Generic Semantics of Feature Diagrams
Variants

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

a Computer Science Departement, University of Namur

Abstract. A large number of extensions of Feature Oriented Domain Analysis (FODA) Feature Diagrams were introduced to compensate for a purported ambiguity and lack of precision and expressiveness of the original FODA feature diagrams (OFD). However, they never received a formal semantics, which is the hallmark of precision and unambiguity. We propose here a formal semantics for all these diagrams, thanks to a generic construction that we call Free Feature Diagrams (FFD). 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 express naturalness of a language. This tool shows that some extensions indeed bring some naturalness, but are harmfully redundant and that the same naturalness can be attained with the simpler varied FD (VFD).

Keywords. Feature Diagrams, Semantics, Feature Interactions

1. Introduction

The description of systems in terms of features is used in many domains such as telecommunication or embedded systems. Two perspectives are used in turn:

- In the beginning of a system, features are “units of evolution” that adapt it to an optional user requirement [5]. Because of feature interaction, adding new features may affect already implemented features. These feature interactions must be detected, resolved if possible, or the combination of features must be forbidden.
- In the production phase of the same system, the features and their interactions are well identified, and this knowledge must be packaged in a form usable by the customers selecting their options, and by engineers deploying them. This is the perspective of the software product lines. The deployment problem is that: “individual features do not typically trace directly to an individual component - this means, as a product is defined by selecting a group of features, a carefully coordinated and complicated mixture of parts of different components are involved.”[7]

In this paper, we address the second perspective, in which industry commonly constructs feature diagrams (FD) to describe allowed variabilities between products. In addition, FD provides a concise and explicit way to

- represent features dependencies,
• guide the selection of features allowing the construction of a specific product,
• facilitate the reuse and the evolution of software components implementing these features.

The Original Feature Diagrams (OFD) were introduced in the FODA methodology back in 1990 [10,11] (Fig. 1). Since then, they have undergone several extensions: RESB Feature Diagram (RFD) [6], Generative Programming Feature Diagram (GPFD) [3], Graphically Extended Feature Diagram (GEFD) [16,15], 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 folklore, we prove formally that these extensions add no expressiveness to OFD, since OFD are maximally expressive (Theorem 4.2). We do not question the fact that extended FD have advantages over the original ones, but they are succinctness (Theorem 4.7) and naturalness (Theorem 4.6) rather than expressiveness. Since more variants of FD can be expected, we take care to obtain these results in a generic way where these next variants of FD are included.

We first overview in Section 2 the FD variants found in literature. We then abstract their commonalities. They are rigorously defined in Section 3.1, 3.2. Surprisingly, although extremely concise and translated almost literally from [10], such a definition has never been published before, while it is well known [8] that formal semantics is the best way (1) to make sure that models are unambiguous (Def. 4.4) (2) to start building safe automated reasoning tools, be it for verification, transformation or, more specifically, for assisting stakeholders in selecting features (Section 4.5). We compare their semantics in Section 4.

In most extensions, there are synonymous constructs, i.e. they are redundant (Def. 4.4, Section 4). Thus we propose a new, natural, succinct, and non-redundant variant of FD: VFD. VFD use a single construct to express all the extensions above.

2. State of the art

2.1. FODA

The reference definition of FD is provided in the Feature Oriented Domain Analysis (FODA) method [10]. As depicted in Fig. 1, OFD are composed of (1) a concept, (2) nodes and (3) relations between them.

1. The concept is the root node and refers to the complete system.
2. The nodes can be mandatory (by default) or optional (with a hollow circle, e.g. coolant) and be subject to decomposition (see below).
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:
      i. requires
      ii. mutex, an abbreviation for “mutually-exclusive-with”.


Constraints are presented as a textual annotation.

Figure 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.

Figure 2. FeatuRSEB: Monitor Engine System

2.3. Generative programming

Czarnecki and Eisenecker [3] have extended OFD with the Generative Programming (GP) approach. They also use “or-features” (so this is RFD) and later add cardinalities [2] borrowed from [16]. Cardinality (called “multiplicity” in UML) denotes the minimum and maximum number of edges to be chosen among those originating from the node. The authors argue that FD have always been directed graphs and not just trees. They introduce graphical constraints, i.e., an edge between features bearing mutex or requires. We call these diagrams GPFD.
2.4. Feature diagrams with multiplicities

Riebisch claims that multiplicities are partially represented with those previous notations [16]. Moreover, he indicates that “these combinations of mandatory and optional features with alternatives, OR and XOR relations could lead to ambiguities” [15, p.4]. These “ambiguities” are due a different conception of what mandatory or optional means (see Example 3.2), and better termed redundancy (see Section 4.4).

In order to limit these drawbacks, he introduces *multiplicities* (i.e. cardinalities) and *mandatory and optional edges* [16]. As illustrated in Fig. 4, he marks mandatory relations with a filled circle at the lower end. Optional edges are useful since a feature can be optional on one side and mandatory on another, see Fig.3. We call these diagrams GEFD.

![Figure 4. FD with multiplicities: Monitor Engine System](image)

Their treatment of optional edges is different from ours for *xor* and *or* nodes: they define it as a redundant semanticless annotation.

3. Formal definition

3.1. Syntax

It is natural when using OFD to ask what motivated the use of the particular operators *xor* and *and*, and to add operators as they seems handy in the specific application. This explains this wide variety of variants of FD. Surprisingly, it is easy to cater for this need in a generic way, which is the main topic of this paper.
Definition A node type is a set of boolean operators, with at most one operator per arity.

Example and is the set of operators that returns true iff all their arguments are true. E.g. Monitor Engine Performance in Fig. 1.

Example The set requires = {⇒} has only a binary operator. Thus such nodes must have exactly two sons. If this node is in, its left son must be out or its right son must be in.

Example Similarly, the set mutex = {∥}, where ∥ is Sheffer’s bar, meaning ~(p1 ∨ p2). has only a binary operator, so that its nodes must have exactly two sons.

Example The previous example can be generalised to nand = {¬ and}. Electronic circuits mainly use this family, since it is easily implemented from transistors.

Example xor is the set operators that returns true iff exactly one of its s arguments is true. See Methods in Fig. 1.

Example or is the set of operators that returns true iff any of their arguments is true. E.g. Methods in Fig. 2, Fig.12 (1), (2).

Example VP(i . . . j), where i ∈ N and j ∈ N∪{*}, is the family where VP(i . . . j), returns true iff at least i and at most j of its s sons are true. A VP(i . . . j)-node is called a variation point i . . . j is the multiplicity or cardinality. See Measures, Methods in Fig. 4.

Similarly, why restrict to just two constraint types?

Definition A graphical constraint type is a binary boolean operator.

Definition A textual constraint type is a boolean operator.

Definition A option type b ∈ B is either 1 (optional), or 0 (mandatory).

We will note optional edges like n ← m, e.g. the edge from Measures to 1/km in Fig. 4, and mandatory ones like n → m. n →?m notes either. Optional nodes n ∈ O are drawn with the small hollow circle above, e.g. coolant in Fig. 1, and in text as ˚n.

Free FD A free feature diagram (FFD) d ∈ FFD(NT,TCT,GCT,OE,ON) has node types are taken from NT, textual constraints types from TCT, graphical constraints from types from GCT, edge option types from OE, nodes option types from ON. d is a labeled graph with:

1. a set of nodes n ∈ N, (e.g. all text in any Figure), that can be labeled as:
   (a) the concept or root r. It is drawn at the top (e.g. Monitor Engine System in Fig. 1).
   (b) one node type from NT
   (c) one option type from ON

2. edges e ∈ E labeled either:
   (a) by a option type in OE
   (b) or by a constraint type from GCT
3. a set of **textual constraints** $\Phi$ with an $\alpha$-ary operator $o$ from TCT and $\alpha$ arguments nodes $n_1, n_2, \ldots, n_\alpha$. We will represent them by their syntax tree: an edge from the root to a node labeled by the constraint type $o$ and edges labeled $1, 2, \ldots, \alpha$ to the nodes $n_1, n_2, \ldots, n_\alpha$.

4. The **leaf features** or **leaves** $F$ are a distinguished subset of the nodes. We will draw them as nodes without sons. Three special nodes: $f$ (false), $t$ (true) and $v$ (base version) are nodes without sons that are not leaves.

$v$ is the and-node with 0 sons, $f$ the xor-node with 0 sons, $t$ is an and-node with 0 sons and an edge from the root.

Most authors use the word “feature” sometimes to mean a node, sometimes a leaf (e.g. the definition of feature in Section 1 can only refer to leaf features, but at most other places in [10] it means a node). Thus from now on, we use either “node” or “leaf” but not “feature”.

FFD collect whatever can be drawn. FD furthermore have minimal well-formedness conditions.

**Feature Diagram** A **feature diagram** (FD) is a FFD where:

1. There is a **unique** concept
2. Only the concept has no father
3. The hierarchy is acyclic: $\neg(n_1 \rightarrow \ldots \rightarrow n_k \rightarrow n_1)$

It is now easy to define all of the above variants, and more, simply by giving the operators, constraints, and options:

**Definition** An **original feature diagram** (OFD) [10] is a FD with:

1. operators *and*, *xor*,
2. textual constraints *mutex*, *requires*,
3. no graphical constraints,
4. optional or mandatory nodes,
5. all edges are mandatory.

**Definition** **RSEB feature diagrams** (RFD or SFD) [6] are FD with:

1. operators *and*, *xor*($\otimes$), *or*($\oplus$),
2. textual constraints *mutex*, *requires*,
3. no graphical constraints,
4. optional or mandatory nodes,
5. all edges are mandatory.

For any variant FD with binary textual constraints XFD (where X can be replaced e.g. by P, R, O, E), we define its graphical version GXFD where the same operators are noted graphically. The translations between these variants are obvious.

**Definition** **extended feature diagrams** (GEFD) [16] use VP and optional links:

1. operator $VP(l,u)$,
2. no textual constraints,
3. graphical constraints *mutex, requires*
4. all nodes are mandatory,
5. edges are optional (→) or mandatory (⇒).

**Definition** generative programming feature diagrams (GPFD) [2] use all of the above:

1. operator and, xor, or, VP(l..u),
2. no textual constraints,
3. graphical constraints mutex, requires
4. optional or mandatory nodes,
5. all edges are mandatory.

We will later use two more Spartan variants:

**Definition** Constraintless optionless FD (CFD(L)) use:

1. operators in L,
2. no textual constraints,
3. no graphical constraints,
4. all nodes are mandatory,
5. all edges are mandatory.

**Definition** Constraintless optionless OFD (COFD) = CFD(and, xor).

**Definition** Varied FD (VFD) = CFD(VP).

This leads to syntactical inclusions:

**Theorem 3.1**

1. COFD ⊂ OFD ⊂ RFD ⊂ PFD
2. VFD ⊂ EFD
3. VFD ⊂ PFD

and the corresponding inclusions for the graphical variants.

### 3.2. Semantics

The semantics of a FD is defined as a PL (Def. 3.2), i.e. a set of products (P). A product provides a set of features (F) (Def. 3.2).

The notion of model is introduced in [10, p.64], with the examples of models of X10 terminals.

**Definition** A model of a FD is a subset of its nodes, its edges, and its option circles: 

\[ M = \mathcal{P}(N + E + \dot{N}) \]

**Definition** A model \( m \in M \) is valid for a \( d \in FD \), noted \( m \vdash d \) iff:

1. The concept is in: \( r \in m \)
2. \( t \) is in: \( t \in m \)
3. The meaning of nodes is satisfied: If a o-node \( p \in m \) with \( s \) outgoing edges \( e_1, \ldots, e_s \) is in, then \( o_s(e_1 \in m, \ldots, e_s \in m) \) must evaluate to true.
4. The model must satisfy all textual constraints: \( \forall \phi \in \Phi, M \vdash \phi \), where \( M \vdash \phi \) means that we replace each node name \( n \) in \( \phi \) by its value \( n \in m \), evaluate it and get true.
5. The model must satisfy all graphical constraints: if there is an edge from \( n_1 \) to \( n_2 \) with binary operator \( o \), \( o(n_1 \in m, n_2 \in m) \) must be true.

6. Two semantics are possible here:
   - **node-based**: A mandatory edge \((f, s) \in m\) is in iff its son \( s \in m\) is in.
   - **edge-based**: If a mandatory edge \((f, s) \in m\) is in, then its son \( s \in m\) is in.

7. If a non-concept, non-true node is in the model, some of its fathers, called its *justification*, must be too: \( m \in M \land m \neq r \land m \neq t \Rightarrow \exists n. n \rightarrow ?m \land n \in M \)

We will often describe models by their nodes only, leaving to the reader to complete it according to the rules above.

**Example** In fig 5, \([l, m]\) is a valid edge-based model, but not a valid node-based model.

![Figure 5. edge- or node-based?](image)

Our semantics is node-based, following [10, p.70].

**Definition**
1. A *product* \( p \in P \) is a set of leaves: \( P \subseteq P F \).
2. The *product named by a model* \( M \), noted \([M]\), is \( M \cap F\). Note that many models are just names for the same product.
3. A *product line* is a set of products: \( PL \subseteq P \subseteq PP \).
4. The *product line of a FD* \( d \) is formed of the products named by its valid models: \([d] = \{M \cap F | M \models d\}\)

**Example** In our semantics, Fig. 4 admits a model with neither \( 1/km \) or \( miles/gallon \). Indeed, since the edges to these nodes are optional, one edge can be true even if its feature is false. When we compute the truth value of the parent *Measures*, we thus obtain true. This is clearly not the semantics intended in [16].

**Example** Our semantics also contradict the sentence: “All mandatory features are part of all [models],[22, p.2], [16, p.2]” Our notion of “mandatory” is only relative to the incoming edge(s), thus a mandatory feature will not be part of the model if none of its fathers is. [22, p.2], [16, p.2] nevertheless seems to agree with our interpretation:

In Fig. 6, extracted from [22, p.2], [16, p.2], *Email* is mandatory, but they make clear that it is not included in models where *Net* is not present.

**Example** The source of a *require* link cannot be a justification. Therefore composition edges and *require* links have a different semantics. Fig 7, for instance, has no model, while the same figure with *require* links replaced by edges has two models.

**Note** 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 [21] will be needed, and gives the same result for our simple case.
4. Formal comparison

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

4.1. Expressiveness

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

**Definition** The expressiveness of a language $L$ is the set $E(L) = \{\|D\| | D \in L\}$. A language $L_1$ is more expressive than a language $L_2$ if $E(L_1) \supset E(L_2)$. A language $L$ with semantic domain $\mathcal{M}$ (i.e. its semantics is $\|\cdot\| : L \rightarrow \mathcal{M}$) is expressively complete if $E(L) = \mathcal{M}$.

The usual way to prove that a language $L_2$ is at least as expressive as $L_1$ is to provide a translation from $L_1$ to $L_2$:

**Definition** A translation is a total function $T : L_1 \rightarrow L_2$ that is correct, i.e. preserves semantics: $\|T(l_1)\| = \|l_1\|$.  

The semantic domain of FD is $\mathcal{M} = PL \in \mathcal{PPF}$. 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 mutex, requires. It is the only part of OFD that nobody has ever extended, so every critic implicitly agreed that it was already “complete”. We can actually prove the reverse.

**Theorem 4.1** There is an OFD D (Fig. 12) and a set of models that no set \( \Phi \) of mutex, requires constraints can express, i.e., \( \{ M | M \models D \land M \models \varphi \} \neq \{ M | M \models D \land M \models \varphi \} \) \( \neq \{ M | M \models \varphi \} \)

**Proof** Any FD on the top row of Fig. 12 has 4 nodes and allows to choose any combination of the 3 leaves. Its PL has 8 models and products. Impose the constraint \( \varphi = C \lor D \). Its PL has 6 models and products. There are 4 values for \( n_1 \) and \( n_2 \), so 16 possible mutex constraints, and the same for requires. Thus \( 2^{4} \times 2^{4} \) sets are possible, but many express the same PL. None of these sets expresses \( \{ C \}, \{ D \}, \{ C, D \} \times \{ \emptyset, B \} \).

Note that this proof is expressed in COFD. Since COFD is included in all other FD variants, this proof also applies to them.

One should not over-interpret this theorem. First, it is true only of some fixed FD. The FD of Fig. 8 expresses this PL, and indeed OFD are expressively complete, Theorem 4.2. Second, if nesting of constraints were allowed, Sheffer [20] has shown that mutex alone 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.

**Definition** The first normal form of a pl \( N_1(pl) \) has as xor-nodes the root \( X = \{ r \} \), as and-nodes the products \( A = pl \), no optional nodes \( O = \emptyset \) edges from the root to each product, and from each product to its constituent features: \( n \rightarrow p \) iff \( n = r \land p \in pl \lor n \in pl \land p \in n \), and no constraints, see e.g. Fig. 8.

The product with no feature is represented by \( v \). The language of first normal forms is thus called \( N_1(PL) \).

![Figure 8. First normal form of the product line \( f1 \lor f2 \)](image)

**Theorem 4.2** Every PL can be expressed by a COFD, e.g. by its first normal form: \( \forall P \subseteq 2^F : \exists N_1(P) \in OFD.P = \llbracket N_1(P) \rrbracket \)

Thus all extensions of COFD (including all known FD variants) are exactly as expressive as COFD, since the latter already have maximal expressiveness.

Kang [10, p.87] also noted that OFD are redundant, but he proposed to get rid of the graphical part and use only textual constraints. Theorem 4.1 excludes this. One can, on the contrary, get rid of the constraints, and they disappeared indeed in a later version of his method (FORM: Feature-Oriented Reuse Method [11]), then reappeared in FOPLE (Feature-Oriented Product Line Engineering [12]). Another alternative is to choose a rich, but fixed, graphical part. Then all the information is in the constraints:
Definition The second normal form \( N_2(L) \) of \( L \), noted \( r, \{ r \}, P(F), \emptyset, F, \{ x \to p \mid x = r \wedge p \subseteq F \vee x \in A \wedge l \in x \} \), has the root as single xor-node, all possible combinations of features as and-nodes, no optional nodes. Then we use a mutex with root to exclude the combinations that should not be included in the PL.

Theorem 4.3 For any set of leaf features \( F \), there is a FD \( N_2(F) \) such that any PL \( L \) on \( F \) can be expressed by constraints on \( N_2(F) \).

This does not contradict Theorem 4.1, but shows that some FD with many nodes provide enough grip to constraints, while simple FD do not.

One might ask whether COFD are the smallest expressively complete sublanguage. On one hand, we cannot remove one of its two operators.

Theorem 4.4 CFD(and) is expressively incomplete.

Proof \( P \in \| \text{CFD(and)} \| \) iff \( P \) has exactly one model.

Theorem 4.5 CFD(xor) is expressively incomplete.

Proof \( P \in \| \text{CFD(xor)} \| \) iff each model of \( P \) contains exactly one path from the concept.

On the other hand, one node type is enough (e.g. VP, see Theorem 4.6 or nand, see [20]).

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. A natural translation [4,13] should be done without a global reorganization:

Definition A context-free language \( L_1 \) is embeddable into \( L_2 \) iff there is a translation \( T : L_1 \to L_2 \) that is:

1. correct: \( \forall l_1 \in L_1, \| T(l_1) \| = \| l_1 \| \).
2. compositional (or modular or homomorphic or natural): \( T(C_1(e)) = C_2(T(e)) \), for all constructs \( C_1 \) of \( L_1 \), where \( e \) is a vector of meta-variables, adequately typed wrt. \( C_1 \)'s subterms expected types, and \( C_2 \) is an expression in language \( L_2 \) containing the same meta-variables. \( C_2 \) is usually noted \( T(C_1) \). A natural translation is called an embedding.

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 compositionaly cannot apply. We generalize it:

Graphical embeddability A graphical language \( L_1 \) is embeddable into \( L_2 \) iff there is a translation \( T : L_1 \to L_2 \) that is:
1. *node-controlled*[9]: \( T \) is expressed as a set of rules \( D_1 \rightarrow D_2 \), where \( D_1 \) is a diagram containing a distinguished node (or edge) \( n \), and all possible relations with this node. Its translation \( D_2 \) is a subgraph in \( L_2 \), plus how the existing relations should be connected to nodes of this new subgraph.

Our second definition is a generalisation of the first one, if we view syntax trees as labeled graphs, and allow syntax trees with sharing. Due to this sharing, a graphical embedding is always linear in graph size.

Embeddability is widely considered as an adequate notion for comparing “natural expressiveness” of languages. Specially, non-trivial self-embeddability is considered as *harmful redundancy*. Said otherwise, the language is unnecessarily complex because it contains easily definable constructs. 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. The embedding also gives this constant, so it is worth looking at the proof.

We first show that COFD can embed (G)OFD, then that EFD have such a sublanguage, VFD.

**Theorem 4.6**

1. OFD is embeddable into COFD (and of course conversely);
2. EFD, PFD, VFD are mutually embeddable.

**Proof**

1. Fig. 9,
2. Fig. 10,

Note that variation points with multiplicity 0, noted VP(0 \ldots 0), cannot be removed: they express negation.

The rules of Fig 10 show that PFD and EFD are harmfully redundant. Obviously, no construct can be removed from VFD, since it has only one.

### 4.3. Succinctness

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

**Succinctness** Let \( \mathcal{G} \) be a set of functions from \( \mathbb{N} \rightarrow \mathbb{N} \). A language \( L_1 \) is \( \mathcal{G} \)-as succinct as \( L_2 \), noted \( L_1 \leq \mathcal{G} L_2 \), if there is a translation \( T : L_1 \rightarrow L_2 \) that is within \( \mathcal{G} \): \( \exists g \in \mathcal{G}, \forall n \in \mathbb{N}, \forall l_1 \in L_1, |l_1| \leq n \Rightarrow |T(l_1)| \leq g(n) \). 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”.

Thus, this implies that \( L_2 \) is at least as expressive as \( L_1 \).

In Fig. 11, BC are the combinatorial boolean electronic circuits [18,19,23]. They are built from mutex gates [20]. Their semantics is different, because the value of the node is simply \( n(e_1, \ldots, e_n) \) and there is no need for justifications. Thus, a BC and FD with the same drawing have a different semantics.

**Theorem 4.7**

Note: the graphical variants are linearly translatable.

1. \( VFD \leq 3 \cdot EFD, VFD \leq 3 \cdot PFD \).
2. \( COFD \leq O(EFD^3) \).
Instead of . . . write . . .

\begin{center}
\begin{tabular}{|l|l|}
\hline
Instead of . . . & write . . . \\
\hline
an optional node \( \hat{o} \) & root \( \hat{o} \) \\
a xor-node & \( m \n \) \\
an or-node & \( m \& n \) \\
a and-node with number of sons \( s \) & \( m \& n \) \\
\( n_1 \) mutex \( n_2 \) & \( m \& \neg n \) \\
\( n_1 \) requires \( n_2 \) & \( m \) \hspace{1cm} \text{not-}n \\
\hline
\end{tabular}
\end{center}

Figure 9. Embedding OFD into COFD

\begin{center}
\begin{tabular}{|l|l|}
\hline
Instead of . . . & write . . . \\
\hline
an optional node \( o \) & a fresh VP 0 . . . 1 with son \( o \) \\
a xor-node & a VP 1 . . . 1 \\
an or-node & a VP 1 . . . * \\
a and-node with number of sons \( s \) & a VP \( s \ldots s \) \\
\( n_1 \) mutex \( n_2 \) & a VP 1 . . . 1 with sons \( n_1, n_2 \) \\
\hline
\end{tabular}
\end{center}

Figure 10. Embedding PFD into VFD in 3n
3. \( COFD \leq O(BC) \).
4. \( BC \leq O(RFD) \).

**Proof**

1. Fig 10.
2. We translate each variation point \( v \) of multiplicity \( \mu \ldots \nu \) by a subgraph of quadratic size in the number of sons \( q \). Assume its sons are ordered as \( s_1, \ldots, s_q \).

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 \((i + 1, j)\).

3. A \((x|y)\) gate is translated to a xor2(and2(x, y), t). This translation loses justifications. To re-instate them, we add an optional node from the root to each feature.
4. COFD, OFD and RFD can be translated linearly to BC. Indeed, their semantics (Definition 3.2) is simply a boolean formula, that can be seen as a tree-shaped BC with \( s \)-ary operators and, or, xor. But a and, or, can be translated linearly in \( s \) by associativity to the binary case, then by the formulae below. The xor translation is done by introducing two wires going from one son to the next: the first says whether we have seen exactly 0 true sons up to now, the second whether we have seen exactly 1 true sons up to now (and it will be the final result). They cannot be both true, but if they are both false, it means that we have already seen too many true nodes and we can stop the evaluation.

\[
\neg x = x|xx \land y = \neg(x|y)x \lor y = \neg x|\neg x y \Rightarrow y = x|\neg yx \lor y = (x|y) \land (x \lor y) \quad (1)
\]

**Example**

The COFD of Fig 5 is translated to the formula \( r \land (r \Rightarrow x \lor y) \land (x \Rightarrow l \lor m) \land (y \Rightarrow m \lor n) \land (l \Rightarrow x) \land (m \Rightarrow (x \lor y)) \land (n \Rightarrow y) \) that we can see as a BC (with arbitrary operators). Then we translate all gates of this BC to \(|\). Putting this result together with result 2 above, we see that we don’t have to pay the quadratic a second time, since VP are translated to xor2 nodes with two sons only. Thus the translation of EFD, VFD, PFD to BC is cubic.

These results compose, e.g. \( VFD \leq 3\cdot OFD \) by using (1) and (3). Further, the inclusions of Theorem 3.1 can be used, as well as the linear translations between a textual and the
corresponding graphical variant. Thus we have essentially two levels of succinctness: COFD, OFD, RFD, BC are at the same level, while VP make a cubic difference for VFD, EFD, PFD.

All formalisms agree on the constraints that can be expressed polynomially, which is surprising when considering the necessary complexity of any PL language.

**Theorem 4.8** Any expressively complete language for PL must have PL that are expressed exponentially in the number of leaf features.

**Proof** If $f$ is the number of leaf features, there are $2^f$ PL to express. In order to give a different encoding to each of them, $2^f$ bits are required.

Thus this common polynomial family is well-known:

**Theorem 4.9** A series of PL is expressed polynomially by FD with operators and textual constraints having PBC (including COFD, OFD, RFD, GPFD, EFD, VFD) iff it has a polynomial boolean circuit (PBC) [14].

4.4. Redundancy

Another criticism [15] of OFD is that they are ambiguous:

**Ambiguity** A diagram (or sentence) is ambiguous iff it has two different meanings: $\|D_1\| \neq \|D_2\|$. A language is ambiguous iff it contains an ambiguous diagram or sentence.

This is obviously impossible with a formal semantics, since $\|\cdot\|$ is a function. The semantics presented here is already present (albeit in English) in [10], thus we were surprised to see this claim in [15].

Let us examine the “proof” of [15] in Fig. 12.

![Ambiguity example](image)

**Figure 12.** Ambiguity example [15]

This shows that $\|D_1\| = \|D_2\|$. Let us name this property:
**Harmless redundancy, normal form** A language $\mathcal{L}$ is harmlessly redundant iff $\exists D_1, D_2 \in \mathcal{L} \cdot D_1 \neq D_2 \land \|D_1\| = \|D_2\|$. Otherwise, it is a normal form (NF).

All the FD seen here, except the two NF (Def. 4.1 and 4.1), are harmlessly redundant. This is the price to pay to have a decently natural language: in a normal form, all structure has to be lost (Corollary 4.1).

Riebisch proposed to solve the problem by adding multiplicities and optional edges to OFD, stating:

Such multiplicities cannot be expressed using the previous notations. [15, p.67] contradicting thus Theorem 4.2. While the idea is good *per se*, it obviously cannot solve any of the alleged problems:

**Theorem 4.10** If a language is ambiguous, adding constructs will keep it ambiguous.

**Theorem 4.11** If a language is harmlessly redundant, adding constructs will keep it harmlessly redundant.

**Theorem 4.12** If a language is harmfully redundant, adding constructs will keep it harmfully redundant.

We cannot make a theorem about lack of precision, since the FD seen here are always fully precise due to their formal definition; but it is clear that adding constructs without a clear semantics (e.g. [15] never distinguished node- and edge-based semantics) also cannot help.

4.5. Decision problems.

In this section, we consider three decision problems, which could be implemented by CASE tools 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 and its product. 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.

**satisfiability** A FD diagram $d$ is satisfiable iff $\|d\| \neq \emptyset$.

**Theorem 4.13** Deciding whether a FD is satisfiable 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. 10, 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 provide examples of products that are in the difference.
Theorem 4.14 Deciding whether two FD have the same valid models is coNP-complete.

Theorem 4.15 Deciding whether two FD are equivalent is coNP-complete. Giving a product in their difference is NP-complete.

Proof Boolean circuit equivalence is coNP-complete [14].

Theorem 4.1 For any normal form NF for which equivalence can be decided in polynomial time, putting a FD into NF is coNP-complete.

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 must inspect their embeddings and their succinctness. VFD is then a clear 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.

5.1. Past and future work

A feature is formalized here as a meaningless symbol: we just follow [10] in that. In [17], 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 (MDA) 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 [17], 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


