# Towards the Comparative Evaluation of Feature Diagram Languages

Patrick Heymans, Pierre-Yves Schobbens, Jean-Christophe Trigaux*, Raimundas Matulevičius, Andreas Classen and Yves Bontemps

PReCISE research centre, Computer Science Faculty, University of Namur
21, rue Grandgagnage – B-5000, Namur (Belgium)
{phe,pys,jtr,rma,aclassen,ybo}@info.fundp.ac.be

**Abstract.** This paper proposes a path to defragmenting research on feature diagram languages: (1) a global quality framework to serve as a language quality improvement roadmap; (2) a set of formally defined criteria to assess the semantics-related qualities of feature diagram languages; (3) a systematic method to formalise these languages and make them ready for comparison and efficient tool automation. The novelty of this paper resides in the latter point and the integration. The results obtained so far are summed up and future works are identified.

## 1 Introduction

During the last fifteen years or so, more than ten different *Feature Diagram* (FD) languages were proposed starting from the seminal work of Kang *et al.* on FODA back in 1990 [1]. An example of a FODA FD appears in Fig. 1. We assume the reader is familiar with the notation.

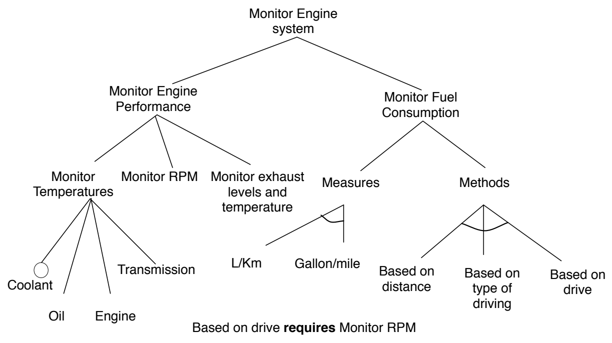

**Fig. 1.** FODA example (inspired from [2])

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

Since Kang *et al.*'s proposal, several extensions to FODA were devised [3–9] (see also Table 1). When looking at these FD languages, one immediately sees aesthetic differences (see, e.g., Fig.2). Although concrete syntax is an important issue in its own right [10], this work focusses on what is really behind the pictures: *semantics*. We noticed that proponents of FD languages often claimed for added value of their language in terms of precision, unambiguity or expressiveness. Nevertheless, our previous work [11–15] demonstrated that the terminology and evaluation criteria that they used to justify these claims were often vague, and sometimes even misleading. We also tried to give a precise meaning to the constructs of those languages.

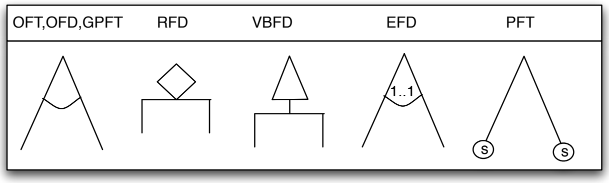

**Fig. 2.** Concrete syntaxes for *xor*-decomposition

Although we note that recent research has devoted more attention to the semantic foundations of these languages [16–22], we still lack concrete means to discriminate between these proposals.

This paper suggests a method to evaluate and compare FD languages focused on the study of their semantics. This method relies on formally defined criteria and terminology, based on the highest standards in formal language definition [23]. It is also situated with respect to SEQUAL [24, 25], a comprehensive framework for assessing and improving the quality of modelling languages.

In Section 2, we briefly present SEQUAL. Section 3 recalls good language definition principles from [23]. On these grounds, Section 4 continues with the definition of the criteria that our method aims to investigate: *expressiveness*, *embeddability* (also called *naturalness*), *succinctness* and (computational) *complexity*. The method is described in Section 5 and constitutes the main contribution of this paper. Section 6 summarises the results obtained so far [13–15]. The paper finishes by discussing the current limitations of the method and the remaining research challenges (Section 7), before it concludes (Section 8). This paper is short version of a technical report [26][1].

## 2 Quality of Models and Languages

Assessing and improving the quality of modelling is a complex and multidimensional task. A comprehensive view of the concerns involved is given in the SEQUAL (semiotic

[1] Available at `http://www.info.fundp.ac.be/~phe/docs/papers/TechRep_Eval_FP_of_FDL_06.pdf`

quality) framework, developed over the last decade by Krogstie *et al.* [25]. SEQUAL is based on a distinction between *semiotic levels*: syntactic, semantic and pragmatic. It adheres to a constructivistic world-view that recognises model creation as part of a dialog between participants whose knowledge changes as the process takes place.

SEQUAL is amenable to specific criteria and guidelines by tailoring. Its main advantages are that (1) it helps situate one's investigations within a comprehensive quality space, (2) it acts as a checklist of qualities to be pursued and (3) it recommends general guidelines on how to proceed.

Our investigation is targeted *semantic* and *pragmatic* qualities of FDs which we have found to be somehow neglected in the current state of the art. So doing, we will see that we inevitably interfere with the other qualities, mainly *syntactic* quality.

The problem we encounter is that representative objects of study – models – do not always exist, or at least are not easily available. And this is indeed the case for FDs which (1) are an emerging modelling paradigm, and (2) have the purpose of representing highly strategic company information. Since representative models[2] are almost nowhere to find, we concentrate on improving the quality of FD *languages*.

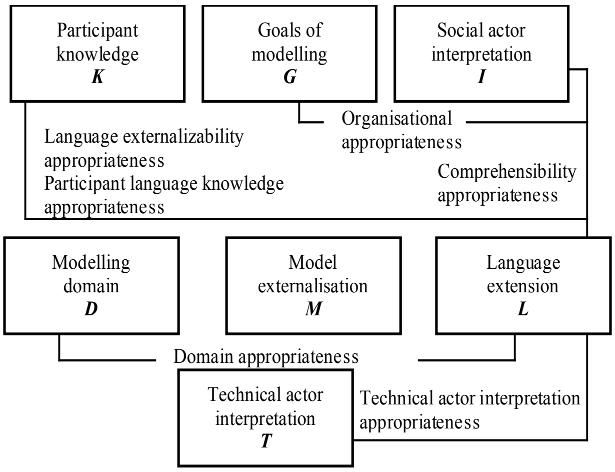
Participant knowledge K
Goals of modelling G
Social actor interpretation I
Organisational appropriateness
Language externalizability appropriateness
Participant language knowledge appropriateness
Comprehensibility appropriateness
Modelling domain D
Model externalisation M
Language extension L
Domain appropriateness
Technical actor interpretation T
Technical actor interpretation appropriateness

**Fig. 3.** SEQUAL : Language Quality [24, 25]

SEQUAL has been adapted to evaluate language appropriateness [24] (see Fig. 3). Six quality areas were proposed. *Domain appropriateness* means that language *L* must be powerful enough to express anything in the domain *D*, and that, on the other hand it should not be possible to express things that are not in *D*. *Participant language knowledge appropriateness* measures how the statements of *L* used by the participants match the explicit knowledge *K* of the participants. *Knowledge externalisability appropriate-*

[2] Except illustrative examples used in research papers.

*ness* means that there are no statements in *K* that cannot be expressed in *L*. *Comprehensibility appropriateness* means that language users understand all possible statements of *L*. *Technical actor interpretation appropriateness* defines the degree to which the language lends itself to automatic reasoning and supports analysability and executability. Finally, *organisational appropriateness* relates *L* to standards and other needs within the organisational context of modelling.

Not being able to assess model qualities directly, our investigations were re-targeted at three main language qualities: *domain appropriateness*, *comprehensibility appropriateness* and *technical actor interpretation appropriateness*. The matching of the investigated criteria wrt these qualities is further discussed in Section 7. In the next section, we will first introduce the basic notions behind these criteria (Section 3), and then the criteria themselves (Section 4).

## 3 Formal definition of visual languages

In [23], Harel and Rumpe recognise that: *"Much confusion surrounds the proper definition of complex modelling languages [...]. At the root of the problem is insufficient regard for the crucial distinction between syntax and true semantics and a failure to adhere to the nature and the purpose of each."* [23] Although they are far less complex than, e.g., the UML[3], we demonstrated in previous papers [11–14] that FDs were also "victims" of similar "mistreatments".

Harel and Rumpe make it clear that the unambiguous definition of any modelling language must consist of three equally necessary elements: a *syntactic domain* ($\mathcal{L}$), a *semantic domain* ($\mathcal{S}$) and a *semantic function* ($\mathcal{M}$) (see Fig. 4). All three should be defined through explicit, rigid and unambiguous rules, hence the use of mathematics.

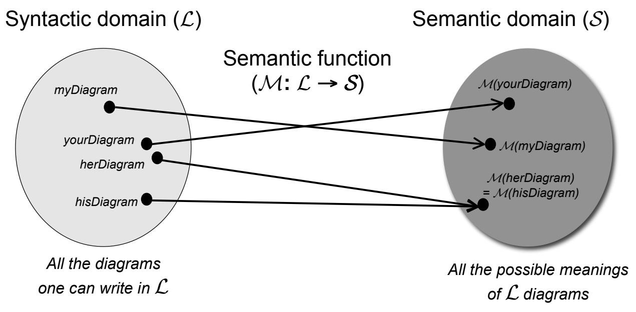
Syntactic domain ($\mathcal{L}$)
Semantic domain ($\mathcal{S}$)
Semantic function
($\mathcal{M}: \mathcal{L} \rightarrow \mathcal{S}$)
myDiagram
yourDiagram
herDiagram
hisDiagram
$\mathcal{M}$(yourDiagram)
$\mathcal{M}$(myDiagram)
$\mathcal{M}$(herDiagram)
= $\mathcal{M}$(hisDiagram)
All the diagrams
one can write in $\mathcal{L}$
All the possible meanings
of $\mathcal{L}$ diagrams

**Fig. 4.** The 3 constituents of a formal language

During our survey, we could observe that many FD languages were never formally defined. Maybe, some answers to why this is so are given in [23] where the authors point

[3] In [23], one of Harel and Rumpe's main motivations is to suggest how to improve the UML.

out of set of frequent misconceptions about formal semantics, e.g., “Semantics is the metamodel”, “Semantics is dealing with behaviour”, “Semantics is being executable”, “Semantics means looking mathematical”, etc. This folklore is demystified [23]. For now, we turn to the definitions of $\mathcal{L}$, $\mathcal{S}$ and $\mathcal{M}$.

## 3.1 Syntax

*Concrete syntax* is the *physical representation* of the data (on screen, or on paper) in terms of lines, arrows, closed curves, boxes and composition mechanisms involving connectivity, partitioning and “insideness” [23].

Although discouraged by best pratice, most of the (informal) definitions of the semantics of FDs we found in the literature were based on concrete syntax, usually discussed on FD examples. Most of the time, a substantial part of the semantics was implicit, leaving it to the diagrams to “speak for themselves”.

The *abstract syntax* ($\mathcal{L}$) is a representation of data that is independent of its physical representation and of the machine-internal structures and encodings. It thus makes the syntactic rules simpler and more portable. The set of all data that comply with a given abstract syntax is called the *syntactic domain*.

In [13, 14], we provided an abstract syntax (and semantics) for several FD languages at once through a generic mathematical structure we called FFD (see Fig. 5 and Table 1). $\mathcal{L}_{FFD}$ has 4 parameters reflecting the 4 abstract syntax variation points we observed among languages: the *graph type* ($GT$ = TREE or DAG[4]), the *node types* ($NT$, i.e. what decomposition operators can be used: *and*, *xor*, *or*,...), the additional *graphical constraint types* used ($GCT$, usually `requires`/$\Rightarrow$ and `mutex`/$|$), and the *texual constraint language* ($TCL$, usually Composition Rules (CR) [1]).

## 3.2 Semantics

The *semantic domain* ($\mathcal{S}$) *“[...] specifies the very concepts that exist in the universe of discourse. As such, it serves as an abstraction of reality, capturing decisions about the kinds of things the language should express”.* $\mathcal{S}$ is a mathematical domain built to have the same structure as the real-world objects the language is used to account for, up to some level of fidelity. The semantic domain that we have proposed for FODA-inspired languages is named $PL$ (Product Lines) [13, 14]. It is recalled in Definition 1. It assumes that FDs are graphs whose nodes ($N$) represent features and where $P$, a subset of $N$, is the set of features that the user considers relevant. We call $P$ the set of *primitive features/nodes*[5]:

**Definition 1 (Configuration, Product, Product Line).** *(1) A* configuration *is a set of nodes, i.e., any element of $\mathcal{P}N$. (2) A* product *is a configuration that contains only primitive features, i.e., any element of $\mathcal{P}P$. (3) A* product line *is a set of products, i.e., any element of $PL = \mathcal{P}\mathcal{P}P$.*

[4] Directed Acyclic Graph.

[5] Hence, *primitive* nodes and *leaf* nodes are different concepts, although the former usually includes the latter, but can include intermediate nodes as well; this is up to the modeller.

Other formalisations [16–22] chose semantic domains different from *PL*, for example using lists instead of sets [22] or keeping the full shape of the FD [19]. How to compare *PL* with other semantic domains will be discussed in Section 5.

The *semantic function* $\mathcal{M} : \mathcal{L} \to \mathcal{S}$ eventually assigns a meaning in $\mathcal{S}$ to each syntactically correct diagram $d$, noted $\mathcal{M}[\![d]\!]$. Again, a mathematical definition is recommended. In [13, 14], we defined a generic semantic function ($\mathcal{M}_{FFD}$) giving a semantics to several FD languages at once (see Fig. 5).

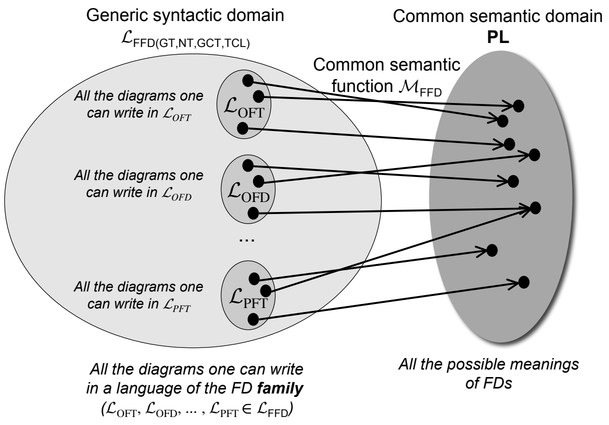
Generic syntactic domain
$\mathcal{L}_{\text{FFD(GT,NT,GCT,TCL)}}$
Common semantic domain
**PL**
Common semantic function $\mathcal{M}_{\text{FFD}}$
All the diagrams one can write in $\mathcal{L}_{OFT}$
$\mathcal{L}_{\text{OFT}}$
All the diagrams one can write in $\mathcal{L}_{OFD}$
$\mathcal{L}_{\text{OFD}}$
...
All the diagrams one can write in $\mathcal{L}_{PFT}$
$\mathcal{L}_{\text{PFT}}$
All the diagrams one can write in a language of the FD **family** ($\mathcal{L}_{\text{OFT}}, \mathcal{L}_{\text{OFD}}, \dots, \mathcal{L}_{\text{PFT}} \in \mathcal{L}_{\text{FFD}}$)
All the possible meanings of FDs

**Fig. 5.** Semantics for a family of FD languages

Since $\mathcal{M}$ is a *function*, there is at most one semantics for each diagram. Ambiguity in this context is therefore not possible. The term “ambiguity” was not always properly used in the surveyed literature. For example, FODA FDs have been criticised for being ambiguous [8]. However, having reconstructed a proper formal semantics from the original plain English definition [1], we could check that this was not the case [11].

Finally, the semantic function should be *total*, that is, it should not be possible to have a diagram in $\mathcal{L}$ which is not given a meaning in $\mathcal{S}$ by $\mathcal{M}$. The converse question (is every element in $\mathcal{S}$ expressible by a diagram in $\mathcal{L}$?) is called the *expressiveness* of a language and is another term used confusingly in the literature. It is clarified, together with other comparison criteria, in the next section.

## 4 Comparison criteria

When a language receives a formal semantics, it can then be evaluated according to various objective criteria. We first address (computational) *complexity*. In a formal language, we can precisely define *decision problems*, i.e., tasks to be automated. A mathematical definition of the tasks is necessary to prove the correctness of algorithms. It also allows to study complexity, thereby assessing their scalability. Results give an indication about the worst case, and how to handle it. Heuristics taking into account the most usual cases can be added to the backbone algorithm, to obtain practical efficiency.

In [13], we studied the complexity of a selection of FD-related decision problems: (1) *satisfiability*: given a diagram $d$, is $\mathcal{M}[\![d]\!] = \emptyset$ true? (2) *equivalence*: given two diagrams $d_1$ and $d_2$, is $\mathcal{M}[\![d_1]\!] = \mathcal{M}[\![d_2]\!]$ true? (3) *model-checking* (called *product-checking* for FDs): given a product $c$ and a diagram $d$, is $c \in \mathcal{M}[\![d]\!]$ true? (4) *intersection*: compute a new diagram $d_3$ such that $\mathcal{M}[\![d_3]\!] = \mathcal{M}[\![d_1]\!] \bigcap \mathcal{M}[\![d_2]\!]$. (5) *union*: compute a new diagram $d_3$ such that $\mathcal{M}[\![d_3]\!] = \mathcal{M}[\![d_1]\!] \bigcup \mathcal{M}[\![d_2]\!]$. (6) *reduced product*: compute a new diagram $d_3$ such that $\mathcal{M}[\![d_3]\!] = \{c_1 \cup c_2 | c_1 \in \mathcal{M}[\![d_1]\!], c_2 \in \mathcal{M}[\![d_2]\!]\}$.

When languages, in addition to having a formal semantics, also share a *common* semantic domain ($\mathcal{S}$), we can compare them with additional criteria. We use three common criteria:

- *expressiveness*: what can the language express?
- *embeddability* (or *macro-eliminability*): when translating a diagram to another language, can we keep its structure?
- *succinctness*: how big are the expressions of a same semantic object?

Formal semantics opens the way for a fully formal definition and objective assessment of these criteria. For example, Def. 2 naturally formalizes *expressiveness* as the part of a languages's semantic domain that its syntax can express. Fig. 6 illustrates it.

**Definition 2 (Expressiveness).**
*The* expressiveness *of a language* $\mathcal{L}$ *is the set* $E(\mathcal{L}) = \{\mathcal{M}[\![d]\!] | d \in \mathcal{L}\}$*, also noted* $\mathcal{M}[\![\mathcal{L}]\!]$*. 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* $\mathcal{S}$ *is* expressively complete *if* $E(\mathcal{L}) = \mathcal{S}$*.*

Since languages compete for expressiveness, it often happens that they reach the same maximal expressiveness (like $\mathcal{L}_W$ in Fig. 6). This is for instance the case for programming languages, that are almost all Turing-complete and can thus express the same computable functions. Consequently, we need finer criteria than expressiveness to compare these languages.

In [13], we recalled equally formal definitions of embeddability and succinctness, which are widely accepted criteria in the formal methods community. We cannot reproduce them here for lack of space, so we just present them informally and motivate them. The results obtained by applying them to FDs are detailed in Section 6.

*Embeddability* is of practical relevance because it questions the existence of a transformation from one language to the other which preserves the whole shape of the diagrams and generates only a linear increase in size. This way, traceability between the two diagrams is greatly facilitated and tool interoperability is made more transparent. Furthermore, limiting the size of diagrams helps avoiding tractability issues for reasoning algorithms taking the diagrams as an input. Most importantly, embeddability can also exist between a language and a subset of itself. A language that is non-trivially self-embeddable [13] is called *harmfully redundant*. This means that it is unnecessarily complex: all diagrams can be expressed in the simpler sublanguage without loss of structure and with only a linear increase in size.

In case linear translations are not possible, the blow-up in the size of the diagram must be measured by *succinctness*. If $\mathcal{L}_1$ is more succinct than $\mathcal{L}_2$, this usually entails that $\mathcal{L}_1$'s diagrams are likely to be more readable. Also, if one needs to translate from $\mathcal{L}_1$ to $\mathcal{L}_2$[6], succinctness will be an indicator of the difficulty to maintain traceability between the orginal and the generated diagram. Traceability of linear translations is usually easier but is likely to become more difficult as the size of the generated diagrams grows bigger. However, this should not be concluded too hastily since succinctness does not provide information on the structure of the generated diagrams[7]. In this sense, succinctness is a coarser-grained criteria than embeddability.

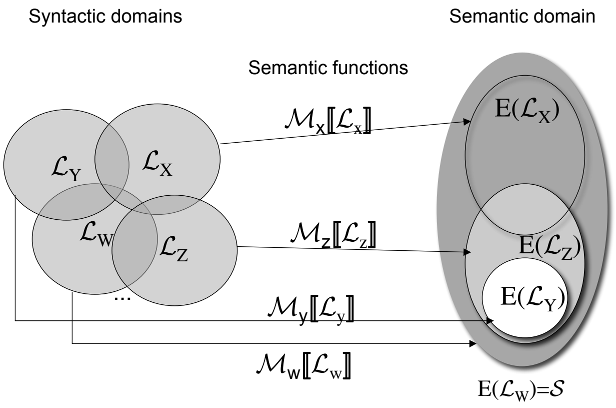
Syntactic domains
Semantic domain
Semantic functions
$\mathcal{M}_x[\![\mathcal{L}_x]\!]$
$\mathcal{M}_z[\![\mathcal{L}_z]\!]$
$\mathcal{M}_y[\![\mathcal{L}_y]\!]$
$\mathcal{M}_w[\![\mathcal{L}_w]\!]$
$\mathcal{L}_Y$
$\mathcal{L}_X$
$\mathcal{L}_W$
$\mathcal{L}_Z$
...
$E(\mathcal{L}_X)$
$E(\mathcal{L}_Z)$
$E(\mathcal{L}_Y)$
$E(\mathcal{L}_W)=\mathcal{S}$

**Fig. 6.** Comparing expressiveness

## 5 A Comparison Method for FD languages

In order to compare FD languages $X_1, \ldots, X_n$ according to the criteria exposed in the previous section, we need formally defined languages. That is, for language $X_i$, we need $\mathcal{L}_{X_i}$, $\mathcal{S}_{X_i}$ and $\mathcal{M}_{X_i}$. To compare expressiveness, embeddability and succinctness, we also need to have $\mathcal{S}_{X_1} = \mathcal{S}_{X_2} = \ldots = \mathcal{S}_{X_n}$. Unfortunately, this ideal situation almost never occurs in practice. Instead, we have to cope with:

- languages that have *no formal semantics* (this is the most frequent case [13, 14]),
- languages with a *formal semantics defined in a different way from* [23],
- or languages with a *formal semantics* compliant with [23] *but different semantic domains*.

[6] E.g., because a tool for achieving some desired functionality is only available in $\mathcal{L}_2$.
[7] However, looking at the transformation's definition will provide the information.

Hence, the overall comparison process should be carried out in two steps: (1) make the languages suitable for comparison, (2) make the comparisons. We now detail the first step.

Let $X_1$ be the language we want to compare with the others $(X_2, ..., X_n)$ which, we assume, are fully and clearly formalised according to [23] and have identical semantic domains. We distinguish three cases:

### 5.1 Case 1: $X_1$ has no formal semantics

There are two alternatives:

- The first alternative is to define the syntax and semantics for each FD language individually following [23]. That is, we define $X_1$ independently from $X_2, ..., X_n$. This is what we did in [11] where we formalised FODA FDs (OFT) [1]. FORM FDs (OFD) [3] are treated the same way in [26].
- The second alternative is to make scale economies and define several languages at once. In [14], we observed that most of the FD languages largely share the same goals, the same constructs and, as we understood from the informal definitions, the same (FODA-inspired) semantics. For this reason, we proposed to define not one FD language but a family of related FD languages (see Fig. 5). We defined a parametric abstract syntax, called FFD, in which parameters correspond to variations in $\mathcal{L}_{X_1}, ..., \mathcal{L}_{X_n}$. This definition follows, but slightly adapts, the principles of Section 3. The semantic domain ($PL$) and semantic function are common to all FD variants, maximizing semantic reusability. With this method, we are confined to handle languages whose only significant variations are in abstract syntax. For languages with very different semantic choices, e.g. [19], it is much harder to describe (and justify) the introduction of variation points in the semantics. Then, we should rather follow either the first alternative in Case 1 if the language is informal, or Cases 2 or 3 otherwise.

### 5.2 Case 2: $X_1$ has formal semantics but $\mathcal{L}_{X_1}$, $\mathcal{S}_{X_1}$ and $\mathcal{M}_{X_1}$ need to be clarified

Another frequent case is when $X_1$ actually has a formal semantics, but irrespective of [23]. That is, we cannot see explicit and self-contained mathematical definitions of $\mathcal{L}_{X_1}$, $\mathcal{S}_{X_1}$ and $\mathcal{M}_{X_1}$. Typically, $\mathcal{L}_{X_1}$ is clear and self-contained, but $\mathcal{S}_{X_1}$ and $\mathcal{M}_{X_1}$ are not. Most of the time, the semantics of $X_1$ is given by describing a transformation of $X_1$'s diagrams to another language, say $W$, which is formal. $W$ does not even need to be a FD language, and usually it is not. Therefore, the semantic domain might be very different from the one intuitively thought of for FDs. The main motivation for formalising this way is usually because $W$ is supported by tools. The problem is that these kinds of "indirect", or tool-based, semantics complicate the assessment of the language[8].

Several proposals of this kind for FDs can be found in recent work [17–22]. We thus need to reformulate the semantics of those languages. In [15], we treated the FD language proposed by van Deursen and Klint [22] (renamed vDFD) before comparing

[8] Even more if $W$'s semantics also does not follow [23].

it to FFD. The main difference w.r.t. Case 1 is that here formalisation decisions are usually much more straightforward since they have already been made. However, they might be hard to dig out if they are coded in some tool. Also, formalisations are not necessarily error-free, and errors can thus be discovered when re-formalising [15].

### 5.3 Case 3: $X_1$ has formal semantics with clear $\mathcal{L}_{X_1}$, $\mathcal{S}_{X_1}$ and $\mathcal{M}_{X_1}$ but $\mathcal{S}_{X_1} \neq \mathcal{S}_{X_2}, ..., \mathcal{S}_{X_n}$

The third and last case is when we have a clear and self-contained mathematical definition of $\mathcal{L}$, $\mathcal{S}$ and $\mathcal{M}$ for all languages (either from the origin, or having previously gone through Case 1 or 2) but the semantic domains of the languages differ. We thus need to define a relation between the semantic domains. We met this problem, for instance, when comparing vDFD with FFD [15]. On the one hand, we had $\mathcal{S}_{FFD} = PL = \mathcal{PP}P$ (sets of sets of nodes), and on the other, $\mathcal{S}_{vDFD} = OON$ (lists of lists of nodes). The latter introduces an order relation on features, and one on products. Comparing languages with different semantic domains is actually possible, but it requires preliminary work which is now explained.

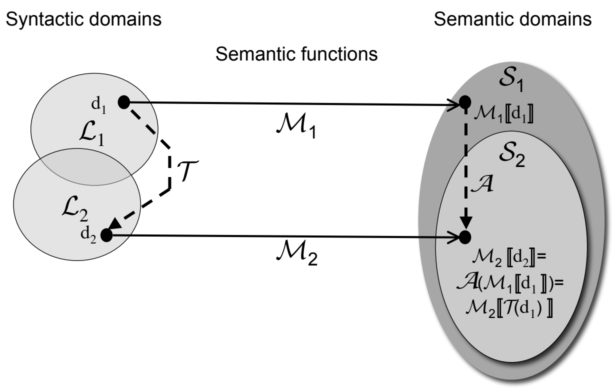

**Fig. 7.** Abstracting a semantic domain

We need to define an *abstraction function* ($\mathcal{A}$ in Fig. 7) whose purpose is to remove extra information from the richer domains and keep the "core" of the semantic domain, where we will perform the comparisons. We used such a function to remove the ordering of features and products from $\mathcal{S}_{vDFD}$ [15]. However, the question of the relevance of this discarded information remains and should be studied carefully. A fairly general case is illustrated in Fig. 7, where domain $\mathcal{S}_1$ contains more information than $\mathcal{S}_2$; we then take $\mathcal{S}_2$ as the common domain. $\mathcal{A}$ removes extra information from elements of $\mathcal{S}_1$ and maps them in $\mathcal{S}_2$. It then makes sense to look for quasi-translations $\mathcal{T} : \mathcal{L}_1 \to \mathcal{L}_2$ between their syntactic domains. They are translations for the abstracted semantics $\mathcal{A} \circ \mathcal{M}_1$, and can thus be used to compare languages for expressiveness, embeddability or succinctness. Hence, if we apply $\mathcal{T}$ to a diagram $d_1$ in the syntactic domain $\mathcal{L}_1$ we will obtain a diagram $d_2$ in the syntactic domain $\mathcal{L}_2$ with the same abstracted semantics. Semantically, if we apply the semantic function $\mathcal{M}_1$ to $d_1$ and then the abstraction function $\mathcal{A}$, we will map on the same element of $\mathcal{S}_2$ as if we apply $\mathcal{T}$ to $d_1$ and then $\mathcal{M}_2$: $\mathcal{A}(\mathcal{M}_1[\![d_1]\!]) = \mathcal{M}_2[\![\mathcal{T}(d_1)]\!]$.

When applied to more than two languages, this method will create many semantic domains related by abstraction functions. The abstraction functions can be composed and will describe a category of the semantic domains. At the syntactic level, the translations can also be composed to yield expressiveness and succinctness results. Similarly, the composition of embeddings yields an embedding.

## 6 Language Evaluation Results

We summarise the results obtained by applying our general comparative semantics method. For the languages defined generically with FFD (see Table 1), the details and proofs can be found in [13]. The treatment of vDFD [22] is found in [15].

| Survey short name | GT | NT | GCT | TCL |
|---|---|---|---|---|
| **OFT [1]** | TREE | $and \cup xor \cup \{opt_1\}$ | $\emptyset$ | CR |
| **OFD [3]** | DAG | $and \cup xor \cup \{opt_1\}$ | $\emptyset$ | CR |
| **RFD [4]=VBFD [9]** | DAG | $and \cup xor \cup or \cup \{opt_1\}$ | $\{\Rightarrow, \mid\}$ | CR |
| **EFD [7,8]** | DAG | $card \cup \{opt_1\}$ | $\{\Rightarrow, \mid\}$ | CR |
| **GPFT [5]** | TREE | $and \cup xor \cup or \cup \{opt_1\}$ | $\emptyset$ | CR |
| **PFT [6]** | TREE | $and \cup xor \cup or \cup \{opt_1\}$ | $\{\Rightarrow, \mid\}$ | $\emptyset$ |
| **VFD [13]** | DAG | $card$ | $\emptyset$ | $\emptyset$ |

**Table 1.** FD languages defined through FFD

### 6.1 Complexity

For FDs, solving all the standard problems of Section 4 turns out to be practically useful:

- *Equivalence* of two FD is needed whenever we want to compare two versions of a product line (for instance, after a refactoring). When they are not equivalent, the algorithm can produce a product showing their difference. For FD languages based on DAGs, and that allow non-primitive features, such as OFD, EFD, VFD, this problem is $\Pi_1$-complete [13] (just above NP-complete).
- *Satisfiability* is a fundamental property. It must be checked for the product line but also for the intermediate FDs produced during a staged configuration [20]. For FD languages based on DAGs, this problem is NP-complete.

- *Model-checking* verifies whether a given product (made of primitive features) is in the product line of a FD. It is not as trivial as expected, because the selection performed for non-primitive nodes must be reconstructed. This gives an NP-complete problem. When recording this selection, the problem becomes linear again.
- *Union* is useful when parallel teams try to detect feature interference in FDs. Their work can be recorded in separate FDs, the union of which will represent the validated products. For FD languages based on DAGs, this problem is solved in linear time, but the resulting FD should probably be simplified for readability. *Intersection* and *reduced product* are similar.

The complexity results show the role of non-primitive features. On one hand, it is useful to record them to accelerate the checking of products. However, they should not become part of the semantics since this would restrict the expressiveness and strongly reduce the possible transformations of diagrams.

## 6.2 Expressiveness

The distinction between languages that only admit trees and the ones that allow sharing of features by more than one parent (DAGs or vDFD) turns out to be important. While tree-shaped languages are usually incomplete, OFD [3] are already expressively complete without the constraints, and thus *a fortiori* RFD [4], EFD [7, 8] and VFD [13]. vDFD are “almost” trees in that only terminal features (i.e. the leaves) can have multiple parents (justifications), but this is sufficient to obtain expressive completeness.

In contrast, tree-shaped diagrams turned out to be expressively incomplete; in particular, OFT [1] cannot express disjunction. This justifies *a posteriori* the proposal [9] (VBFD) to add the *or* operator to OFT. But even so, we do not attain expressive completeness: this language is still unable to express $card_3[2..2]$, the choice of two features among three[9]. This justifies similarly the proposal [7] (EFD) to use the *card* operators. Both [9] and [7] also propose to allow DAGs: this extension alone, as we have seen, ensures expressive completeness. But we will see below better justifications in terms of embeddability rather than succinctness.

When designing a FD language, is thus essential to have more than trees to reach expressive completeness. Trees, however, are easier to understand and manipulate because they have a compositional semantics. vDFD [22] manage to have both advantages.

## 6.3 Embeddability

An optional node $n$ can be translated into a $xor_2$-node, say $n?$, with two sons: the original node $n$, and the TRUE node $v$ which is an $and_0$-node (i.e., with no son). As we see in Fig.8, all incoming edges from parents of $n$ are redirected to the new top node ($n?$), and all outgoing edges to sons start from the node $n$. This supports our view [13] that optionality is better treated as a decomposition operator ($opt_1$).

We constructed an embedding from OFD without constraints (called COFD in [13]) to VFD, presented in Table 2. To save space, we use the textual form for the graphs. For

[9] Operator arity is denoted by an underscript.

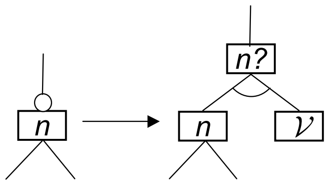

**Fig. 8.** Graphical embedding of redundant optional node (in OFD concrete syntax)

instance, a node bearing a $xor_m$ operator is translated to a node bearing a $card_m[1 \ldots 1]$ operator. In the next section, we will consider how those embeddings increase the size of the graph. Here we see that the VFD resulting from the embedding of a COFD diagram has the same size. This result indicates that *card*-nodes proposed by [7] can embed all the other constructs. We proposed thus to use them systematically inside tools. We slightly differ from [7] that also uses optional edges: these can be modelled by $card_1[0..1]$-nodes and would be harmfully redundant. We proposed VFD to eliminate this slight drawback. Please note that this latter suggestion only concerns abstract syntax. In the concrete syntax, it is probably a good idea to keep optional nodes as this would decrease the size and visual complexity of the diagrams.

| **Instead of ...** | **write ...** |
|---|---|
| $opt_1(f)$ | $card_1[0 \ldots 1](f)$ |
| $xor_m(f_1, \ldots, f_m)$ | $card_m[1 \ldots 1](f_1, \ldots, f_m)$ |
| $and_s(f_1, \ldots, f_s)$ | $card_s[s \ldots s](f_1, \ldots, f_s)$ |

**Table 2.** Embedding COFD into VFD

### 6.4 Succinctness

When translations are not linear, it is still interesting to compute the increase in size of the graph, as measured by succinctness. RFD and OFD are of similar succinctness, but when translating VFD or EFD to OFD we translate a $card_k$-node to a OFD graph of size $O(k^2)$ [13]. A VFD of size $O(k)$ could contain $k$ $card_k$-nodes, giving a cubic translation at the end: COFD $\leq O(VFD^3)$. This result indicates again that *card*-nodes are a useful addition, but for different reasons than presented in [7].

## 7 Discussion

The main limitation of our work is explicit in its scope: we address only *formal semantics*-related properties. In order not to over-interpret our conclusions, one should keep comprehensive view of model quality in mind. With respect to SEQUAL (Section 2), in order to be accurate and effective, we deliberately chose to address only part of the required qualities: *Domain appropriateness* is addressed by looking at language *expressiveness*. *Comprehensibility appropriateness* is addressed by looking at *embeddability* and *succinctness*. *Technical actor interpretation appropriateness* is addressed by looking at *complexity* and also *embeddability* and *succinctness*. Furthermore, our criteria cover only part of each of the three qualities. Future research should therefore devote similar attention to other qualities and criteria.

In contrast, a more holistic (quality-wise) attempt to compare FD languages is reported in [27]. It is specific though in the sense that it concerns the usage of FDs in a particular company, for a given kind of project. This leads us to point out that the notion of a "good" modelling language is only relative to the context of use of the language. The priorities to be put on the expected qualities and criteria are very likely to be different from one company, or projet, to another. This could lead us to relativise in some contexts the importance of formality. Still, we think that for FDs formality is very likely to deliver more than it will cost since (1) languages are relatively simple, (2) formality can be made largely transparent to the users (hidden behind a graphical concrete syntax), (3) the automation possibilities are many [13, 14, 28], and (4) correct FDs are mission-critical company assets that should suffer no ambiguity.

SEQUAL also helps identify another limitation: for now, we have only looked at *language* quality adopting a theoretical approach. A complementary work is to investigate models *empirically*. In Section 2, we emphasised the difficulty of such an endeavour because of the limited availability of "real" FDs. Nevertheless, we do not consider it impossible and can certainly learn a lot by observing how practitioners create and use FDs. Although we have focussed on studying theoretical properties of FD languages, we need to recognise that no formal semantics, nor criteria, can ever guarantee by itself that the languages help capture the right information (neither too little, nor too much) about the domain being modelled. Only empirical research can help us give a convincing answer to this other aspect of domain appropriateness.

A *threat to validity* is that all our reasoning (comparisons, demonstrations of theorems) was done by humans only, no tools. Human errors, miss- or over-interpretations are thus possible. Also, our formalisations were made only by considering the published documents, and without contacting the authors for clarifications, nor testing their tools. However, making $\mathcal{L}$, $\mathcal{S}$ and $\mathcal{M}$ explicit, we open the way for constructive discussion.

Finally, our method is yet to be applied to some relevant FD language proposals [16–21]. This is a prioritary topic of future work.

## 8 Conclusion

The bad news confirmed by this paper is that current research on variability modelling is fragmented. Existing research in the field is characterised by a growing number of proposals and a lack of accurate comparisons between them. In particular, the formal underpinnings of feature diagrams need more careful attention.

The nocuous consequences of this situation are: (1) the difficulty for practitioners to choose appropriate feature modelling techniques, (2) an increased risk of ambiguity in models, (3) underdeveloped, suboptimal or unsafe (i.e., not proved correct) tool support for reasoning on feature diagrams.

The good news that this paper delivers is that there are remedies to this situation. The ones that we propose are: (1) a global quality framework (e.g. Krogstie *et al.*'s SEQUAL) to serve as a roadmap for improving the quality of feature modelling techniques; (2) a set of formally defined criteria to assess the semantics-related qualities of feature diagram languages; (3) a systematic method to formalise these languages and make them ready for comparison and efficient tool automation; and (4) a first set of results obtained from the application of this systematic method on a substantial part of the feature modelling languages encountered in the literature.

Although the road ahead is still quite long, we are confident that the community can take profit of our proposal. It could be used for example as part of an arsenal to elaborate a standard feature modelling language. This standard would not suffer from ambiguity, and its formal properties (among others) would be well known, allowing to devise proved correct efficient reference algorithms. A similar approach could also be transposed to cognate areas where existing modelling techniques face similar challenges. In particular, we think of goal modelling techniques.

## References

1. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)
2. Cohen, S., Tekinerdogan, B., Czarnecki, K.: A case study on requirement specification: Driver Monitor. In: Workshop on Techniques for Exploiting Commonality Through Variability Management at the Second International Conference on Software Product Lines (SPLC2). (2002)
3. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: A feature-oriented reuse method with domain-specific reference architectures. Annals in Software Engineering **5** (1998) 143–168
4. Griss, M., Favaro, J., d'Alessandro, M.: Integrating Feature Modeling with the RSEB. In: Proceedings of the 5th International Conference on Software Reuse (ICSR'98), Vancouver, BC, Canada (1998) 76–85
5. Eisenecker, U.W., Czarnecki, K.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)
6. Eriksson, M., Börstler, J., Borg, K.: The PLUSS Approach - Domain Modeling with Features, Use Cases and Use Case Realizations. In: Proceedings of the 9th International Conference on Software Product Lines (SPLC 2005). (2005) 33–44
7. Riebisch, M., Böllert, K., Streitferdt, D., Philippow, I.: Extending Feature Diagrams with UML Multiplicities. In: Proceedings of the Sixth Conference on Integrated Design and Process Technology (IDPT 2002), Pasadena, CA (2002)
8. Riebisch, M.: 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)
9. van Gurp, J., Bosch, J., Svahnberg, M.: On the Notion of Variability in Software Product Lines. In: Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01). (2001)
10. Moody, D.L.: What Makes a Good Diagram? Improving the Cognitive Effectiveness of Diagrams in IS Development. In: Proceedings of the 15th international conference in Information Systems Development (ISD 2006). (2006)

11. Bontemps, Y., Heymans, P., Schobbens, P.Y., Trigaux, J.C.: Semantics of FODA Feature Diagrams. In Männistö, T., Bosch, J., eds.: Proceedings of Workshop on Software Variability Management for Product Derivation: Towards Tool Support, Boston (2004) 48–58
12. Bontemps, Y., Heymans, P., Schobbens, P.Y., Trigaux, J.C.: Generic Semantics of Feature Diagrams Variants. In: Proceedings of the 8th International Conference on Feature Interactions in Telecommunications and Software Systems(ICFI), IOS Press (2005) 58–77
13. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Computer Networks (2007) special issue on feature interactions in emerging application domains **51** (2007) 456–479
14. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Feature Diagrams: A Survey and A Formal Semantics. In: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06), Minneapolis, Minnesota, USA (2006) 139–148
15. Trigaux, J.C., Heymans, P., Schobbens, P.Y., Classen, A.: Comparative semantics of Feature Diagrams : FFD vs vDFD. In: Proceedings of Workshop on Comparative Evaluation in Requirements Engineering (CERE'06), Minneapolis, Minnesota, USA (2006)
16. Asikainen, T., Mannisto, T., Soininen, T.: A Unified Conceptual Foundation for Feature Modelling. In: Proceedings of the 10th International Software Product Line Conference. (2006) 31–40
17. Batory, D.S.: Feature Models, Grammars, and Propositional Formulas. In: Proceedings of the 9th International Conference on Software Product Lines (SPLC 2005). (2005) 7–20
18. Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Automated Reasoning on Feature Models. LNCS, Advanced Information Systems Engineering: Proceedings of the 17th International Conference, CAiSE 2005 **3520** (2005) 491–503
19. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing Cardinality-based Feature Models and their Specialization. Software Process: Improvement and Practice **10** (2005) 7–29
20. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged Configuration Using Feature Models. Software Process Improvement and Practice, special issue on Software Variability: Process and Management **10** (2005) 143 – 169
21. Sun, J., Zhang, H., Li, Y.F., Wang, H.: Formal Semantics and Verification for Feature Modeling. In: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2005. (2005) 303–312
22. van Deursen, A., Klint, P.: Domain-Specific Language Design Requires Feature Descriptions. Journal of Computing and Information Technology **10** (2002) 1–17
23. Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer **37** (2004) 64–72
24. Krogstie, J.: Using a semiotic framework to evaluate UML for the development of models of high quality. Unified Modeling Language: System Analysis, Design and Develoment Issues, IDEA Group Publishing (2001) 89–106
25. Krogstie, J., Sindre, G., Jørgensen, H.: Process Models Representing Knowledge for Action: a Revised Quality Framework. Eur. J. Inf. Syst. **15** (2006) 91–102
26. Heymans, P., Schobbens, P.Y., Trigaux, J.C., Matulevičius, R., Bontemps, Y., Classen, A.: Evaluating Formal Properties of Feature Diagrams. Technical report, University of Namur (2006)
27. Djebbi, O., Salinesi, C.: Criteria for Comparing Requirements Variability Modeling Notations for Product Lines. Workshop on Comparative Evaluation in Requirements Engineering (CERE'06) **0** (2006) 20–35
28. Benavides, D., Ruiz-Cortés, A., Trinidad, P., Segura., S.: A Survey on the Automated Analyses of Feture Models. In: Jornadas de Ingeniería del Software y Bases de Datos (JISBD). (2006)