Intersection of Feature Models

Pim van den Broek
Department of Computer Science
University of Twente
Enschede, the Netherlands
Tel: +31 534893762
pimvdb@ewi.utwente.nl

ABSTRACT
In this paper, we present an algorithm for the construction of the intersection of two feature models. The feature models are allowed to have "requires" and "excludes" constraints, and should be parent-compatible. The algorithm is applied to the problem of combining feature models from stakeholders with different viewpoints.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms
Algorithms, Design

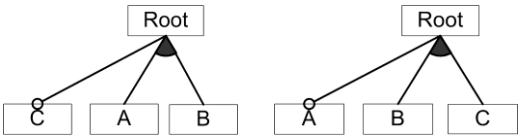
Keywords
feature models, intersection, viewpoints

1. INTRODUCTION
Feature models are used to specify the variability of software product lines [1,2]. The semantics of a feature model is a set of configurations of a product line, where each configuration is a set of features [3]. When performing calculations with product lines, the challenge is to avoid to explicitly compute their set of configurations, as the size of this set may be exponential in the number of features [4]. In this paper, we consider the problem of constructing an intersection of feature models. An intersection of two feature models is a feature model whose set of products is the intersection of the sets of products of its constituents. Recently, several authors have considered the problem of determining an intersection of feature models; we will review these briefly in section 3, and identify their shortcomings. Our algorithm will overcome these shortcomings.

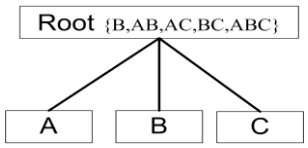
We will consider feature models which are based on trees, i.e. there is no sharing of features, since these are the models that are most widely used in practice. The cross-tree constraints we will consider are the familiar "requires" and "excludes" constraints.

SPLC - Vol. II, September 02 - 07 2012, Salvador, Brazil

We will not stick to one particular graphical notation for our feature models. Instead, we assume for each feature in a feature model a set of possible sets of child features; whereas graphical notations are used to denote these possible sets of child features, there does not exist a notation for every such set. Therefore, where we adopt common notations where possible, we use explicit set notation if needed. Consider for example the following feature models:



The first feature model has or-group {A,B} and optional feature C. The possible sets of child features of the root (suppressing Root) are {A,B,AB,AC,BC,ABC}. The second feature model has or-group {B,C} and optional feature A. Here the possible sets of child features of the root are {B,C,BC,AB,AC,ABC}. The possible sets of child features of the intersection therefore is {B,AB,AC,BC,ABC}. For this set, there is no graphical notation in any of the graphical formalisms. We therefore have to denote the intersection of the feature models with explicit set notation, for instance as follows:



We require different feature models to be parent-compatible. Suppose feature A has parent feature B in one feature model, and it has parent feature C in another feature model. Which parent should A have in the intersection? To avoid problems of this kind we assume parent-compatibility. Two feature models are parent-compatible if equal features have equal parents (or both are the root). In our method, feature models FM1 and FM2 can only have an intersection FM3 when FM1 and FM2 are parent-compatible, and FM3 will be parent-compatible with both FM1 and FM2.

The paper is organized as follows. In the next section we briefly provide a formal definition of feature models. In section 3 we examine two existing intersection methods for feature models, and identify their short-comings. In section 4 we present our algorithm. Section 5 gives an example. In section 6 we apply our method to the case where feature models from stakeholders with different viewpoints must be combined. Finally section 7 concludes the paper with a discussion.

2. FEATURE MODELS

Features in this paper are considered to be atomic entities. A feature model FM in this paper consists of:

1. A set of features FE(FM); if it is non-empty, then it contains a designated element called the root feature, denoted by RO(FM).
2. A tree structure on FE(FM) whose root is RO(FM); for each feature F □ RO(FM) its parent is denoted by PA(FM,F).
3. For each feature F of FE(FM) a set PC(FM,F) whose elements are sets of features. Elements of this set are the sets of features which are considered to be possible sets of children of F.
4. A set RE(FM) of ordered pairs of features; these are the "requires" constraints.
5. A set EX(FM) of pairs of features; these are the "excludes" constraints.

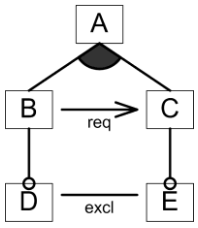
We will take the liberty to omit the first argument of FE, RO, PA, PC, RE and EX in case where it is clear from the context to which feature model they refer.

A nonempty subset P of FE(FM) is a product of FM if and only if the following four conditions are satisfied:

1. $\forall F \in P: F \neq RO \Rightarrow PA(F) \in P$.
2. $\forall F \in P: \{G \in P \mid PA(G) = F\} \in PC(F)$.
3. $\forall (A,B) \in RE: A \in P \Rightarrow B \in P$.
4. $\forall (A,B) \in EX: A \in P \Rightarrow B \notin P$.

The semantics of a feature model is the set of all its products [3]; we denote the semantics of FM by SEM(FM). The feature model whose set of features is empty is denoted by NIL; it has no products. When SEM(FM1) = SEM(FM2) then FM1 and FM2 are said to be equivalent, denoted by FM1 ~ FM2.

Feature model languages normally use a graphical notation or annotations to express PC. Our specification will be independent of the notational conventions of feature modeling languages. In this paper we adopt the convention that features are denoted by capitals, and products by strings. An example of a feature model is:



Here FE = {A,B,C,D,E}, RO = A, PA(B) = A, PA(C) = A, PA(D) = B, PA(E) = C, PC(A) = {{B},{B,C},{C}}, PC(B) = {{},{D}}, PC(C) = {{},{E}}, PC(D) = {{}}, PC(E) = {{}}, RE = {(B,C)} and EX = {(D,E)}. The semantics of this feature model is {ABC, ABCD, ABCE, AC, ACE}.

3. INTERSECTIONS OF FEATURE MODELS IN THE LITERATURE

In [3] and [6] methods have been given for the construction of an intersection of feature models. Let FM1 and FM2 be the feature models to be intersected.

The method of Schobbens et al. [3] goes as follows. First the features of FM2 are renamed, so that the feature sets of FM1 and FM2 are disjoint. Then a new root is taken, which has the roots of FM1 and FM2 as mandatory children. Finally requires constraints are added between any pair of corresponding features in both subtrees. This method works also for feature models with shared features, and for feature models which are not parent-compatible. However, it doesn't work when FE(FM1) ≠ FE(FM2), which means that features which are not in FE(FM1) ∩ FE(FM2) have to be eliminated first. The main drawback of this method is the duplication of features, and the multitude of requires constraints. For instance, if FM1 = FM2, the intersection contains two identical subtrees, and all corresponding features of these identical subtrees are connected via two requires constraints. In our method, given in the next section, the intersection of FM1 and FM1 is just FM1.

The method of Acher et al. [6] assumes that each feature in a feature model belongs to one of four types: a type A feature has only mandatory children, a type B feature has only optional children, the children of a type C feature form an Or-group and the children of a type D feature form an Alternative-group. The type of a feature in the intersection, given its types in FM1 and FM2, is then read from a table; its children are obtained as the intersection of the sets of children in FM1 and FM2. This method does not work, however, if either of the features is of type A. For instance, the intersection of an A-type feature with children {A,B} and an A-type feature with children {B,C} is NIL, and not an A-type feature with child B. A second drawback of this method is that it does not take into account that the intersection of sets of children can be empty, which may be erroneous. For instance, the intersection of a C-type feature with children {A,B} and a C-type feature with children {C,D} is not a C-type feature without child features, but is NIL. Finally, this method does not consider cross-tree constraints. Our method, given in the next section, does take requires and excludes constraints into account, and does not suffer from the problems identified above.

A different approach to the construction of an intersection of feature models is to use the methods of Czarnecki and Wasovski [7]. Feature models can be translated to propositional formulas, and the conjunction of two such formulas represents the intersection of the corresponding feature models. Their method to retrieve this intersection from its propositional formula requires, however, user interactivity.

4. INTERSECTION OF FEATURE MODELS

Let FM1 and FM2 be the two feature models to be intersected. Let T1 be the set of products whose features belong to FE(FM1) and satisfy the tree constraints of FM1. Let C1 be the set of products whose features belong to FE(FM1) and satisfy the non-tree constraints of FM1. Likewise we define the sets T2 and C2. Our algorithm is based on the following identity, which is an immediate consequence of the associativity and commutativity of the intersection operator:

$(T1 \cap C1) \cap (T2 \cap C2) = C1 \cap C2 \cap (T1 \cap T2)$

Here (T1 ∩ C1) is the set of products which satisfy all constraints of FM1, i.e. it is equal to SEM(FM1). The left-hand side of the equation is thus SEM(FM1) ∩ SEM(FM2), which is the set of products of the intersection we are constructing. The set (T1 ∩ T2) is the set of products which are common to FM1 and FM2

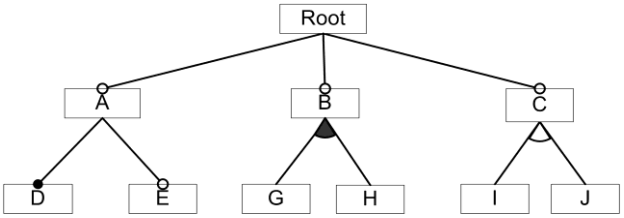
when cross-tree constraints are not considered. Our algorithm therefore consists of two phases: in phase 1, we construct the intersection of FM1 and FM2 without cross-tree constraints, leading to a feature model with semantics (T1 ∩ T2), and in phase 2 we add the cross-tree constraints, such that the semantics (T1 ∩ T2) gets intersected with C1 and C2.

The first phase of the construction goes as follows. First we construct FM3, a feature model with semantics (T1 ∩ T2). We let FE(FM3) be equal to FE(FM1) ∩ FE(FM2), i.e. FM3 consists of the features which are common to FM1 and FM2. The tree structure of FM3 is inherited from FM1 and FM2: for each F in FE(FM3) which is not the root of FM1, we define PA(FM3,F) = PA(FM1,F). This is equal to PA(FM2,F) also, due to the requested parent-compatibility. For each F in FE(FM3) we define PC(FM3,F) to be equal to PC(FM1,F) ∩ PC(FM2,F). The products of FM3 are those products which satisfy the tree-constraints of both FM1 and FM2, so SEM(FM3) = (T1 ∩ T2), as intended. We would be ready if none of the PC(FM3,F) would be empty. Note that if PC(FM3,F) is empty for some F, this does not mean that F is a leaf of the tree. For a leaf F we would have PC(FM3,F) = {{}}, i.e. the only possibility is that there are no children. If PC(FM3,F) is empty, there are no legal possibilities for the children of F. This means that F is a dead feature, i.e there are no products containing F. We may therefore apply a semantics preserving refactoring of FM3, by eliminating F from FM3. Eliminating F means discarding F and all its successors, and removing all sets containing F from PC(FM3,PA(F)). Note that if F is a mandatory child of PA(F), then PC(FM3,PA(F)) becomes empty, and PA(F) must be eliminated as well. If this procedure is finished, we have obtained a feature model FM3, without cross-tree constraints, whose products are the common products of FM1 and FM2 when no cross-tree constraints are taken into account, for which PC(FM3,F) is non-empty for all F from FE(FM3).

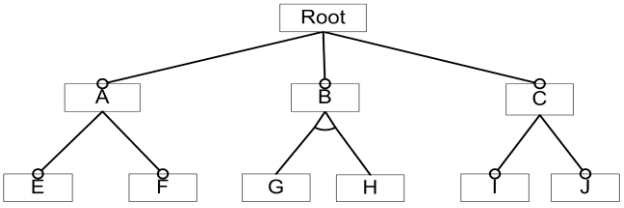
In the second phase of the algorithm, FM3 has to be modified in such a way that products which do not belong to C1 ∩ C2 disappear from its semantics. So we have to apply all cross-tree constraints from FM1 and FM2 to FM3. For constraints whose features are present in FE(FM3) this is easy: these constraints are put in RE(FM3) or EX(FM3). Constraints with features which are not present in FM3 (dangling constraints) need closer attention. Consider first a constraint of the form "A requires B". A product P satisfies this constraint if $A \in P \Rightarrow B \in P$ is true. If FE(FM3) does not contain A, we may replace $A \in P$ by "false", and the constraint is seen to be satisfied. If FE(FM3) does contain A and does not contain B, the constraint becomes $A \in P \Rightarrow$ "false", which is equivalent to $A \notin P$. So the constraint is satisfied if no product contains A; we therefore eliminate feature A. Next consider a constraint of the form "A excludes B". A product P satisfies this constraint if $A \in P \Rightarrow B \notin P$ is true. If FE(FM3) does not contain both A and B, this constraint is seen to be satisfied. So, Summarizing, from all dangling constraints only constraints of the form "A requires B". for which A is present and B is not present need attention, by eliminating A. This elimination can give rise to further eliminations: features whose set of sets of possible children becomes empty must be eliminated (as in phase 1), and if "A requires B" is in RE(FM3) and becomes dangling by an elimination of B, then A must be eliminated also. Cross-tree constraints of other types which become dangling, may simply be removed.

5. EXAMPLE

Let FM1 be the feature model with the following tree:



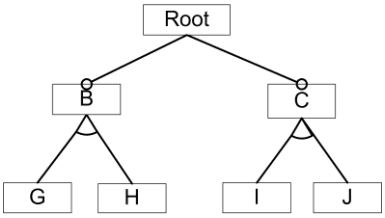
and cross-tree constraints "G requires E" and "E requires J". Let FM2 be the feature model with the following tree:



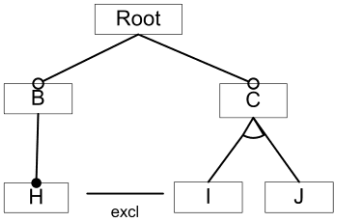
and cross-tree constraints "I excludes F" and "I excludes H". Note that FM1 and FM2 are parent-compatible, so their intersection FM3 can be computed. Starting with phase 1, we compute FE(FM3) = FE(FM1) ∩ FE(FM2), which is equal to {Root,A,B,C,E,G,H,I,J}. For each feature Fe in this set we compute PC(FM3,Fe) = PC(FM1,Fe) ∩ PC(FM2,Fe):

PC(FM3,Root} = {{},A,B,C,AB,AC,BC,ABC}
PC(FM3,A) = {}
PC(FM3,B) = {G,H}
PC(FM3,C) = {I,J}
PC(FM3,Fe) = {{}} for Fe ∈ {E,G,H,I,J}

Since PC(FM3,A) is empty, feature A must be is removed, together with its child feature E. Further, sets containing A are removed from PC(FM3,Root}, which becomes {{},B,C,BC}. The resulting feature model has a graphical notation, which can be determined with a simple algorithm, given in [8]. The result is:



In phase 2, all cross-tree constraints of FM1 and FM2 are considered, in any order. The constraint "G requires E" implies that feature G has to be removed, since feature E is not present anymore. Sets containing G should be removed from PC(FM3,G), which becomes {H}. The constraint "E requires J" can be neglected, as is the constraint "I excludes F". Finally the constraint "I excludes H" must be added to EX(FM3). The following feature model:

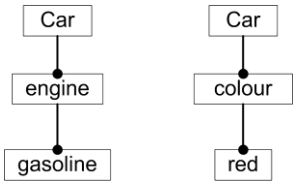


is the final result of the intersection of FM1 and FM2.

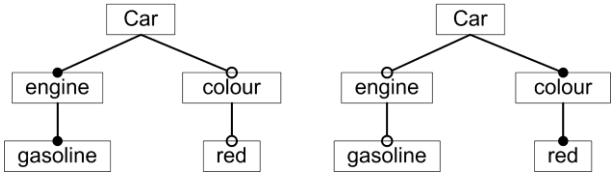
6. APPLICATION

In this section we will show how our algorithm to construct the intersection of two parent-compatible feature models can be used to compose the feature models of stakeholders with different viewpoints. Let FM1 and FM2 be two feature models, developed by stakeholders with different viewpoints. Let FS = FE(FM1) ∪ FE(FM2), FS1 = FE(FM1) \ FE(FM2) and FS2 = FE(FM2) \ FE(FM1), i.e. FS is the set of common features and FS1 (FS2) is the set of features which are specific to FM1 (FM2). Note that for all F in FS except RO(FM1), also PA(F) belongs to FS, due to parent-compatibility. An algorithm to compose both feature models has been given in [8]. Here a product of the composed feature model always is a product of one of the constituent feature models. So, no product can have features from both FS1 and FS2. We consider this to be a unwanted situation. Suppose my wife and I want to buy a new car. My wife has a technical viewpoint: she wants the car to have a gasoline engine, she does not care about the colour of the car. My viewpoint is an aesthetical one: I want the car to have a red colour and I don't care about the type of engine. According to [7], there does not exist a car which suits both of us. However, it is clear that a red car with a gasoline engine should be a perfect choice.

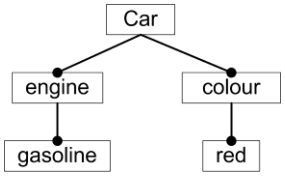
Our proposal is as follows. The composition of FM1 and FM2 is a feature model FM3 whose products are all products which are acceptable to both FM1 and FM2. A product P is acceptable to a feature model FM if P ∩ FE(FM) ⊆ SEM(FM). FM3 can be constructed as follows. We add to FM1 all features from F2 as optional features. From parent-compatibility it is known to which feature each new optional feature should be added as a child. The products of the resulting feature model are all products from FM1 enhanced with features from F2 (in a parent-compatible way with FM2). Likewise we add to FM2 all features from F1 as optional features. The intersection of both new feature models is the wanted composition of FM1 and FM2. As an illustration, consider the following small example alluded to above:



Adding the features from the first model as optional features to the second model, and the features from the second model as optional features to the first model gives:



The intersection of these feature models is the feature model which is a proper composition of feature models FM1 and FM2:



7. DISCUSSION

We have presented an algorithm for the construction of the intersection of two feature models, thereby correcting and extending previous algorithms. The worst case computational time-complexity of our algorithm is quadratic in the number of features. In practice this time-complexity will even be linear, since the number of cross-tree constraints, which can be quadratic in the number of features, is usually smaller than the number of features. This time-complexity is much better than the time-complexity of a recent algorithm to compute the union of feature models [9]. Not only does a feature model whose semantics is equal to the union of the semantics of two given feature models not always exist, its computation in [9] has exponential time complexity. The reason that it is so much easier to compute the intersection than it is to compute the union is the equation given in section 4; an analogous equation for the case of union does not hold.

The correctness of our algorithm can be verified easily. The equation in section 4 shows that the partition in two phases is correct. The correctness of the second phase is explained in section 4. The correctness of the first phase is a straightforward verification that the correct intersection is obtained in case there are no cross-tree constraints. One verifies that a product of the intersection is a product of both constituents, and vice versa. This reasoning needs however the assumption of parent-compatibility of the constituents.

If the constituents are not parent-compatible, it might be that the designers did have different perceptions when modeling their feature models. Consider the following trivial example. One designer has a feature model with four features: root "Car" has mandatory child "colour" which has an Or-group with features "red" and "blue". The other designer has a feature model with three features: root "Car" has an Or-group with features "red" and "yellow". The feature models are not parent-compatible; it seems that the designers have different opinion about explicitly modeling feature "colour". If they reach consensus, their feature models have an intersection containing a red car. As they are, their intersection is empty, which might not be an adequate description of the intentions of the designers. In case of parent-compatibility, designers agree on the parent-child relationships of their common features. We feel that designers should reach such agreement, in order for the intersection of their feature models to be meaningful.

It is straightforward to generalize our method to general constraints. In phase 2 of the algorithm, in each constraint the features which have been eliminated must be replaced by "false".; this is what has been done explicitly in section 4 for "requires" and "excludes" constraints. If a constraint becomes "true" it can be removed; if it becomes "false" the resulting intersection is NIL; if it takes the form "F implies false" then feature F should be eliminated. Constraints of other forms become constraints of the intersection feature model.

8. REFERENCES

[1] Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E. and Peterson, A.S. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University.

[2] Czarnecki, K. and Eisenecker, U. 2000. *Generative Programming: Methods Tools and Applications*. Addison-Wesley.

[3] Schobbens, P. –Y., Heymans, P., Trigaux, J. –Chr. and Bontemps, Y. 2007. Generic semantics of feature diagrams. *Computer Networks* 51, 456-479.

[4] Benavides, D., Segura, S. and Ruiz-Cortés, A. 2010. Automated analysis of feature models 20 years later: a literature review. *Information Systems* 35, 615-636.

[5] Van den Broek, P. and Galvão, I. 2009. Analysis of feature models using generalised feature trees, In: Benavides, D., Metzger, A. and Eisenecker, U. (eds.), *3th International Workshop on Variability Modelling of Software-intensive Systems*, ICB-Research Report 29, University of Duisburg-Essen, 29--36 http://eprints.eemcs.utwente.nl/15045.

[6] Acher, M., Collet, Ph., Lahire, Ph. and France, R. 2010. Composing feature models, In: *Proceedings of the Second International Conference on Software Language Engineering (SLE'09)*, Lecture Notes in Computer Science, vol. 5969, Springer-Verlag, 62-81.

[7] Czarnecki, K. and Wasowski, A. 2007. Feature Diagrams and Logics: There and Back Again. In: *11th International , Software Product Line Conference (SPLC 2007)*, IEEE, 23-34.

[8] Atilgan Aydin, E., Oguztüzün, H., Dogru, A.H. and Karatas, A.S. 2011. Merging Multi-View Feature Models by Local Rules. In*: 9th International Conference on Software Engineering Research, Management and Applications*, IEEE, 140-147

[9] Van den Broek, P., Galvão, I. and Noppen, J. 2010. Merging feature models. In: *14th International Software Product Line Conference*, Volume 2, 83-89. http://eprints.eemcs.utwente.nl/18520/.