to M. The configuration ordering relation, denoted by $\leq$, is a partial order on S. □

A set S is said to be a *partially ordered set* (or *ordered set*), if it is equipped with a partial order relation [16]. Therefore, the set of all valid configurations represented by a feature model is a (partially) ordered set.

If S is an ordered set and $Q \subseteq S$, then an element $l \in S$ is called a *lower bound* of Q if $l \leq q$ for all $q \in Q$ [16]. Moreover, l is called the *greatest lower bound* of Q (or the *infimum* of Q) if $t \leq l$ for all lower bounds t of Q. When Q has two members such as $Q = \{x, y\}$, the infimum of Q can be denoted as $x \wedge y$ (i.e., x *meet* y). Next, we show that given a feature model it is always possible to find the infimum of any two valid configurations represented by this feature model (see Appendix 1 for the proof of Proposition 2).

Proposition 2 Let M be a feature model, and $C_1 = (I_1, E_1)$ and $C_2 = (I_2, E_2)$ two valid configurations represented by M. Then, $C_1 \wedge C_2$ is $C_{inf} = (I_1 \cap I_2, E_1 \cap E_2)$. □

If S is an ordered set and $Q \subseteq S$, then an element $u \in S$ is called an *upper bound* of Q if $q \leq u$ for all $q \in Q$ [16]. We do not always have an upper bound in the ordered set of all valid configurations represented by a given feature model. For instance, assume that f and g are two features that exclude each other in a given feature model, and C_f and C_g are two valid configurations such that C_f includes the feature f and excludes the feature g, and C_g includes the feature g and excludes the feature f. In any valid configuration $C = (I, E)$, where $C_f \leq C$, f must be included and g must be excluded. However, if a valid configuration C includes the feature f, then $f \notin E$; thus, $C_g \leq C$ will not be true. Hence, it is not possible to find an upper bound for $\{C_f, C_g\}$.

If S is a non-empty ordered set, and $x \wedge y$ exists for all $x, y \in S$, then S is called a *meet semilattice* [16]. Next, we conclude that the set of all valid configurations represented by a non-void feature model is a meet semilattice (see Appendix 1 for the proof of Proposition 3).

Proposition 3 Let M be a feature model that is not void. The set of all valid configurations represented by M is a meet semilattice. □

If S is the set of all valid configurations represented by a given non-void feature model, then the infimum of S will be the empty configuration $(\emptyset, \emptyset)$, since $\emptyset \subseteq I$ and $\emptyset \subseteq E$ for any sets of features I and E. The maximal elements will be the valid full configurations (i.e., valid products).

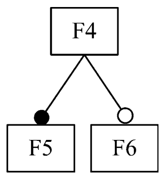


Fig. 6 An example feature model

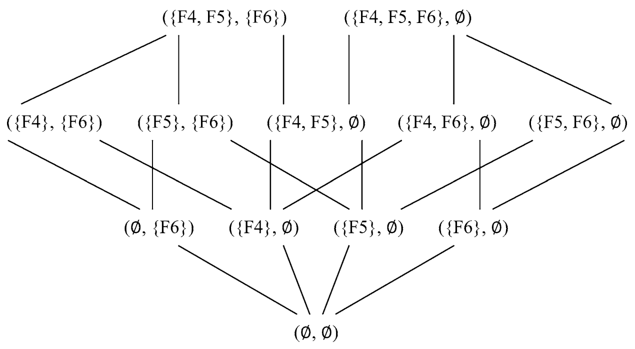


Fig. 7 The Hasse diagram of the meet semilattice

Example Consider the feature model given in Fig. 6.

The Hasse diagram [16] depicting the meet semilattice formed by the valid configurations represented by this model is given in Fig. 7. □

Next, we extend the ordering relation to fit the extended configuration definition.

Definition 13 *(Extended configuration ordering)*: Let $C_1 = (I_1, E_1, \Delta_1)$ and $C_2 = (I_2, E_2, \Delta_2)$ be two configurations represented by a feature model, where $\Delta_1 = \{a_1 \in A_{1\text{-}1}, \ldots, a_n \in A_{1\text{-}n}\}$ and $\Delta_2 = \{a_1 \in A_{2\text{-}1}, \ldots, a_n \in A_{2\text{-}n}, \ldots\}$. C_1 is less or equally specialized with respect to C_2, denoted by $C_1 \leq C_2$, if and only if the following three conditions hold:

- $I_1 \subseteq I_2$,
- $E_1 \subseteq E_2$,
- $A_{2\text{-}1} \subseteq A_{1\text{-}1}, \ldots, A_{2\text{-}n} \subseteq A_{1\text{-}n}$ □

Using the ideas presented in the proof of Proposition 1, it can be shown that the extended configuration ordering is a partial order on the set of all valid extended configurations (both valid F-configurations and P-configurations) represented by a given feature model. Here, we will show that for any two valid extended configurations x and y, $x \wedge y$ exists (see Appendix 1 for the proof of Proposition 4).

Proposition 4 :Let M be an extended feature model, and $C_1 = (I_1, E_1, \Delta_1)$ and $C_2 = (I_2, E_2, \Delta_2)$ two valid extended configurations represented by M, where $\Delta_1 = \{a_1 \in A_{1\text{-}1}, \ldots, a_n \in A_{1\text{-}n}, \ldots\}$ and $\Delta_2 = \{a_1 \in A_{2\text{-}1}, \ldots, a_n \in A_{2\text{-}n}, \ldots\}$; thus, the set of attributes that appear in both Δ_1 and Δ_2 are $\{a_1, \ldots, a_n\}$. Then, $C_1 \wedge C_2$ is $C_{inf} = (I_1 \cap I_2, E_1 \cap E_2, \Delta_{inf})$, where $\Delta_{inf} = \{a_1 \in A_{1\text{-}1} \cup A_{2\text{-}1}, \ldots, a_n \in A_{1\text{-}n} \cup A_{2\text{-}n}\}$. □

Finally, we conclude that the set of all valid extended configurations represented by a non-void extended feature model is a meet semilattice (see Appendix 1 for the proof of Proposition 5).

Proposition 5 Let M be an extended feature model that is not void. The set of all valid extended configurations represented by M is a meet semilattice. □

If S is the set of all valid extended configurations represented by a given non-void extended feature model, then the infimum of S will be the empty configuration ($\emptyset$, $\emptyset$, $\emptyset$). The maximal elements will be the valid full F-configurations (i.e., valid F-products).

If S is an ordered set and any two elements of S are comparable (i.e., for all x, $y \in S$, either $x \le y$ or $y \le x$), then S is called a *chain* (or a *totally ordered set*) [16]. Any increasing chain in the set of valid configurations represented by a given feature model corresponds to a derivation of the final configuration (the largest element in the chain) from the initial configuration (the smallest element in the chain). In particular, a chain starting from the empty configuration ($\emptyset$, $\emptyset$, $\emptyset$) to some maximal element reflects the sequence of decisions made in the whole process of deriving an F-product. A decision can be basic (include a single feature, exclude a single feature, or reduce the A-domain of some attribute by removing a single value) or a combination of basic decisions.

If S is an ordered set and x, $y \in S$, x is *covered by* y (or y covers x) if $x < y$ and $x \le z < y$ implies $z = x$ (i.e., there is no element z of S such that $x < z < y$) [16]. The case where a valid configuration C_1 is covered by another valid configuration C_2 (equivalently, C_2 covers C_1) indicates that C_2 is obtained from C_1 by a basic decision.

Let S be an ordered set and $Q \subseteq S$. For any $x \in Q$ and $y \in S$, if we have $y \in Q$ whenever $y \le x$, then Q is called a *down-set*, if we have $y \in Q$ whenever $x \le y$, then Q is called an *up-set* (i.e., down-sets are "closed under going down," and up-sets are "closed under going up") [16]. Hence, if C is a valid configuration, the smallest down-set that includes C (i.e., C is the least upper bound of the down-set) represents all possible ways of deriving C starting from the empty configuration. Dually, the smallest up-set that includes C (i.e., C is the greatest lower bound of the up-set) represents all possible ways of deriving F-products starting from C.

5.3 A closer look at P-products

There is variability in the syntactical sense in P-products, since at least one of the A-domains in the product must have two or more values that can be assigned to the respective attribute. However, this variability may not actually lead to a variety of products. For instance, consider the feature model EM and the P-product ({F0, F1, F2}, {F0.a = 2, F1.b ∈ 2..4}). Although it is possible to derive three F-products from this P-product, ({F0, F1, F2}, {F0.a = 2, F1.b = 2}) through ({F0, F1, F2}, {F0.a = 2, F1.b = 4}), only the first is a valid F-product (the remainder all fail Ctr2). Clearly, variability in such P-products is pseudo-variability, and it does not result in a variety of products; thus, it can give the user wrong indication. Hence, we introduce a subcategory for P-products that do not have this deficiency.

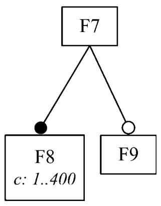


Fig. 8 An extended feature model

Definition 14 *(Proper P-product)*: Let M be a feature model, P a P-product, and S the set of all F-products that can be derived from P by assigning values to the attributes from their respective A-domains. P is called a *proper P-product* if and only if two or more of the F-products in S are valid with respect to M. □

The definition given for proper P-products also enables a semantic interpretation, since there is no proper P-product that is invalid. This is not surprising since the category of proper P-products is a subcategory of all valid P-products, where the membership of this category is based on semantics.

Although proper P-products fit the idea of product lines in a better way, since they possess a real variability that leads to a variety of products, this may still be far from being an efficient representation. For instance, consider the model given in Fig. 8.

Assume that there is a cross-tree constraint "*F9* requires *F8.c* = 200 or *F8.c* = 400" and a proper P-product P14 = ({F7, F8, F9}, {F8.c ∈ 1..400}). When we remove all the variability to achieve a set of fully specialized products we obtain 400 different F-products $P14_1$ = ({F7, F8, F9}, {F8.c = 1}) through $P14_{400}$ = ({F7, F8, F9}, {F8.c = 400}). A quick analysis reveals that only two of these F-products, namely $P14_{200}$ and $P14_{400}$, are valid, while the remainder of the products are all semantically invalid. A simple calculation shows that 99.5 % of the fully specialized products that can be derived from P14 are semantically invalid, which means the resources (e.g., time, power, space.) used to derive these 398 F-products will be wasted.

Recalling the feature model EM and assuming that there is a P-product P15 = ({F0, F1, F2}, {F1.a ∈ 1..2, F2.b ∈ 1..2}) there are four F-products that can be derived from this P-product in total:

- $P15_1 = (\{F0, F1, F2\}, \{F1.a = 1, F2.b = 1\})$
- $P15_2 = (\{F0, F1, F2\}, \{F1.a = 1, F2.b = 2\})$
- $P15_3 = (\{F0, F1, F2\}, \{F1.a = 2, F2.b = 1\})$
- $P15_4 = (\{F0, F1, F2\}, \{F1.a = 2, F2.b = 2\})$

A simple analysis shows that these four F-products are semantically valid since they satisfy all the constraints in *EM*. Thus, any assignment to the attributes from their respective A-domains in P15 derives a semantically valid F-product. Such situations are desirable as no resource expended on the derivation of fully specialized products will be wasted. Moreover, if we know in advance that a P-product has a valid F-product derivation rate of 100 %, we can safely eliminate the cost of checking the validity of the derived F-products. Hence, such P-products deserve a classification of their own.

Definition 15 *(C-product)*: Let *M* be a feature model, *P* a proper P-product, and *S* the set of all F-products that can be derived from *P* by assigning values to the attributes from their respective A-domains. *P* is called a *free choice product* if and only if all of the F-products in *S* are valid with respect to *M*. □

Some of the C-products that can be derived from *EM* are:

- $(\{F0, F1, F2\}, \{F1.a \in 1..2, F2.b \in 1..2\})$
- $(\{F0, F1, F2\}, \{F1.a \in 3..4, F2.b \in 3..4\})$
- $(\{F0, F1, F2, F3\}, \{F1.a \in 5..6, F2.b = 4\})$

Similar to proper P-products, the definition for C-products also enables a semantic interpretation, since there is no C-product that is invalid.

C-products may be of great importance especially when there are limited resources available to derive F-products. For instance, handheld devices have limited capabilities in terms of resources; thus, deploying C-products in such devices may be desirable since they provide a number of advantages. First, as they still possess variability, they allow for a delay in binding the attribute values. Second, they demand less computing power, since the validity of every derived F-product is guaranteed and there is no need to perform a check. Consequently, there is no need to store the cross-tree constraints in the model and C-products will demand less storage space than the other P-products.

6 Effects on analyses

6.1 Existing analyses

The extended definitions proposed in the previous sections inevitably affect the analysis operations. For instance, the analysis operation *number of products* is used to compute the number of products that are represented by a feature model. However, as there are a number of product types (i.e., F-product, P-product, proper P-product, and C-product) this definition becomes ambiguous, since it is not possible to infer the product type that was intended.

In the following subsections we discuss some of the well-known analysis operations on feature models, propose a reformulation for some of them, and suggest a revised understanding for others, considering the effects of the new type of variability that has been introduced. However, we do not attempt to cover all the operations that can be revised; instead, our aim is to present selected illustrative examples.

6.1.1 Valid partial configuration

This operation is used to discover whether a given configuration is valid with respect to a given feature model. The original version of this operation takes a feature model and a partial configuration as the input and returns a value that notifies whether the configuration is valid [6].

However, with the introduction of new classifications for configurations this operation should be reformulated as follows:

Valid partial configuration: This operation takes a feature model *M*, a partial configuration *C*, and a configuration type *T* (i.e., F-configuration or P-configuration) as inputs, and returns a value which confirms whether *C* is a valid partial configuration of type *T* with respect to *M*.

The *valid product* analysis operation [6] can also be reformulated in a similar manner.

6.1.2 All products

This operation is used to compute all the products represented by a given feature model. The original version of this operation requires a feature model as input, and outputs all products represented by the model [6]. However, since we have classified products into different categories, the type of products to be outputted must be specified as well. Hence, the all products analysis operation is redefined as follows:

All Products: This operation takes a feature model *M* and a product type *T* as inputs, and returns all the products of type *T* that are represented by *M*.

The *number of products* analysis operation [6] can also be redefined in a similar manner.

6.1.3 Filter

This operation is used to compute the set of valid products that include a given configuration. The original version of this operation requires a feature model and a configuration

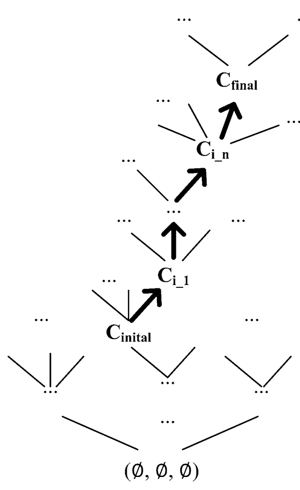


Fig. 9 A chain from the initial configuration to the final configuration

as inputs, and outputs all products that are represented by the model including the input configuration [6]. We use the ordering relation presented in Sect. 5.2 to reformulate this operation.

Filter: This operation takes a feature model *M*, a configuration *C*, and a product type *T* as inputs, and returns all the products $P_1, \ldots, P_n$, which are of type *T*, represented by *M*, and $C \leq P_1, \ldots, C \leq P_n$.

Note that, if the input product type is F-product, then the filter operation is equivalent to finding all the maximal elements in the smallest up-set that includes *C*.

6.1.4 Multi-step configuration

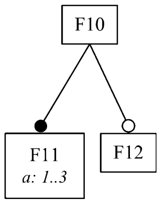
This operation is used to solve a multi-step configuration problem, which is known as producing a specialization path from one configuration to another [6]. As discussed in Sect. 5.2, for a given feature model the set of all valid configurations represented by the model is a meet semi-lattice. Thus, this operation is actually finding a chain from the initial, to the final configuration, as depicted in Fig. 9.

Multi-step configuration: This operation takes a feature model *M*, an initial configuration $C_{initial}$, a desired final configuration C_{final}, a number *n*, a global constraint *G* that cannot be violated, and a function determining the cost to transition from one configuration to another as inputs, and returns an ordered list, *L*, of valid configurations with respect to *M* as the output, where $L = (C_{i\_1}, \ldots, C_{i\_n})$, such that: *(i)* $C_{initial} \leq C_{i\_1}$ and $C_{i\_n} \leq C_{final}$ *(ii)* elements of *L* form an increasing chain, *(iii)* *L* has exactly *n* elements, *(iv)* no transition in the chain violates the global constraint *G*.

6.1.5 Core features

This operation is used to find the set of features that are included in all the products represented by a feature model [6]. Before discussing how this analysis operation is affected by the extended product definition, we will consider the example given in Fig. 10.

Fig. 10 An extended feature model



The feature model given in Fig. 10 represents six F-products:

- ({F10, F11}, {F11.a = 1}) through ({F10, F11}, {F11.a = 3})
- ({F10, F11, F12}, {F11.a = 1}) through ({F10, F11, F12}, {F11.a = 3})

It also represents eight P-products:

- ({F10, F11}, {F11.a ∊ {1, 2}}), …, ({F10, F11}, {F11.a ∊ {1, 2, 3}})
- ({F10, F11, F12}, {F11.a ∊ {1, 2}}), …, ({F10, F11, F12}, {F11.a ∊ {1, 2, 3}})

The cases for the features F10 and F12 are obvious (i.e., F10 is a core feature and F12 is not). However, the case for feature F11 is more interesting, since although it is included in all of the products the A-domains of its attribute are different in some of the products. Consequently, the specialization of the feature F11 can be different in different products. However, regardless of its specialization, the feature will continue to be included in those products. Thus, we conclude that F11 is a core feature.

The extension we have proposed to the product definition does not affect the inclusion state of a feature in a product; it only provides extra information regarding the attributes. Hence, the core features analysis operation remains unaffected. Using a similar inference it is easy to show that the *variant features* analysis operation [6] also remains unaffected.

6.1.6 Commonality

This operation is used to compute a measure of how common a given configuration is among the products represented by a given feature model. The original version of this operation requires a feature model and a configuration as inputs and returns the percentage of products including the input configuration, as computed using Formula 1 [6].

$$\text{Commonality}(M, C) = \frac{|\text{filter}(M, C)|}{\text{Number of Products}(M)} \qquad (1)$$

As discussed in Sects. 6.1.2 and 6.1.3, number of products and filter analysis operations require a product type as input, and the outcome of these operations depends on this input. Thus, the commonality analysis operation is redefined as follows:

Commonality: This operation takes a feature model M, a configuration C, and a product type T as input, and returns the percentage of products including the input configuration, which is computed using Formula 2.

$$\text{Commonality}(M, C, T) = \frac{|\text{filter}(M, C, T)|}{\text{Number of Products}(M, T)} \tag{2}$$

If the number of products is equal to 0 (e.g., when considering P-products and no feature having an attribute) the operation must automatically return 0 to avoid division by zero.

6.1.7 Variability factor

This operation computes the ratio between the actual number of products and the potential number of products represented by a given feature model. The original version of this operation requires a feature model as the input and returns the variability factor computed by Formula 3 [6].

$$\left(\begin{array}{c}\left(\left(2^{|D_{1.1}|}-1\right)\times\ldots\times\left(2^{|D_{1.p}|}-1\right)+1\right)\times\ldots\times\left(\left(2^{|D_{m.1}|}-1\right)\times\ldots\times\left(2^{|D_{m.q}|}-1\right)+1\right)\\ -\left(|D_{1.1}|\times\ldots\times|D_{1.p}|+1\right)\times\ldots\times\left(|D_{m.1}|\times\ldots\times|D_{m.q}|+1\right)\end{array}\right)\times 2^{n-m} \tag{6}$$

$$\text{Variability Factor}(M) = \frac{\text{Number of Products}(M)}{2^n} \tag{3}$$

In particular, 2^n, where n is the number of features to be considered, is the *number of potential products* represented by a feature model assuming that any combination of features is allowed. The formula 2^n follows the assumption that a feature has two possible states with respect to a product: it is either included in the product or not. However, this assumption does not hold if a feature has an attribute that has two or more values in its specified domain. Hence, the computation for the number of potential products and the variability factor analysis operation must be reformulated.

Variability Factor: This operation takes a feature model M and a product type T as the input, and returns the ratio between the actual and potential number of products represented by M, which is computed with Formula 4.

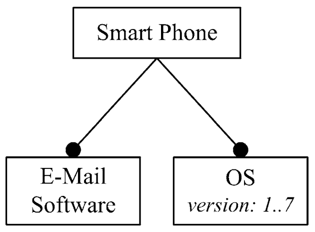


Fig. 11 An extremely simplified feature model for a smart phone family

The appropriate formulas for the F-product and P-product types are given in Formula 5 and Formula 6, respectively. In these formulas n represents the total number of features, m the number of features that have at least one attribute with a domain whose cardinality is greater than 1 such that $0 \leq m \leq n$, $D_{i.j}$ the domain of the jth attribute of the ith feature and |D| the cardinality of domain D.

$$\text{Variability Factor}(M, T) = \frac{\text{Number of Products}(M, T)}{\text{Appropriate formula for T}} \tag{4}$$

$$2^{n-m} \times \left(|D_{1.1}| \times \ldots \times |D_{1.p}| + 1\right) \times \ldots \times \left(|D_{m.1}| \times \ldots \times |D_{m.q}| + 1\right) \tag{5}$$

Since we assume that any A-domain specification for the attributes is possible, Formula 6 can also be used to calculate the potential number of proper P-products and C-products. If the number of potential products is equal to 0 (e.g., when considering P-products and no feature having an attribute), the operation must automatically return 0 to avoid division by zero.

6.2 New analyses

The new variability type and the extended definitions of configuration and product also open the way to new analysis needs. In this section we provide examples of such analyses.

6.2.1 Dead attribute values

Consider the extremely simplified extended feature model for a smart phone given in Fig. 11.

Assume that there is a cross-tree relation *E-Mail Software requires OS.version* $\geq$ *6* in this model. As *E-Mail Software* is a core feature, the attribute *OS.version* can never obtain a value from 1..5 when all the variability (in terms of features and attributes) in the model is removed. In this case values 1 through 5 are called *dead attribute values*. The model as it is gives the user the impression that early versions of the operating system (i.e., versions 1 through 5 in this case) can also be installed in this device. However, as the quick analysis reveals, no valid F-product that includes an early version can exist. Thus, such cases are undesirable since they give the user incorrect ideas about the domains of the attributes.

To find the dead attribute values in a feature model we introduce a new analysis operation, *dead attribute values*, as follows.

Dead Attribute Values: This operation takes a feature model as the input, and returns the set of *(attribute, set of dead values)* pairs in the given feature model.

Dead attribute values cannot exist in an A-domain of an attribute in any valid F-product (and consequently cannot exist in any C-product), but may exist in some A-domains of the attributes in some P-products and proper P-products. However, eventually these values will have to be eliminated when removing variability involving attributes.

6.2.2 Generating the corresponding F-product set

Sets of products are often encountered in the feature model representations and analyses. For instance, a feature model can also be expressed as the set of all products that can be derived from the model. Outputs of a number of analysis operations such as *all products* and *filter* are sets of products. Since there are four product types, consequently, there will be product sets including different types of products. In this section we will present a correspondence relation between product sets.

Definition 16 *(Corresponding F-set)*: Let *M* be a feature model, and *S* a set of products derived from *M*. Then *the corresponding F-set* for *S* with respect to *M*, which will be denoted as F-set(*S*, *M*), is the set of all valid F-products (with respect to *M*) that can be derived from the products included in *S* (i.e., the set of all maximal elements in the smallest up-set that includes all the elements of *S*). □

Example Consider the feature model *EM* and the product set S = {P16}, where P16 = ({F0, F1, F2}, {F1.a ∊ 1..3, F2.b ∊ 1..3}). Then the corresponding F-set is F-set(S, *EM*) = {$P16_1$, $P16_2$, $P16_3$, $P16_4$, $P16_5$}, where:

- $P16_1$ = ({F0, F1, F2}, {F1.a = 1, F2.b = 1})
- $P16_2$ = ({F0, F1, F2}, {F1.a = 1, F2.b = 2})
- $P16_3$ = ({F0, F1, F2}, {F1.a = 2, F2.b = 1})
- $P16_4$ = ({F0, F1, F2}, {F1.a = 2, F2.b = 2})
- $P16_5$ = ({F0, F1, F2}, {F1.a = 3, F2.b = 3}) □

All product sets, except for those that consist of only F-products, will include products that still have some unresolved variability involving attributes; hence, it is possible to further specialize them. Thus, we introduce a new analysis operation to undertake this task.

Generating the Corresponding F-Set: This operation takes a feature model *M* and a set of products *S* as the input and outputs the corresponding F-set for *S* with respect to *M* (i.e., F-set(*S*, *M*)).

6.2.3 Generating a minimum set

Recall that, although P-products still have unresolved variability involving attributes, this variability may not lead to an actual variety of valid products. C-products, on the other hand, offer true variability and an efficient way of representing a set of products by a single product, since all the F-products that can be derived from a C-product will be valid. Thus, if a product set includes only C-products and valid F-products, then it will be ultimately prolific, since any F-product obtained from the set will be valid. Therefore, we start by introducing such product sets.

Definition 17 *(Prolific product set)*: Let *M* be a feature model, and *S* a set of products. *S* is called *prolific* if *(i)* *S* includes only C-products and F-products, *(ii)* all products in *S* are valid with respect to *M*. □

Prolific sets enable efficient derivation of F-products, since we can arbitrarily assign values to the attributes from their A-domains in order to derive fully specialized products without being concerned about violating a constraint in the feature model. However, still leading to the derivation of the same set of F-products, some prolific sets may offer a more efficient representation than other sets. For instance, consider the following set of products, S, derived from *EM*:

- ({F0, F1, F2}, {F1.a = 1, F2.b = 1})
- ({F0, F1, F2}, {F1.a = 1, F2.b = 2})
- ({F0, F1, F2}, {F1.a = 2, F2.b = 1})
- ({F0, F1, F2}, {F1.a = 2, F2.b = 2})

A prolific set, S_1, where F-set(S_1, *M*) = F-set(*S*, *M*) can contain the following three products:

- ({C, F1, F2}, {F1.a = 1, F2.b ∊ 1..2})
- ({C, F1, F2}, {F1.a ∊ 1..2, F2.b = 1})
- ({C, F1, F2}, {F1.a = 2, F2.b ∊ 1..2})

Another prolific set, S_2, where F-set(S_2, *M*) = F-set(S, *M*) can contain only the following product:

- ({C, F1, F2}, {F1.a ∊ 1..2, F2.b ∊ 1..2})

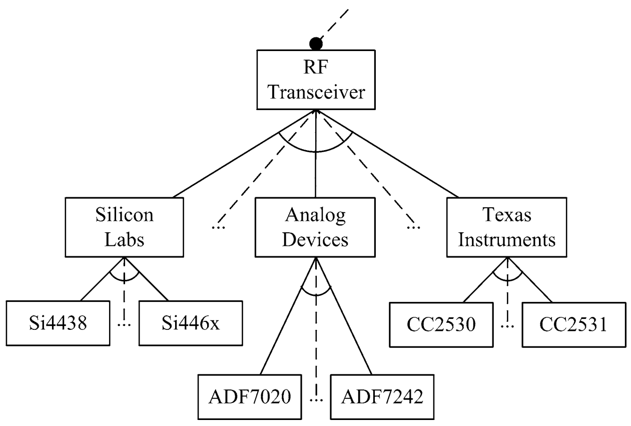


Fig. 12 Decomposition of the feature *RF Transceiver*

It is also possible to find other prolific sets whose corresponding F-set will be equal to the F-set(S, *M*). Although the corresponding F-sets of S_1 and S_2 are both equal to the F-set(S, *M*), S_2 offers a more efficient representation since S_2 contains only a single product, whereas S_1 contains three products. Moreover, it is not possible to find a prolific set S_x such that F-set(S_x, *M*) = F-set(S, *M*) and $|S_x| < |S_2|$. Thus, S_2 has a special property among the other prolific sets.

Definition 18 *(A minimum prolific set)*: Let *S* be a set of products, *M* a feature model, and $\{S_1, \ldots, S_n\}$ the set of all prolific sets such that F-set(S_i, *M*) = F-set(S, *M*) for i = 1, …, n. Then, S_k is called *a minimum prolific set* corresponding to S with respect to *M* if $|S_k| \leq |S_i|$ where $1 \leq k \leq n$. □

Using a minimum prolific set would be desirable to represent a set of products, since it provides a smaller set, which includes fewer products, but capable of representing the same F-product set. A minimum set can be used to discover individual C-products that represent a wider set of F-products compared with similar C-products (i.e., the C-products that include the same set of features). Thus, to compute a minimum prolific set corresponding to a given set of products and a feature model we introduce a new analysis operation, generating a minimum prolific set, as follows.

Generating a Minimum Prolific Set: This operation takes a feature model *M* and a product set *S* as input and returns a minimum prolific set corresponding to *S* with respect to *M*.

7 Effects in practice

The feature model we built for the KaVIS project includes many features that represent off-the-shelf components that are readily available in the market. For instance, the *RF Transceiver* feature is further decomposed into actual components as depicted in Fig. 12.

The RF Transceiver component is used for radio communication between the mobile unit and the beacon nodes in an industrial, scientific and medical (ISM) [25] frequency band. There are various ISM frequency bands available in different regions as specified by the International Telecommunications Union. For instance, Turkey is located in Region-1; thus, frequency ranges such as 40.66–40.70 MHz and 2.4–2.5 GHz are available for use in Turkey.

First, we utilized the filter operation in order to find the products that can be marketed in specific regions. Note that, the value of the attribute *frequency*, which belongs to the feature *RF Transceiver*, determines whether a product can be marketed in a region. In order to find the products that qualify, regarding the ISM regulations, to be marketed in Region-2, we applied the *filter* operation on the extended configuration ({RF Transceiver}, {}, {RF Transceiver.frequency ∊ {13.553–13.567 MHz, 26.957–27.283 MHz, …, 5.725–5.875 GHz, 24–24.25 GHz}). Thus, we clearly needed the revised version of the filter operation for this task.

Next, we decided to focus on the products that can be marketed locally in Turkey. As discussed above, this goal can also be achieved by applying the filter operation. However, we chose to modify our model to make it specifically tailored for Region-1. Therefore, we added the constraint "*Mobile Unit* requires an *RF Transceiver th*at is capable of operating in an ISM band in Turkey (i.e., Region-1)." Then, we applied the *dead attribute values* operation. This operation eliminated the values, which cannot appear in any of the valid F-products, from the domains of the attributes (e.g., values that do not belong to Region-1 ISM band frequencies from the domains of the *frequency* attributes of the RF Transceiver components). Hence, we obtained a simplified and more realistic model.

We also sought and produced C-products that serve for different purposes. Since C-products still possess variability involving attributes where values for the attributes can freely be selected from their A-domains (i.e., all possible assignments will result in valid F-products) they facilitate the delaying of the binding on certain properties. For instance, we sought C-products that have a fixed hardware structure, but can operate in different regions by simply configuring the RF Transceiver operation frequency. Another example was that we sought products where the parameters of the installed ranging algorithms (e.g., the maximum number of beacons that will be taken into account in the computations, the size of the measurement pool that will be used in the computations) can be configured before deployment in accordance with the capabilities of the micro controller unit (the clock speed, the size of the flash and RAM memories, etc.) to obtain better performance for different considerations from the selected hardware.

After obtaining various C-products we generated the corresponding F-product sets to check the product diversity these C-products offer by simply configuring some parameters before the deployment. We also generated corresponding minimum prolific sets for different F-product sets in order to obtain C-products that lead to a greater variety of F-products compared with similar C-products.

8 Related work

As the success of a product family is highly dependent on effective variability handling [8], variability modeling and management is an essential activity in SPLs. There are several variability modeling and management approaches reported in the literature such as feature modeling (e.g., [26]), decision modeling (e.g., [46]), using UML and its extensions (e.g., [51]), using domain ontologies (e.g., [1]), using use cases (e.g., [22]), and using the orthogonal variability model (e.g., [40]). Within this wide variety of techniques, feature modeling is the most popular choice [11]. The variability modeling and management method we discussed in this study are feature modeling. We refer the reader to [45] and [11] for detailed literature reviews and classifications of various variability modeling and management techniques used in SPLs.

The existing approaches that use feature models manage variability at the level of features (i.e., features are the units of variability in feature models) [14], whereas our work extends the notion of variability in feature models to also include the feature attributes as variability units. The existing approaches model variability in feature models through decomposition and cross-tree relations. The decomposition relations include the *mandatory*, *optional*, and *alternative* relations, as introduced in [26], the *or* relation, as introduced in [21], the *group cardinality* relation, as proposed in [41] and [15], and the *feature cardinality* relation, as proposed in [15]. The cross-tree relations include the basic *requires* and *excludes* constraints, as introduced in [26], formulas constructed by applying propositional connectives on arbitrary features (e.g., *Feature X requires Feature Y or Feature Z*), as described in [3], and complex cross-tree relations involving both features and feature attributes, as described in [31]. Our work does not introduce a new relation to enhance the expressiveness of the feature models; instead, it focuses on the effects of the variability modeling relations, primarily the complex cross-tree relations involving both features and feature attributes, used in feature models. The variability modeling relations involving only features and the related analysis operations are a well-studied subject in the literature. However, to the best of our knowledge, this article is the first to elaborate variability modeling in feature models at the level of feature attributes, hence extending the units of variability to include feature attributes.

There are various classifications for features reported in the literature. For instance, Kang et al. [27] consider different levels of abstraction and classify features into four categories according to the types of information they represent: capability features, operating environment features, domain technology features, and implementation technique features. Later, Lee et al. [33] discuss the different viewpoints that exist in the problem and solution spaces and refine this classification as goal features, usage context features, quality features, capability features, operating environment features, and design features. Loesch et al. [34] present a classification for features according to their usage in real products and describe the following four categories for the variable features: always used, never used, only used mutually exclusively, and only used in pairs. Botterweck et al. [9] propose to leverage visualization techniques in order to support fundamental development tasks and classify some of the features as cost-driving features (i.e., features with a high relative contribution to the products' total development costs) and high-risk features (i.e., features that represent critical capabilities of the product line). In their work, high-risk features are further classified into four subcategories as follows: critical innovative features, critical routine features, critical third party features, and critical systemic features. In one of their studies, Lee et al. [32] classify features into feature groups based on an analysis regarding binding information. They adopt a two-dimensional viewpoint, with the first dimension being the product lifecycle and the second being the feature binding state, and in which categories are described to present different types of features such as features that will be included at the product development stage, features that will become available at the installation phase, and so on.

The feature classification we present in this article is solely based on the new variability type (i.e., variability involving attributes). Therefore, it is not an alternative or complement to any of the aforementioned classifications; rather, it is a classification that is to be used in parallel. For instance, a feature that belongs to the category capability features as described in [33] will also be an F-feature or P-feature. Furthermore, a critical innovative feature as described in [34] will also be an F-feature or P-feature. Moreover, we do not categorize the features in a feature model until they appear in a configuration or a product as an included feature. Hence, the classification we present for features is dynamic in the sense that it may vary for a feature with respect to the configurations or products in which it is included. For instance, a feature may be classified as an F-feature in one configuration and as a P-feature in another.

Feature classifications also exist that are introduced in accordance to the characteristics of the domain of the product family. For instance, Lopez-Herrejon et al. [35] define a standard problem from the domain of classical graph applications in computer science for evaluating product-line methodologies and propose a classification for the features extracted from this domain. Later, Wang et al. [49] explicitly list this classification as algorithm, graph type, and search features, and build a feature model using them in order to test their approach. Clearly, since our classification is independent of domain, it can coexist with such domain-specific classifications. For instance, a graph-type feature will also be an F-feature or P-feature, depending on the configuration or product in which it is included.

The classifications for products reported in the literature are essentially based on domain characteristics. For instance, Hartmann et al. [23] give product classifications such as DVDs, MP3 players, and hard disk recorders, or budget products, mid-range products, and high-end products where the motivations for these categorizations originate from the domain characteristics. Similarly, Dumitru et al. [18] classify products into types such as antivirus software, browsers, and file managers while recommending features for a product family in the computer software domain. Our classification for products is solely built on the requirements that the new variability type demands meaning that it is not domain specific. Therefore, it can coexist with such domain-specific product classifications. For instance, a mid-range product as described in [23] will also be an F-product or P-product (if so can also be a proper P-product and/or C-product).

Using lattices in support of feature modeling is not new. For instance, Ryssel et al. [42] make use of a lattice to build a feature model from an incidence matrix that describes the common and different artifacts of variants. Niu et al. [38] use a lattice ordering to characterize variability degrees and evolutionary properties in order to integrate different viewpoints of a product line. However, our purpose in using partial orders and lattices is completely different since we use them primarily to support product derivation. A recent work by Höfner et al. [24] presents a semi-ring of product families to express variabilities in the form of features. They too define a partial order, the natural order associated with the semi-ring. The natural order is defined on product families (i.e., sets of products), whereas our partial order is defined on configurations. There is a connection so that if two configurations are related in our order, say $C_1 \leq C_2$, then we have $p(C_2) \leq p(C_1)$ in the natural order (in fact, $p(C_2) \subseteq p(C_1)$), where $p(C_1)$ is the set of maximal elements in the smallest up-set that includes C_1, (i.e., the set of all products derivable from C_1), and similarly for C_2. In line with most publications in the field, they consider features as variability units, whereas we consider both features and their attributes and extend our ordering relation accordingly. We establish the fact that the set of all valid configurations represented by a feature model, whether basic or extended, is a meet semilattice.

Kang et al. [26] were the first to propose *valid partial configuration* and *valid product* analysis operations in a work in which they introduced feature models. The first references to *all products* and *number of products* analysis operations exist in Mannion [36] and van Deursen et al. [17]. The first references to *filter*, *variability factor*, and *commonality* analysis operations were in the work of Benavides et al. [4, 5, 7], and Fernandez-Amoros et al. [19] were the first to suggest the *homogeneity* analysis operation. The *multi-step configuration* analysis operation is defined in [52]. The *core features* and *variant features* analysis operations were first discussed in [48]. We refer the reader to [6] for a detailed literature review on the analysis operations on feature models. However, all of these analysis operations consider only the features as variability units. For instance, the valid partial configuration analysis accepts the configurations specified by only the set of features to be included and excluded. Our study includes feature attributes as variability units in addition to features and revisits these analysis operations to discuss the effects of the new variability type, variability involving attributes. We provide slight modifications to some operations (e.g., number of products), a revised understanding of other operations (e.g., filter), maintain some operations (e.g., core features), and reformulate others (e.g., variability factor) while keeping the purpose and essence of the operations unchanged. We must also note that the new analysis operation *dead attribute values* that we introduce was inspired from the analysis operation *dead features*, first introduced by Kang et al. in [26], and finds the features that are included in none of the valid products represented by a feature model.

The sequence of development, starting with the seminal work of Czarnecki et al. [13] and Benavides et al. [7], continuing with Karataş et al. [29, 31] and culminating in the present study, has established feature attributes as first-class entities in variability modeling. In the same way as features, feature attributes can also be treated as units of variability in the modeling and analysis of variability.

9 Discussion and challenges

9.1 An alternative approach?

Recall that in this work, we allow only finite sets to be the domains of attributes. Therefore, theoretically, it can be

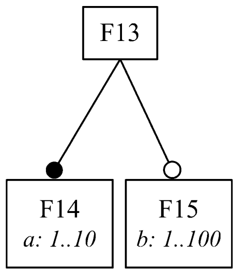


Fig. 13 A simple extended feature model

possible to transform the extended feature models into equivalent basic feature models by enumerating the values in the attribute domains and introducing a new child feature for each enumeration. Such an approach can facilitate the use of the existing theory and tools on the extended feature models that we consider. However, theoretical possibility does not necessarily lead to efficient and practical solutions. For instance, consider the simple extended feature model given in Fig. 13.

A straightforward way to adopt the strategy mentioned above is to introduce 10 new features for *F14* and 100 new features for *F15*, one for each value in the domains of these features' attributes, and connect these new features to their parents using the *alternative* decomposition relation. Therefore, the resulting transformed model will contain a total of 113 features. Even this simple example indicates that such an approach can introduce massive complexity to the model. Moreover, the complexity introduced will not be limited to the new features.

Assume that the model in Fig. 13 has a cross-tree constraint "If *F15.b* $\leq$ 80, then *F15* requires F14.a > 8." This constraint should also be adopted in the transformation by introducing new constraints. Thus, 8 new basic constraints must be introduced for each enumeration of *F15* for the values 1 through 80 such as "F15$_{\_1}$ excludes F14$_{\_1}$,"…, "F15$_{\_1}$ excludes F14$_{\_8}$,"…, "F15$_{\_80}$ excludes F14$_{\_1}$,"…, "F15$_{\_80}$ excludes F14$_{\_8}$." In this case, 640 basic cross-tree constraints would be added to the transformed model for a single cross-tree constraint involving attributes, and it could be even worse if the numbers in the constraint were different.

In the above case, an extremely simple extended feature model is analyzed in which there are only 3 features that only have single attributes. It is obvious that the situation can become much more complicated when there are more features, and some features have multiple attributes. Hence, such a straightforward approach would not be feasible in practice.

A study by Karataş et al. [30] attempts to eliminate attributes from cross-tree constraints to transform extended feature models into equivalent basic feature models in order to benefit from existing tools. This work uses a more efficient strategy than the straightforward approach discussed above, as it does not introduce a new feature for every value in the attributes' domains, but groups these values according to the cross-tree constraints and introduces a single feature for each group. Even in this case, assuming there are *n* cross-tree constraints involving attributes and *k* attributes that appear in cross-tree constraints, the growth in the number of features will be O(*nk*), which can be very high. Moreover, this attempt assumes that there are harsh restrictions on the structure of the cross-tree constraints (e.g., in the constraints that have the form "Feature requires Expression" Expression is not allowed to include attributes of two different features), which significantly limits the expressivity of the complex cross-tree constraints that can be used.

Apart from the complexities that can be caused by the number of new features to be introduced, other issues that would cause problems in a transformation attempt include the transformation of global constraints and complex cross-tree constraints. The study presented in this article allows extended feature models to include global constraints as described in [28], and a rich set of complex cross-tree constraints as described in [31]. Global constraints can provide significant efficiency in certain analysis operations [28]. However, it is not clear how global constraints can be expressed without using attributes. Similarly, it will be quite cumbersome to express complicated cross-tree constraints, such as "If $X.a < 10$ and ($Y.b < 20$ or $Z.c \neq 30$) then *W* requires $S.d = 40$ and $T.e < 50$ or $S.f + Q.g < P.h - R.i$," which are allowed in this study, in terms of basic cross-tree constraints.

Assuming a method is found to overcome all the issues discussed in this section in an efficient way, the practicality of such a method would still be questionable. As the model will include artificial features and constraints contrived for the purpose of transformation, which are not evident in the actual requirements, the model would look unnatural.

9.2 Challenges ahead

The list of existing analysis operations reformulated in this article is by no means exhaustive in terms of the known analysis operations to be revised. For instance, the analysis operation *degree of orthogonality* must be reformulated as the outcome of this operation is dependent on the type of products to be considered. This operation can be reformulated using the ideas presented for the reformulation of the *commonality* analysis operation. However, it is not easy to foresee how analysis operations such as *explanations* and *anomalies detection* will be affected from the introduced extended concepts, and thus, there is a need for rigorous dedicated studies. Thus, the analysis operations revised in this article are examples intending to provide insight for some of the well-known, frequently used operations. The examination of the remainder of the existing analysis operations that have been reported is subject to further research.

The new analysis operations introduced in this article can be useful in order to take full advantage of extended feature models. However, more analysis operations can be introduced, for instance, an analysis operation to find the *conditionally dead attribute values* can be defined in a similar fashion to the definition of the *dead attribute values* analysis operation. The extra constraint representativeness (ECR) analysis operation, which is defined for features, can be adopted to compute the *ECR for attributes (ECR-A)*. Hence, it is possible to define new analysis operations as required using the definitions in this article as a guide.

Feature models for realistic product families can easily grow to include hundreds of features, attributes and cross-tree constraints. Manually applying the concepts presented in this article to real-life problems can be very time-consuming and error prone; hence, automated support is highly desirable. There are many popular commercial and academic tools available for feature model management and analyses. Adopting the ideas proposed in this article to the existing tools and developing new tools with such capabilities are subject to further study.

As Capilla et al. [10] suggest, dynamic SPLs must provide late binding times. C-products would be very useful in this context to delay the binding of variability involving attributes, especially if the resources available to derive F-products are limited. If there is a C-product, then there is no need to check the validity of an F-product derived from the C-product, since every derived F-product will be valid; thus, the processing power requirements to check the validity can be eliminated together with the storage space needed to store the constraints. However, the degree of variability provided by C-products is limited to variability involving attributes. Configurations, on the other hand, can possess more variability, i.e., variability involving features in addition to variability involving attributes. The introduction of a new partial configuration type, which will guarantee that all products derived from this type of configurations will be valid, would prove to be very useful. For instance, such configurations can delay not only the binding of attribute values, but also binding of the features that are not specified in the configuration. Clearly, this topic needs further research and a rigorous examination of the requirements that such configurations should meet and the ramifications they will bring.

It may be desirable to employ a post-processing step on the results provided by some of the analysis operations in order to produce refined and, consequently, more efficient outcomes. For instance, assume that the model given in Fig. 14 has a single cross-tree constraint:

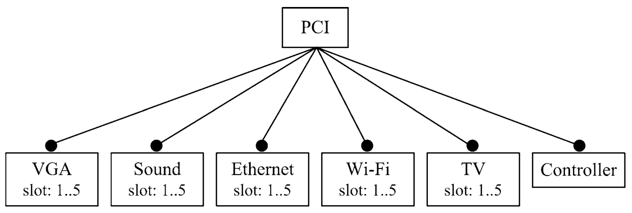


Fig. 14 A simple extended feature model

- *Controller* requires *VGA.slot* $<$ *Sound.slot* and *Sound.slot* $<$ *Ethernet.slot* and *Ethernet.slot* $<$ *Wi-Fi.slot* and *Wi-Fi.slot* $<$ *TV.slot*

This model represents only a single F-product, which is ({PCI, VGA, Sound, Ethernet, Wi-Fi, TV, Controller}, {VGA.slot = 1, Sound.slot = 2, Ethernet.slot = 3, Wi-Fi.slot = 4, TV.slot = 5}), and no proper P-products (consequently no C-products). However, when P-products are considered, the result will be enormous: 1,048,575 P-products. More interesting at this point is that although this number is very high, all of the resulting P-products will be equivalent, since the only F-product represented by the model will be derivable from each one of these P-products. Thus, if only one of the semantically equivalent products is to be output, then the result would be much more concise, since only one P-product would exist in the outcome.

This example suggests that the outcomes of some of the analysis operations such as *all products*, *number of products*, and *filter* can be processed to eliminate all but one of the semantically equivalent products. Considering that two F-products (or two C-products) are semantically equivalent only if these two products are the same, such post-processing will not have any effect on the sets of F-products or C-products. For P-products and proper P-products, on the other hand, it is necessary to investigate the types of benefits to be gained and the types of complications that can arise from employing such post-processing steps.

10 Conclusions

This article has elaborated the role of feature attributes in the modeling and analysis of commonality and variability. The existing semantic groundwork has been extended to enable the full use of extended feature models for the analysis of variability and for product derivation. This extension is predicated upon treating feature attributes as first-class entities, that is, on the same footing as features.

We discussed the effects of the inclusion of feature attributes in cross-tree constraints on feature models. Including attributes in complex cross-tree constraints increases the expressive power of the constraints, hence allows the modeling of complex requirements that can originate in realistic scenarios. As existing configuration and product definitions do not include information regarding the values that can be assigned to the attributes of the included features, they are not sufficiently powerful to deal with situations that can arise due to an extended type of variability, variability involving attributes. Therefore, we proposed extending the existing configuration and product definitions to overcome this deficiency. It is worth noting that the extended definitions include the existing definitions as special cases (i.e., they assume that the specified domains of the attributes in the model serve as their A-domains); thus, we achieve full backward compatibility. As the expressive power of configurations and products increase with these extensions, the extended definitions would promote the wider use of extended feature models.

Next, we elaborated on the ramifications of these extensions and presented categorizations to identify the configurations and products with similar properties. In order to provide a semantic foundation for our proposal, we examined the validity of different types of configurations and products based on constraint satisfaction. We also showed that there exists an ordering relation on the valid configurations (including products as maximal elements), and the resulting partially ordered set is indeed a meet semilattice. Since the theory of partial orders and lattices is a mature mathematical subject, we believe that this connection will prove to be valuable for the formal foundations of variability management.

We also presented a number of examples showing how the existing analysis operations must be revised in order to meet the new conditions that arise from this study. The list of examples presented is, of course, not exhaustive; however, we hope that these examples, along with the new analysis operations proposed, will serve as a start-up guide for those that will conduct further research on this subject.

Finally, we shared some experiences gained from an ongoing industrial R&D project that employs an extended feature model. Although the experiences we present in this study are at an introductory level, we believe that they will provide an insight into the ideas in action for practitioners in the field. As researchers and practitioners adopt the ideas presented in this study, a collective know-how on the effects of these ideas in action will emerge.

Appendix 1

Proof of Proposition 1 We first start by observing that the subset relation is also reflexive (i.e., $X \subseteq X$), antisymmetric (i.e., $X \subseteq Y$ and $Y \subseteq X$ imply $X = Y$), and transitive (i.e., $X \subseteq Y$ and $Y \subseteq Z$ imply $X \subseteq Z$), where X, Y, and Z are sets. We define the configuration ordering using the subset relation between the sets of included features and the sets of excluded features of two valid configurations; thus, it quickly follows that the configuration ordering relation is a partial order, as follows.

Let $C_1 = (I_1, E_1)$, $C_2 = (I_2, E_2)$ and $C_3 = (I_3, E_3)$ be three valid configurations with respect to M. The configuration ordering relation is

- (reflexive) $C_1 \leq C_1$, since $I_1 \subseteq I_1$ and $E_1 \subseteq E_1$.
- (antisymmetric) $C_1 \leq C_2$ and $C_2 \leq C_1$ imply $C_1 = C_2$, since $I_1 \subseteq I_2$ and $I_2 \subseteq I_1$ imply $I_1 = I_2$, and $E_1 \subseteq E_2$ and $E_2 \subseteq E_1$ imply $E_1 = E_2$
- (transitive) $C_1 \leq C_2$ and $C_2 \leq C_3$ imply $C_1 \leq C_3$, since $I_1 \subseteq I_2$ and $I_2 \subseteq I_3$ imply $I_1 \subseteq I_3$, and $E_1 \subseteq E_2$ and $E_2 \subseteq E_3$ imply $E_1 \subseteq E_3$ □

Proof of Proposition 2 We first start by observing that $C_{inf} \leq C_1$ (since $I_1 \cap I_2 \subseteq I_1$ and $E_1 \cap E_2 \subseteq E_1$) and $C_{inf} \leq C_2$ (since $I_1 \cap I_2 \subseteq I_2$ and $E_1 \cap E_2 \subseteq E_2$). Thus, C_{inf} is a lower bound of $\{C_1, C_2\}$.

Then, we show that C_{inf} is the greatest lower bound of $\{C_1, C_2\}$. Assume that there exists another configuration $C = (I, E)$, where $C \neq C_{inf}$, such that C is a lower bound of $\{C_1, C_2\}$ and $C_{inf} \leq C$. Due to the assumption $C \neq C_{inf}$, at least one of the following two conditions must be true: *(i)* $I \neq I_1 \cap I_2$, *(ii)* $E \neq E_1 \cap E_2$.

We will first examine *(i)*. Since we have assumed that $C_{inf} \leq C$, then, due to the definition of the configuration ordering relation, it must be the case that $I_1 \cap I_2 \subseteq I$. If *(i)* is true, then there must be at least one feature f such that f is a member of I, but not $I_1 \cap I_2$. f is not a member of $I_1 \cap I_2$ if and only if $f \notin I_1$ or $f \notin I_2$. However, since we assumed that C is a lower bound of $\{C_1, C_2\}$, it must be the case that $C \leq C_1$ and $C \leq C_2$. If $C \leq C_1$, then $I \subseteq I_1$; thus, every member of I, including f, must also be a member of I_1; hence, $f \notin I_1$ cannot be true. For similar reasons, $f \notin I_2$ cannot also be true. Therefore, it is not possible to find such an f. Consequently, *(i)* cannot be true.

Similarly, it is easy to show that *(ii)* cannot be true as well. Thus, our assumption on the existence of such a C fails. Therefore, we conclude that $C_{inf} = (I_1 \cap I_2, E_1 \cap E_2)$ is the infimum of $\{C_1, C_2\}$ (i.e., $C_{inf} = C_1 \wedge C_2$). □

Proof of Proposition 3 Let the set of all valid configurations represented by M be S. Since M is not void, it represents at least one valid product; thus, S will be non-empty. It follows from Proposition 1 that S is equipped with the partial order relation $\leq$; thus, it is a partially ordered set. It follows from Proposition 2 that for any two valid configurations $C_1 = (I_1, E_1)$ and $C_2 = (I_2, E_2)$ such that $C_1, C_2 \in S$, $C_1 \wedge C_2$ exists and is equal to $(I_1 \cap I_2, E_1 \cap E_2)$. Therefore, S is a meet semilattice. □

Proof of Proposition 4 We first start by observing that $C_{inf} \leq C_1$ (since $I_1 \cap I_2 \subseteq I_1$, $E_1 \cap E_2 \subseteq E_1$, and

$A_{1\text{-}1} \subseteq A_{1\text{-}1} \cup A_{2\text{-}1}, \ldots, A_{1\text{-}n} \subseteq A_{1\text{-}n} \cup A_{2\text{-}n}$). Similarly, $C_{inf} \leq C_2$ is also true. Thus, C_{inf} is a lower bound of $\{C_1, C_2\}$.

Then, we show that C_{inf} is the greatest lower bound of $\{C_1, C_2\}$. Assume that there exists another configuration $C = (I, E, \Delta)$, where $C \neq C_{inf}$, such that C is a lower bound of $\{C_1, C_2\}$ and $C_{inf} \leq C$. Due to the assumption $C \neq C_{inf,}$ at least one of the following three conditions must be true: *(i)* $I \neq I_1 \cap I_2$, *(ii)* $E \neq E_1 \cap E_2$, *(iii)* $\Delta \neq \Delta_{inf}$. The proofs that show *(i)* and *(ii)* cannot be true can be directly borrowed from the proof of Proposition 2. Therefore, we will focus on *(iii)*.

Let $\Delta = \{a_1 \in A_1, \ldots, a_n \in A_n\}$. Since we have assumed that $C_{inf} \leq C$, then, due to the definition of the extended configuration ordering relation, it must be the case that $A_1 \subseteq A_{1\text{-}1} \cup A_{2\text{-}1}, \ldots, A_n \subseteq A_{1\text{-}n} \cup A_{2\text{-}n}$. If *(iii)* is true, then there must be at least one value v such that v is a member of $A_{1\text{-}x} \cup A_{2\text{-}x,}$ but not A_x, where $1 \leq x \leq n$. v is a member of $A_{1\text{-}x} \cup A_{2\text{-}x}$ if and only if $v \in A_{1\text{-}x}$ or $v \in A_{2\text{-}x}$. However, since we assumed that C is a lower bound of $\{C_1, C_2\}$, it must be the case that $C \leq C_1$ and $C \leq C_2$. If $C \leq C_1$, then $A_{1\text{-}x} \subseteq A_x$; thus, a value that is not member of A_x cannot be a member of $A_{1\text{-}x}$; hence, $v \in A_{1\text{-}x}$ cannot be true. For similar reasons, $v \in A_{2\text{-}x}$ cannot also be true. Therefore, it is not possible to find such a v and *(iii)* cannot be true.

Thus, our assumption on the existence of such a C fails. Therefore, we conclude that C_{inf} is the infimum of $\{C_1, C_2\}$ (i.e., $C_{inf} = C_1 \wedge C_2$). □

Proof of Proposition 5 Let the set of all valid configurations represented by M be S. Since M is not void, it represents at least one valid product; thus, S will be non-empty. As discussed above, S is equipped with the partial order relation $\leq$; thus, it is a partially ordered set. It follows from Proposition 4 that for any two configurations $C_1 = (I_1, E_1, \Delta_1)$ and $C_2 = (I_2, E_2, \Delta_2)$ such that $C_1, C_2 \in S$, $C_1 \wedge C_2$ exists and is equal to $(I_1 \cap I_2, E_1 \cap E_2, \Delta_{inf})$, where $\Delta_1 = \{a_1 \in A_{1\text{-}1}, \ldots, a_n \in A_{1\text{-}n}, \ldots\}$, $\Delta_2 = \{a_1 \in A_{2\text{-}1}, \ldots, a_n \in A_{2\text{-}n}, \ldots\}$, and $\Delta_{inf} = \{a_1 \in A_{1\text{-}1} \cup A_{2\text{-}1}, \ldots, a_n \in A_{1\text{-}n} \cup A_{2\text{-}n}\}$. Therefore, S is a meet semilattice. □

References

1. Asikainen T, Mannisto T, Soininen T (2007) Kumbang: a domain ontology for modelling variability in software product families. Adv Eng Inform 21:23–40
2. Bagheri E, Ensan F (2014) Dynamic decision models for staged software product line configuration. Requir Eng 19:187–212
3. Batory D (2005) Feature models, grammars, and propositional formulas. In: Proceedings of the 9th International software product line conference (SPLC'05), LNCS 3714, pp 7–20
4. Benavides D, Ruiz-Cortés A, Trinidad P (2004) Coping with automatic reasoning on software product lines. Second Groningen workshop on software variability management
5. Benavides D, Ruiz-Cortés A, Trinidad P (2005) Using constraint programming to reason on feature models. In: Proceedings of the 17th international conference on software engineering and knowledge engineering (SEKE'05), pp 677–682
6. Benavides D, Segura S, Ruiz-Cortés A (2010) Automated analysis of feature models 20 years later: A literature review. Inform Syst 35:615–636
7. Benavides D, Trinidad P, Ruiz-Cortés A (2005) Automated reasoning on feature models. In: Proceedings of the 17th international conference on advanced information systems engineering (CAISE'05), LNCS 3520, pp 491–503
8. Bosch J, Florijn G, Greefhorst D, Kuusela J, Obbink H, Pohl K (2001) Variability issues in software product lines. In: Proceedings of the 4th international workshop on product family engineering (PFE'01), pp 11–19
9. Botterweck G, Thiel S, Nestor D, Abid S, Cawley C (2008) Visual tool support for configuring and understanding software product lines. In: Proceedings of the 12th international software product line conference (SPLC'08), IEEE computer society, pp 77–86
10. Capilla R, Bosch J, Trinidad P, Ruiz-Cortés A, Hinchey M (2014) An overview of dynamic software product line architectures and techniques: observations from research and industry. J Syst Softw 91:3–23
11. Chen L, Babar MA (2011) A systematic review of evaluation of variability management approaches in software product lines. Inform Softw Tech 53:344–362
12. Cheng L, Wu C, Zhang Y, Wu H, Li M, Maple C (2012) A survey of localization in wireless sensor network. Int J Distrib Sens N 962523(1–962523):12
13. Czarnecki K, Bednasch T, Unger P, Eisenecker U (2002) Generative programming for embedded software: an industrial experience report. In: Proceedings of the ACM SIGPLAN/SIGSOFT conference on generative programming and component engineering (GPCE'02), LNCS 2487, pp 156–172
14. Czarnecki K, Grünbacher P, Rabiser R, Schmid K, Wasowski A (2012) Cool features and tough decisions: a comparison of variability modeling approaches. In: Proceedings of the 6th international workshop on variability modelling of software-intensive systems (VAMOS'12), pp 173–182
15. Czarnecki K, Helsen S, Eisenecker U (2005) Formalizing cardinality-based feature models and their specialization. Software Process Improv Pract 10:7–29
16. Davey BA, Priestley HA (2002) Introduction to lattices and order, 2nd edn. Cambridge University Press, Cambridge
17. Deursen A, Klint P (2002) Domain-specific language design requires feature descriptions. J Comput Inform Tech 10:1–17
18. Dumitru H, Gibiec M, Hariri N, Leland-Huang J, Mobasher B, Castro-Herrera C, Mirakhorli M (2011) On-demand feature recommendations derived from mining public product descriptions. In: Proceedings of the 33rd international conference on software engineering (ICSE'11), ACM, pp 181–190
19. Fernandez-Amoros D, Heradio R, Cerrada J (2009) Inferring information from feature diagrams to product line economic models. In: Proceedings of the 13th international software product line conference (SPLC'09), pp 41–50
20. Goldoni E, Savioli A, Risi M, Gamba P (2010) Experimental analysis of RSSI-based indoor localization with IEEE 802.15.4. In: Proceedings of the 16th European wireless conference (EW'10), pp 71–77
21. Griss M, Favaro J, d'Alessandro M (1998) Integrating feature modeling with the RSEB. In: Proceedings of the 5th international conference on software reuse (ICSR'98), pp. 76-85

22. Halmans G, Pohl K (2003) Communicating the variability of a software product family to customers. Softw Syst Model 2:15–36
23. Hartmann H, Trew T (2008) Using feature diagrams with context variability to model multiple product lines for software supply chains. In: Proceedings of the 12th international software product line conference (SPLC'08), IEEE computer society, pp 12–21
24. Höfner P, Khedri R, Möller B (2011) An algebra of product families. Softw Syst Model 10:161–182
25. ITU-R (2011) Impact of industrial, scientific and medical (ISM) equipment on radiocommunication services ITU-R SM.2180
26. Kang K, Cohen S, Hess J, Novak W, Peterson S (1990) Feature-oriented domain analyses (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh
27. Kang K, Kim S, Lee J, Kim K (1998) FORM: a feature-oriented reuse method with domain-specific reference architectures. Ann Soft Eng 5:143–168
28. Karataş AS, Oğuztüzün H, Doğru A (2010) Global constraints on feature models. In: Proceedings of the 16th international conference on principles and practices of constraint programming (CP'10), LNCS 6308, pp 537–551
29. Karataş AS, Oğuztüzün H, Doğru A (2010) Mapping extended feature models to constraint logic programming over finite domains. In: Proceedings of the 14th international software product line conference (SPLC'10), LNCS 6287, pp 286–299
30. Karataş AS, Oğuztüzün H, Doğru A (2011) Transforming cross-tree relations involving attributes into basic constraints in feature models, In: Proceedings of the 5th international conference on application of information and communication technologies (AICT'11), pp 157–161
31. Karataş AS, Oğuztüzün H, Doğru A (2013) From extended feature models to constraint logic programming. Sci Comput Program 78:2295–2312
32. Lee J, Kang, K (2006) A feature-oriented approach for developing dynamically reconfigurable products in product line engineering. In: Proceedings of the 10th international software product line conference (SPLC'06), IEEE CS Press, pp 131–140
33. Lee J, Kang K, Sawyer P, Lee H (2013) A holistic approach to feature modeling for product line requirements engineering. Requir Eng. doi:10.1007/s00766-013-0183-6
34. Loesch F, Ploedereder E (2007) Restructuring variability in software product lines using concept analysis of product configurations. In: Proceedings of the 11th European conference on software maintenance and reengineering (CSMR'07), IEEE computer society, pp 159–170
35. Lopez-Herrejon RE, Batory D (2001) A standard problem for evaluating product-line methodologies, In: Proceedings of the 3rd international conference on generative and component-based software engineering (GCSE'01), Springer, pp 10–24
36. Mannion M (2002) Using first-order logic for product line model validation. In: Proceedings of the 2nd software product line conference (SPLC'02), LNCS 2379, pp 176–187
37. Mao G, Fidan B, Anderson BDO (2007) Wireless sensor network localization techniques. Comput Netw 51:2529–2553
38. Niu N, Savolainen J, Yu Y (2010) Variability modeling for product line viewpoints integration. In: Proceedings of the IEEE 34th computer software and applications conference, pp 337–346
39. Peng R, Sichhitiu ML (2006) Angle of arrival localization for wireless sensor networks. In: Proceedings of the 3rd annual IEEE conference on sensor and ad hoc communications and networks, pp 374–382
40. Pohl K, Böckle G, Linden F (2005) Software product line engineering: foundations, principles, and techniques. Springer, Berlin
41. Riebisch M, Bollert K, Streitferdt D, Philippow I (2002) Extending feature diagrams with UML multiplicities. 6th conference on integrated design and process technology (IDPT'02), Pasadena, California
42. Ryssel U, Ploennigs J, Kabitzsch K (2011) Extraction of feature models from formal contexts. SPLC Workshops, ACM
43. Schobbens P, Trigaux JC, Heymans P, Bontemps Y (2007) Generic semantics of feature diagrams. Comput Netw 51:456–479
44. Simos M et al (1996) Software technology for adaptable reliable systems (STARS) organization domain modeling (ODM) guidebook version 2.0. STARS-VC-A025/001/00, Manassas, VA, lockheed martin tactical defense systems
45. Sinnema M, Deelstra S (2006) Classifying variability modeling techniques. Inform Softw Tech 49:717–739
46. Software Productivity Consortium Services Corporation (1993) Reuse-driven software processes guidebook version 02.00.03, Technical Report SPC-92019-CMC
47. Stoiber R, Glinz M (2010) Supporting stepwise, incremental product derivation in product line requirements engineering. In: Proceedings of the 4th international workshop on variability modeling of software-intensive systems (VaMoS'10), pp 77–84
48. Trinidad P, Ruiz-Cortés A (2009) Abductive reasoning and automated analysis of feature models: how are they connected? In: Proceedings of the 3rd international workshop on variability modelling of software-intensive systems (VAMOS'09), pp 145–153
49. Wang H, Li YF, Sun J, Zhang H, Pan J (2007) Verifying feature models using OWL. J Web Semant 5:117–129
50. Wang J, Ghosh RK, Das SK (2010) A survey on sensor localization. J Control Theory Appl 8:2–11
51. Webber DL, Gomaa H (2004) Modeling variability in software product lines with the variation point model. Sci Comput Program 53:305–331
52. White J, Doughtery B, Schmidt D, Benavides D (2009) Automated reasoning for multi-step software product-line configuration problems. In: Proceedings of the 13th software product line conference (SPLC'09), pp 11–20