Automated Analysis of Feature Models 20 Years Later: A Literature Review[☆]

David Benavides, Sergio Segura and Antonio Ruiz-Cortés

Dpto. de Lenguajes y Sistemas Informáticos, University of Seville
Av. Reina Mercedes s/n, 41012, Seville - Spain

Abstract

Software product line engineering is about producing a set of related products that share more commonalities than variabilities. Feature models are widely used for variability and commonality management in software product lines. Feature models are information models where a set of products are represented as a set of features in a single model. The automated analysis of feature models deals with the computer–aided extraction of information from feature models. The literature on this topic has contributed with a set of operations, techniques, tools and empirical results which have not been surveyed until now. This paper provides a comprehensive literature review on the automated analysis of feature models 20 years after of their invention. This paper contributes by bringing together previously-disparate streams of work to help shed light on this thriving area. We also present a conceptual framework to understand the different proposals as well as categorise future contributions. We finally discuss the different studies and propose some challenges to be faced in the future.

Key words: Feature models, automated analyses, software product lines, literature review

1. Introduction

Mass production is defined as the production of a large amount of standardized products using standardized processes that produce a large volume of the same product in a reduced time to market. Generally, the customers' requirements are the same and no customization is performed (think of Japanese watches of the nineties). After the industrial revolution, large companies started to organise –and are still organising– their production in a mass production environment.

However, in this highly competitive and segmented market, mass production is not enough anymore and *mass customization* is due to become a must for market success. According to Tseng and Jiao [83], mass customization is about "*producing goods and services to meet individual customer's needs with near mass production efficiency*". There are two key parts in this definition. Firstly, mass customization aims to meet as many individual customer's needs as possible (imagine current mobile phones). Secondly, this has to be done while maintaining the highest mass production efficiency as possible. To achieve this efficiency, practitioners propose building products from existing assets that share more commonalities than singularities.

The information systems market is a peculiar branch of industry compared to more traditional branches. Making the parallelism with the history of traditional industries, the industrialization of information systems started with artisanal methods, evolved to mass production and is now pursuing mass customization to succeed in the market. In the software engineering literature, the mass customization of software products is known as *software product lines* [24] or *software product families* [62]. In order to achieve customer's personalization, *software product line engineering* promotes the production of a family of software products from common features instead of producing them one by one from scratch. This is the key change: software product line engineering is about producing families of similar systems rather than the production of individual systems.

Software product lines have found a broad adoption in several branches of software production such as embedded systems for mobile devices, car embedded software and avionics [85]. However, adopting other types of software and systems applications such as desktop, web or data–intensive applications is a current challenge.

An organisation decides to set up a software product line and faces the following issues, How is a particu-

[☆]A very preliminay version of this paper was published in Jornadas de Ingeniería del Software y Bases de Datos (JISBD'06)

Email address: {benavides,sergiosegura,aruiz}@us.es (David Benavides, Sergio Segura and Antonio Ruiz-Cortés) lar product specified?, and How is the software product line itself specified? When this question was first posed, there was ample evidence for a solution: in other industries product lines are specified in terms of *features*. Products in a software product line are differentiated by their features, where a feature is an increment in program functionality [7]. Individual products are specified using features, software product lines are specified using *feature models*.

Feature model languages are a common family of visual languages to represent software product lines [69]. The first formulation of a feature model language is due by Kang et al. in 1990 [43]. A feature model captures software product line information about common and variant features of the software product line at different levels of abstraction. A feature model is represented as a hierarchically arranged set of features with different relationships among those features. It models all possible products of a software product line in a given context. Unlike traditional information models, feature models not only represent a single product but a family of them in the same model.

The automated analysis of feature models is about extracting information from feature models using automated mechanisms [7]. Analysing feature models is an error–prone and tedious task, and it is infeasible to do manually with large–scale feature models. It is an active area of research and is gaining importance in both practitioners and researchers in the software product line community [7, 9]. Since the introduction of feature models, the literature has contributed with a number of operations of analysis, tools, paradigms and algorithms to support the analysis process.

In this article, we present a structured literature review [46, 94] of the existing proposals for the automated analysis of feature models. The main contribution of this article is to bring together previously–scattered studies to set the basis for future research as well as introduce new researchers and practitioners in this thriving area. We present a conceptual framework to understand the different proposals and classify new contributions in the future. 53 primary studies were analysed from where we report 30 operations of analysis and 4 different groups of proposals to automate those operations. As a result of our literature review, we also report some challenges that remain open to research.

The main target audience of this literature review are researchers in the field of automated analysis, tool developers or practitioners who are interested in analysis of feature models as well as researchers and professionals of information systems interested in software product lines, their models and analyses.

The remainder of the paper is structured as follows: Section 2 presents feature models in a nutshell. Section 3 presents the method used in the literature review. Section 4 describes the conceptual framework that we use to classify primary studies and define recurring concepts in the paper. Section 5 presents the different analysis operations. Section 6 presents the automated techniques used for analysis. Section 7 discusses the results of performance analysis of feature models. Section 8 discusses the results obtained and describes some challenges to be faced in the future. Finally, Section 9 presents some conclusions.

2. Feature Models

A feature model represents the information of all possible products of a software product line in terms of features and relationships among them. Feature models are a special type of information model widely used in software product line engineering. A feature model is represented as a hierarchically arranged set of features composed by:

1. relationships between a parent (or compound) feature and its child features (or subfeatures).
2. cross–tree (or cross–hierarchy) constraints that are typically inclusion or exclusion statements in the form: *if feature F is included, then features A and B must also be included (or excluded).*

Figure 1 depicts a simplified feature model inspired by the mobile phone industry. The model illustrates how features are used to specify and build software for mobile phones. The software loaded in the phone is determined by the features that it supports. According to the model, all phones must include support for *calls*, and displaying information in either a *basic*, *colour* or *high resolution* screen. Furthermore, the software for mobile phones may optionally include support for *GPS* and multimedia devices such as *camera*, *MP3* player or both of them.

Feature models are used in different scenarios of software production ranging from model driven development [81], feature oriented programming [6], software factories [40] or generative programming [27], all of them around software product line development. Although feature models are studied in software product line engineering, these information models can be used in different contexts ranging from requirements gathering [23] to data model structures, hence the potential importance of feature models in the information systems domain.

The term *feature model* was coined by Kang et al. in the FODA report back in 1990 [43] and has been one of the main topics of research in software product

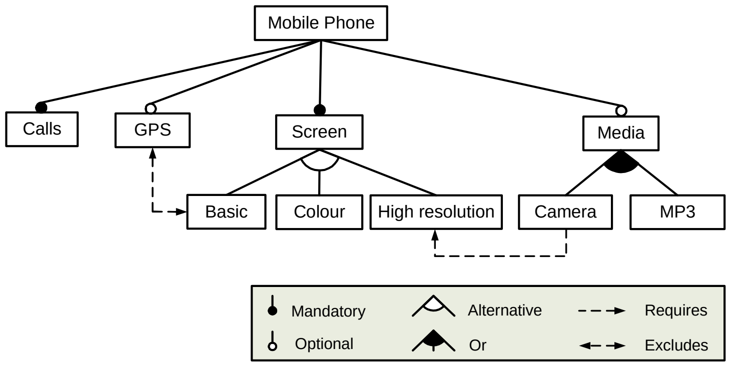


Figure 1: A sample feature model

lines since then. There are different feature model languages. We refer the reader to [69] for a detailed survey on the different feature model languages. Below, we review the most well known notations for those languages.

2.1. Basic feature models

We group as basic feature models those allowing the following relationships among features:

- **Mandatory.** A child feature has a mandatory relationships with its parent when the child is included in all products in which its parent feature appears. For instance, every mobile phone system in our example must provide support for *calls*.

- **Optional.** A child feature has an optional relationship with its parent when the child can be optionally included in all products in which its parent feature appears. In the example, software for mobile phones may optionally include support for *GPS*.

- **Alternative.** A set of child features have an alternative relationship with their parent when only one feature of the children can be selected when its parent feature is part of the product. In the example, mobile phones may include support for a *basic*, *colour* or *high resolution* screen but only one of them.

- **Or.** A set of child features have an or-relationship with their parent when one or more of them can be included in the products in which its parent feature appears. In Figure 1, whenever *Media* is selected, *Camera*, *MP3* or both can be selected.

Notice that a child feature can only appear in a product if its parent feature does. The root feature is a part of all the products within the software product line. In addition to the parental relationships between features, a feature model can also contain cross-tree constraints between features. These are typically in the form:

- **Requires.** If a feature A requires a feature B, the inclusion of A in a product implies the inclusion of B in such product. Mobile phones including a *camera* must include support for a *high resolution* screen.

- **Excludes.** If a feature A excludes a feature B, both features cannot be part of the same product. *GPS* and *basic* screen are incompatible features.

More complex cross-tree relationships have been proposed later in the literature [5] allowing constraints in the form of generic propositional formulas, e.g. “A and B implies not C”.

2.2. Cardinality–based feature models

Some authors propose extending FODA feature models with UML-like multiplicities (so-called *cardinalities*) [28, 65]. Their main motivation was driven by practical applications [26] and “conceptual completeness”. The new relationships introduced in this notation are defined as follows:

- **Feature cardinality.** A feature cardinality is a sequence of intervals denoted [$n..m$] with n as lower bound and m as upper bound. These intervals determine the number of instances of the feature that can be part of a product. This relationship may be used as a generalization of the original mandatory ($[1, 1]$) and optional ($[0, 1]$) relationships defined in FODA.

- **Group cardinality.** A group cardinality is an interval denoted $\langle n..m\rangle$, with n as lower bound and m as upper bound limiting the number of child features that can be part of a product when its

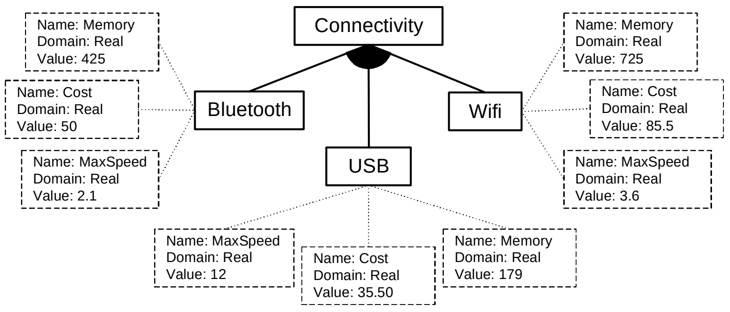


Figure 2: A sample extended feature model

parent feature is selected. Thus, an alternative relationship is equivalent to a $\langle 1..1 \rangle$ group cardinality and an or–relationship is equivalent to $\langle 1..N \rangle$, being N the number of features in the relationship.

2.3. Extended feature models

Sometimes it is necessary to extend feature models to include more information about features. This information is added in terms of so–called *feature attributes*. This type of models where additional information is included are called *extended, advanced or attributed feature models*.

FODA [43], the seminal report on feature models, already contemplated the inclusion of some additional information in feature models. For instance, relationships between features and feature attributes were introduced. Later, Kang et al. [44] make an explicit reference to what they call "non–functional" features related to feature attributes. In addition, other groups of authors have also proposed the inclusion of attributes in feature models [5, 7, 11, 12, 29, 30, 73, 96]. There is no consensus on a notation to define attributes. However, most proposals agree that an attribute should consist at least of a *name*, a *domain* and a *value*. Figure 2 depicts a sample feature model including attributes using the notation proposed by Benavides et al. in [11]. As illustrated, attributes can be used to specify extra-functional information such as cost, speed or RAM memory required to support the feature.

Extended feature models can also include complex constraints among attributes and features like: "*If attribute A of feature F is lower than a value X, then feature T can not be part of the product*".

3. Review method

We have carried out a literature review in order to examine studies proposing automated analysis of feature models. To perform this review we followed a systematic and structured method inspired by the guidelines of Kitchenham [46] and Webster et al. [94]. Below, we detail the main data regarding the review process and its structure. For further details on the method followed we refer the reader to [13].

3.1. Research questions

The aim of this review is to answer the following research questions:

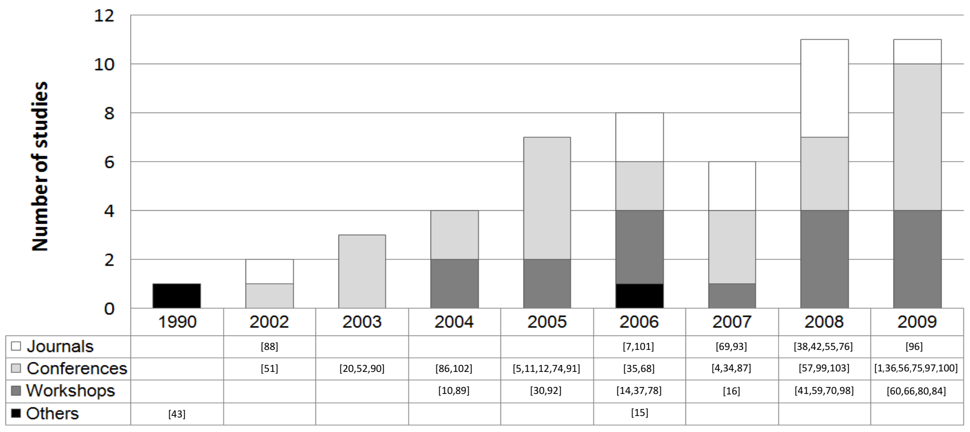
- *RQ1: What operations of analysis on feature models have been proposed?* This question motivates the following sub-questions:
  - What operations have been formally described?
- *RQ2: What kind of automated support has been proposed and how is it performed?* This question motivates the following sub-questions:
  - Which techniques have been proposed to automate the analysis?
  - What is the feature modelling notation supported by each approach?
  - Which analysis operations have been automated?
  - Which proposals present a performance evaluation of their results?

After reviewing all this information we also want to answer a more general question:

- *RQ3: What are the challenges to be faced in the future?*

3.2. Source material

As recommended by Webster et al. [94], we used both manual and automated methods to make a selection of candidate papers in leading journals and conferences and other related events. We reviewed 72 papers, 19 were discarded resulting in a total of **53 papers** that were in the scope of this review. These 53 papers are referred as *primary studies* [46].



	1990	2002	2003	2004	2005	2006	2007	2008	2009
☐ Journals		[88]				[7,101]	[69,93]	[38,42,55,76]	[96]
▨ Conferences		[51]	[20,52,90]	[86,102]	[5,11,12,74,91]	[35,68]	[4,34,87]	[57,99,103]	[1,36,56,75,97,100]
▩ Workshops				[10,89]	[30,92]	[14,37,78]	[16]	[41,59,70,98]	[60,66,80,84]
■ Others	[43]					[15]			

Figure 3: Classification of papers per year and type of publication

Figure 3 classifies primary studies according to the year and type of publication. Of the 53 papers included in the review, 10 were published in journals, 25 in conferences, 16 in workshops, 1 in the formal post–proceeding of a summer school and 1 in a technical report. The graph indicates that there was an important gap between 1990 and 2002 and since then the tendency seems to be ascendant.

3.3. *Inclusion and exclusion criteria*

Articles on the following topics, published between January 1st 1990 and December 31st 2009, were included: *i*) papers proposing any analysis operation on feature models in which the original model is not modified, *ii*) papers proposing the automation of any analysis on feature models, and *iii*) performance studies of analysis operations.

Works of the same authors but with very similar content were intentionally classified and evaluated as separate primary studies for a more rigorous analysis. Later, in the presentation of results, we grouped those works with no major differences.

Some related works were discarded to keep the size and complexity of the review at a manageable level, namely: *i*) papers proposing operations where the input feature model is modified by returning a new feature model, i.e. only operations proposing information extraction where considered, *ii*) papers presenting any application of the analysis of feature models rather than proposing new analyses, and *iii*) papers dealing with the analysis of other kinds of variability models like OVM [62], decision models [67] and further extensions of feature models like *probabilistic feature models* [31].

4. Conceptual framework

In this section, we propose a conceptual framework that attempts to provide a high-level vision of the analysis process and clarify the meaning of various usually ambiguous terms found in the literature. This is the result of the common concepts and practices identified in the primary studies of our review.

As a result of the literature review we found that the automated analysis of feature models can be defined as the *computer–aided extraction of information from feature models*. This extraction is mainly carried out in a two–step process depicted in Figure 4. Firstly, the input parameters (e.g. feature model) are translated into a specific representation or paradigm such as propositional logic, constraint programming, description logic or ad–hoc data structures. Then, off–the–shelf solvers or specific algorithms are used to automatically analyse the representation of the input parameters and provide the result as an output.

The analysis of feature models is performed in terms of *analysis operations*. An operation takes a set of parameters as input and returns a result as output. In addition to feature models, typical input and output parameters are:

- *Configuration*. Given a feature model with a set of features F, a configuration is a 2–tuple of the form (S,R) such that $S, R \subseteq F$ being S the set of features to be selected and R the set of features to be removed such that $S \cap R = \emptyset$.
  - *Full configuration*. If $S \cup R = F$ the configuration is called *full configuration*.
  - *Partial configuration*. If $S \cup R \subset F$ the configuration is called *partial configuration*

  As an example, consider the model in Figure 1 and the full (FC) and partial (PC) configurations described below:

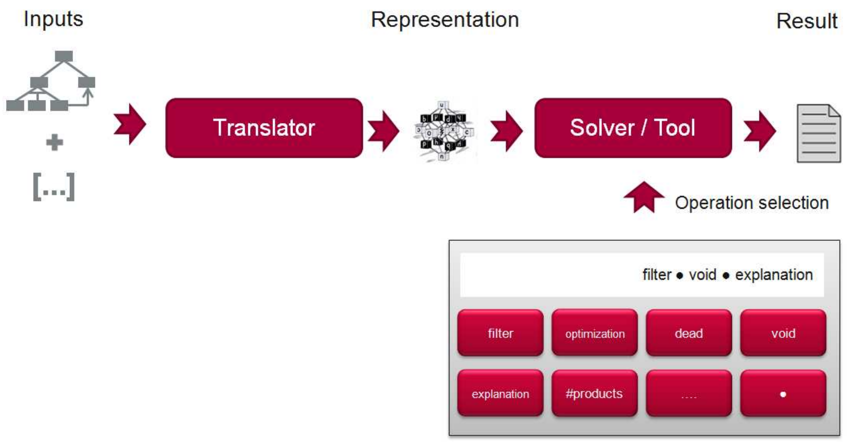


Figure 4: Process for the automated analysis of feature models

```
FC = ({MobilePhone,Calls,Screen,Colour},
      {GPS,Basic,High resolution,Media,Camera,MP3})

PC = ({MobilePhone,Calls,Camera},{GPS})
```

- *Product*. A product is equivalent to a full configuration where only selected features are specified and omitted features are implicitly removed. For instance, the following product is equivalent to the full configuration described above:

```
P = {MobilePhone,Calls,Screen,Colour}
```

5. Analysis operations on feature models

In this section, we answer *RQ1 : What operations of analysis on feature models have been proposed?* For each operation, its definition, an example and possible practical applications are presented.

5.1. Void feature model

This operation takes a feature model as input and returns a value informing whether such feature model is void or not. A feature model is *void* if it represents no products. The reasons that may make a feature model void are related with a wrong usage of cross–tree constraints, i.e. feature models without cross-tree constraints cannot be void.

As an example, Figure 5 depicts a void feature model. Constraint *C-1* makes the selection of the mandatory features *B* and *C* not possible, adding a contradiction to the model because both features are mandatory.

The automation of this operation is especially helpful when debugging large scale feature models in which the manual detection of errors is recognized to be an error-prone and time–consuming task [5, 43, 76]. This operation is also referred to by some authors

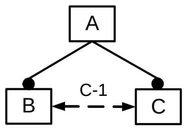


Figure 5: A void feature model

as *"model validation"*, *"model consistency checking"*, *"model satisfiability checking"*, *"model solvability checking"* and *"model constraints checking"*.

5.2. Valid product

This operation takes a feature model and a product (i.e. set of features) as input and returns a value that determines whether the product belongs to the set of products represented by the feature model or not. For instance, consider the products $P1$ and $P2$, described below, and the feature model of Figure 1.

```
P1={MobilePhone,Screen,Colour,Media,MP3}
P2={MobilePhone,Calls,Screen,High resolution,GPS}
```

Product $P1$ is not valid since it does not include the mandatory feature *Calls*. On the other hand, product $P2$ does belong to the set of products represented by the model.

This operation may be helpful for software product line analysts and managers to determine whether a given product is available in a software product line. This operation is sometimes also referred to as *"valid configuration checking"*,*"valid single system"*, *"configuration consistency"*, *"feature compatibility"*, *"product checking"* and *"product specification completeness"*.

5.3. Valid partial configuration

This operation takes a feature model and a partial configuration as input and returns a value informing whether the configuration is valid or not, i.e. a partial configuration is valid if it does not include any contradiction. Consider as an example the partial configurations $C1$ and $C2$, described below, and the feature model of Figure 1.

```
C1 = ({MobilePhone,Calls,Camera}, {GPS,High resolution})
C2 = ({MobilePhone,Calls,Camera}, {GPS})
```

$C1$ is not a valid partial configuration since it selects support for the camera and removes the high resolution screen that is explicitly required by the software product line. $C2$ does not include any contradiction and therefore could still be extended to a valid full configuration.

This operation results helpful during the product derivation stage to give the user an idea about the progress of the configuration. A tool implementing this operation could inform the user as soon as a configuration becomes invalid, thus saving time and effort.

5.4. *All products*

This operation takes a feature model as input and returns all the products represented by the model. For instance, the set of all the products of the feature model presented in Figure 1 is detailed below:

```
P1 = {MobilePhone,Calls,Screen,Basic}
P2 = {MobilePhone,Calls,Screen,Basic,Media,MP3}
P3 = {MobilePhone,Calls,Screen,Colour}
P4 = {MobilePhone,Calls,Screen,Colour,GPS}
P5 = {MobilePhone,Calls,Screen,Colour,Media,MP3}
P6 = {MobilePhone,Calls,Screen,Colour,Media,MP3,GPS}
P7 = {MobilePhone,Calls,Screen,High resolution}
P8 = {MobilePhone,Calls,Screen,High resolution,Media,MP3}
P9 = {MobilePhone,Calls,Screen,High resolution,Media,MP3,Camera}
P10 = {MobilePhone,Calls,Screen,High resolution,Media,Camera}
P11 = {MobilePhone,Calls,Screen,High resolution,GPS}
P12 = {MobilePhone,Calls,Screen,High resolution,Media,MP3,GPS}
P13 = {MobilePhone,Calls,Screen,High resolution,Media,Camera,GPS}
P14 = {MobilePhone,Calls,Screen,High resolution,Media,Camera,MP3,GPS}
```

This operation may be helpful to identify new valid requirement combinations not considered in the initial scope of the product line. The set of products of a feature model is also referred to in the literature as *"all valid configurations"* and *"list of products"*.

5.5. *Number of products*

This operation returns the number of products represented by the feature model received as input. Note that a feature model is void iff the number of products represented by the model is zero. As an example, the number of products of the feature model presented in Figure 1 is *14*.

This operation provides information about the flexibility and complexity of the software product line [11, 30, 88]. A big number of potential products may reveal a more flexible as well as more complex product line. The number of products of a feature model is also referred to in the literature as *"variation degree"*.

5.6. *Filter*

This operation takes as input a feature model and a configuration (potentially partial) and returns the set of products including the input configuration that can be derived from the model. Note that this operation does not modify the feature model but filters the features that are considered.

For instance, the set of products of the feature model in Figure 1 applying the partial configuration $(S, R) = (\{Calls, GPS\}, \{Colour, Camera\})$, being S the set of features to be selected and R the set of features to be removed, is:

```
P1 = {MobilePhone,Calls,Screen,High resolution,GPS}
P2 = {MobilePhone,Calls,Screen,High resolution,Media,MP3,GPS}
```

Filtering may be helpful to assist users during the configuration process. Firstly, users can filter the set of products according to their key requirements. Then, the list of resultant products can be inspected to select the desired solution [30].

5.7. *Anomalies detection*

A number of analysis operations address the detection of anomalies in feature models i.e. undesirable properties such as redundant or contradictory information. These operations take a feature model as input and return information about the anomalies detected. We identified five main types of anomalies in feature models reported in the literature. These are:

Dead features. A feature is *dead* if it cannot appear in any of the products of the software product line. Dead features are caused by a wrong usage of cross–tree constraints. These are clearly undesired since they give the user a wrong idea of the domain. Figure 6 depicts some typical situations that generate dead features.

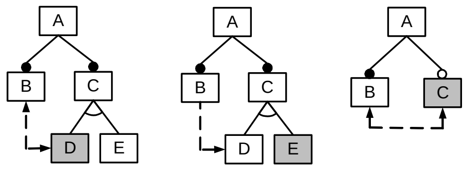


Figure 6: Common cases of dead features. Grey features are dead

Conditionally dead features. A feature is *conditionally dead* if it becomes dead under certain circumstances (e.g. when selecting another feature) [41]. Both unconditional and conditional dead features are often referred to in the literature as *"contradictions"* or *"inconsistencies"*. In Figure 7 feature B becomes dead whenever feature D is selected. Note that, with this definition, features in an alternative relationship are conditionally dead.

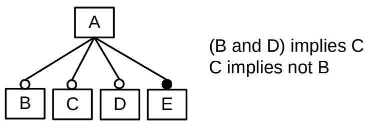


Figure 7: An example of a conditionally dead feature

False optional features. A feature is *false optional* if it is included in all the products of the product line despite not being modelled as mandatory. Figure 8 depicts some examples of false optional features.

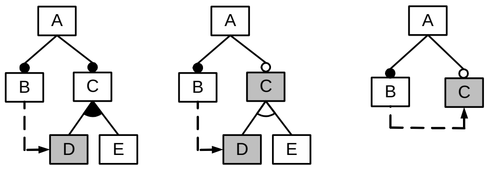


Figure 8: Some examples of false optional features. Grey features are false optional

Wrong cardinalities. A group cardinality is wrong if it cannot be instantiated [80]. These appear in cardinality–based feature models where cross–tree constraints are involved. An example of wrong cardinality is provided in Figure 9. Notice that features *B* and *D* exclude each other and therefore the selection of three subfeatures, as stated by the group cardinality, is not possible.

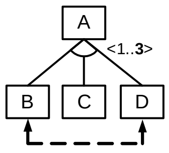


Figure 9: An example of wrong cardinality

Redundancies. A feature model contains redundancies when some semantic information is modelled in multiple ways [89]. Generally, this is regarded as a negative aspect since it may decrease the maintainability of the model. Nevertheless, it may also be used as a means of improving readability and comprehensibility of the model. Figure 10 depicts some examples of redundant constraints in feature models.

5.8. Explanations

This operation takes a feature model and an analysis operation as inputs and returns information (so-called explanations) about the reasons of *why* or *why not* the corresponding response of the operation [80]. Causes are mainly described in terms of features and/or relationships involved in the operation and explanations are ofter related to anomalies. For instance, Figure 11 presents a feature model with a dead feature. A possible explanation for the problem would be *"Feature D is dead because of the excludes constraint with feature B"*. We refer the reader to [80] for a detailed analysis of explanation operations.

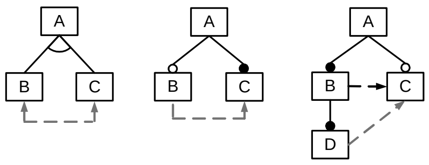


Figure 10: Some examples of redundancies. Gray constraints are redundant

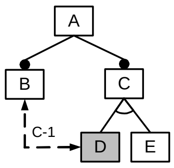


Figure 11: Grey feature is dead because relationship C–1

Explanations are a challenging operation in the context of feature model error analysis, (a.k.a. feature model debugging) [7, 76, 80]. In order to provide an efficient tool support, explanations must be as accurate as possible when detecting the source of an error, i.e. it should be minimal. This becomes an even more challenging task when considering extended feature models and relationships between feature attributes.

5.9. Corrective explanations

This operation takes a feature model and an analysis operation as inputs and returns a set of corrective explanations indicating changes to be made in the original inputs in order to change the output of the operation. In general, a *corrective explanation* provides suggestions to solve a problem, usually once this has been detected and explained.

For instance, some possible corrective explanations to remove the dead feature in Figure 11 would be *"remove excludes constraint C-1"* or *"model feature B as optional"*. This operation is also referred to in the literature as *"corrections"*.

5.10. Feature model relations

These operations take two different feature models as inputs and returns a value informing how the models are related. The set of features in both models are not necessarily the same. These operations are useful for determining how a model has evolved over time. Thüm et al. [75] classify the possible relationships between two feature models as follows:

Refactoring. A feature model is a refactoring of another one if they represent the same set of products while having a different structure. For instance, model in Figure 12(b) is a refactoring of model in Figure 12(a) since they represent the same products i.e. {{*A*,*B*},{{*A*,*B*,*C*}, {*A*,*B*,*D*},{*A*,*B*,*C*,*D*}}. Refactorings are useful to restructure a feature model without changing its semantics. When this property is fulfilled the models are often referred to as *"equivalent"*.

Generalization. A feature model, *F*, is a generalization of another one, *G*, if the set of products of *F* maintains and extends the set of products of *G*. For example, feature model in Figure 12(c) is a generalization of the model in Figure 12(a) because it adds a new product ({*A*}) without removing an existing one. Generalization occurs naturally while extending a software product line.

Specialization. A feature model, *F*, is a specialization of another one, *G*, if the set of products of *F* is a subset of the set of products of *G*. For example, Figure 12(d) depicts a specialization of the model in Figure 12(a) since it removes a product from the original model ({*A*,*B*,*C*,*D*}) and adds no new ones.

Arbitrary edit. There is no explicit relationship between the input models, i.e. there are non of the relationships defined above. Models in Figure 12(a) and Figure 12(e) illustrate an example of this. Thüm et al. [75] advise avoiding arbitrary edits and replacing these by a sequence of specialization, generalizations and refactorings edits for a better understanding of the evolution of a feature model.

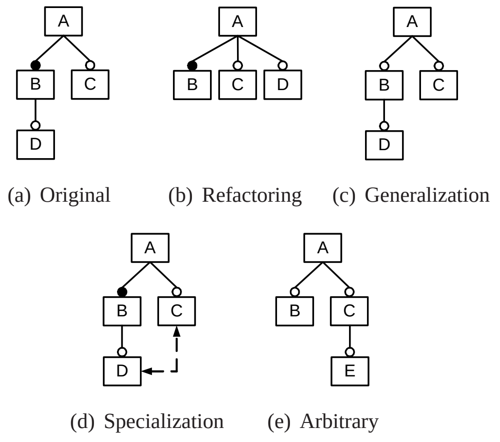


Figure 12: Types of relationships between two feature models

5.11. Optimization

This operation takes a feature model and a so-called objective function as inputs and returns the product fulfilling the criteria established by the function. An objective function is a function associated with an optimization problem that determines how good a solution is.

This operation is chiefly useful when dealing with extended feature models where attributes are added to features. In this context, optimization operations may be used to select a set of features maximizing or minimizing the value of a given feature attribute. For instance, mobile phones minimizing connectivity cost in Figure 2 should include support for *USB* connectivity exclusively, i.e. *USB* is the cheapest.

5.12. Core features

This operation takes a feature model as input and returns the set of features that are part of all the products in the software product line. For instance, the set of core features of the model presented in Figure 1 is {*MobilePhone*,*Calls*,*Screen*}.

Core features are the most relevant features of the software product line since they are supposed to appear in all products. Hence, this operation is useful to determine which features should be developed in first place [77] or to decide which features should be part of the core architecture of the software product line [61].

5.13. Variant features

This operation takes a feature model as input and returns the set of variant features in the model [80]. Variant features are those that do not appear in all the products of the software product line. For instance, the set of variant features of the feature model presented in Figure 1 is {*Basic*,*Colour*,*High resolution*,*Media*,*Camera*, *MP3*,*GPS*}.

5.14. Atomic sets

This operation takes a feature model as input and returns the set of atomic sets of the model. An *atomic set* is a group of features (at least one) that can be treated as a unit when performing certain analyses. The intuitive idea behind atomic sets is that mandatory features and their parent features always appear together in products and therefore can be grouped without altering the result of certain operations. Once atomic sets are computed, these can be used to create a reduced version of the model simply by replacing each feature with the atomic set that contains it.

Figure 13 depicts an example of atomic sets computation. Four atomic sets are derived from the original model, reducing the number of features from 7 to 4.

Note that the reduced model is equivalent to the original one since both represent the same set of products.

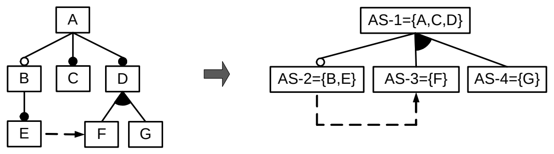


Figure 13: Atomic sets computation

Using this technique, mandatory features are safely removed from the model. This operation is used as an efficient preprocessing technique to reduce the size of feature models prior to their analysis [70, 102].

5.15. Dependency analysis

This operation takes a feature model and a partial configuration as input and returns a new configuration with the features that should be selected and/or removed as a result of the propagation of constraints in the model [55]. As an example, consider the input and output configurations described below and the model in Figure 1.

```
Input = ({MobilePhone,Calls,Camera}, {MP3})
Output =
  ({MobilePhone,Calls,Camera,Media,Screen,High resolution},
   {MP3,Basic,Colour})
```

Features *Screen* and *High resolution* are added to the configuration to satisfy the requires constraint with *Camera*. *Media* is also included to satisfy the parental relationship with *Camera*. Similarly, features *Basic* and *Colour* are removed to fulfil the constraints imposed by the alternative relationship.

This operation is the basis for constraint propagation during the interactive configuration of feature models [55]

5.16. Multi–step configuration

A multi–step configuration problem is defined as the process of producing a series of intermediate configurations, i.e. a configuration path, going from a feature model configuration to another [97]. An analysis operation solving a multi–step configuration problem takes as input a feature model, an initial configuration, a desired final configuration, a number of steps in the configuration path K, a global constraint that can not be violated (usually referred to feature attributes) and a function determining the cost to transition from one configuration in step T to another in step U. As a result, the operation provides an ordered list of K configurations that determines the possible steps that can be taken to go from the initial configuration to the desired final configuration without violating the feature model and global constraints. For a detailed example and a rigorous definition of the operation we refer the reader to [97].

5.17. Other operations

In this section, we group those operations that perform some computation based on the values of previous operations. We also classify in this group those analysis operations proposed as part of other algorithms.

Homogeneity. This operation takes a feature model as input and returns a number that provides an indication of the degree to which a feature model is homogeneous [36]. A more homogeneous feature model would be one with few unique features in one product (i.e. a unique feature appears only in one product) while a less homogeneous one would be one with a lot of unique features. According to [36] it is calculated as follows:

$$Homogeneity = 1 - \frac{\#uf}{\#products}$$

$\#uf$ is the number of unique features in one product and *#products* denotes the total number of products represented by the feature model. The range of this indicator is [0,1]. If all the products have unique features the indicator is 0 (lowest degree of homogeneity). If there are no unique features, the indicator is 1 (highest degree of homogeneity).

Commonality. This operation takes a feature model and a configuration as inputs and returns the percentage of products represented by the model including the input configuration. An as example, consider the partial configurations described below and the model in Figure 1:

```
C1 = {{Calls}, {}}
C2 = {{Calls},{MP3}}
```

The commonality of both configurations is calculated as follows:

$$Comm(C1) = \frac{|filter(FM, \{\{Calls\}, \{\}\})|}{\#products(FM)} = \frac{14}{14} = 1$$

$$Comm(C2) = \frac{|filter(FM, \{\{Calls\}, \{MP3\}\})|}{\#products(FM)} = \frac{7}{14} = 0.5$$

The range of this indicator is [0,1]. Configuration $C1$ appears in 100% of the products whereas $C2$ is included only in 50% of them.

This operation may be used to prioritize the order in which the features are going to be developed [77] or to decide which features should be part of the core architecture of the software product line [61].

Variability factor. This operation takes a feature model as input and returns the ratio between the number of products and 2^n where n is the number of features considered. In particular, 2^n is the potential number of products represented by a feature model assuming that any combination of features is allowed. The root and non-leaf features are often not considered. As an example, the variability of the feature model presented in Figure 1 taking into account only leaf features is:

$$\frac{N.Products}{2^n} = \frac{14}{2^7} = 0.0625$$

An extremely flexible feature model would be one where all its features are optionals. For instance, the feature model of Figure 14 has the following variability factor:

$$\frac{N.Products}{2^n} = \frac{8}{2^3} = 1$$

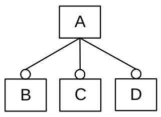


Figure 14: Sample feature model with three optional features

The range of this indicator would depend on the features considered to calculate the factor. The feature model variability can be used to measure the flexibility of the feature model. For instance, a small factor means that the number of combinations of features is very limited compared to the total number of potential products.

Degree of orthogonality. This operation takes a feature model and a subtree (represented by its root feature) as input and returns their degree of orthogonality. Czarnecki et al. [30] defines the *degree of orthogonality* as the ratio between the total number of products of the feature model and the number of products of the subtree. Only local constraints in the subtree are considered for counting the products. For instance, the formula below shows the degree of orthogonality for the subtree *Screen* in Figure 1.

$$Orthogonality(Screen) = \frac{14}{3} = 4.66$$

The range of this indicator is $(0,\infty)$. A high degree of orthogonality indicates that decisions can be taken locally without worrying about the influence in the configuration of other parts of the tree [30].

Extra Constraint Representativeness (ECR). This operation takes a feature model as input and returns the degree of representativeness of the cross-tree constraints in the tree. Mendonça et al. [57, 56] defines the *Extra Constraint Representativeness (ECR)* as the ratio of the number of features involved in cross-tree constraints (repeated features counted once) to the number of features in the feature tree. For instance, ECR in Figure 1 is calculated as follows:

$$ECR = \frac{4}{10} = 0.4$$

The range of this indicator is [0,1]. This operation has been used successfully to design and evaluate heuristics for the automated analysis of feature models [57].

Lowest Common Ancestor (LCA). This operation takes a feature model and a set of features as input and returns a feature that is the lowest common ancestor of the input features. Mendonça et al. [57] defines the *Lowest Common Ancestor (LCA)* of a set of features, $LCA(FM, \{f_1, ..., f_n\})$, as the shared ancestor that is located farthest from the root. In Figure 1, $LCA(FM, \{Basic, Camera\}) = MobilePhone$.

Root features. This operation takes a feature model and a set of features as inputs and returns a set of features that are the *roots* features in the model. Given $l = LCA(FM, \{f_1, ..., f_n\})$, Mendonça et al. [57] defines the roots of a set of features, $Roots(FM, \{f_1, ..., f_n\})$ as the subset of child features of l that are ancestors of $f_1, ..., f_n$. In Figure 1, $Roots(FM, \{Basic, Camera\}) = \{Media, Screen\}$.

6. Automated support

Previously, we presented the different analysis operations that we found in the literature. In this section, we address *RQ2: What kind of automated support has been proposed and how is it performed?* To answer this question, we classified the primary studies in four different groups according to the logic paradigm or method used to provide the automated support. In particular, we next present the group of approaches using *Propositional Logic* (PL), *Constraint Programming* (CP), *Description Logic* (DL), and other contributions not classified in the former groups proposing ad–hoc solutions, algorithms or paradigms.

6.1. Propositional logic based analyses

A *propositional formula* consists of a set of primitive symbols or variables and a set of logical connectives constraining the values of the variables, e.g. $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

A *SAT solver* is a software package that takes as input a propositional formula and determines if the formula is satisfiable, i.e. there is a variable assignment that makes the formula evaluate to true. Input formulas are usually specified in *Conjunctive Normal Form* (CNF). CNF is a standard form to represent propositional formulas that is used by most of SAT solvers where only three connectives are allowed: $\neg, \wedge, \vee$. It has been proved that every propositional formula can be converted into an equivalent CNF formula [25]. SAT solving is a well known NP-complete problem [25], however, current SAT solvers can deal with big problems where in most of the cases the performance is not an issue [53].

Similarly, a *Binary Decision Diagram* (BDD) solver is a software package that takes a propositional formula as input (not necessarily in CNF) and translates it into a graph representation (the BDD itself) which allows determining if the formula is satisfiable and providing efficient algorithms for counting the number of possible solutions [19]. The size of the BDD is crucial because it can be exponential in the worst case. Although it is possible to find a good variable ordering that reduces the size of the BDD, the problem of finding the best variable ordering remains NP-complete [18].

The mapping of a feature model into a propositional formula can change depending on the solver that is used later for analysis. In general, the following steps are performed: *i*) each feature of the feature model maps to a variable of the propositional formula, *ii*) each relationship of the model is mapped into one or more small formulas depending on the type of relationship, in this step some auxiliary variables can appear, *iii*) the resulting formula is the conjunction of all the resulting formulas of step *ii* plus and additional constraint assigning true to the variable that represents the root, i.e. $root \iff true$.

Concrete rules for translating a feature model into a propositional formula are listed in Figure 15. Also, the mapping of our running example of Figure 1 is presented. We may mention that the mapping of the propositional formulas listed in Figure 15 into CNF is straightforward (see [25]).

There are some works in the literature that propose the usage of propositional formulas for the automated analysis of feature models (see Table 3). In these studies the analysis is performed in two steps. Firstly, the feature model is translated into a propositional formula. Then, an off–the–shelf solver is used to automatically analyse the formula and subsequently the feature model. A summary of the solvers used for analysis is shown in Table 1.

Tool	Primary study
SAT Solver [17]	[5, 14, 16, 56, 70, 75]
Alloy [2]	[37, 74]
BDD Solver [95]	[14, 16, 30, 57, 70, 86, 87, 103, 100]
SMV [71]	[101, 102]
Not specified	[51, 52]

Table 1: Propositional logic based tools used for FM analysis

To underline the most important contributions in terms of innovation with respect to prior work we may mention the following studies: Mannion et al. [51, 52] was the first to connect propositional formulas and feature models. Zhang et al. [102] reported a method to calculate *atomic sets*, later explored by Segura [70]. Batory [5] shows the connections among grammars, feature models and propositional formulas, this was the first time that a SAT solver was proposed to analyse feature models. In addition, a *Logic Truth Maintenance System* (a system that maintains the consequences of a propositional formula) was designed to analyse feature models. Sun et al. [74] propose using Z, a formal specification language, to provide semantics to feature models. Alloy was used to implement those semantics and analyse feature models. Benavides et al.[14, 16, 70] propose using a multi–solver approach where different solvers are used (e.g. BDD or SAT solvers) depending on the kind of analysis operations to be performed. For instance, they suggest that BDD solvers seem to be more efficient in general than SAT solvers for counting the number of products of a feature model. Mendonca et al. [57] also used BDDs for analysis and compared different classical heuristics found in the literature for variable ordering of BDDs with new specific heuristics for the analysis of BDDs representing feature models. They experimentally showed that existing BDD heuristics fail to scale for large feature models while their novel heuristics can scale for models with up to 2,000 features. Thüm et al. [75] present an automated method for classifying feature model edits, i.e. changes in an original feature model, according to a taxonomy. The method is based on propositional logic algorithms using a SAT solver and constraint propagation algorithms. Yan et al. [100] propose an optimization method to reduce the size of the logic representation of the feature models by removing irrelevant constraints. Mendonca et al. [56] shows by means of an experiment that the analysis of feature models with similar properties to those found in the literature using SAT solvers is computationally affordable.

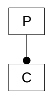
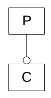
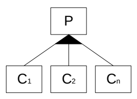
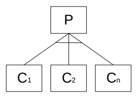
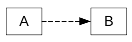
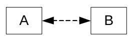
Relationship		PL Mapping	Mobile Phone Example
MANDATORY		$P \leftrightarrow C$	$\text{MobilePhone} \leftrightarrow \text{Calls}$ $\text{MobilePhon e} \leftrightarrow \text{Screen}$
OPTIONAL		$C \rightarrow P$	$\text{GPS} \rightarrow \text{MobilePhone}$ $\text{Media} \rightarrow \text{MobilePhone}$
OR		$P \leftrightarrow (C_1 \vee C_2 \vee \ldots \vee C_n)$	$\text{Media} \leftrightarrow (\text{Camera} \vee \text{MP3})$
ALTERNATIVE		$(C_1 \leftrightarrow (\neg C_2 \wedge \ldots \wedge \neg C_n \wedge P)) \wedge$ $(C_2 \leftrightarrow (\neg C_1 \wedge \ldots \wedge \neg C_n \wedge P)) \wedge$ $(C_n \leftrightarrow (\neg C_1 \wedge \neg C_2 \wedge \ldots \wedge \neg C_{n-1} \wedge P))$	$(\text{Basic} \leftrightarrow (\neg\text{Color} \wedge \neg\text{Highresolution} \wedge \text{Screen})) \wedge$ $(\text{Color} \leftrightarrow (\neg\text{Basic} \wedge \neg\text{Highresolution} \wedge \text{Screen})) \wedge$ $(\text{Highresolution} \leftrightarrow (\neg\text{Basic} \wedge \neg\text{Color} \wedge \text{Screen}))$
IMPLIES		$A \rightarrow B$	$\text{Camera} \rightarrow \text{Highresolution}$
EXCLUDES		$\neg(A \wedge B)$	$\neg(\text{GPS} \wedge \text{Basic})$

Figure 15: Mapping from feature model to propositional logic

6.2. Constraint programming based analyses

A *Constraint Satisfaction Problem* (CSP) [82] consists of a set of variables, a set of finite domains for those variables and a set of constraints restricting the values of the variables. *Constraint programming* can be defined as the set of techniques such as algorithms or heuristics that deal with CSPs. A CSP is solved by finding states (values for variables) in which all constraints are satisfied. In contrast to propositional formulas, CSP solvers can deal not only with binary values (true or false) but also with numerical values such as integers or intervals.

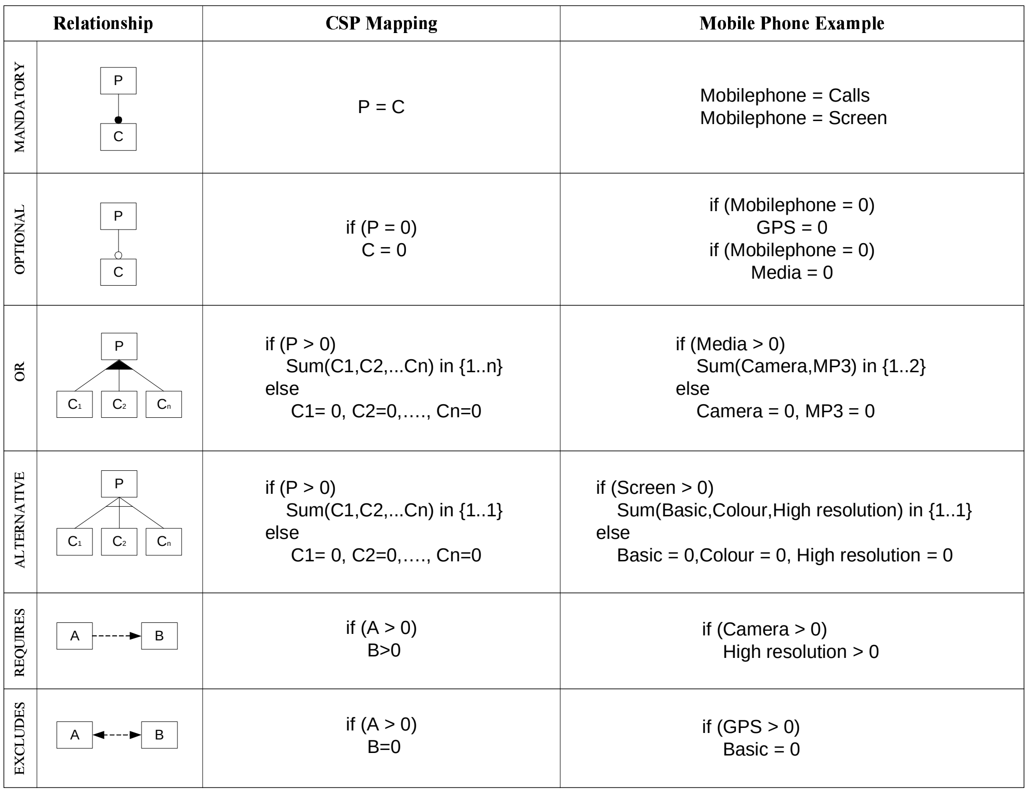
A CSP solver is a software package that takes a problem modelled as a CSP and determines whether there exists a solution for the problem. From a modelling point of view, CSP solvers provide a richer set of modelling elements in terms of variables (e.g. sets, finite integer domains, etc.) and constraints (not only propositional connectives) than propositional logic solvers.

The mapping of a feature model into CSP can vary depending on the concrete solver that is used later for the analysis. In general, the following steps are performed: *i*) each feature of the feature model maps to a variable of the CSP with a domain of 0..1 or TRUE, FALSE, depending on the kind of variable supported by the solver, *ii*) each relationship of the model is mapped into a constraint depending on the type of relationship, in this step some auxiliary variables can appear, *iii*) the resulting CSP is the one defined by the variables of steps *i* and *ii* with the corresponding domains and a constraint that is the conjunction of all precedent constraints plus and additional constraint assigning true to the variable that represents the root, i.e. $root \iff true$ or $root == 1$, depending on the variables' domains.

Concrete rules for translating a feature model into a CSP are listed in Figure 16. Also, the mapping of our running example of Figure 1 is presented.

There are some works in the literature that propose the usage of constraint programming for the automated analysis of feature models (see Table 3). Analyses are performed in two steps. Firstly, the feature model is translated into a CSP. Then, an off–the–shelf solver is used to automatically analyse the CSP and subsequently the feature model. A summary of the solvers used for analysis is shown in Table 2.

Benavides et al. were the first authors proposing the usage of constraint programming for analyses on feature models [10, 11, 12]. In those works, a set of mapping rules to translate feature models into a CSP were provided. Benavides et al. proposals pro-



Relationship		CSP Mapping	Mobile Phone Example

MANDATORY	P, C	P = C	Mobilephone = Calls
			Mobilephone = Screen

OPTIONAL	P, C	if (P = 0)
		C = 0
			if (Mobilephone = 0)
			GPS = 0
			if (Mobilephone = 0)
			Media = 0

OR	P, C1, C2, Cn	if (P > 0)
		Sum(C1,C2,...Cn) in {1..n}
		else
		C1= 0, C2=0,…., Cn=0
			if (Media > 0)
			Sum(Camera,MP3) in {1..2}
			else
			Camera = 0, MP3 = 0

ALTERNATIVE	P, C1, C2, Cn	if (P > 0)
		Sum(C1,C2,...Cn) in {1..1}
		else
		C1= 0, C2=0,…., Cn=0
			if (Screen > 0)
			Sum(Basic,Colour,High resolution) in {1..1}
			else
			Basic = 0,Colour = 0, High resolution = 0

REQUIRES	A, B	if (A > 0) B>0	if (Camera > 0) High resolution > 0

EXCLUDES	A, B	if (A > 0)
		B=0
			if (GPS > 0)
			Basic = 0

Figure 16: Mapping from feature model to CSP

Tool	Proposals
JaCoP [33]	[14, 15, 16, 70]
Choco [21]	[15, 99, 97]
OPL studio [58]	[10, 11, 12]
GNU Prolog [39]	[34]
Not specified	[78, 76]

Table 2: CSP based tools used for FM analysis

vide support for the analysis of extended feature models (i.e. including feature attributes) and the operation of optimization. The authors also provide tool support [16, 79] and they have compared the performance of different solvers when analysing feature models [15, 14, 70]. Trinidad et al. [78, 76] focus on the detection and explanation of errors in feature models based on Reiter's theory of diagnosis [64] and constraint programming. Djebbi et al. [34] propose a method to extract information from feature models in terms of queries. A set of rules to translate feature models to boolean constraints are given. They also describe a tool under development enabling the analysis of feature models using constraint programming. White et al. [99] propose a method to detect conflicts in a given configuration and propose changes in the configuration in terms of features to be selected or deselected to remedy the problem. Their technique is based on translating a feature model into a CSP and adding some extra variables in order to detect and correct the possible errors after applying optimization operations. In [97], White et al. provide support for the analysis of multi–step configuration problems.

6.3. Description logic based analyses

Description logics are a family of knowledge representation languages enabling the reasoning within knowledge domains by using specific logic reasoners [3]. A problem described in terms of description logic is usually composed by a set of concepts (a.k.a. classes), a set of roles (e.g. properties or relationships) and set of individuals (a.k.a. instances).

A description logic reasoner is a software package that takes as input a problem described in description logic and provides facilities for consistency and correctness checking and other reasoning operations.

We found four primary studies proposing the usage of description logic to analyse feature models. Wang et al. [92] were the first to propose the automated analysis of feature models using description logic. In their work, the authors introduce a set of mapping rules to translate feature models into OWL-DL ontologies [32]. OWL-DL is an expressive yet decidable sub language of OWL [32]. Then, the authors suggest using description logic reasoning engines such as RACER[63] to perform automated analysis over the OWL representations of the models. In [93], the authors extend their previous proposal [92] with support for explanations by means of an OWL debugging tool. Fan et al. [35] also propose translating feature models into description logic and using reasoners such as RACER to perform their analyses. In [1], Abo Zaid et al. propose using semantic web technologies to enable the analyses. They use OWL for modelling and the Pellet [22] reasoner for the analysis.

6.4. Other studies

There are some primary studies that are not classified in the former groups, namely: *i*) studies in which the conceptual logic used is not clearly exposed and *ii*) studies using ad–hoc algorithms, paradigms or tools for analysis.

Kang et al. mentioned explicitly the automated analysis of feature models in the original FODA report [43, pag. 70]. A prolog–based prototype is also reported. However, no detailed information is provided to replicate their prolog coding. After the FODA report, Deursen et al. [88] were the first authors proposing some kind of automated support for the automated analysis of feature models. In their work, they propose a textual feature diagram algebra together with a prototype implementation using the ASF+SDF Meta-Environment [47]. Von der Massen et al. [90] present Requiline, a requirement engineering tool for software product lines. The tool is mainly implemented by using a relational data base and ad–hoc algorithms. Later, Von der Massen et al. [91] propose a method to calculate a rough approximation of the number of products of a feature model, which they call *variation degree*. The technique is described using mathematical expressions. In [4], Bachmeyer et al. present *conceptual graph feature models*. Conceptual graphs are a formalism to express knowledge. Using this transformation, they provide an algorithm that is used to compute analysis. Hemakumar [41] proposes a method to statically detect conditional dead features. The method is based on model checking techniques and incremental consistency algorithms. Mendonça et al. [54, 55] study dependencies among feature models and cross–tree constraints using different techniques obtaining a noticeable improvement in efficiency. Gheyi et al. [38] present a set of algebraic laws in feature models to check configurability of feature model refactorings. They use the PVS tool to do some analysis although this is not the main focus of the paper. White et al. [96] present an extension of their previous work [98]. The same method is presented but giving enough details to make it reproducible since some details were missed in their previous work. The method is called *Filtered Cartesian Flattering* which maps the problem of optimally selecting a set of features according to several constraints to a *Multi–dimensional Multi–choice Knapsack Problem* and then they apply several existing algorithms to this problem that perform much faster while offering an approximate answer. Van den Broek et al. [84] propose transforming feature models into generalised feature trees and computing some of their properties. A *generalised feature tree* is a feature model in which cross-tree constraints are removed and features can have multiple occurrences. Some algorithms and an executable specification in the functional programming language Miranda are provided. The strength of their proposal lies in the efficiency of the analysis operation. Fernandez et al. [36] propose an algorithm to compute the total number of products on what they call *Neutral Feature Trees*, trees that allow complex cross-tree constraints. Computing the total number of products the authors are also able to calculate the *homogeneity* of a feature tree as well as the *commonality* of a given feature. They finally compare the computational complexity of their approach with respect to previous work.

6.5. Summary and analysis of operations and support

A summary of the analysis operations (RQ1) and automated support (RQ2) identified in the literature is shown in Table 3. Operations are listed horizontally and ordered by the total number of papers mentioning it. Primary studies are listed vertically. Related works of the same author are grouped in a single column. Primary studies are grouped according to the paradigm they use for the analyses as follows: *i*) Propositional Logic (PL), *ii*). Constraint Programming (CP) *iii*) Description Logic (DL), *iv*) works that integrate more than one paradigm and/or solver (Multi), *v*) studies that use their own tools not categorized in the former groups (Others), and *vi*) proposals that present different operations but do not provide automated support for them (No support).

The cells of the matrix indicate the information about a primary study in terms of operations supported. Cells marked with '+' indicate that the proposal of the column provides explicit support for the operation of the row. We use the symbol '$\sim$' for proposals with no automated support for the corresponding operation but explicit definition of it. We also highlight the primary study that first proposed an operation using the symbols '$\oplus$' (when support is provided) and '$\ominus$' (when no support is provided). To fully

	Batory [5]	Czarnecki *et al.* [30]	Gheyi *et al.* [37]	Mannion *et al.* [51, 52]	Mendonca *et al.* [57]	Mendonca *et al.* [56]	Sun *et al.* [74]	Thüm *et al.* [75]	van der Storm [86, 87]	Zhang *et al.* [102, 101]	Zhang *et al.* [103]	Yan *et al.* [100]	Benavides *et al.* [10, 11, 12]	Benavides *et al.* [15]	Djebii *et al.* [34]	Trinidad *et al.* [78, 76]	White *et al.* [99]	White *et al.* [97]	Abo Zaid *et al.* [1]	Fan *et al.* [35]	Wang *et al.* [92, 93]	Benavides *et al.* [14]	Benavides *et al.* [16]	Segura [70]	Bachmeyer *et al.* [4]	Cao *et al.* [20]	Fernandez *et al.* [36]	Hemakumar [41]	Gheyi *et al.* [38]	Kang *et al.* [43]	Mendonca *et al.* [55]	Osman *et al.* [59, 60]	Salinesi *et al.* [66]	Van den Broek *et al.* [84]	Van Deursen *et al.* [88]	Von der Massen *et al.* [90]	Von der Massen *et al.* [91]	White *et al.* [98, 96]	Batory *et al.* [7]	Schobbens *et al.* [42, 68, 69]	Trinidad *et al.* [80]	Von der Massen *et al.* [89]
	PL												CP						DL			Multi			Others														No support			
Void feature model	+	+		+		+	+		+	+	+	+	+	+	+	+				+	+	+	+	+		+			+	⊕		+	+	+	+	+				~	~	
#Products		+		⊕									+	+	+							+	+	+			+							+	⊕		+				~	
Dead features		~				+				+	+	+				+			+											⊕		+	+	+					~		~	~
Valid product	+	+	+	+			+										+				+				+					⊕						+			~	~	~	
All products	+		+	⊕			+						+										+			+								+	⊕						~	
Explanations	+	~					+									+			+		+									⊕		+		+					~		~	
Refactoring			+				⊕	+													+								+											~	~	
Optimization													⊕		+																	+						+	~		~	
Commonality													⊕			+							+				+														~	
Filter		+											⊕		+																			+							~	
Valid partial configuration	+	+							+																					⊕											~	
Atomic sets					+					⊕														+							+											
False optional features										⊕	+					+																									~	⊖
Corrective explanations																	+															+										⊖
Dependency analysis																														⊕	+											
ECR					⊕	+																																				
Generalization			⊕					+																																		
Core features						+																																			⊖	
Variability factor													⊕																												~	
Arbitrary edit								+																																		
Conditional dead features																												+														
Homogeneity																											+															
LCA					+																																					
Muti–step configuration																		+																								
Roots features					+																																					
Specialization								+																																		
Degree of orthogonality		~																																								
Redundancies																																										~
Variant features																																									~	
Wrong cardinalities																																									~	
Feature model notation	B	C	B	B	B	B	B	B	B	B	C	B	B	C	C	B	B	B	B	B	B	B	C	B	B	B	C	B	B	B	C	C	C	B	B	B	B	B	B	C	C	B
Extended feature model													+		+		+	+	+											+		+						+	+		+	
Formalization			+		+	+	+	+		+	+		+			+		+		+					+		+		+			+					+			+		

+ Supported ~ No support ⊕ Supported(first reference) ⊖ No support (first reference) B Basic feature model C Cardinality–based feature models

Table 3: Summary of operations and support

answer the research questions we also extracted some additional information about different aspects of the primary studies, namely: *i*) feature model notations supported: 'B' (basic feature model), 'C' (cardinality–based feature model) *ii*) whether the approach support extended feature models or not, and *iii*) whether the approach is described formally. This information is also reported in the final rows of Table 3.

Table 4 depicts a chronological view of the data presented in Table 3. More specifically, it shows the amount of references to operations, notation, formalization and kind of automated support found in the literature for each year. Vertically, we list all the years where primary studies were published. The last column indicates the total number of primary studies referring the operation, the notation of feature models, the formalization provided and the type of automated support used for analysis. The table also shows the number of new operations proposed each year.

As illustrated in Tables 3 and 4, there are 11 out of 30 operations that only appeared in one primary study. Likewise, 6 operations were treated in more than 10 studies of which 4 were already mentioned in the original FODA report back in 1990 [43]. This denotes, in our opinion, that FODA authors were quite visionary in predicting the importance of automated analysis of feature models and pinpointing some of the most referred operations. We may remark that 11 new operations were proposed in the last two years of our study and 22 of them were referred in 2009 suggesting that the analysis of feature models is an active research field.

Regarding the notation used, 40 out of 53 primary studies used basic feature model notation for the analysis of feature models. However, there seems to be an increasing interest in the analysis of cardinality–based and extended feature models since 2004.

With respect to automated support for analysis, 18 out of 53 studies used propositional logic while only 4 of them used description logic. Constraint programming was referred to in 12 studies leaded by three different groups of authors. We remark that no support for extended feature models was found in the studies using propositional logic. There are also 16 studies proposing ad–hoc solutions and this tendency seems to be in progression in the last years which may suggest that researchers are looking for more specific and efficient algorithms to perform analysis operations.

We also found that there are 22 studies proposing a formal or rigorous definition of analysis operations. This tendency seems to be ascendant since 2004 which may indicate that there is an increasing interest by the research community to accurately define analysis operations.

Explanations are acknowledged to be an important operation for feature model error analysis in the literature [7, 80]. As presented in Sections 5.8 and 5.9, these operations take as input a feature model and an operation and return as a result the source of the errors in the model and the possible actions to correct them respectively. Table 5 shows a detailed view of the operations that haven been used in explanations and corrective explanations. As illustrated, there are only four operations with support for explanations in more than one study. All logical paradigms have been used for explaining different analysis operations. We found that explanations have been largely studied in related problems in the communities of propositional logic, constraint programming and description logic for years. This has provided researchers with helpful guidelines and methods to assist them with the implementation of explanations in the analysis of feature models. We also remark that all the explanations operations refer to the analysis of basic or cardinality–based feature models while we have not found any study dealing with explanations in extended feature models. Only Trinidad et al. [80] attempted an explanation of the optimization operation but no explicit method to support this operation was presented.

7. Performance evaluation

Performance analyses play a key role in the evaluation of the analysis techniques and tools. The results obtained highlight the strengths and weaknesses of the proposals, helping researchers to improve their solutions, identify new research directions and show the applicability of the analysis operations.

Table 7 summarizes the proposals reporting performance results on the analysis of feature models. We consider as performance results any data (e.g. time, memory) suggesting how a proposal behaves in practice. Works based on propositional logic, constraint programming and ad–hoc solutions have presented a similar number of performance evaluations while only one proposal has presented results of description logic based support. Regarding operations, 18 out of 30 analysis operations identified in the literature have been used in performance analyses. However, only 7 of them have been evaluated by more than one proposal, providing some comparable results.

In general terms, the available results suggest that CP-based and PL-based automated support provide similar performance [14, 70]. PL-based solutions relying on BDDs (Binary Decision Diagrams) seem to be an exception as it provides much faster execution times than the rest of known approaches, especially when computing the number of solutions

	1990	2002	2003	2004	2005	2006	2007	2008	2009	Total
	Operations									
Void feature model	░ +	▒ +	▒ +	▒ +	▓ +	▓ +	▓ +	▓ +	▓ +	35
#Products		▒ +	░ +	░ +	▓ +	▒ +	▒ +	░ +	▒ +	16
Dead features	░ +			▒ +	░ ~	▒ +		▒ +	▓ +	17
Valid product	░ +	░ +	▒ +		▓ +	▒ +	▒ +	▒ +	░ ~	17
All products		▒ +	▒ +	░ +	▓ +	░ +	░ +		▒ +	13
Explanations	░ +				▓ +	░ ~	░ +	▒ +	▓ +	13
Refactoring					░ +	▒ +	▒ +	▒ +	▒ +	9
Optimization				░ +	▒ +	░ ~	░ +	▒ +	▒ +	9
Commonality					▒ +	░ +	░ +		▒ +	6
Filter				░ +	▒ +		░ +		▒ +	7
Valid partial configuration	░ +			░ +	▒ +				░ ~	5
Atomic sets				░ +				▒ +		4
False optional features				▒ +		░ +		▒ +	░ ~	6
Corrective explanations				░ ~				▒ +		3
Dependency analysis	░ +							░ +		2
ECR								░ +	░ +	2
Generalization						░ +			░ +	2
Core features									▒ +	2
Variability					▒ +				░ ~	3
Arbitrary edit									░ +	1
Conditional dead features								░ +		1
Homogeneity									░ +	1
LCA								░ +		1
Muti–step configuration									░ +	1
Roots								░ +		1
Specialization									░ +	1
Degree of orthogonality					░ ~					1
Redundancies				░ ~						1
Variant features									░ ~	1
Wrong cardinalities									░ ~	1
New operations	6	2	0	6	4	1	0	4	7	30
	Notation and formalization									
Basic FMs	░ +	▒ +	▒ +	▓ +	▓ +	▓ +	▒ +	▓ +	▓ +	40
Cardinality-based FMs					░ +	▒ +	▒ +	▓ +	▒ +	13
Extended feature models	░ +			░ +	▒ +	░ +	░ +	▒ +	▓ +	13
Formalization				▒ +	▓ +	▓ +	▒ +	▓ +	▓ +	22
	Support									
Propositional logic		░ +	░ +	▒ +	▒ +	▒ +	▒ +	▒ +	▒ +	18
Constraint programming				░ +	▒ +	▒ +	▒ +	▒ +	░ +	12
Description logic					░ +	░ +	░ +		░ +	4
Others	░ +	░ +	▒ +		░ +		░ +	▓ +	▓ +	16

░ 1 study ▒ 2-3 studies ▓ >3 studies

Table 4: Number of primary studies referring operations, notations and support for each year

	Batory [5]	Czarnecki et al. [30]	Sun et al. [74]	Trinidad et al. [78, 76]	White et al. [99]	Abo Zaid et al. [1]	Wang et al. [92, 93]	Kang et al. [43]	Osman et al. [59, 60]	Van den Broek et al. [84]	Batory et al. [7]	Trinidad et al. [80]	Von der Massen et al. [89]
	PL			CP		DL		Others			No		
Valid product	+	~	+		+		+	+			~	~	
Void feature model	+		+	+			+		+	+		~	
Dead features				+		+				+		~	~
Valid partial configuration	+	~						+				~	
False optional				+								~	~
Dependency analysis								+					
Core features												~	
Optimization												~	
Redundancies													~
Variant features												~	
Wrong cardinalities												~	

Table 5: Summary of the proposals reporting explanations

[14, 57, 70, 103]. The major drawback of this technique is the size of the BDD representing the feature model that can be exponential in the worst case. Several authors have worked in the development of new heuristics and techniques to reduce the size of the BDDs used in the analysis of feature models [57, 103]. Others focus on providing automated support using different paradigms in order to combine the best of all of them in terms of performance [14, 16].

A key aspect in the experimental work related to the analysis of feature models is the type of subject problems used for the experiments. We found two main types of feature models used for experimentation: realistic and automatically generated feature models. By *realistic* models we intend those modelling real–world domains or a simplified version of them. Some of the realistic feature models most quoted in the revised literature are: e-Shop [48] with 287 features, graph product line [50] with up to 64 features, BerkeleyDB [45] with 55 features and home integration system product line [11] with 15 features.

Although there are reports from the industry of feature models with hundreds or even thousands of features [7, 49, 72], only a portion of them is typically published. This has led authors to generate feature models automatically to show the scalability of their approaches with large problems. These models are generated either randomly [14, 15, 55, 60, 70, 96, 97, 99, 100, 103] or trying to imitate the properties of the realistic models found in the literature [56, 75]. Several algorithms for the automated generation of feature models have been proposed [70, 75, 100].

In order to understand the relationship between realistic feature models and automatically generated models in experimentation, we counted the number of works using each type by year. The results are shown in Figure 17. For each type of model, we also show the number of features of the largest feature model for each year. The figure shows an increasing trend in the number of empirical works since 2004 being specially notable in the last two years. The first works used small realistic feature models in their experiments. However, since 2006, far more automatically generated feature models than realistic ones have been used. Regarding the size of the problems, there is a clear ascendant tendency ranging from the model with 15 features used in 2004 to the model with 20 000 features used in 2009. These findings reflect an increasing concern to evaluate and compare the performance of different solutions using larger and more complex feature models. This suggests that the analysis of feature models is maturing.

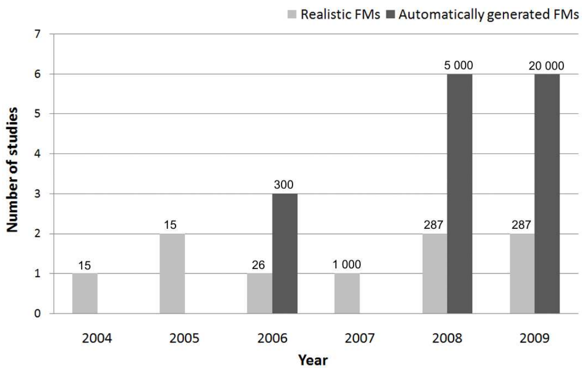


Figure 17: Type and maximum size of the feature models used in performance evaluations for each year

8. Discussions and challenges

In this section, we discuss the results obtained from the literature review. Based on these results, we identify a number of challenges (RQ3) to be addressed in the future. Challenges are part of the authors' own personal view of open questions, based on the analysis presented in this paper.

- *Formal definition of analysis operations*. As we mentioned, most of the proposals define operations in terms of informal descriptions. To implement a tool, it is desirable to have precise definition of the operations. Formal definitions of operations would facilitate both communication among the community and tool development. Schobbens et al. [68, 69] and Benavides [8] have made some progress in this direction. Note that [8] was not included as a primary study because it was not published in a peer reviewed format.

  Challenge 1: Formally describe all the operations of analysis and provide a formal framework for defining new operations.

- *Extended feature model analyses*. Analysis on basic or cardinality–based feature models are covered by most of the studies. However, extended feature models where numerical attributes are included, miss further coverage. When including attributes in feature models the analysis becomes more challenging because not only attribute–value pairs can be contemplated, but more complex relationships can be included like *"feature Camera requires Scree.resolution ≥ 640x480"*. This type of relationships can affect operations of analysis and can include new ones. For instance, the number of products of a feature model can be reduced or increased if these relationships are considered.

	Gheyi et al. [37]	Mendonca et al. [56]	Thüm et al. [75]	Zhang et al. [103]	Yan et al. [100]	Benavides et al. [10, 11, 12]	Benavides et al. [15]	White et al. [99]	White et al. [97]	Wang et al. [93]	Benavides et al. [14]	Segura [70]	Hemakumar [41]	Mendonca et al. [55]	Osman et al. [60]	White et al. [98, 96]
	PL					CP				DL	Multi		Others			
Void feature model		+		+	+		+				+	+				
#Products							+				+	+				
Dead features															+	
Valid product	+							+		+						
All products	+					+										
Explanations										+					+	
Refactoring	+		+													
Optimization																+
Atomic sets												+				
Corrective explanations								+								
Dependency analysis														+		
Generalization	+		+													
Arbitrary edit			+													
Conditional dead features													+			
Muti-step configuration									+							
Specialization			+													

Table 6: Summary of the studies reporting performance results for analysis operations

Challenge 2: Include feature attribute relationships for analyses on feature models and propose new operations of analysis leveraging extended feature models.

- *Performance and scalability of the operations.* Performance testing is being studied more and more and recent works show empirical evidences of the computational complexity of some analysis operations. We believe that a more rigorous analysis of computational complexity is needed. Furthermore, a set of standard benchmarks would be desirable to show how the theoretical computational complexity is run in practice.

Challenge 3: Further studies about computational complexity of analysis.

Challenge 4: Develop standard benchmarks for analysis operations.

- *Tools used for analysis.* As mentioned in Section 6, there are mainly three groups of solvers used for analysis: constraint programming, description logic and propositional logic based solvers. From the primary studies, we detected that proposals using constraint programming–based solvers are the most indicated to deal with extended feature models, i.e. feature models with attributes. Propositional logic–based solvers that use binary decisions diagrams as internal representations seem to be much more efficient for counting the number of products but present serious limitations regarding memory consumption. Description logic–based solvers have not been studied in depth to show their strengths and limitations when analysing feature models. Finally, it seems clear that not all solvers and paradigms will perform equally well for all the identified operations. A characterisation of feature models, operations and solvers seems to be an interesting topic to be explored in the future.

Challenge 5: Study how propositional logic and description logic–based solvers can be used to add attributes on feature models.

Challenge 6: Compare in depth description logic–based solvers with respect to analysis operations and other solvers.

Challenge 7: Characterise feature models, analysis operations and solvers to select the best choice in each case.

9. Conclusions

The automated analysis of feature models is thriving. The extended use of feature models together with the many applications derived from their analysis has allowed this discipline to gain importance among researchers in software product lines. As a result, a number of analysis operations and approaches providing automated support for them are rapidly proliferating. In this paper, we revised the state of the art on the automated analysis of feature models by running a structured literature review covering 53 primary studies and outlining the main advances made up to now. As a main result, we presented a catalogue with 30 analysis operations identified in the literature and classified the existing proposal providing automated support for them according to their underlying logical paradigm. We also provided information about the tools used to perform the analyses and the results and trends related to the performance evaluation of the published proposals. From the analysis of current solutions, we conclude that the analysis of feature models is maturing with an increasing number of contributions, operations, tools and empirical works. We also identified a number of challenges for future research mainly related to the formalization and computational complexity of the operations, performance comparison of the approaches and the support of extended feature models.

Acknowledments

We would like to thank Lamia Abo Zaid, Don Batory, Deepak Dhungana, David Fernández-Amorós, Jose Galindo, Ruben Heradio, Christian Kästner, Abdelrahman Osman Elfaki, Anna Queralt, Fabricia C. Roos, Ernest Teniente, Thomas Thum, Pablo Trinidad, Pim Van den Broek and Wei Zhang for their helpful commments in earlier versions of this article.

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT projects SETI (TIN2009-07366) and WebFactories(TIN2006-00472) and by the Andalusian Government under ISABEL project (TIC-2533)

References

[1] L. Abo, F. Kleinermann, and O. De Troyer. Applying semantic web technology to feature modeling. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1252–1256, New York, NY, USA, 2009. ACM.

[2] Alloy analyzer, `http://alloy.mit.edu/`. accessed January 2010.

[3] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003.

[4] R. Bachmeyer and H. Delugach. A conceptual graph approach to feature modeling. In *Conceptual Structures: Knowledge Architectures for Smart Applications, 15th International Conference on Conceptual Structures, ICCS*, pages 179–191, 2007.

[5] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines Conference*, volume 3714 of *Lecture Notes in Computer Sciences*, pages 7–20. Springer–Verlag, 2005.

[6] D. Batory. A tutorial on feature oriented programming and the ahead tool suite. In *Summer school on Generative and Transformation Techniques in Software Engineering*, 2005.

[7] D. Batory, D. Benavides, and A. Ruiz-Cortés. Automated analysis of feature models: Challenges ahead. *Communications of the ACM*, December:45–47, 2006.

[8] D. Benavides. *On the Automated Analyisis of Software Product Lines using Feature Models. A Framework for Developing Automated Tool Support.* PhD thesis, University of Seville, 2007.

[9] D. Benavides, D. Batory, P. Heymans, and A. Ruiz-Cortés, editors. *First Workshop on Analyses of Software Product Lines*, September 2008.

[10] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Coping with automatic reasoning on software product lines. In *Proceedings of the 2nd Groningen Workshop on Software Variability Management*, November 2004.

[11] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. In *Advanced Information Systems Engineering: 17th International Conference, CAiSE*, volume 3520 of *Lecture Notes in Computer Sciences*, pages 491–503. Springer–Verlag, 2005.

[12] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Using constraint programming to reason on feature models. In *The Seventeenth International Conference on Software Engineering and Knowledge Engineering, SEKE 2005*, pages 677–682, 2005.

[13] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models: A detailed literature review. Technical Report ISA-09-TR-04, ISA research group, 2009. Available at `http://www.isa.us.es/`.

[14] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. In *Managing Variability for Software Product Lines: Working With Variability Mechanisms*, 2006.

[15] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Using java CSP solvers in the automated analyses of feature models. *LNCS*, 4143:389–398, 2006.

[16] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. FAMA: Tooling a framework for the automated analysis of feature models. In *Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, pages 129–134, 2007.

[17] D. Le Berre. SAT4J solver, `http://www.sat4j.org`. accessed January 2010.

[18] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[19] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[20] F. Cao, B. Bryant, C. Burt, Z. Huang, R. Raje, A. Olson, and M. Auguston. Automating feature-oriented domain analysis. In *International Conference on Software Engineering Research and Practice (SERP'03)*, pages 944–949, June 2003.

[21] CHOCO solver, `http://choco.emn.fr/`. accessed January 2010.

[22] Clark and Parsia. Pellet: the open source owl reasoner, http://clarkparsia.com/pellet/. published on line.

[23] A. Classen, P. Heymans, and P.Y. Schobbens. What's in a feature: A requirements engineering perspective. In *Fundamental Approaches to Software Engineering*, volume 4961, pages 16–30. Springer, 2008.

[24] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, August 2001.

[25] S. Cook. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[26] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker. Generative programming for embedded software: An industrial experience report. In *Generative Programming and Component Engineering, ACM SIGPLAN/SIGSOFT Conference, GPCE*, pages 156–172, 2002.

[27] K. Czarnecki and U.W. Eisenecker. *Generative Program-*

ming: Methods, Techniques, and Applications. Addison–Wesley, may 2000. ISBN 0–201–30977–7.

[28] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[29] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[30] K. Czarnecki and P. Kim. Cardinality-based feature modeling and constraints: A progress report. In *Proceedings of the International Workshop on Software Factories At OOPSLA 2005*, 2005.

[31] K. Czarnecki, S. She, and A. Wasowski. Sample spaces and feature models: There and back again. In *proceedings of the Software Product Line Conference (SPLC)*, pages 22–31, 2008.

[32] M. Dean and G. Schreiber. OWL web ontology language reference. W3C recommendation, W3C, February 2004.

[33] JaCoP development team. accessed January 2010.

[34] O. Djebbi, C. Salinesi, and D. Diaz. Deriving product line requirements: the red-pl guidance approach. In *14th Asia-Pacific Software Engineering Conference (APSEC)*, pages 494–501, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[35] S. Fan and N. Zhang. Feature model based on description logics. In *Knowledge-Based Intelligent Information and Engineering Systems, 10th International Conference, KES, Part II*, volume 4252 of *Lecture Notes in Computer Sciences*. Springer–Verlag, 2006.

[36] D. Fernandez-Amoros, R. Heradio, and J. Cerrada. Inferring information from feature diagrams to product line economic models. In *Proceedings of the Sofware Product Line Conference*, 2009.

[37] R. Gheyi, T. Massoni, and P. Borba. A theory for feature models in alloy. In *Proceedings of the ACM SIGSOFY First Alloy Workshop*, pages 71–80, Portland, United States, nov 2006.

[38] R. Gheyi, T. Massoni, and P. Borba. Algebraic laws for feature models. *Journal of Universal Computer Science*, 14(21):3573–3591, 2008.

[39] GNU prolog, `http://www.gprolog.org`. accessed January 2010.

[40] J. Greenfield, K. Short, S. Cook, and S. Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, August 2004.

[41] A. Hemakumar. Finding contradictions in feature models. In *First International Workshop on Analyses of Software Product Lines (ASPL'08)*, pages 183–190, 2008.

[42] P. Heymans, P.Y. Schobbens, J.C. Trigaux, Y. Bontemps, R. Matulevicius, and A. Classen. Evaluating formal properties of feature diagram languages. *Software IET*, 2(3):281–302, 2008.

[43] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[44] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A feature–oriented reuse method with domain–specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[45] C. Kastner, S. Apel, and D. Batory. A case study implementing features using aspectj. In *SPLC '07: Proceedings of the 11th International Software Product Line Conference*, pages 223–232, Washington, DC, USA, 2007. IEEE Computer Society.

[46] B. Kitchenham. Procedures for performing systematic reviews. Technical report, Keele University and NICTA, 2004.

[47] P. Klint. A meta-environment for generating programming environments. *ACM Trans. Softw. Eng. Methodol.*, 2(2):176–201, April 1993.

[48] S.Q. Lau. Domain analysis of e-commerce systems using feature–based model templates. master's thesis. Dept. of ECE, University of Waterloo, Canada, 2006.

[49] F. Loesch and E. Ploedereder. Optimization of variability in software product lines. In *Proceedings of the 11th International Software Product Line Conference, SPLC*, pages 151–162, Washington, DC, USA, 2007. IEEE Computer Society.

[50] R.E Lopez-Herrejon and D. Batory. A standard problem for evaluating product-line methodologies. In *GCSE '01: Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, pages 10–24, London, UK, 2001. Springer-Verlag.

[51] M. Mannion. Using first-order logic for product line model validation. In *Proceedings of the Second Software Product Line Conference (SPLC'02)*, volume 2379 of *Lecture Notes in Computer Sciences*, pages 176–187, San Diego, CA, 2002. Springer–Verlag.

[52] M. Mannion and J. Camara. Theorem proving for product line model verification. In *Software Product-Family Engineering (PFE)*, volume 3014 of *Lecture Notes in Computer Science*, pages 211–224. Springer–Verlag, 2003.

[53] F. Marić. Formalization and implementation of modern sat solvers. *Journal of Automated Reasoning*, 43(1):81–119, June 2009.

[54] M. Mendonça, T.T. Bartolomei, and D. Cowan. Decision-making coordination in collaborative product configuration. In *Proceedings of the 2008 ACM symposium on Applied computing (SAC '08)*, pages 108–113, New York, NY, USA, 2008. ACM.

[55] M. Mendonça, D.D. Cowan, W. Malyk, and T. Oliveira. Collaborative product configuration: Formalization and efficient algorithms for dependency analysis. *Journal of Software*, 3(2):69–82, 2008.

[56] M. Mendonça, A. Wasowski, and K. Czarnecki. SAT–based analysis of feature models is easy. In *Proceedings of the Sofware Product Line Conference*, 2009.

[57] M. Mendonça, A. Wasowski, K. Czarnecki, and D. Cowan. Efficient compilation techniques for large scale feature models. In *Generative Programming and Component Engineering, 7th International Conference, GPCE , Proceedings*, pages 13–22, 2008.

[58] OPL studio, `http://www.ilog.com/products/oplstudio/`. accessed January 2010.

[59] A. Osman, S. Phon-Amnuaisuk, and C.K. Ho. Knowledge based method to validate feature models. In *First International Workshop on Analyses of Software Product Lines*, pages 217–225, 2008.

[60] A. Osman, S. Phon-Amnuaisuk, and C.K. Ho. Using first order logic to validate feature model. In *Third International Workshop on Variability Modelling in Software-intensive Systems (VaMoS)*, pages 169–172, 2009.

[61] J. Pena, M. Hinchey, A. Ruiz-Cortés, and P. Trinidad. Building the core architecture of a multiagent system product line: With an example from a future nasa mission. In *7th International Workshop on Agent Oriented Software Engineering*, Lecture Notes in Computer Sciences. Springer–Verlag, 2006.

[62] K. Pohl, G. Böckle, , and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer–Verlag, 2005.

[63] Racer Systems GmbH Co. KG. RACER, `http://www.racer-systems.com/`. accessed January 2010.

[64] R Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[65] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. Ex-

tending feature diagrams with UML multiplicities. In *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, June 2002.

[66] C. Salinesi, C. Rolland, and R. Mazo. Vmware: Tool support for automatic verification of structural and semantic correctness in product line models. In *Third International Workshop on Variability Modelling of Software-intensive Systems*, pages 173–176, 2009.

[67] K. Schmid and I. John. A customizable approach to full lifecycle variability management. *Science of Computer Programming*, 53(3):259–284, 2004.

[68] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis, Minnesota, USA, September 2006.

[69] P. Schobbens, J.C. Trigaux P. Heymans, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, Feb 2007.

[70] S. Segura. Automated analysis of feature models using atomic sets. In *First Workshop on Analyses of Software Product Lines (ASPL 2008). SPLC'08*, pages 201–207, Limerick, Ireland, September 2008.

[71] SMV system , `http://www.cs.cmu.edu/~modelcheck`. accessed January 2010.

[72] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber. Introducing pla at bosch gasoline systems: Experiences and practices. In *SPLC*, pages 34–50, 2004.

[73] D. Streitferdt, M. Riebisch, and I. Philippow. Details of formalized relations in feature models using OCL. In *Proceedings of 10th IEEE International Conference on Engineering of Computer–Based Systems (ECBS 2003), Huntsville, USA. IEEE Computer Society*, pages 45–54, 2003.

[74] J. Sun, H. Zhang, Y.F. Li, and H. Wang. Formal semantics and verification for feature modeling. In *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2005.

[75] T. Thüm, D. Batory, and C. Kästner. Reasoning about edits to feature models. In *International Conference on Software Engineering*, pages 254–264, 2009.

[76] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81(6):883–896, 2008.

[77] P. Trinidad, D. Benavides, and A. Ruiz-Cortés. Improving decision making in software product lines product plan management. In *Proceedings of the V ADIS 2004 Workshop on Decision Support in Software Engineering*, volume 120. CEUR Workshop Proceedings (CEUR-WS.org), 2004.

[78] P. Trinidad, D. Benavides, and A. Ruiz-Cortés. A first step detecting inconsistencies in feature models. In *CAiSE Short Paper Proceedings*, 2006.

[79] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A.Jimenez. FaMa framework. In *12th Software Product Lines Conference (SPLC)*, 2008.

[80] P. Trinidad and A. Ruiz Cortés. Abductive reasoning and automated analysis of feature models: How are they connected? In *Third International Workshop on Variability Modelling of Software-Intensive Systems. Proceedings*, pages 145–153, 2009.

[81] S. Trujillo, D. Batory, and O. Díaz. Feature oriented model driven development: A case study for portlets. In *International Conference on Software Engineering*, pages 44–53, 2007.

[82] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1995.

[83] M. Tseng and J. Jiao. *Handbook of Industrial Engineering: Technology and Operations Management*, chapter Mass Customization, page 685. Wiley, 2001.

[84] P. van den Broek and I. Galvao. Analysis of feature models using generalised feature trees. In *Third International Workshop on Variability Modelling of Software-intensive Systems*, number 29 in ICB-Research Report, pages 29–35, Essen, Germany, January 2009. Universität Duisburg-Essen.

[85] F. van der Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[86] T. van der Storm. Variability and component composition. In *Software Reuse: Methods, Techniques and Tools: 8th International Conference, ICSR 2004. Proceedings*, volume 3107 of *Lecutre Notes in Computer Sciences*, pages 157–166. Springer, July 2004.

[87] T. van der Storm. Generic feature-based software composition. In *Software Composition*, volume 4829 of *Lecture Notes in Computer Sciences*, pages 66–80. Springer–Verlag, 2007.

[88] A. van Deursen and P. Klint. Domain–specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10(1):1–17, 2002.

[89] T. von der Massen and H. H. Lichter. Deficiencies in feature models. In Tomi Mannisto and Jan Bosch, editors, *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, 2004.

[90] T. von der Massen and H. Lichter. Requiline: A requirements engineering tool for software product lines. In F. van der Linden, editor, *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE)*, volume 3014 of *Lecture Notes in Computer Sciences*, Siena, Italy, 2003. Springer–Verlag.

[91] T. von der Massen and H. Litcher. Determining the variation degree of feature models. In *Software Product Lines Conference*, volume 3714 of *Lecture Notes in Computer Sciences*, pages 82–88. Springer–Verlag, 2005.

[92] H. Wang, Y. Li, J. Sun, H. Zhang, and J. Pan. A semantic web approach to feature modeling and verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, November 2005.

[93] H. Wang, Y.F. Li, J. un, H. Zhang, and J. Pan. Verifying Feature Models using OWL. *Journal of Web Semantics*, 5:117–129, June 2007.

[94] J. Webster and R. Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, 26(2):xiii–xxiii, 2002.

[95] J. Whaley. JavaBDD, `http://javabdd.sourceforge.net/`. accessed January 2010.

[96] J. White, B. Doughtery, and D. Schmidt. Selecting highly optimal architectural feature sets with filtered cartesian flattening. *Journal of Systems and Software*, 82(8):1268–1284, 2009.

[97] J. White, B. Doughtery, D. Schmidt, and D. Benavides. Automated reasoning for multi-step software product-line configuration problems. In *Proceedings of the Sofware Product Line Conference*, pages 11–20, 2009.

[98] J. White and D. Schmidt. Filtered cartesian flattening: An approximation technique for optimally selecting features while adhering to resource constraints. In *First International Workshop on Analyses of Software Product Lines (ASPL)*, pages 209–216, 2008.

[99] J. White, D. Schmidt, D. Benavides P. Trinidad, and A. Ruiz-Cortes. Automated diagnosis of product-line configuration errors in feature models. In *Proceedings of the Sofware Product Line Conference*, 2008.

[100] H. Yan, W. Zhang, H. Zhao, and H. Mei. An optimization strategy to feature models' verification by eliminating verification-irrelevant features and constraints. In *ICSR*,

pages 65–75, 2009.

[101] W. Zhang, H. Mei, and H. Zhao. Feature-driven requirement dependency analysis and high-level software design. *Requirements Engineering*, 11(3):205–220, June 2006.

[102] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. In J. Davies, editor, *ICFEM 2004*, volume 3308 of *Lecture Notes in Computer Sciences*, pages 115–130. Springer–Verlag, 2004.

[103] Wei Zhang, Hua Yan, Haiyan Zhao, and Zhi Jin. A bdd–based approach to verifying clone-enabled feature models' constraints and customization. In *High Confidence Software Reuse in Large Systems, 10th International Conference on Software Reuse, ICSR, Proceedings*, volume 5030 of *Lecture Notes in Computer Sciences*, pages 186–199. Springer–Verlag, 2008.