



# Formalizing Feature Models

Uni.Lu - May 24<sup>th</sup> 2006

[Pierre-Yves.Schobbens@info.fundp.ac.be](mailto:Pierre-Yves.Schobbens@info.fundp.ac.be)

[Patrick.Heymans@info.fundp.ac.be](mailto:Patrick.Heymans@info.fundp.ac.be)

[Jean-Christophe.Trigaux@info.fundp.ac.be](mailto:Jean-Christophe.Trigaux@info.fundp.ac.be)

[Yves.Bontemps@smals-mvm.be](mailto:Yves.Bontemps@smals-mvm.be)

FUNDP - Institut d'Informatique  
21 rue Grandgagnage  
B-5000 Namur  
<http://www.info.fundp.ac.be>



# PRECISE

**7 professors**

Jean-Luc Hainaut

Pierre-Yves Schobbens

Naji Habra

Vincent Englebert

Michaël Petit

Patrick Heymans

Stéphane Faulkner



**Axes of research**

Database Engineering

Interoperability &  
Re-Engineering

Software Methodology  
& Modelling

CASE tools

Software Quality  
Metrics

Requirements  
Engineering

**30 researchers**



**network of  
industrial partners**



**spinoff**



**technology  
transfer center**

**20 on-going  
research projects**



## Formalizing Feature Models: Outline

- Research context
  - Software Product Lines
  - Feature Models
- Research method
- Achievements
- Future works



# Research context

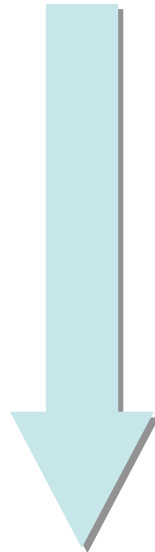
Software Product Lines

Feature Models



## Approaches to Software Reuse

**Least abstract**



1960s - Subroutines

1970s - Modules

1980s - Objects

1990s - Component-based systems

21<sup>st</sup> century - Software Product Lines

**Most abstract**

[Michael D. Conahan,2002]



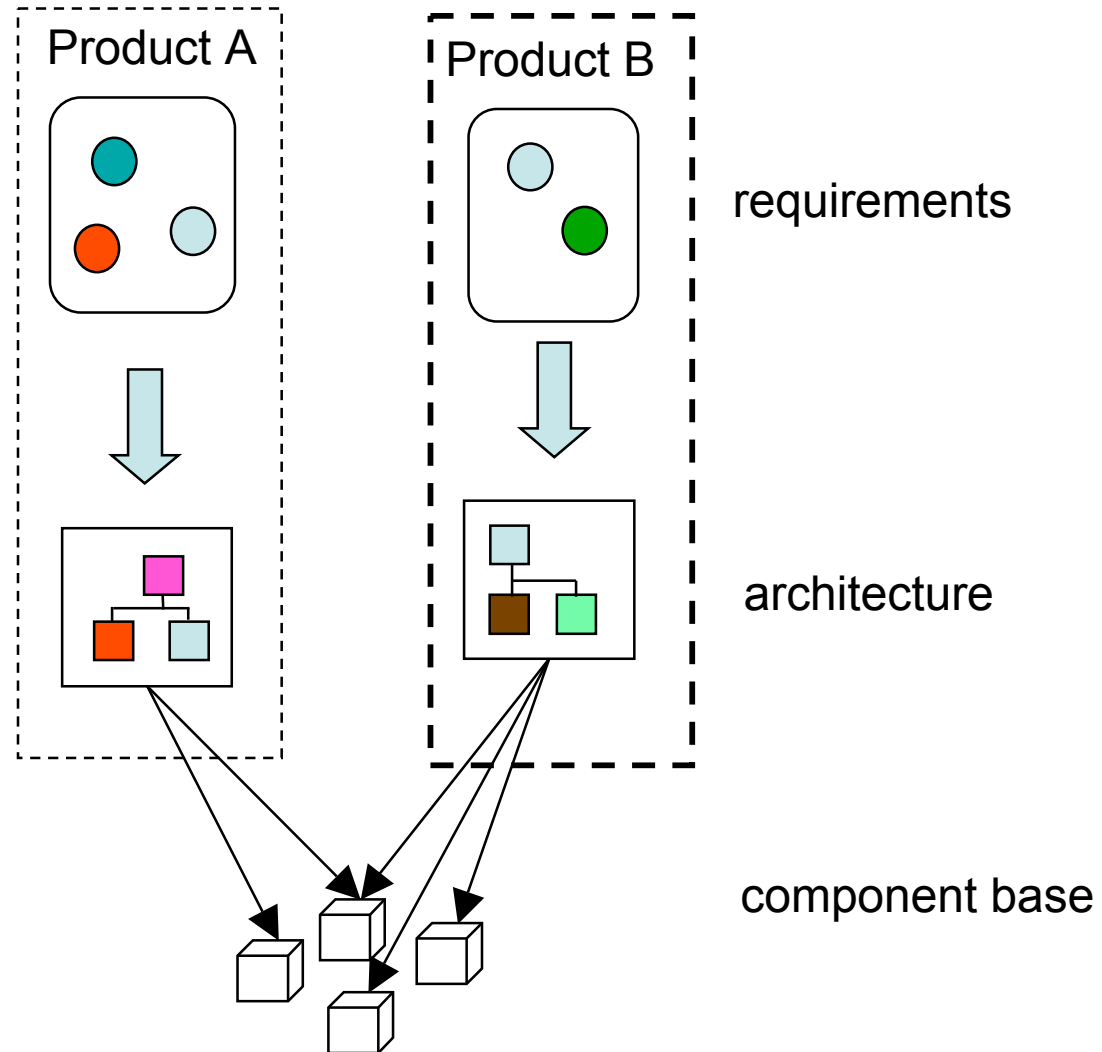
## Single Product Devlpt vs. SPL Devlpt

- Not one product but several which
  - Share commonalities
  - Differ through key variabilities
- **Improve reusability** and **Manage variability** during the entire software development:
  - Requirements
  - Architecture
  - Code
  - Tests



## Component-Reuse vs. SPL Devlpt

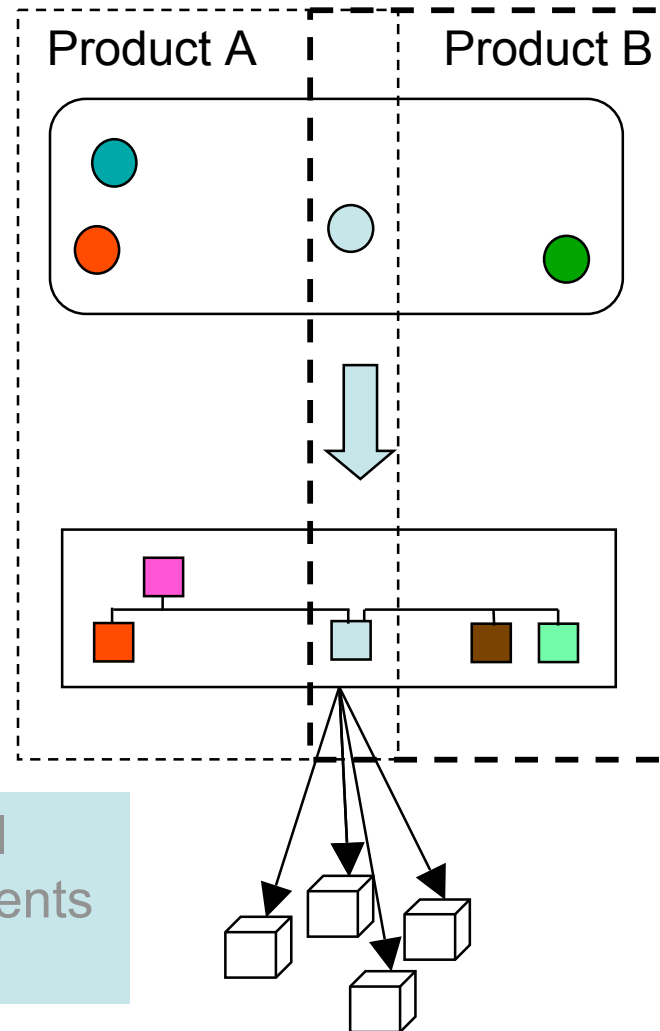
component-  
reuse





## Component-Reuse vs. SPL Devlpt

software  
product lines



requirements

architecture

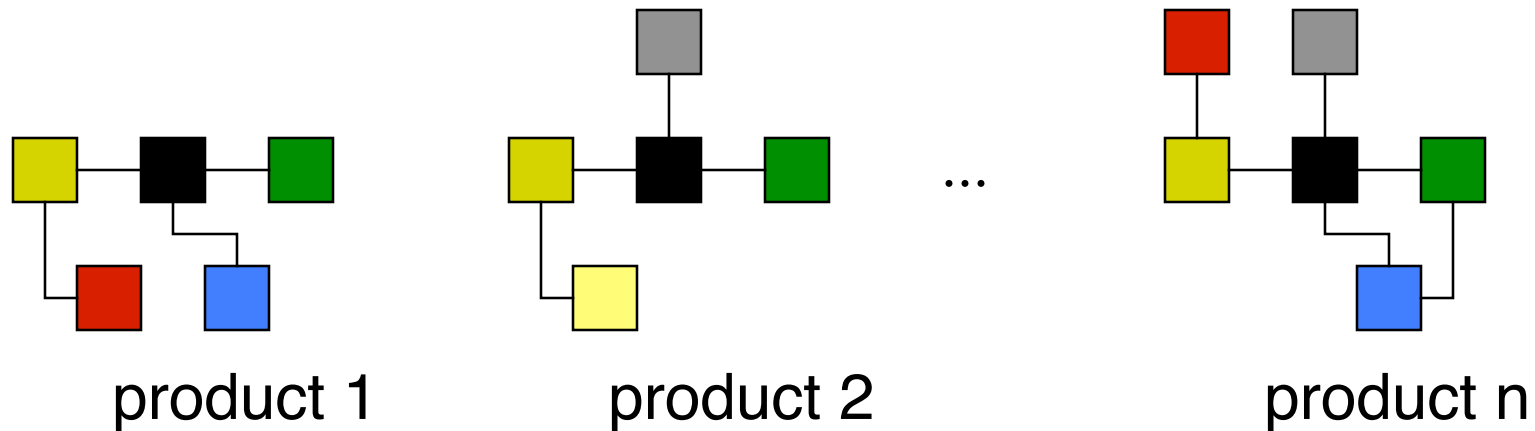
component base

→ Planned reuse of all  
assets - from requirements  
to test cases



## Traditional Approach

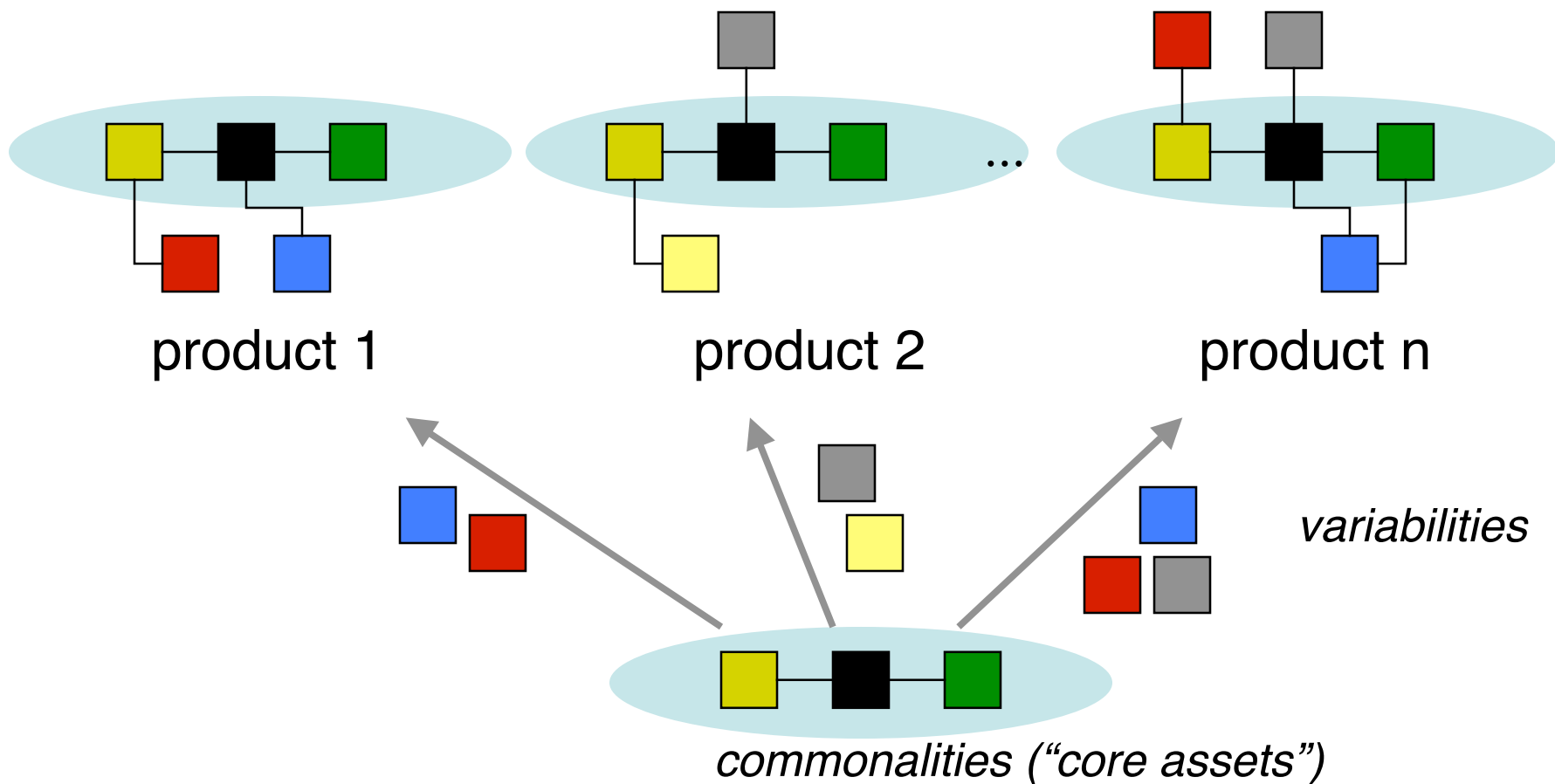
- From mass production...





## Software Product Lines approach

- ...to mass customization



adapted from [Benavides,2005]



## Software Product Lines: Hall of Fame

- Mainly Embedded Systems
  - Television Systems (Philips)
  - Medical Systems (Philips)
  - Mobile Phones (Nokia)
  - Printers (HP)
  - Car Systems (Daimler Chrysler)
  - Home Service Robots
  - Home automation Systems
- Few Information Systems
  - Stock market software



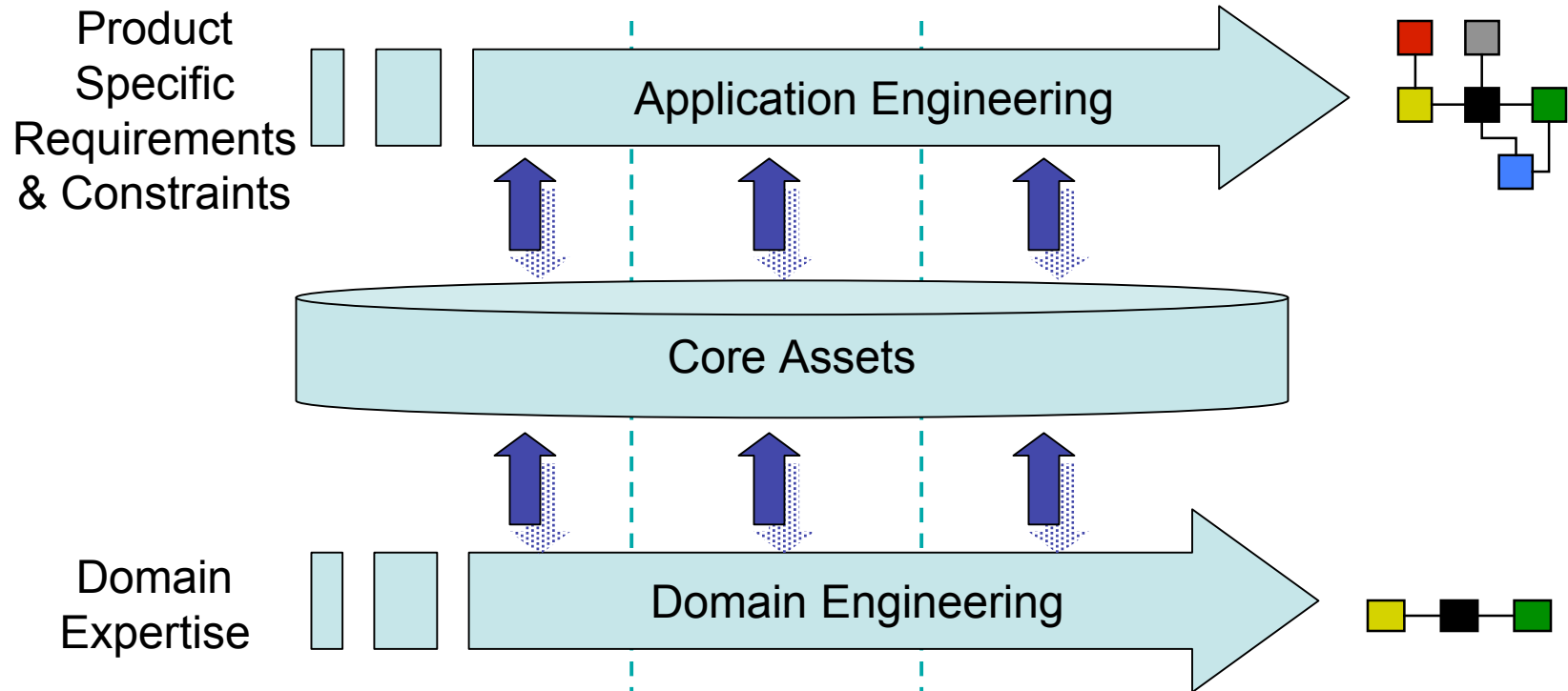
## Software Product Lines: A Definition

“A *software product line* is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

Carnegie Mellon Software Engineering Institute



# Software Product Lines Engineering

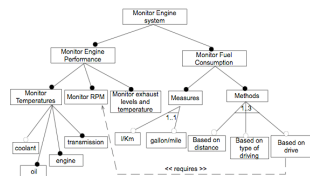


Requirements

Design

Code

*Feature Model*



adapted from [Benavides,2005]



## Feature Models (Diagrams, Trees)

- What is a Feature Model?
  - “A feature model describes all possible configurations of a software system, focusing on the features that may differ in each of the configuration.” [Van Deursen and Klint]
  - “A feature diagram is a graphical notation for describing dependencies between (variable) features.” [Van Deursen and Klint]



## Feature Models (Diagrams, Trees)

- Two views on Feature Models :
  - A feature model is used to represent variabilities and to organize, classify and represent dependencies between features.
  - A feature model is used as a decision tree to select features and to derive a configuration corresponding to one product.

## Feature

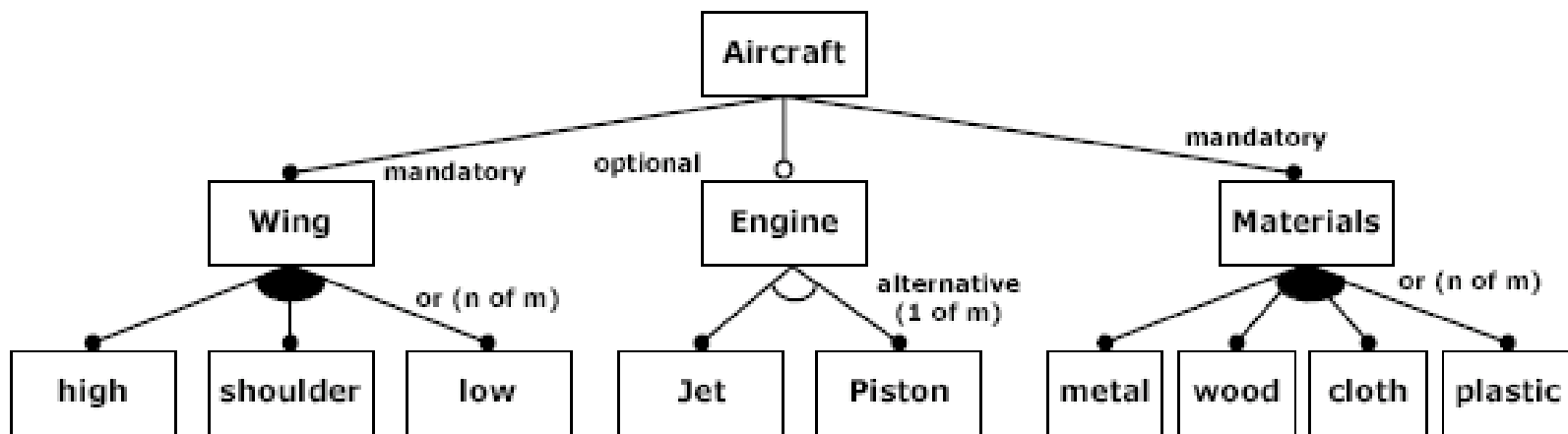
- How to define a Feature?
  - A **feature** is “anything users or client programs might want to control about a concept” [K. Czarnecki and U.W. Eisenecker]
  - A **feature** is “a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems” [Kang]
  - A **feature** is “an increment in functionality” [Batory]
  - A **feature** is “a distinguishable characteristic of a system that is relevant to a stakeholder of the system” [ODM]
  - A **feature** describes “a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements” [Chen]

## Feature

- In the SPL context, we “use” this definition:
  - A **feature** is a distinguishable characteristic of a (software) product that is relevant to a stakeholder and allows to differentiate one (software) product to another in a (software) product line.
  - A **characteristic** mainly consists of a cohesive set of requirements but could also include other software artefacts (components, tests, (sub-)models, codes,...).

## Feature Models (Diagrams, Trees)

### How it looks like?



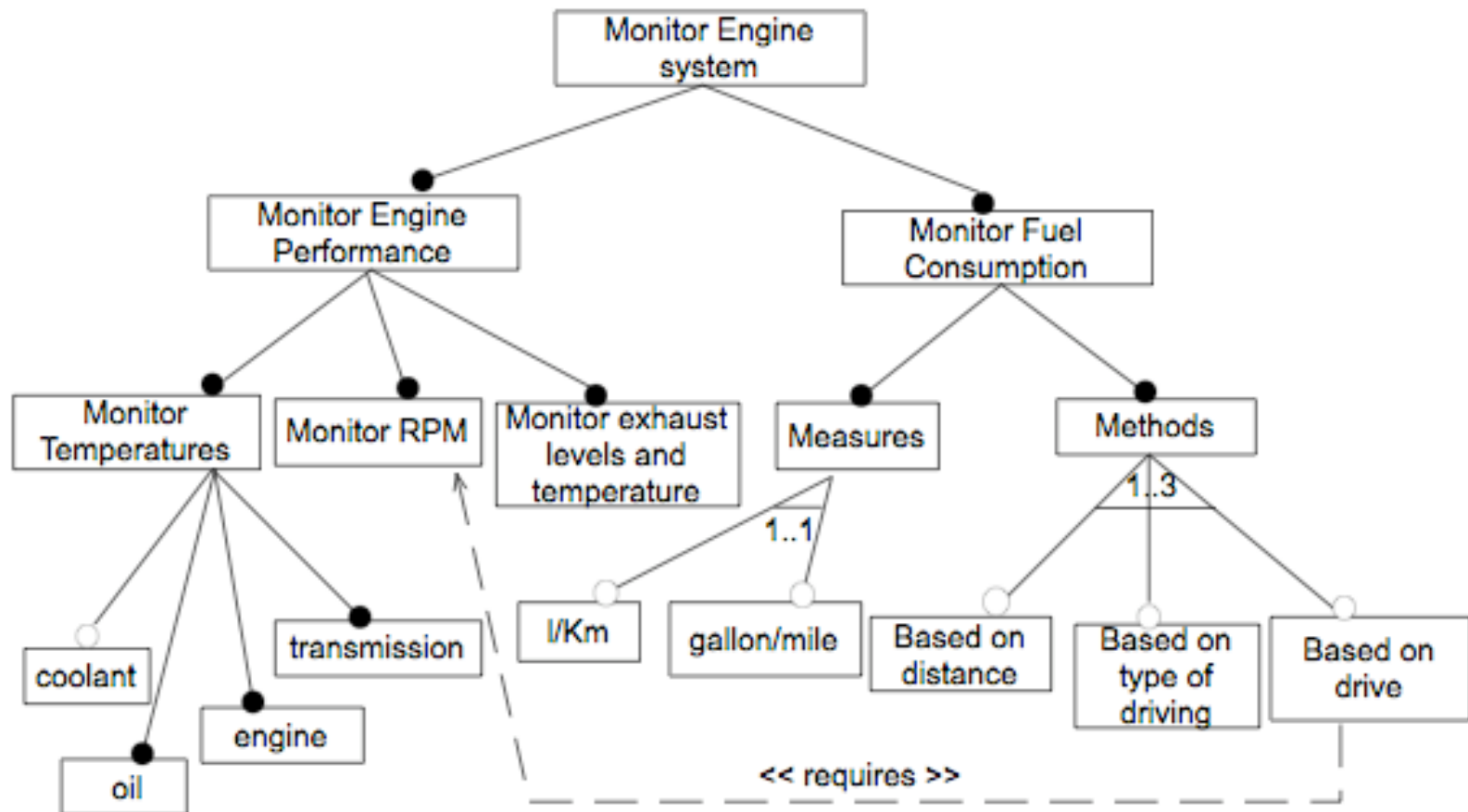
#### Example products:

- An aircraft with a low wing, piston engine and made of metal, wood and cloth: **Robin DR-400**
- An aircraft with shoulder wing, no engine and made of plastic: **ASW-27**
- An aircraft with low wing, jet engine(s) and made of metal: **Airbus A320**





## Feature Models (Diagrams, Trees)





# Research method



## Research Method

- The problems we encounter with FD
  - Many different syntaxes and few semantics
  - How to compare these syntaxes and semantics?
  - Interoperability?
- The solutions we propose
  - Define a generic semantics for a family of FD
    - Generic construction for FD (FFD)
    - A semantics for FFD
  - Formal comparison criteria (expressiveness, succinctness,...)
  - Comparative semantics approach



## Research Method: Semantics

Modelling languages should be defined as rigorously as programming languages to allow computer processing. They should be formal languages with syntaxes and a semantics.

- A concrete syntax defines how the language elements appear in a concrete, human-usable form
- An abstract syntax characterizes in an abstract form the kinds of elements that make up the language and the rules for combining them
- The semantics defines the meaning of the language.



## Research Method: Semantics

- Define a formal and abstract semantics as a function from Feature Graphs to a mathematical structure (Semantic domain) chosen for being as close as possible to our intuitive understanding.
- Why another new semantics?
  - Generic
  - More abstract
  - Not tool dependant

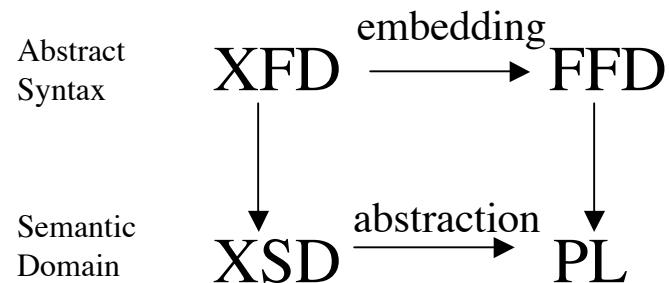


## Research Method: Comparison Criteria

- Problem: How to compare FD Languages ?
- Solutions:
  - Define formal comparison criteria
    - Expressiveness
    - Embeddability
    - Succinctness
    - Redundancy
- Results:
  - Classes of Expressiveness, Embeddability and Succinctness

## Research Method: Comparative Semantics

- Other authors have and will defined semantics for FD
- Compare our semantics with other works
  - Hypothesis: Our semantic is more abstract
  - Reason: Keep syntactic information
  - Solution: Discard syntactic information  $\Rightarrow$  abstraction functions
  - Result: Construct categories of semantic domains linked by abstraction functions





# Achievements



## Achievements

- Survey on FD
- Semantic Domain for FD
- Free Feature Diagrams Formalization
- Comparison criteria
- Comparison with Batory Semantics
- Comparison with van Deursen and Klint's semantics



## Achievements: Semantic Domain

**Definition 5 (Product and Product Line, or PL).** *We define:*

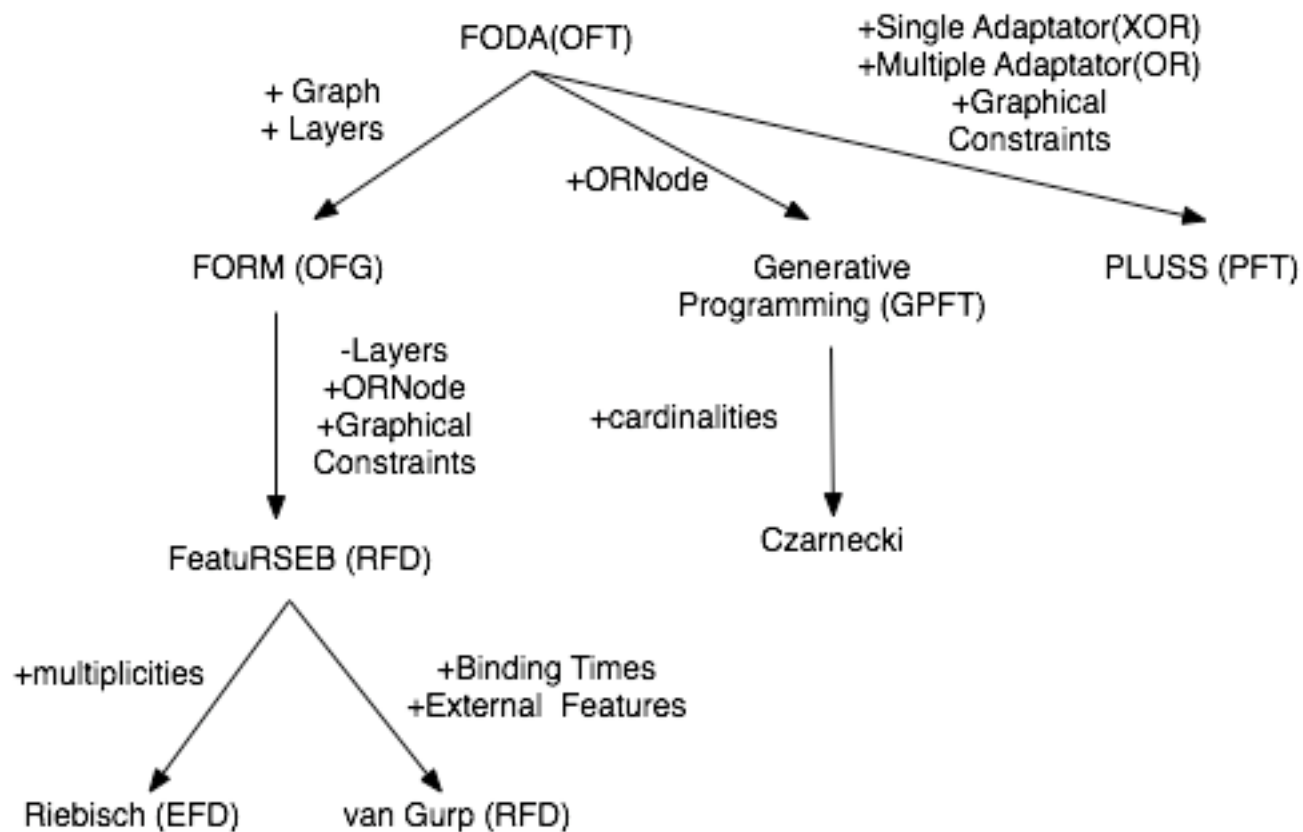
1. *A product  $c$  is a set of primitive nodes:  $c \in \mathcal{P}P$ .*
2. *The product named by a model  $m$ , noted  $\llbracket m \rrbracket$ , is  $m \cap P$ .*
3. *A product line (PL)  $pl$  is a set of products:  $pl \in \mathcal{P}P\mathcal{P}$ .*
4. *The product line of a FD  $d$  consists of the products named by its valid models:  
 $\llbracket d \rrbracket = \{m \cap P \mid m \models d\}$*

## Achievements: Free FD Formalization

- Free FD are a parametric construction that generalizes the syntax of FD languages
- Free FD represent a product line of FD for which we can explicit some variation points (parameters):
- $\text{FFD}(\text{GT}, \text{NT}, \text{GCT}, \text{TCL}) = \dots$ 
  - Graph Type (GT)
  - Node Type (NT) is a set of Boolean operators
    - And is a set of operator  $\text{and}_s$  that return true iff all their  $s$  arguments are true
    - $\text{vp}_s(i, j)$  return true iff at least  $i$  and at most  $j$  of their arguments are true
  - Graphical Constraint Type (GCT)
  - Textual Constraint Language (TCL)



## Achievements: FD Family





## Achievements: FD Family

Short Name	GT	NT	GCT	TCL
OFT	1	$and \cup xor \cup \{opt_1\}$	$\emptyset$	CR
OFD	0	$and \cup xor \cup \{opt_1\}$	$\emptyset$	CR
RFD	0	$and \cup xor \cup or \cup \{opt_1\}$	$\{\Rightarrow, \mid\}$	CR
EFD	0	$card \cup \{opt_1\}$	$\{\Rightarrow, \mid\}$	CR
GPFT	1	$and \cup xor \cup or \cup \{opt_1\}$	$\emptyset$	CR
PFT	1	$and \cup xor \cup or \cup \{opt_1\}$	$\{\Rightarrow, \mid\}$	$\emptyset$
CFD(OP)	0	$OP$	$\emptyset$	$\emptyset$
COFD	0	$and \cup xor$	$\emptyset$	$\emptyset$
VFD	0	$card$	$\emptyset$	$\emptyset$
CFT(OP)	1	$OP$	$\emptyset$	$\emptyset$
COFT	1	$and \cup xor$	$\emptyset$	$\emptyset$
CRFT	1	$and \cup xor \cup or$	$\emptyset$	$\emptyset$

## Achievements: Free FD Formalization

### ▪ Abstract Syntax

**Definition 1 (Free Feature Diagram, or FFD).** A FFD  $d \in FFD(GT, NT, GCT, TCL) = (N, P, r, \lambda, DE, CE, \Phi)$  where:

- $N$  is its set of nodes;
- $P \subseteq N$  is its set of primitive nodes;
- $r \in N$  is the root of the FD, also called the concept;
- $\lambda : N \rightarrow NT$  labels each node with an operator from  $NT$ ;
- $DE \subseteq N \times N$  is the set of decomposition edges;  $(n, n') \in DE$  will rather be noted  $n \rightarrow n'$ ;
- $CE \subseteq N \times GCT \times N$  is the set of constraint edges;
- $\Phi \subseteq TCL$  are the textual constraints.



## Achievements: Free FD Formalization

### ▪ Abstract Syntax

**Definition 2 (Feature Diagram, or FD).** A FD is a FFD where:

1. Only  $r$  has no parent:  $\forall n \in N. (\nexists n' \in N. n' \rightarrow n) \Leftrightarrow n = r.$
2. DE is acyclic:  $\nexists n_1, \dots, n_k \in N. n_1 \rightarrow \dots \rightarrow n_k \rightarrow n_1.$
3. If  $GT = 1$ , DE is a tree:  $\nexists n_1, n_2, n_3 \in N. n_1 \rightarrow n_2 \wedge n_3 \rightarrow n_2 \wedge n_1 \neq n_3.$
4. Nodes are labelled with operators of the appropriate arity:  $\forall n \in N. \lambda(n) = op_k \wedge k = \#\{(n, n') | n \rightarrow n'\}$

## Achievements: Free FD Formalization

### ▪ The Semantics

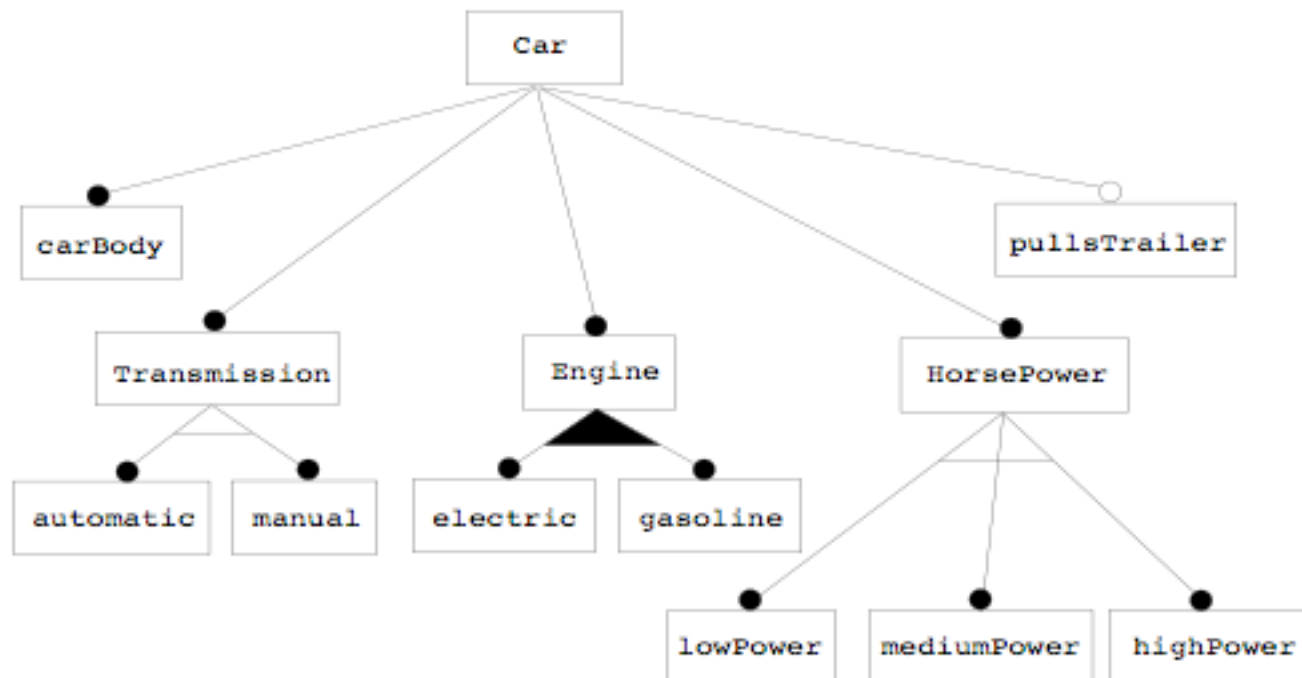
**Definition 3 (Model).** *A model of a FD is a subset of its nodes:  $M = \mathcal{P}N$ .*

**Definition 4 (Valid model).** *[13, p.70] A model  $m \in M$  is valid for a  $d \in FD$ , noted  $m \models d$  iff:*

1. *The concept is in:  $r \in m$*
2. *The meaning of nodes is satisfied: If a node  $n \in m$ , and  $n$  has sons  $s_1, \dots, s_k$  and  $\lambda(n) = op_k$ , then  $op_k(s_1 \in m, \dots, s_k \in m)$  must evaluate to true.*
3. *The model must satisfy all textual constraints:  $\forall \phi \in \Phi, m \models \phi$ , where  $m \models \phi$  means that we replace each node name  $n$  in  $\phi$  by the value of  $n \in m$ , evaluate  $\phi$  and get true.*
4. *The model must satisfy all graphical constraints:  $\forall (n_1, op_2, n_2, ) \in CE, op_2(n_1 \in m, n_2 \in m)$  must be true.*
5. *If  $s$  is in the model and  $s$  is not the root, one of its parents  $n$ , called its justification, must be too:  $\forall s \in N. s \in m \wedge s \neq r : \exists n \in N : n \in m \wedge n \rightarrow s$ .*

## van Deursen and Klint's FD Formalization

- Graphical Concrete Syntax





## van Deursen and Klint's FD Formalization

### ■ Textual Concrete Syntax

- Car: `all( carBody, Transmission, Engine, HorsePower, pullsTrailer? )`
- Transmission: `one-of( automatic, manual )`
- Engine: `more-of( electric, gasoline )`
- HorsePower: `one-of( lowPower, mediumPower, highPower )`



## van Deursen and Klint's FD Formalization

### lexical syntax

<code>[A-Z][a-zA-Z0-9]*</code>	-> FeatureName
<code>[a-z][a-zA-Z0-9]*</code>	-> AtomicFeature

### context-free syntax

<code>FeatureDefinition* Constraint*</code>	-> FeatureDiagram
<code>FeatureName ":" FeatureExpression</code>	-> FeatureDefinition
<code>{ FeatureExpression "," }+</code>	-> FeatureList
<code>all(FeatureList)</code>	-> FeatureExpression
<code>one-of(FeatureList)</code>	-> FeatureExpression
<code>more-of(FeatureList)</code>	-> FeatureExpression
<code>FeatureName</code>	-> FeatureExpression
<code>AtomicFeature</code>	-> FeatureExpression
<code>FeatureExpression "?"</code>	-> FeatureExpression
<code>"default" "=" AtomicFeature</code>	-> FeatureExpression
<code>"..."</code>	-> AtomicFeature
<code>DiagramConstraint</code>	-> Constraint
<code>UserConstraint</code>	-> Constraint
<code>AtomicFeature "requires" AtomicFeature</code>	-> DiagramConstraint
<code>AtomicFeature "excludes" AtomicFeature</code>	-> DiagramConstraint
<code>"include" AtomicFeature</code>	-> UserConstraint
<code>"exclude" AtomicFeature</code>	-> UserConstraint



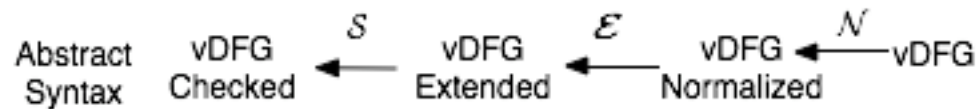
## van Deursen and Klint's FD Formalization

- Semantics
  - Provide a Feature Diagram Language (vDFG);
  - Provide a Feature Diagram Algebra which is a suite of formally defined operations to manipulate vDFG expressions.
    - Normalization rules(N): to simplify the feature expression by eliminating duplicate features and degenerate cases of the various constructors.
    - Variability rules: to count the number of possibilities for a given feature diagram.
    - Expansion rules (E): to expand a normalized feature expression into a *disjunctive normal form*.
    - Satisfaction rules (S): given a feature expression in disjunctive normal form and given constraints, they determine which of the disjuncts satisfy the constraints.



## van Deursen and Klint's FD Formalization

- A serie of transformations is defined



- to produce a disjunctive normal form listing all valid members of the product line

```

one-of(all(carBody, automatic, electric, lowPower, pullsTrailer),
      all(carBody, automatic, electric, gasoline, lowPower,
          pullsTrailer),
      all(carBody, automatic, gasoline, lowPower,
          pullsTrailer),
      all(carBody, automatic, electric, mediumPower,
          pullsTrailer),
      all(carBody, automatic, electric, gasoline, mediumPower,
          pullsTrailer),
      all(carBody, automatic, gasoline, mediumPower,
          pullsTrailer),
      ...
)
  
```



## Semantic Domain

### Semantic Domain

- Van Deursen and Klint :
  - The disjunctive normal form corresponds to an ordered list of ordered lists of primitive features  $O(O(P))$  .

**Definition 9 (Disjunctive normal form).** *A disjunctive normal form [23, p.9] is a one-of feature expression with only alls feature expressions as arguments with only atomic features as arguments. A disjunctive normal form is an expression of the form:*  
 $\text{one-of}(\text{all}(A_{11}, \dots, A_{1n_1}), \dots, \text{all}(A_{m1}, \dots, A_{mn_m}))$

- FFD:

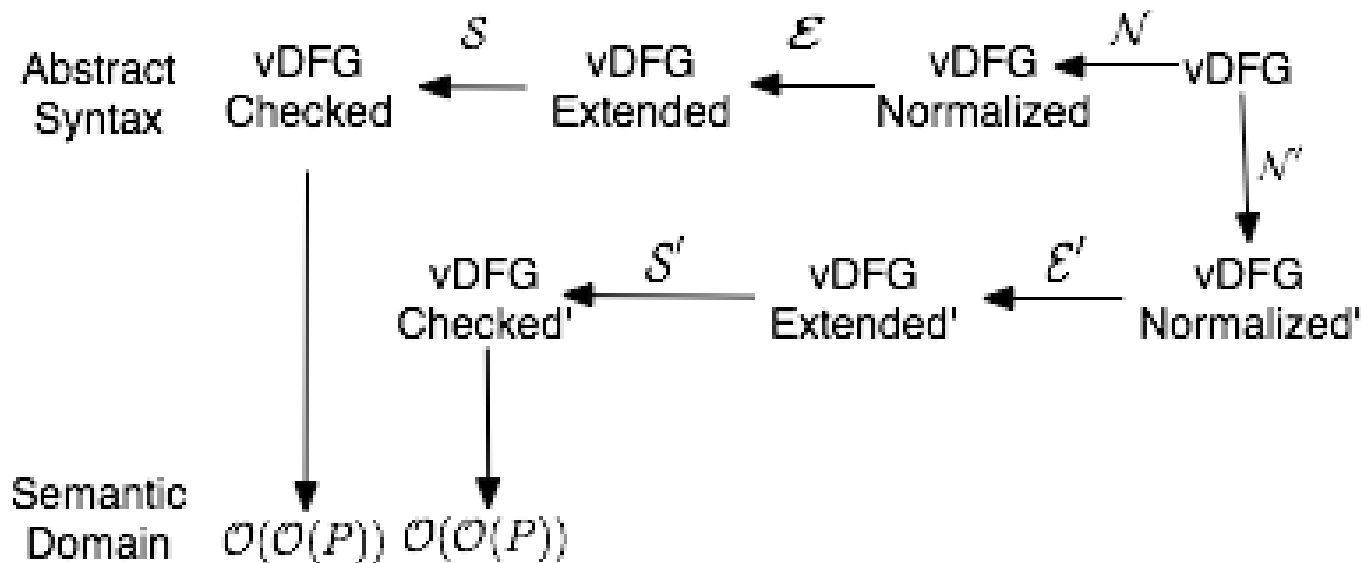
*A product line (PL)  $pl$  is a set of products:  $pl \in \mathcal{PPP}$ .*



## Comparative Semantics

- Three main differences
  - Small mistakes (missing rules)
  - Semantic domain
    - Ordered list of ordered lists of atomic features
      - $All(A,B) \langle \rangle All(B,A)$
    - Set of sets of primitives features
  - vDFG gives preference to inclusion in terms
    - Edge-based Semantics
    - Node-based Semantics
- Can we resolve these differences?
  - Revisit van Deursen Semantics
  - Relate the semantic domain by an abstraction function (R)
  - Apply a preliminary transformation (T) on FFD to switch from node-based semantics to an edge-based semantics
  - Semantic Equivalence

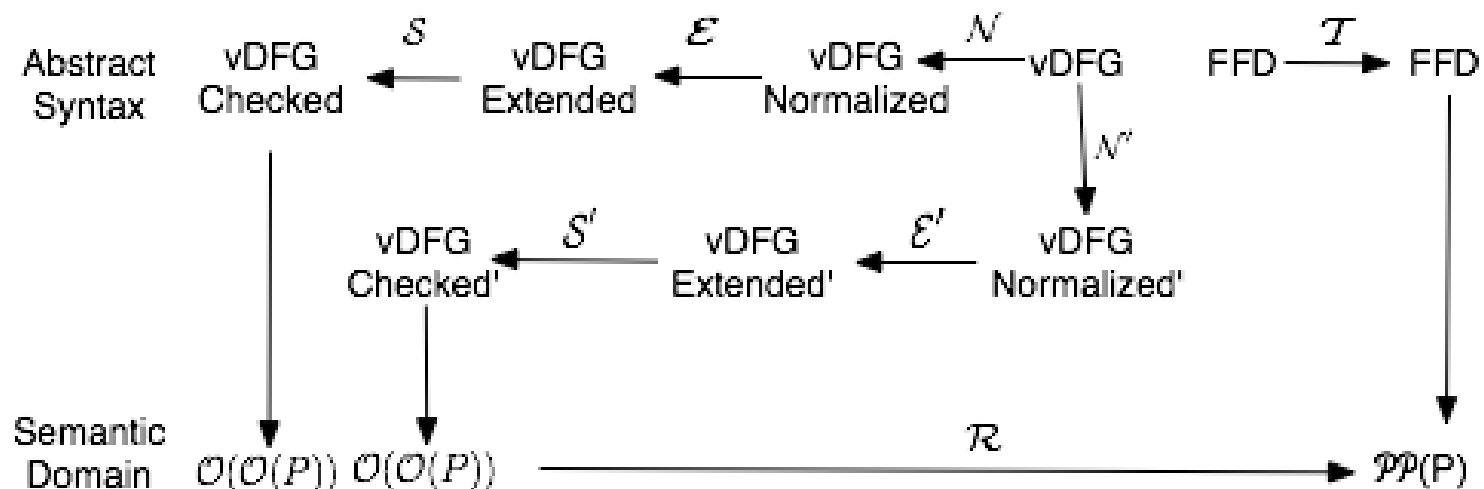
## Revisited Semantics



**Definition 13 (vDFG Semantics).** *The semantic of a vDFG (vdfg) is a function  $\mathcal{L} : vDFG \rightarrow \mathcal{O}(\mathcal{O}(P))$  where  $\mathcal{L}(vdfg) = \mathcal{S}(\mathcal{E}(\mathcal{N}(vdfg)))$ .*

**Definition 17 (Revisited vDFG Semantics).** *The revisited semantics function of van Deursen and Klint is defined as:  $\mathcal{L}' : vDFG \rightarrow \mathcal{O}(\mathcal{O}(P))$  where  $\mathcal{L}'(vdfg) = \mathcal{S}'(\mathcal{E}'(\mathcal{N}'(vdfg)))$*

## Comparative Semantics



**Definition 18 (Semantic Domain Abstraction  $\mathcal{R}$ ).**

$$\mathcal{R} : \mathcal{O}(\mathcal{O}(P)) \rightarrow \mathcal{P}(\mathcal{P}(P))$$

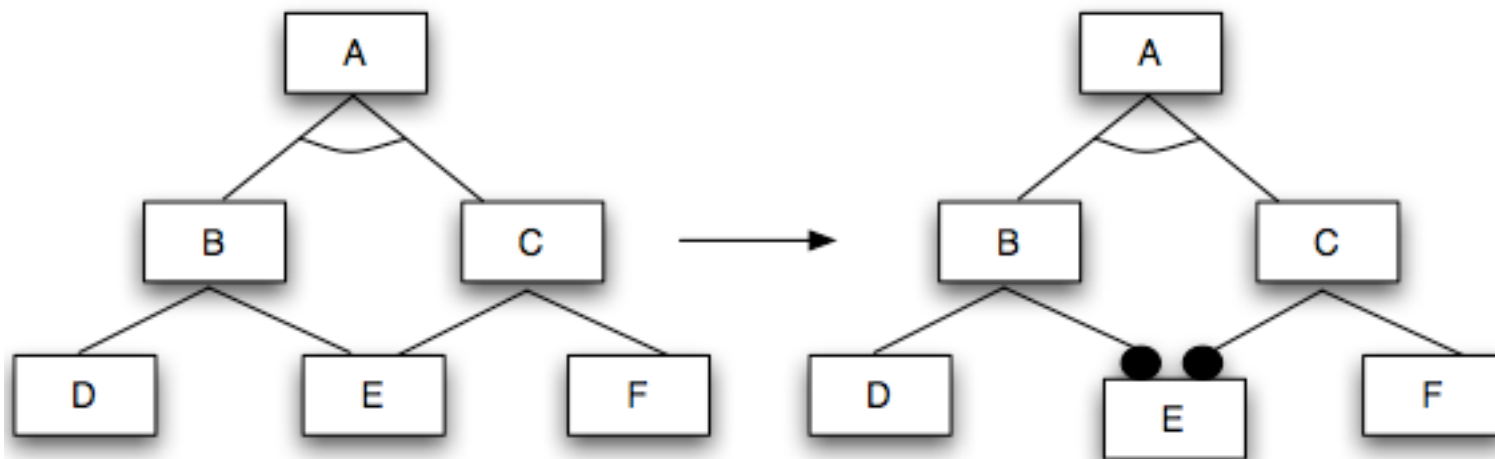
$$\mathcal{R}(\text{all}(f_n \dots f_m), \dots, \text{all}(f_{n'}, \dots, f_{m'})) = \{\{f_n, \dots, f_m\}, \dots, \{f_{n'}, \dots, f_{m'}\}\}$$

**Theorem 1.**

$$\forall t : t \in vDFG : \llbracket T(t) \rrbracket = \mathcal{R}(\mathcal{L}'(t))$$

## Comparative Semantics

- From Node-Based to Edge-Based Semantics (T)



## Comparison Criteria

### ■ Expressiveness

- Definition:

**Definition 14 (Expressiveness)** *The expressiveness of a language  $\mathcal{L}$  is the set  $E(\mathcal{L}) = \{\llbracket D \rrbracket \mid D \in \mathcal{L}\}$ , also noted  $\llbracket \mathcal{L} \rrbracket$ . 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  $\llbracket \cdot \rrbracket : \mathcal{L} \rightarrow \mathbb{M}$ ) is expressively complete if  $E(\mathcal{L}) = \mathbb{M}$ .*

- vDFG expressions  $\Rightarrow$  at least one operand  
 $\Rightarrow$  empty product line and base product line can not be expressed
- vDFG expressions + Textual Constraints  
 $\Rightarrow$  Expressively complete  
empty product line : one-of(all(A)) with exclude A  
base product line: one-of(all(A?)) with exclude A

## Comparison Criteria

### ■ Embeddability

#### • Definition

**Definition 19 (Graphical embeddability)** *A graphical language  $\mathcal{L}_1$  is embeddable into  $\mathcal{L}_2$  iff there is a translation  $T : \mathcal{L}_1 \rightarrow \mathcal{L}_2$  that is node-controlled [17] :  $T$  is expressed as a set of rules of the form  $D_1 \rightarrow D_2$ , where  $D_1$  is a diagram containing a defined node or edge  $n$ , and all possible connections with this node or edge. 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.*

#### • Result

- vDFG is embeddable into FFD



## Comparison Criteria

- Succinctness
  - Definition

**Definition 21 (Succinctness)** *Let  $\mathcal{G}$  be a set of functions from  $\mathbb{N} \rightarrow \mathbb{N}$ . 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 translation  $T : \mathcal{L}_1 \rightarrow \mathcal{L}_2$  that is within  $\mathcal{G}$ :  $\exists g \in \mathcal{G}, \forall n \in \mathbb{N}, \forall l_1 \in \mathcal{L}_1, |l_1| \leq n \Rightarrow |T(l_1)| \leq g(n)$ . 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”.*

- Feature sharing is not allowed at intermediate nodes

⇒ Exponential blow up





## Conclusions & Future Works

- Conclusions
  - Propose a generic formalization of syntax and semantics of FD
  - Present our method based on Comparative semantics
  - Apply this method to study van Deursen & Klint FD
  - Foundations necessary to avoid ambiguities and to build safe and efficient tools.
- Future Works
  - Study and compare other FD semantics
    - Batory, Czarnecki, Wang
  - Complete semantics categories
  - Tools - Decision Problems :
    - Satisfiability
    - Product Checking
    - FD Union Intersection
    - Interoperability