Semantics of FODA Feature Diagrams

Yves Bontemps*, Patrick Heymans,
Pierre-Yves Schobbens, and Jean-Christophe Trigaux**

Institut d'Informatique, University of Namur

Abstract. Extended Feature Oriented Domain Analysis (FODA) Feature Diagrams (EFD) were introduced to compensate for a purported ambiguity and lack of precision and expressiveness of the original FODA feature diagrams (OFD). However, EFD never received a formal semantics, which is the hallmark of precision and unambiguity. We propose here a semantics for both diagrams. From this we demonstrate that OFD are precise, unambiguous, and expressively complete, and thus that all extensions add no expressiveness. A finer notion is thus needed to compare these languages. Two solutions are well-established: succinctness and embeddability, that measures naturalness of a language. This tool shows that EFD indeed bring some naturalness, but are harmfully redundant and that the same naturalness can be attained with the simpler varied FD (VFD). We also show that no ambiguity is present, in fact.

1 Introduction

When engineering software product lines (PL), constructing FD is commonly used to capture commonalities and variabilities between products and to bridge the gap between requirements and design. Kang *et al.* define a *feature* (F) as "any prominent and distinctive aspect or characteristic that is visible to various stakeholders (i.e. end-users, domain experts, developers, etc.)" [9]. FD provide a concise and explicit representation of variability. They guide the choices to be made for determining the features of specific products and facilitate the reuse of software components implementing these features.

The original FD (OFD) were introduced in the FODA methodology back in 1990 [9] (Fig. 1). Since then, they have undergone several extensions [6, 14, 13, 4, 20, 3, 10] intended to "improve their expressiveness". However, this has never been demonstrated. In this paper, we adopt a rigorous approach towards checking the validity of such claims.

Contrary to folk belief, we prove formally that these extensions add absolutely no expressiveness to OFD, since OFD are maximally expressive. We do not question the fact that EFD have important advantages over the original ones, but we show rigorously that they are *succinctness* and *embeddability* rather than *expressiveness*. Then, we revisit the "ambiguity" problem invented in [13]. We show that it is more rigorously termed a harmless redundancy issue.

To get to these results, we first overview the FD found in literature. This is the topic of Section 2 where we focus on syntactic variations, which are rigorously defined in Section 3.1. Semantic concerns are addressed in Section 4. A fully formal definition of FD is given in Section 3.2. Surprisingly, although extremely concise and translated almost literally from [9], such a definition has never been published beforehand, while it is well known [7] that formal semantics is the best way (1) to make sure that models are unambiguous (Def. 14) and (2) to start building safe automated reasoning tools, be it for verification, transformation or, more specifically, for assisting stakeholders in choosing features.

EFD are as expressive (Theorem 2), but more succinct (Theorem 5) and natural (Theorem 4). But these measures also detect that EFD are harmfully redundant and that the simpler VFD gives the same naturalness with a much simpler syntax. VFD is thus the FD language we can

* FNRS Research Fellow

** Work Supported by Walloon Region and FSE in FIRST Europe Objective 3 Project PLENTY EPH3310300R0462 / 215315

recommend for general use: it is simple, natural, succinct, has a simple formal semantics and is expressively complete. The problem of EFD is the presence of synonymous constructs, which are often believed to bear different meanings. This opens the way to misunderstandings as well as endless, pointless arguments between stakeholders.

We terminate Section 4 with the purported [14] problem of ambiguity (Def. 14), that is actually harmless redundancy (Def. 15).

2 State of the art

2.1 FODA

The reference definition of FD is provided in the Feature Oriented Domain Analysis (FODA) method [9]. As depicted in figure 1, the original FD (OFD) are composed of:

1. The *concept* or *root* refers to the complete product line.
2. The *nodes* can be mandatory (by default) or optional (with a hollow circle, e.g. `coolant`).
3. The *relations* between nodes can be:
   (a) *consists-of*: It has two meanings, *and-decomposition* (e.g. between `Monitor Fuel Consumption` and its sons) or *xor-decomposition* (e.g. between `Methods` and its sons).
   (b) *constraints*: either requiresor mutex, an abbreviation for "mutually-exclusive-with". Constraints were first added as textual annotation, but later drawn in the graph, a variant we call Graphical OFD (GOFD).

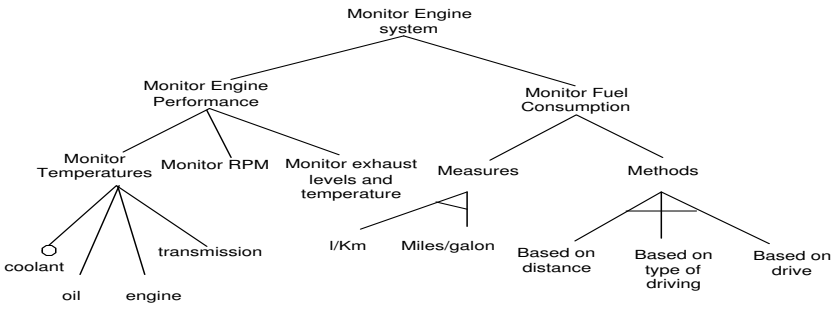

Fig. 1. FODA: Monitor Engine System

2.2 FeatuRSEB

FeatuRSEB [6] is a combination between the FODA method and the Reuse-Driven Software Engineering Business (RSEB) method. RSEB is a use-case driven systematic reuse process, where variability is captured by structuring use cases and object models with variation points and variants. As illustrated in Fig. 2, FeatuRSEB uses *XOR (white diamonds) and OR (black diamonds)* nodes. We call such diagrams RFD.

2.3 Generative programming

Czarnecki and Eisenecker [4] have extended OFD for use in the Generative Programming (GP) approach. Mandatory, alternative and optional nodes are completed with "or-features", "optional alternative features" and "optional or-features". The authors argue that FD are *directed graphs* and not just trees. Their FD can now also contain cardinalities [2, 3] like in EFD (Section 2.4). We call these diagrams GPFD.

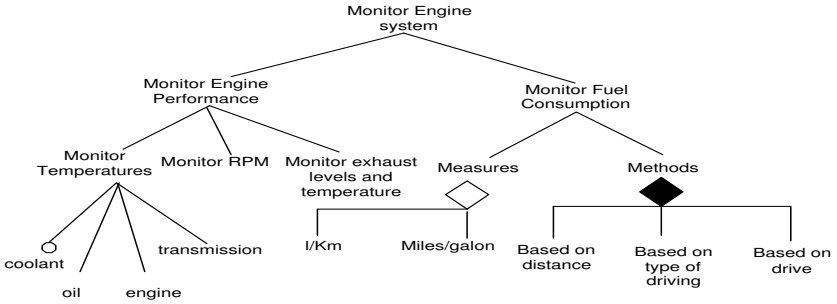

Fig. 2. FeatuRSEB: Monitor Engine System

2.4 Feature diagrams with multiplicities

Riebisch claims that multiplicities are partially represented with those previous notations [14]. Moreover, he indicates that "unfortunately, these combinations of mandatory and optional features with alternatives, OR and XOR relations could lead to ambiguities" [13]. In order to limit these drawbacks, he extends the GPFD notation with *UML-like multiplicities* and *mandatory and optional relations* [14]. As illustrated in Fig. 3, he marks mandatory relations with a filled circle. Multiplicity denotes the minimum and maximum number of edges to be chosen from those originating from the node. We call these diagrams EFD.

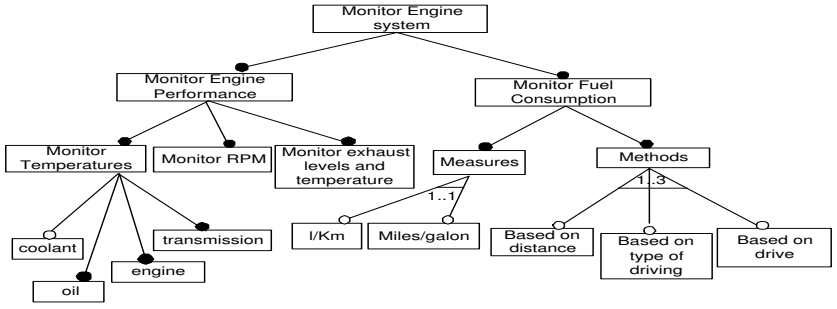

Fig. 3. Feature Diagram with multiplicities: Monitor Engine System

3 Formal definition

3.1 Syntax

In this section, we present a syntax for OFD and EFD, see Fig. 1 and 3.

Definition 1 (Original feature diagram). *An* original feature diagram *(OFD) D [9] is a labeled graph with:*

1. nodes $n \in N$ *(e.g. all text in any Figure), that are labeled as:*
   (a) *One* node type*, either:*
      i. xor-*node* $x \in X$*, denoting features that are realized by including exactly one of their sons. They are drawn with an arc joining their outgoing links (see* `Methods` *in Fig. 1).*
      ii. and-*node* $a \in A$*, denoting features that are realized by including all their sons, such as* `Monitor Engine Performance`*, in Fig. 1;*

and optionally:

(b) as the concept *or* root $r \in N$*. It is drawn at the top (eg.* `Monitor Engine System` *in Fig. 1). There is a unique concept in an FD, conventionally an and-node:* $r \in A$

(c) leaf $f \in F$ *are features in the meaning of Section 1. They are drawn at the bottom of the diagram.*

(d) optional $o \in O$*, drawn with a small hollow circle above, cf.* `coolant` *in Fig. 1.*

2. edges $\rightarrow \subseteq N \times N$ *labeled* **mandatory** *such that*
   (a) Only the concept has no father: $n \not\rightarrow m$ *iff* $n = r$*.*
   (b) The hierarchy is acyclic: $\neg(n_1 \rightarrow \ldots \rightarrow n_k \rightarrow n_1)$
3. *a set of constraints* Φ *of the forms* n_1 *mutex* n_2*, or* n_1 *requires* n_2*. For the sake of Section 4.2, where all must be labeled graph, we will represent constraints by their syntax tree, attached to the root.*

Definition 2 (Extended feature diagram). Extended feature diagrams *(EFD) add a new node type, variation points* $VP(i \ldots j)$*, where* $i \in \mathbb{N}$ *and* $j \in \mathbb{N} \cup \{*\}$ *are called the* multiplicities*.*

3.2 Semantics

The semantics of a FD will be a product line, whose products provide a set of features (F) (Def. 5).

The notion of model is introduced in [9, p.64], with the examples of models of X10 terminals.

Definition 3 (Model). *A* model $m \in M$ *of a FD is a subset of its nodes, plus a second copy noted* o' *of its optional nodes:* $M = \mathcal{P}(N \uplus O')$*. We let* $n' = n$ *for non-optional nodes.*

Definition 4 (Valid model). *[9, p.70] A model* $m \in M$ is valid for *a diagram* $D \in FD$*, noted* $m \vDash D$ *iff:*

1. *The concept is in the model:* $r \in m$
2. *If a xor-node is in the model, exactly one of its sons is:* $x \in m \Rightarrow \ \exists! n.x \rightarrow n \wedge n' \in m$
3. *If a and-node is in the model, all its sons are:* $a \in m \Rightarrow \forall n.(a \rightarrow n \Rightarrow n' \in m)$
4. *If a* $VP(i \ldots j)$ *is in the model, the number of its sons that are in the model is between* i *and* j *(included).* $v \in M \Rightarrow i \leq |\{n | x \rightarrow n \wedge n' \in M\}| \leq j$
5. *If a non-concept node is in the model, some of its fathers, called its* justification*, is:* $n' \in m \wedge n \neq r \Rightarrow \exists n_1.n_1 \rightarrow n \wedge n_1 \in m$
6. *The* justification *of* o *is* o'*:* $o \in m \Rightarrow o' \in m$
7. *The model must satisfy all formulae from the constraint set:* $\forall \phi \in \Phi, m \vDash \phi$*, where:*
   $m \vDash n_1$ *mutex* n_2 *iff* n_1 *and* n_2 *are not both in* m*;*
   $m \vDash n_1$ *requires* n_2 *iff when* n_1 *is in* m*,* n_2 *too.*

Definition 5 (Product line).

1. *A* product $p \in P$ *is a set of leaves:* $P = \mathcal{P}(F)$*.*
2. *The* product of *a model* m*, noted* $[\![m]\!]$*, is* $m \cap F$*. Many models denote the same product.*
3. *A* product line *is a set of products:* $PL = \mathcal{P}(P) = \mathcal{P}(\mathcal{P}(F))$*.*
4. *The* product line of *a FD* d *is the products of its valid models:* $[\![d]\!] = \{m \cap F | m \vDash d\}$

3.3 Discussion of the semantics

Our semantics is node-based, following [9, p.70]; an alternative would be to take an edge-based semantics, where models are sets of edges and the conditions on sons are replaced by conditions on edges. In fig 4, $\{l, m\}$ is a valid edge-based model, but not a valid node-based model.

Almost each paper in Section 2 has understood the meaning of options in a different way. Thus we contradict some. In our semantics, Fig. 3 admits a model with neither `l/km` or `miles/gallon`, for instance.

Our use of justifications works well with a finite acyclic hierarchy, but will not extend to the infinite or cyclic case. In this case, a complex solution in the style of stable models [19] will be needed, and will give the same result for our simple case.

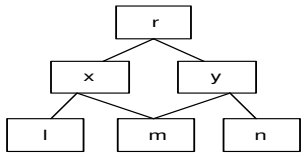

Fig. 4. edge- or node-based?

4 Formal comparison

In this section, we will compare FD w.r.t. expressiveness, embeddings, succinctness, redundancy.

4.1 Expressiveness

The *expressiveness* of a language is the part of its semantic domain that it can describe.

Definition 6 (Expressiveness). *The* expressiveness *of a language $\mathcal{L}$ is the set $E(\mathcal{L}) = \{[\![D]\!] | D \in \mathcal{L}\}$. A language $\mathcal{L}_1$ is more expressive than a language $\mathcal{L}_2$ if $E(\mathcal{L}_1) \supset E(\mathcal{L}_2)$. A language $\mathcal{L}$ with semantic domain $\mathbb{M}$ (i.e. its semantics is $[\![\cdot]\!] : \mathcal{L} \to \mathbb{M}$) is* expressively complete *if $E(\mathcal{L}) = \mathbb{M}$.*

The semantic domain $\mathbb{M}$ of FD is $PL = \mathcal{P}(\mathcal{P}(F))$. In other words: every FD defines a PL. Now, we can ask the converse question: Can every PL be expressed by a FD of a given kind? In other words: are these FD expressively complete?

We start with the two textual constraints mutexand requires. It is the only part of OFD that was never extended, hinting that it is complete. We prove the opposite.

Theorem 1. *There is a FD D (Fig. 9) and a boolean formula φ that no set of mutex, requiresconstraints can express, i.e., $\{m | m \models D \wedge m \models_{\mathcal{B}} \varphi\} \neq \{m | m \models D \wedge m \models \Phi\}$*

Proof. Any FD on the top row of Fig. 9 has 4 nodes and allows to choose any combination of the 3 leaves. Its PL has 8 models and products. Impose the constraint $\varphi = C \vee D$. Its PL has 6 models and products. There are 4 nodes, thus 4 values for n_1 and n_2, so 16 possible mutexconstraints, and the same for requires. Thus 2^{32} sets are possible, but many express the same PL. None of these sets Φ expresses $[\![\varphi]\!]$, more precisely (1) $[\![\Phi]\!] \subset [\![\varphi]\!]$ or (2) $[\![\Phi]\!] \cap \{C, D\} = P(\{C, D\})$. We induce on Φ. If it is empty, we have (2). Else, let ϕ be a constraint in Φ. For $n_1 \text{requires} n_1$ or if n_1 and $n_2 \notin \{C, D\}$ we have (2) ; else, we have (1).

One should not over-interpret this theorem. First, it is true only of some fixed FD. The FD of Fig. 5 expresses this PL, and indeed OFD are expressively complete (Theorem 2). Second, if nesting constraints were allowed, Sheffer [18] has shown that mutexalone is complete, even when base symbols are taken from F instead of N.

Now we ask: Can every PL be expressed by an OFD? Many authors assumed no, and extended OFD. We need a small tool to solve this question: the *first normal form* of a product line.

Definition 7 (First normal form). *The* first normal form *of a PL P $\mathcal{N}_1(P)$ has as xor-nodes the root $X = \{r\}$, as and-nodes the products $A = P$, no optional nodes $O = \emptyset$, edges from the root to each product, and from each product to its constituent features: $n \to p$ iff $n = r \wedge p \in P \vee n \in P \wedge p \in n$, and no constraints, see e.g. Fig. 5.*
The language of first normal forms is thus called $\mathcal{N}_1(PL)$.

Note that this form is expressed in a subset of OFD:

Definition 8. *COFD is OFD without constraints nor options.*

Theorem 2. *Every PL can be expressed by a COFD, e.g. by its first normal form:*

$$\forall P \subseteq 2^F . \exists \mathcal{N}_1(P) \in \mathit{COFD} . P = [\![\mathcal{N}_1(P)]\!]$$

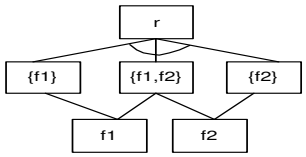

Fig. 5. Normal form of product line $\{\{f1\}, \{f2\}, \{f1, f2\}\}$

Kang [9, p.87] also noted that OFD are redundant. He proposed to get rid of the graphical part and use only textual constraints. Theorem 1 excludes this. One can, on the contrary, get rid of the constraints. Another alternative is to choose a rich, but fixed, graphical part. Then all the information is in the constraints:

Definition 9 (Second normal form). *The* second normal form *of a product line P, noted $\mathcal{N}_2(P)$, is a COFD where the root is the only xor-node, connected all possible sets of leaves which are and-nodes connected to their leaves. Then we use a mutexwith root to exclude the combinations that should not be included in the PL.*

Theorem 3. *For any set of leaf features F, there is a FD $\mathcal{N}_2(\{F\})$ such that any PL L on F can be expressed by constraints on $\mathcal{N}_2(\{F\})$.*

This does not contradict Theorem 1, but shows that some FD with many nodes provide enough grip to constraints, while simple FD do not.

Thus all extensions of COFD are exactly as expressive as COFD, since the latter already have maximal expressiveness.

Probably the critics of OFD confused expressiveness with succinctness or naturalness, that are often used as a substitute for expressiveness when all languages are equally expressive (as is the case, e.g., with programming languages.)

4.2 Embeddability

When all languages have the same expressiveness, one needs finer criteria to compare them. [5], following [11], pointed that to be natural, the translation used for the proof of expressiveness should be done without a global reorganization:

Definition 10 (Embeddability). *$\mathcal{L}_1$ is* embeddable *into $\mathcal{L}_2$ iff there is an* embedding *$T : \mathcal{L}_1 \to \mathcal{L}_2$ that is:*

1. compositional *(also called* modular *or* homomorphic*): $T(C_1(e)) = C_2(T(e))$, for all constructs C_1 of L_1, where $\boldsymbol{e}$ is a vector of meta-variables, adequately typed w.r.t. C_1's subterms expected types, and C_2 is an expression in language $\mathcal{L}_2$ containing the same meta-variables.*
2. correct*: $\forall l_1 \in \mathcal{L}_1, [\![T(l_1)]\!] = [\![l_1]\!]$.*

Said otherwise, an L_1 compiler can be obtained by putting a trivial macro pre-processor in front of an L_2 compiler.

The graphical languages considered in this paper have no context-free syntax, so that compositionality cannot apply. We generalize it:

Definition 11 (Graphical embeddability). *A graphical language $\mathcal{L}_1$ is* embeddable *into $\mathcal{L}_2$ iff there is a correct* embedding *$T : \mathcal{L}_1 \to \mathcal{L}_2$ that is:*

1. node-controlled*[8]: T is expressed as a set of rules $D_1 \to D_2$, where D_1 is a diagram containing a distinguished node, and all possible relations with this node. Its translation D_2 is a subgraph in $\mathcal{L}_2$, plus how the existing relations should be connected to nodes of this new subgraph.*

Our second definition is a generalization of the first one, if we view syntax trees as labeled graphs, and allow syntax trees with sharing.

Embeddability is widely considered as an adequate notion for comparing "natural expressiveness" of languages. Specially, non-trivial self-embeddability is considered as *harmful redundancy*. It means that there is a strict sublanguage where everything can be expressed as naturally, thus that the language is unnecessarily complex. All abstraction and decomposition principles are kept in the smaller language. Indeed, the replacement is purely local and only impacts the diagram size or shape by a constant factor. We now show that EFD have such a sublanguage, that we call VFD.

Definition 12 (VFD). *VFD is the sublanguage of EFD with just VPs.*

Theorem 4. *The normal forms are embeddable into COFD; COFD is embeddable into OFD; OFD is embeddable into EFD; EFD is embeddable into VFD and conversely.*

The first two embeddings are obvious. For the third, we provide the translation in Fig. 7 and an example in Fig. 6.

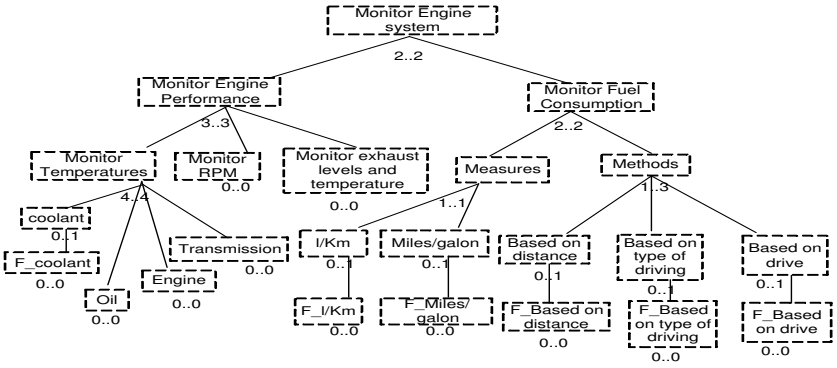

Fig. 6. Varied Feature Diagram(VFD): Monitor Engine System

Instead of ...	**write ...**
a and-node with number of sons s	a VP($s \dots s$)
a xor-node	a VP($1 \dots 1$)
an or-node	a VP($1 \dots *$)
an optional node o	a VP($0 \dots 1$) with son o
n_1 mutexn_2	a VP(1..1), and-son of r, with sons n_1, n_2
n_1 requiresn_2	a VP(1..2), and-son of r, with two sons $n_2, \neg n_1$, himself a VP(0..0) with son n_1.

Fig. 7. Embedding EFD into VFD in $3n$

4.3 Succinctness

Furthermore, the cost of translating between these notations is not prohibitive.

Definition 13 (Succinctness). *Let $\mathcal{G}$ be a set of functions from $\mathbb{N} \to \mathbb{N}$. A language $\mathcal{L}_1$ is* $\mathcal{G}$-as succinct as $\mathcal{L}_2$, *noted $\mathcal{L}_2 \leq \mathcal{G}(\mathcal{L}_1)$, iff there is a correct translation $T : \mathcal{L}_1 \to \mathcal{L}_2$ that is* within $\mathcal{G}$: *$\exists g \in \mathcal{G}, \forall n \in N, \forall l_1 \in \mathcal{L}_1, |l_1| \leq n \Rightarrow |T(l_1)| \leq g(n)$. Common values for $\mathcal{G}$ are "identically" $= \{n\}$, "thrice" $= \{3n\}$, "linearly" $= O(n)$, "cubically" $= O(n^3)$, "exponentially" $= O(2^n)$. We will omit "identically".*

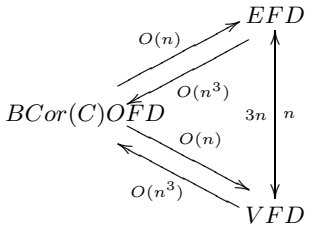

Fig. 8. Translations between Feature Diagram Languages

In Fig. 8, BC are the combinatorial boolean electronic circuits [16, 17, 21]. They are built from mutex gates [18].

Theorem 5. *1. $EFD \leq OFD \leq COFD$.*
2. $EFD \leq VFD$.
3. $VFD \leq 3{\cdot}EFD$.
4. $COFD \leq O(VFD^3)$.
5. $VFD \leq BC$.
6. $BC \leq O(VFD^3)$.

Proof. 1. Let $T(D) = D$.
2. Let $T(D) = D$.
3. Fig 7.
4. We translate each variation point v of multiplicity $\mu \dots \nu$ by a subgraph of quadratic size. Assume its sons are ordered as $s_1, \dots, s_q$. We introduce fresh xor-nodes of the form (i, j) where $0 \leq i \leq q$ is the number of sons treated, and $0 < j \leq \nu \leq q$ is the number of sons in the model among the sons treated. When $i = q$, if $\mu \leq j \leq \nu$, we replace it by the true node (a and-node with 0 sons with a and-link from the root). Else, we replace it by the false node (a xor-node with 0 sons). Each (i, j) with $i < q$ has two and-sons, based on a case analysis of whether the son i is in the model. The first $(i, j)^+$ is and-node with sons s_i and $(i + 1, j + 1)$. The second $(i, j)^-$ is and-node with sons $\neg s_i$ and $(i + 1, j)$. $\neg s_i$ is a xor-node with two sons, s_i and a true node.
5. A mutex$_s$ gate can be translated by a VP$(0 \dots s - 1)$.
6. First translate the VFD to a COFD as above. We note that each node has zero or two sons. It is known that every boolean operator can be translated linearly in BC, so we replace each node by this translation.

These results compose, e.g. VFD $\leq 3{\cdot}$OFD by using (1) and (3). Thus VFD can naturally encode any FD, multiplying its size by at most 3.

This is rather surprising when considering the necessary complexity of any PL language.

Theorem 6. *Any expressively complete language for PL must have PL that are expressed exponentially in the number of leaf features.*

Thus all languages considered here agree on the PL that are expressed polynomially:

Theorem 7. *A series of PL is expressed polynomially by a FD iff it has a polynomial boolean circuit (PBC) [12].*

4.4 Redundancy

Another criticism [13] of OFD is that they are ambiguous:

Definition 14 (Ambiguity). *A diagram (or sentence) is* ambiguous *iff it has two different meanings:* $[\![D]\!] \neq [\![D]\!]$. *A language is ambiguous iff it contains an ambiguous diagram or sentence.*

This is obviously impossible with a formal semantics, since $[\![.]\!]$ is a function. The semantics presented here is already present (albeit in English) in [9], thus we were surprised to see this claim in [13].

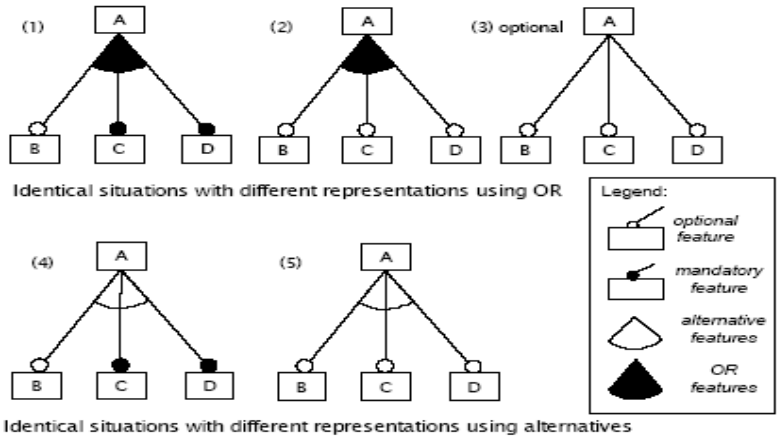

Fig. 9. Ambiguity example [13]

Let us examine the "proof" of [13] in Fig. 9. It shows $[\![D_1]\!] = [\![D_2]\!]$. Let us name this property:

Definition 15 (Harmless redundancy). *A language* $\mathcal{L}$ *is* harmlessly redundant *iff* $\exists D_1, D_2 \in \mathcal{L}. D_1 \neq D_2 \wedge [\![D_1]\!] = [\![D_2]\!]$. *Otherwise, it is a* normal form *(NF).*

All FD except normal forms are thus harmlessly redundant. But a normal form cannot be as natural (Corollary 1).

4.5 Complexity

We consider the complexity of implementing a CASE tool supporting FD. Our envisioned CASE tool shall support two tasks: designing and analyzing new FD, and resolving variation points by interacting with a customer to obtain a model. In the latter problem, the customer is asked questions following the root down to the leaves. When the customer chooses, this choice is propagated, pruning the FD. A choice that leads to no product should never be proposed to the user.

In the theorems of this section, we use FD to stand for COFD, OFD, VFD or EFD.

Theorem 8. *Deciding whether an FD has products is NP-complete.*

The design task is made of several use cases. First, an analyst draws a FD. Since VFD are now less popular than EFD, although better, as advocated in the previous sections, it may be easier to allow input in the syntax of any FD. The tool translates those into VFD, following the rules of Fig. 7, and lets the analyst switch between the two views. Doing so would ensure a better acceptance and a steeper learning curve.

Secondly, when a FD is considered too complex, an engineer could want to manually refactor it. After such a modification, a natural question to ask is whether the PL described by this new FD is the same. Similarly, when PL are developed by several persons, conflicts may arise and, in order to solve them, it can be interesting to first compare PL, and give a product showing the difference.

Theorem 9. *Deciding whether two FD have the same valid models is coNP-complete. Giving a model in their difference is NP-complete.*

Theorem 10. *Deciding whether two FD are equivalent (have the same products) is coNP-complete. Giving a product in their difference is NP-complete.*

Corollary 1. *For any normal form NF for which equivalence can be decided in polynomial time, putting a FD into NF is coNP-complete.*

No NF failing this condition is known: our two NF (Defs. 7 and 9) are in $O(n \log n)$, for instance. All known coNP-complete algorithms take exponential time. So in practice, putting a FD into NF is exponential. An embbeding is linear, so that no NF can be as natural as a FD language.

5 Conclusion

Through this paper, we hope to have clarified feature diagrams. We have shown how to evaluate their various extensions. Expressiveness is not a criterion here, since no extension can increase the expressiveness of OFD. We should use a succinct, natural and non-redundant language. Then VFD is clearly the winner.

Our formalization decides concisely every detail of these diagrams. Some might disagree on the meaning of specific examples. But at least we opened the field for a precise and fruitful discussion, and we believe to have good arguments for our choices.

A feature is formalized here as a meaningless symbol: we just follow [9] in that. In [15], we have shown how to consider a feature as coordinated transformations of models at several levels of abstraction, thus giving it a rich formal semantics. It gives rise to an integrated Transformational Model-Driven Approach (TMDA) for Product Lines (PL).

Variation points can be added to any type of diagrams (or text), obtaining thus "varied diagrams". We will give their generic semantics. The treatment of variation will thus also be harmonized in the whole development. This is a restricted case of the approach above [15], since it cannot deal with addition of unexpected features but it has a more appealing graphical syntax and a simpler semantics.

Our formalization has the same structure as the definition of logics. Many natural questions arise if we see FD as a logic; the main one is to provide a sound and complete inference system. Such a system has to be graphical, since we use a graphical syntax. For instance, in [1], we gave a graphical inference system for a graphically similar, but semantically very different, language: priority graphs.

We only studied static product lines. An important future topic of study is their evolution.

References

1. Andreka, Ryan, and Schobbens. Operators and laws for combining preference relations. *JLC: Journal of Logic and Computation*, 12, 2002.
2. Krzysztof Czarnecki, Thomas Bednasch, Peter Unger, and Ulrich W. Eisenecker. . generative programming for embedded software: An industrial experience report. In Don Batory, Charles Consel, and Walid Taha., editors, *ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering*, pages 156–172, Berlin-Heidelberg, 2002. Springer-Verlag.
3. Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration using feature models. In *Proc. SPLC04*, 2004.

4. Ulrich W. Eisenecker and Krzysztof Czarnecki. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.
5. M. Felleisen. On the expressive power of programming languages. In N. D. Jones, editor, *Proceedings of the Third European Symposium on Programming*, volume 432 of *Lecture Notes in Computer Science*, pages 134–151. Springer Verlag, 1990.
6. M. Griss, J. Favaro, and M. d'Alessandro. Integrating Feature Modeling with the RSEB. In *Proceedings of the Fifth International Conference on Software Reuse*, pages 76–85, Vancouver, BC, Canada, June 1998.
7. D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff, part i: The basic stuff. Technical Report MCS00-16, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, 2000.
8. Dirk Janssens and Grzegorz Rozenberg. On the structure of node label controlled graph languages. *Inform. Sci.*, 20:191–244, 1980.
9. 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.
10. Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe. Feature-Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, July/August 2002.
11. Stephen Cole Kleene. *Introduction to Metamathematics*, volume 1 of *Bibliotheca Mathematica*. North-Holland, Amsterdam, 1952.
12. Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
13. M. Riebisch. Towards a More Precise Definition of Feature Models. *Position Paper. In: M. Riebisch, J. O. Coplien, D, Streitferdt (Eds.): Modelling Variability for Object-Oriented Product Lines*, 2003.
14. Matthias Riebisch, Kai Böllert, Detlef Streitferdt, and Ilka Philippow. Extending Feature Diagrams with UML Multiplicities. In *Proceedings of the Sixth Conference on Integrated Design and Process Technology (IDPT 2002)*, Pasadena, CA, June 2002.
15. Mark Ryan and Pierre Yves Schobbens. Fireworks: A formal transformation-based model-driven approach to features in product lines. In Tomi Männistö and Jan Bosch, editors, *Proc. WS. on Software Variability Management for Product Derivation - Towards Tool Support*, 2004.
16. Claude Shannon. A symbolic analysis of relay and switching circuits. Master's thesis, MIT, 1937.
17. Claude Shannon. A symbolic analysis of relay and switching circuits. *Transactions American Institute of Electrical Engineers*, 57:713–723, 1938.
18. Henry M. Sheffer. A set of five independent postulates for boolean algebras, with application to logical constants. *Trans. AMS*, 14:481–488, 1913.
19. T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In G. Gupta, editor, *Practical Aspects of Declarative Lan-guages First International Workshop, PADL'99, Proceedings*, volume 1551 of *Lecture Notes in Computer Science*, pages 305–319. Springer-Verlag, 1999.
20. Jilles van Gurp, Jan Bosch, and Mikael Svahnberg. On the Notion of Variability in Software Product Lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, 2001.
21. Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag, New York, 1999.