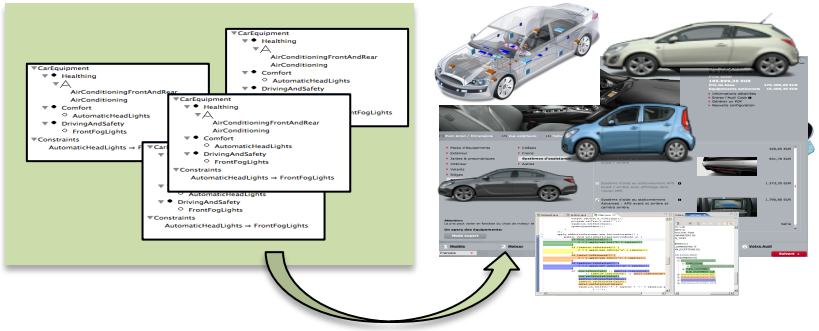


# Model-based Software Product Lines

## Variability Models (Part 3)

Mathieu Acher

Maître de Conférences

mathieu.acher@irisa.fr

UNIVERSITÉ DE RENNES 1

istic Informatique Électronique

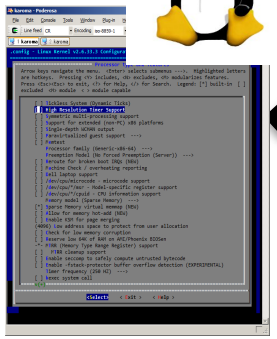
# Material

**http://mathieuacher.com/teaching/M2R/**

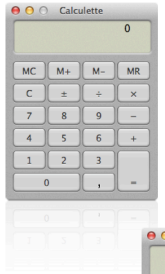
# Agenda

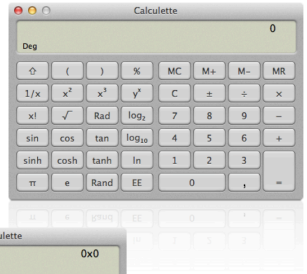
- Previously
- Feature Models
  - Syntax and Semantics (advanced)
  - Feature diagrams and formula
- Compositions
  - Motivation
  - Feature Model Synthesis
  - Techniques
- Two open problems

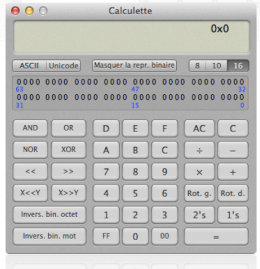
# Previously



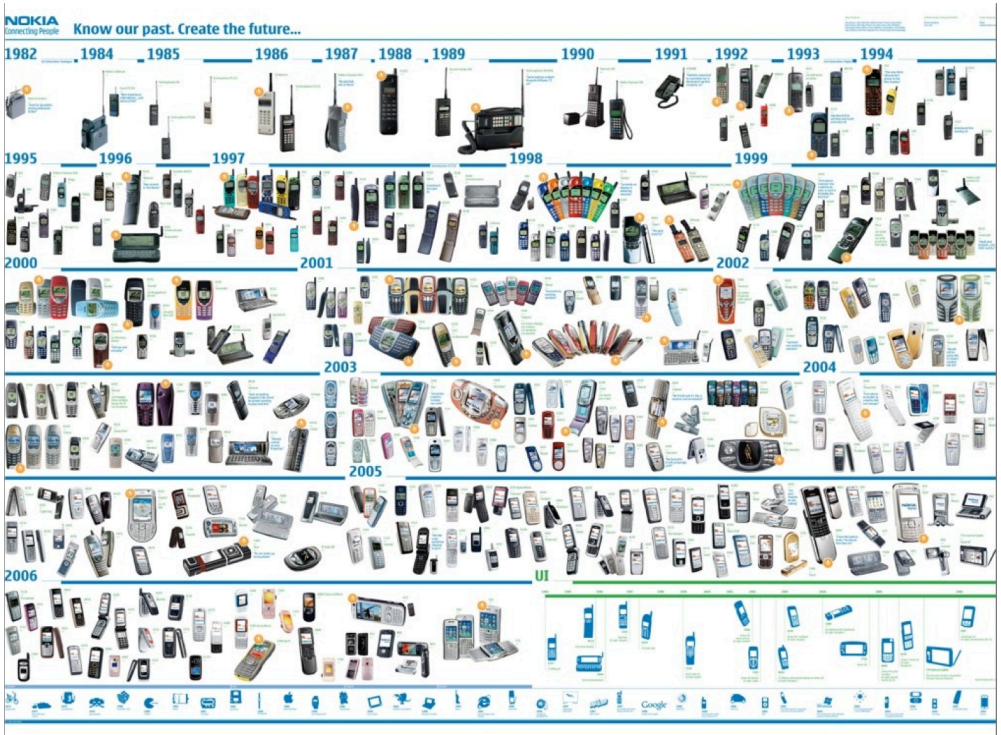
Linux Kernel



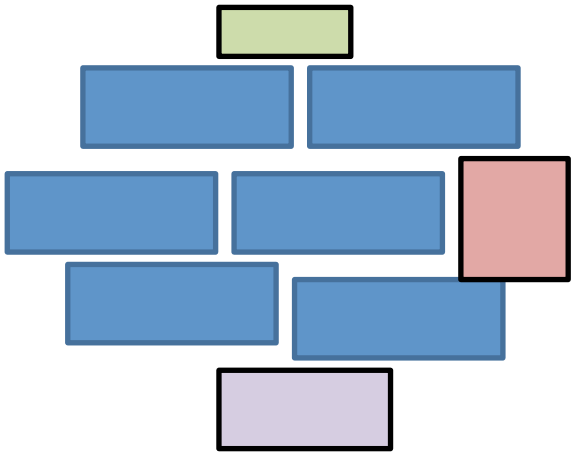

2000 features



# Software Product Line Engineering

Factoring out **commonalities**

for **Reuse** [Krueger et al., 1992] [Jacobson et al., 1997]



Managing **variabilities**

for Software **Mass Customization** [Bass et al., 1998] [Krueger et al., 2001], [Pohl et al., 2005]

# Variability Management

## Common features

print – connect with computer…

## Variable features

fax – scan – USB port…

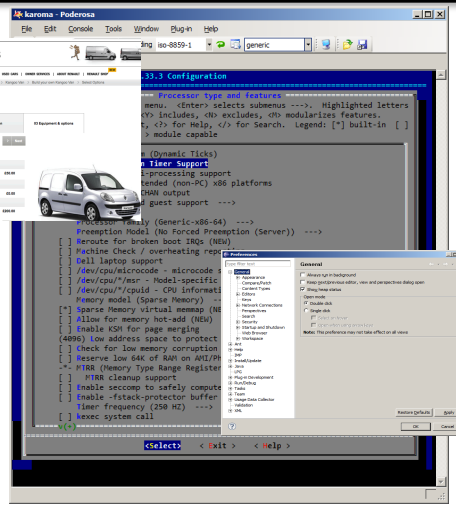
## Product-specific features

serial port

# Variability

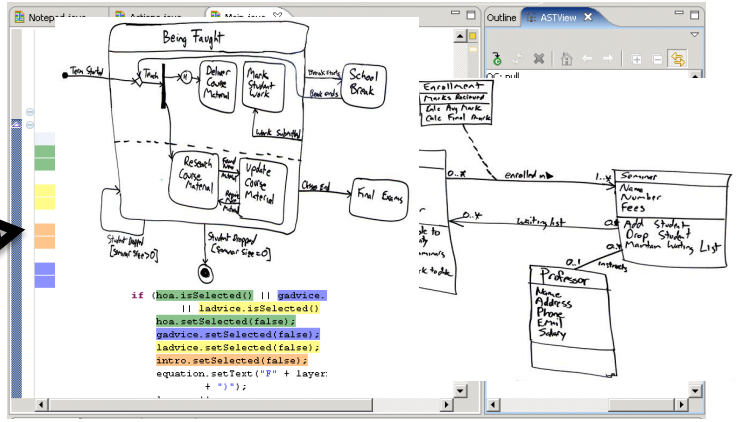
**"the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context"**

*Mikael Svahnberg, Jilles van Gurp, and Jan Bosch (2005)*

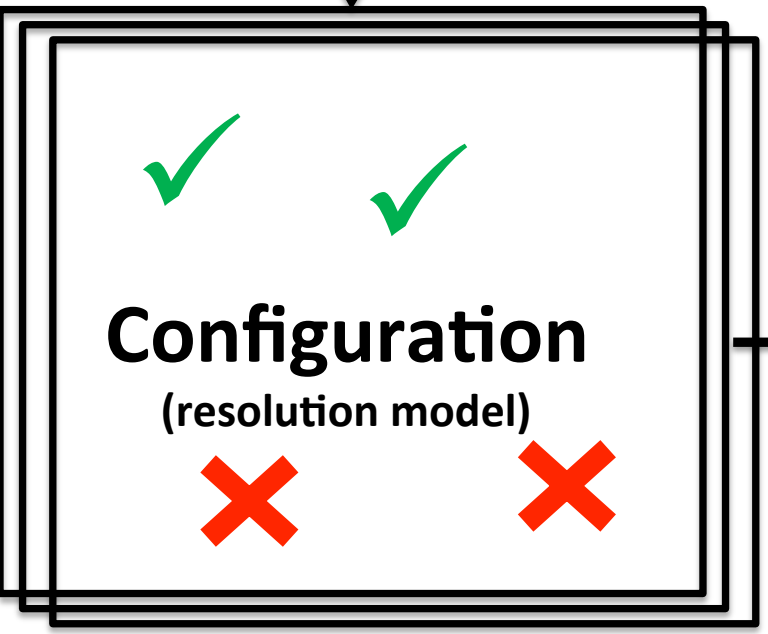


Variability Abstraction Model (VAM)

Variability Realization Model (VRM)



Domain Artefacts (e.g., models)



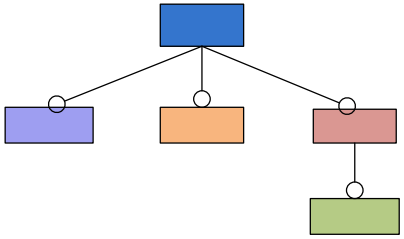
Configuration

(resolution model)

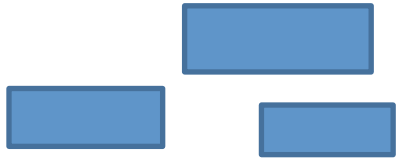
Software Generator

(derivation engine)

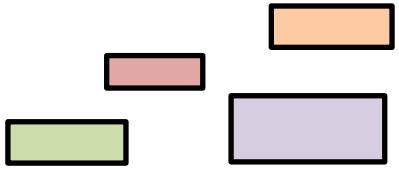
# Domain engineering (development for reuse)



Variability Model

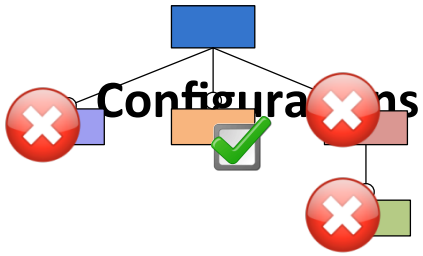


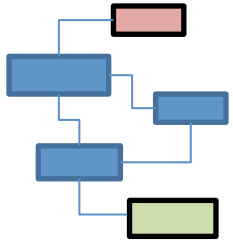
*Common assets*



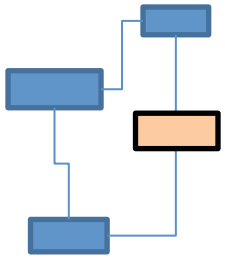
*Variants*

**Reusable Assets**
(e.g., models or source code)

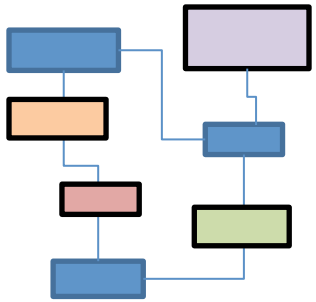


*product*$_1$



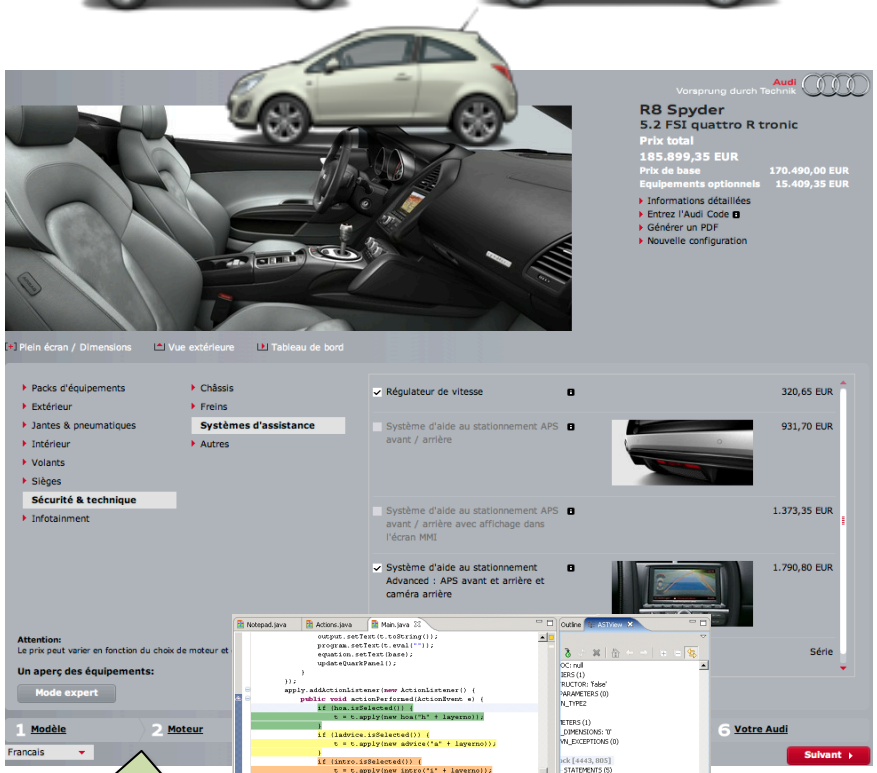
*product*$_2$

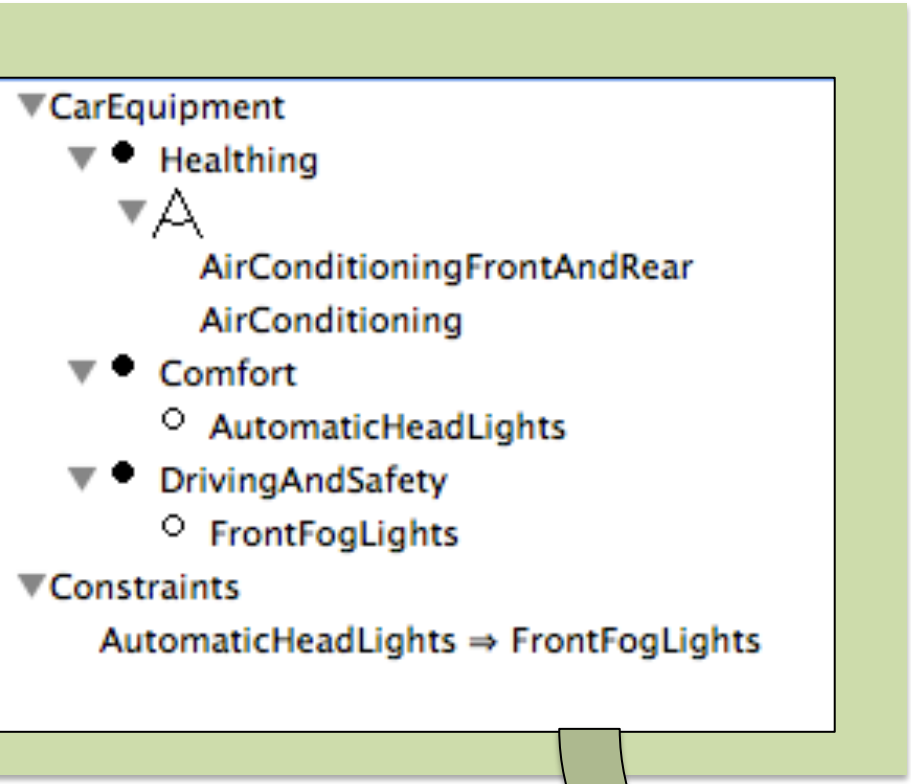


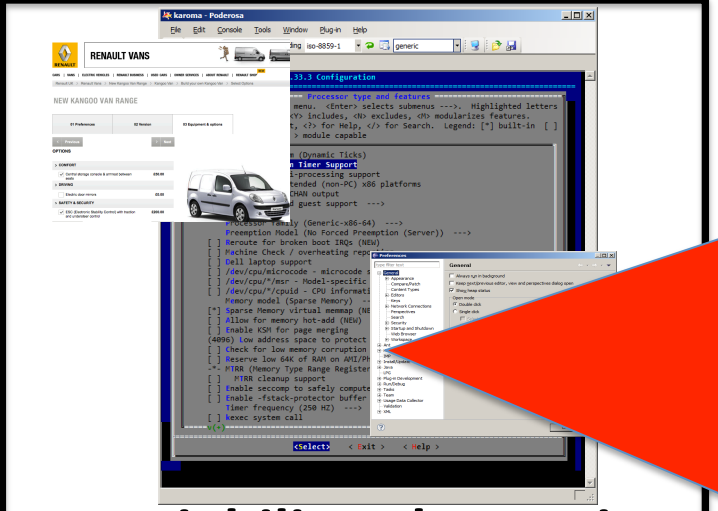
*product*$_n$

# Application engineering (development with reuse)

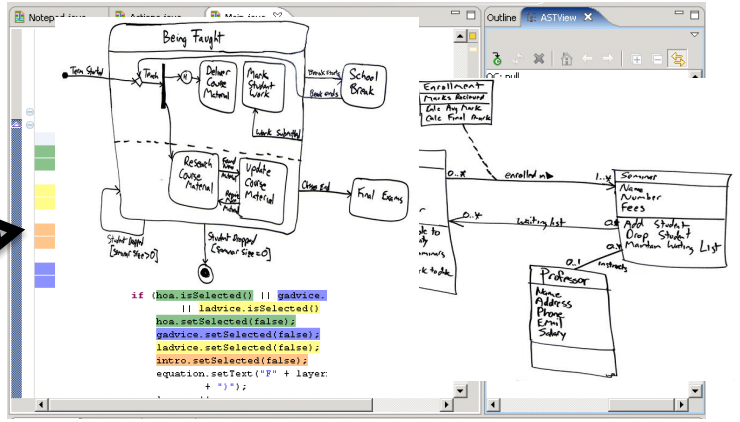
# Variability Models



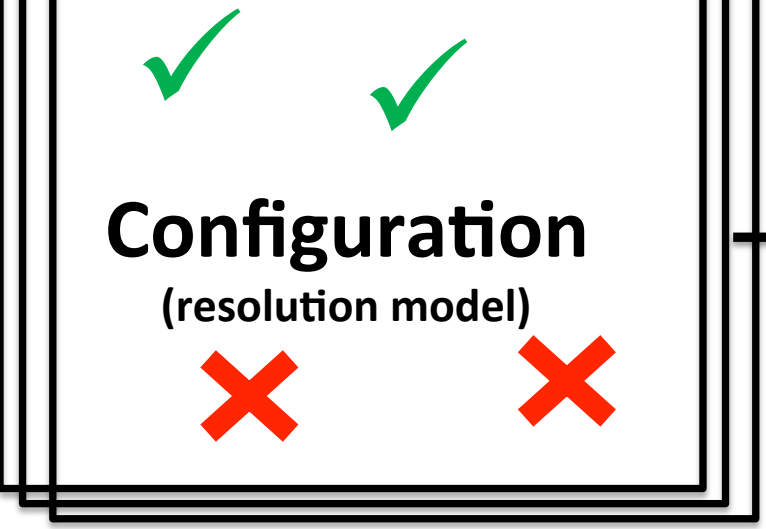






Variability Abstraction Model (VAM)

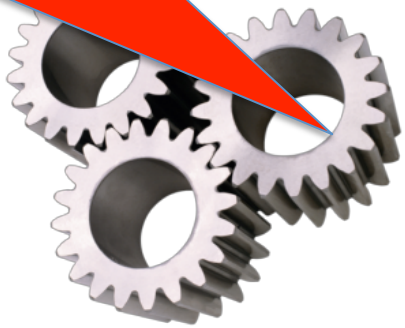



Domain Artefacts (e.g., models)



Configuration

(resolution model)



Software Generator

(derivation engine)

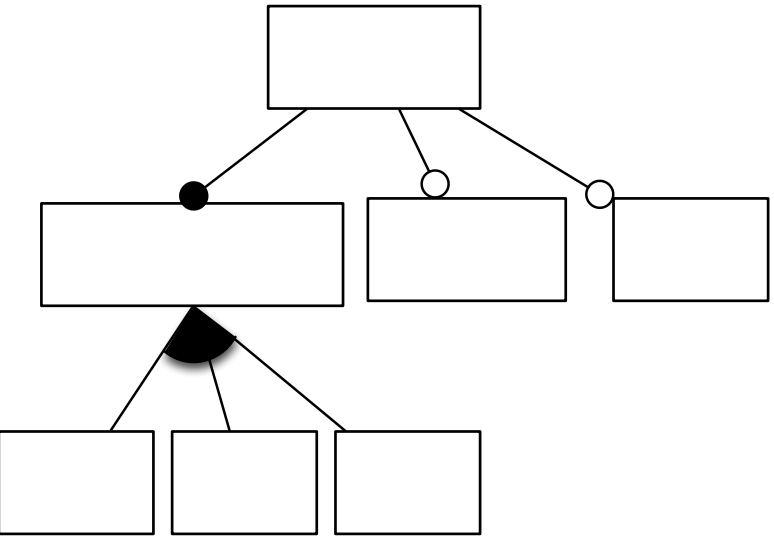

Illegal variant


Unused flexibility

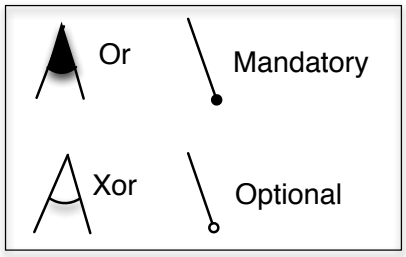
# Feature Models

Syntacs, Semantics
Feature diagram, Formula

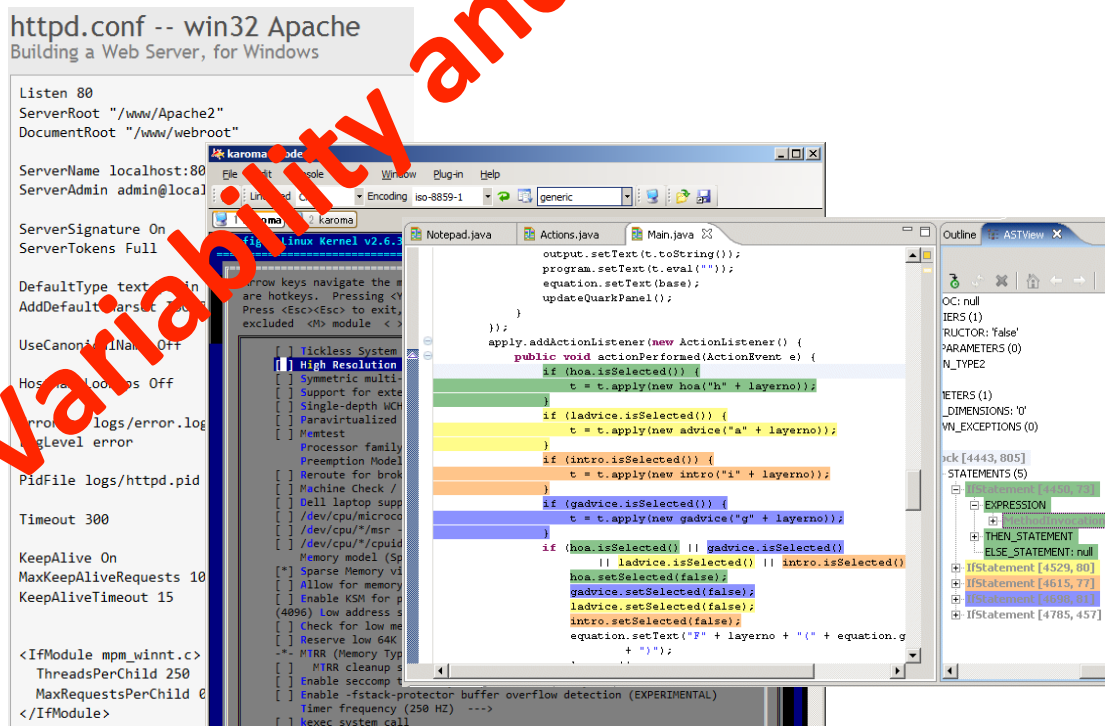
# Feature Models



not, and, or, implies

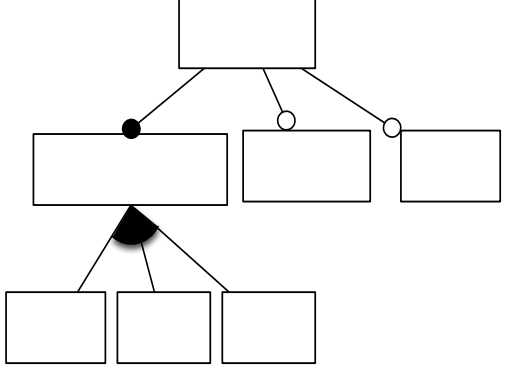






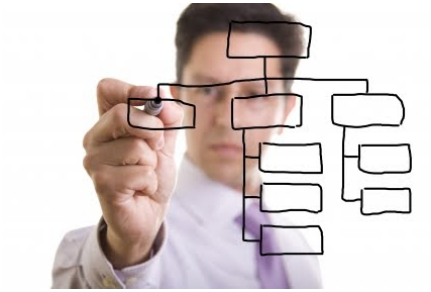

Variability and Configuration

# Feature Model

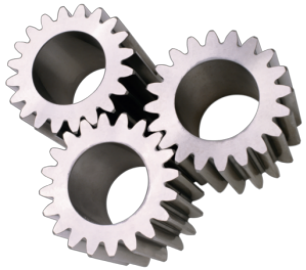


not, and, or, implies

**Communicative**

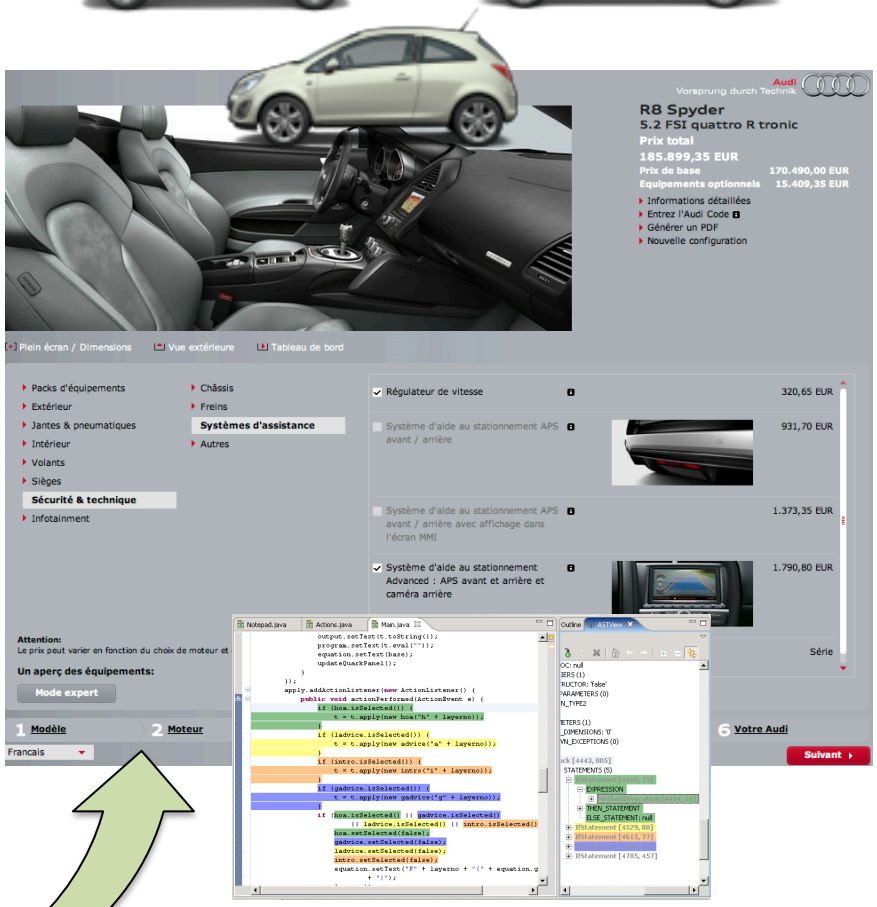


**Analytic**

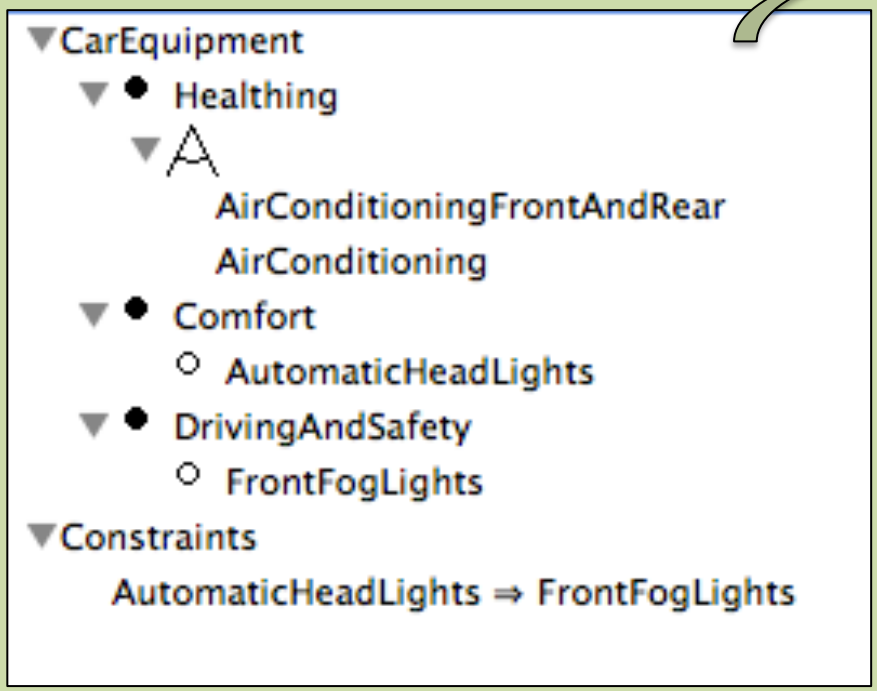


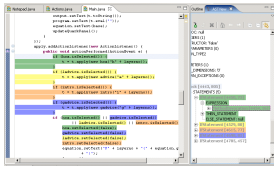
**Generative**

# Feature Models



# Feature Models (Background)

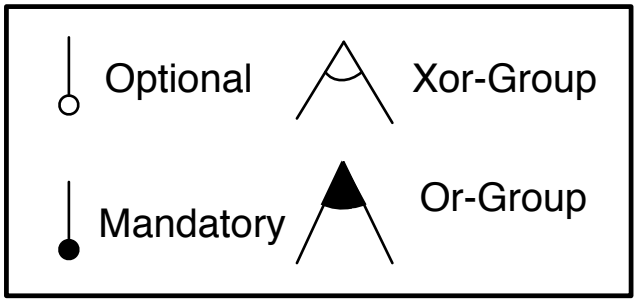


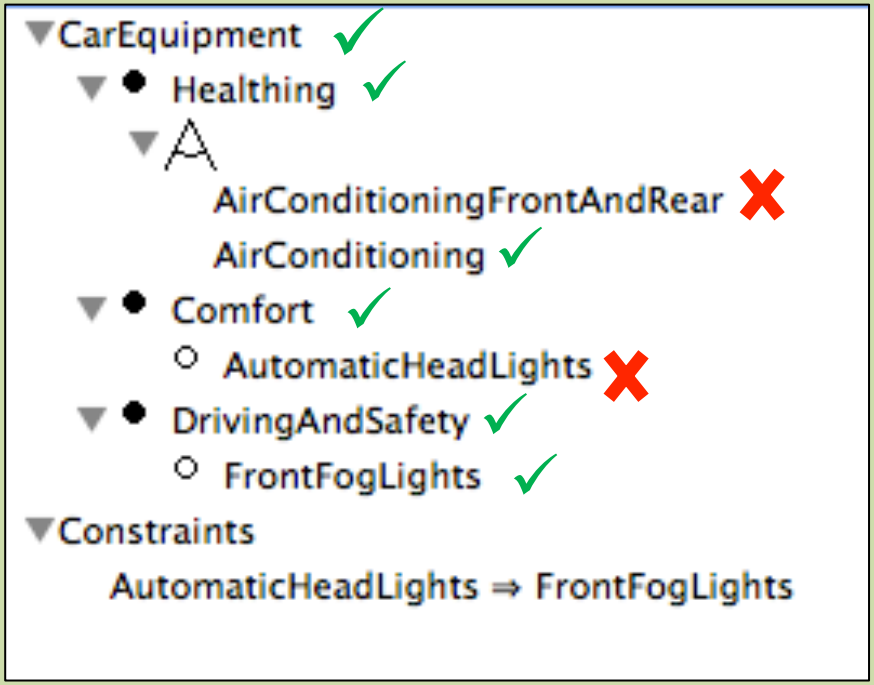



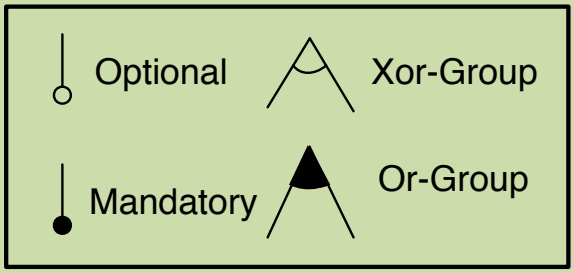
**Hierarchy:** rooted tree

**Variability:**

- mandatory,
- optional,
- Groups: exclusive or inclusive features
- Cross-tree constraints



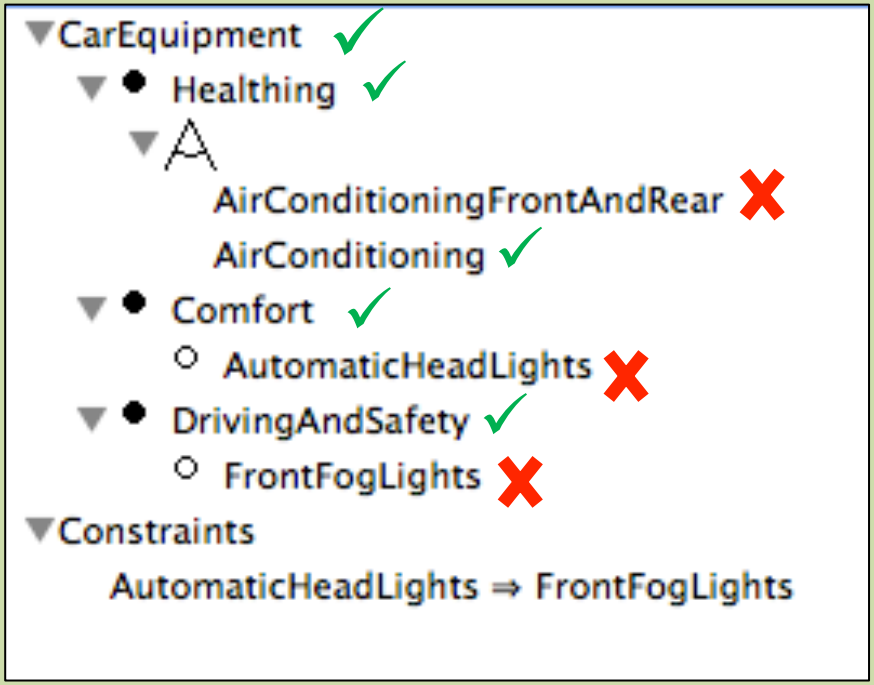


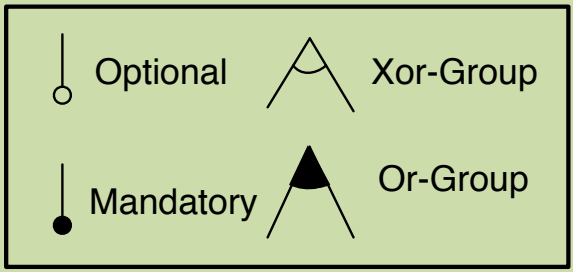




# Hierarchy + Variability
# =
# set of valid configurations

**configuration = set of features selected**

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning, FrontFogLights}

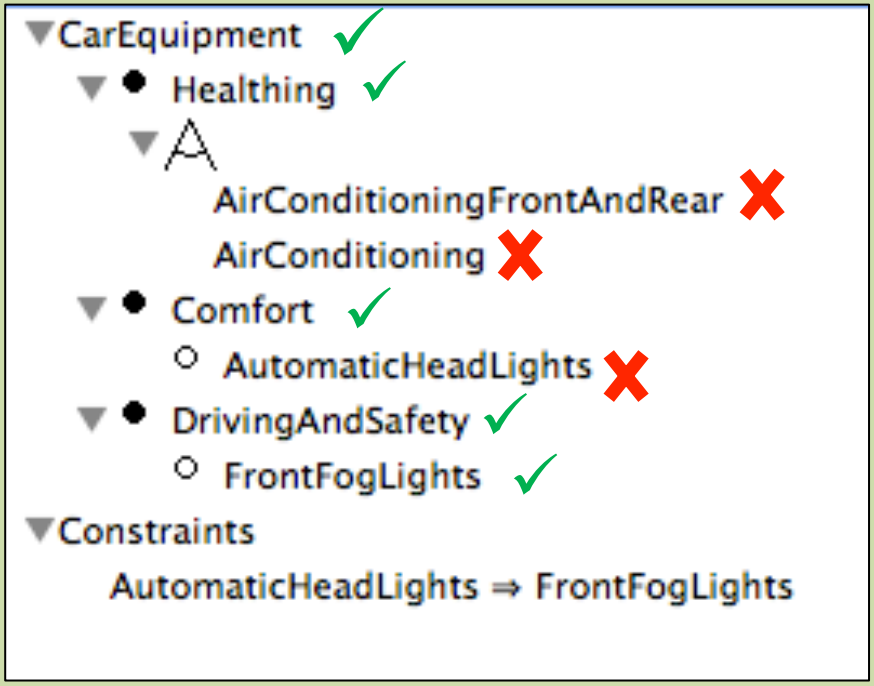


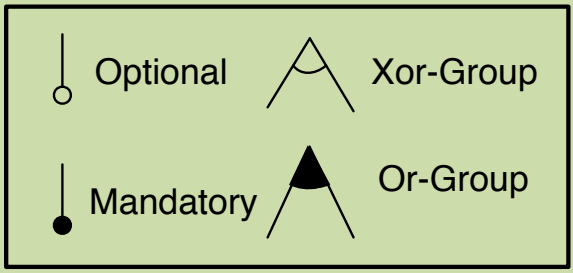




# Hierarchy + Variability
# =
# set of valid configurations

**configuration = set of features selected**

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning}

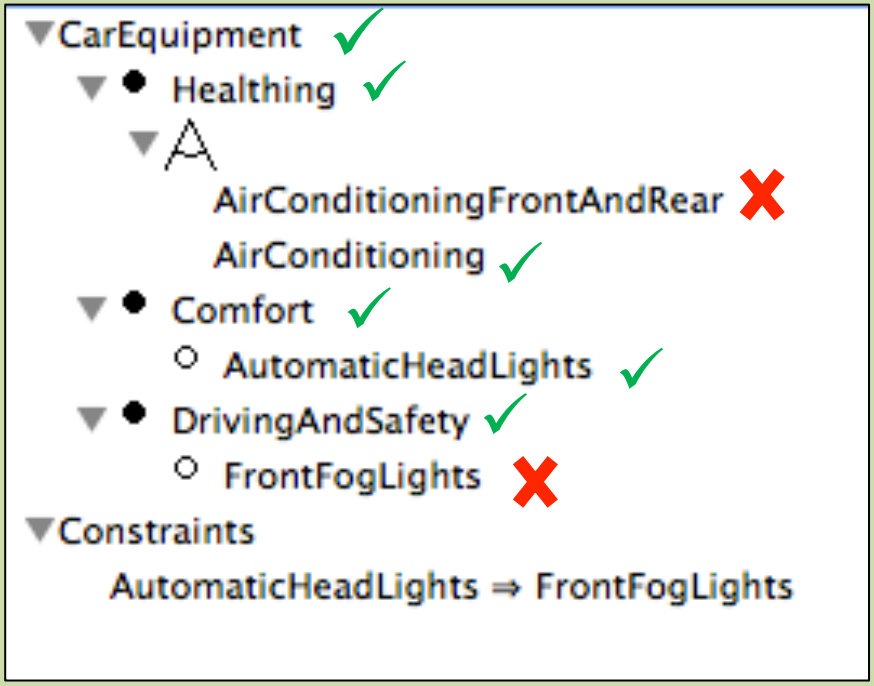


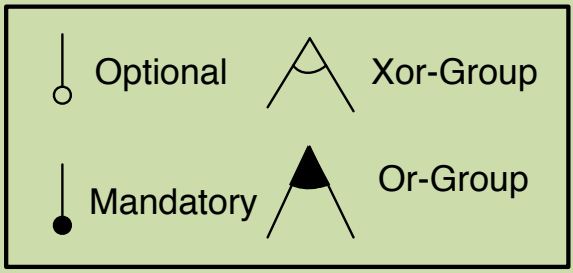




# Hierarchy + Variability
# =
# set of valid configurations

**configuration = set of features selected**

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning, AirConditioningFrontAndRear, FrontFogLights}

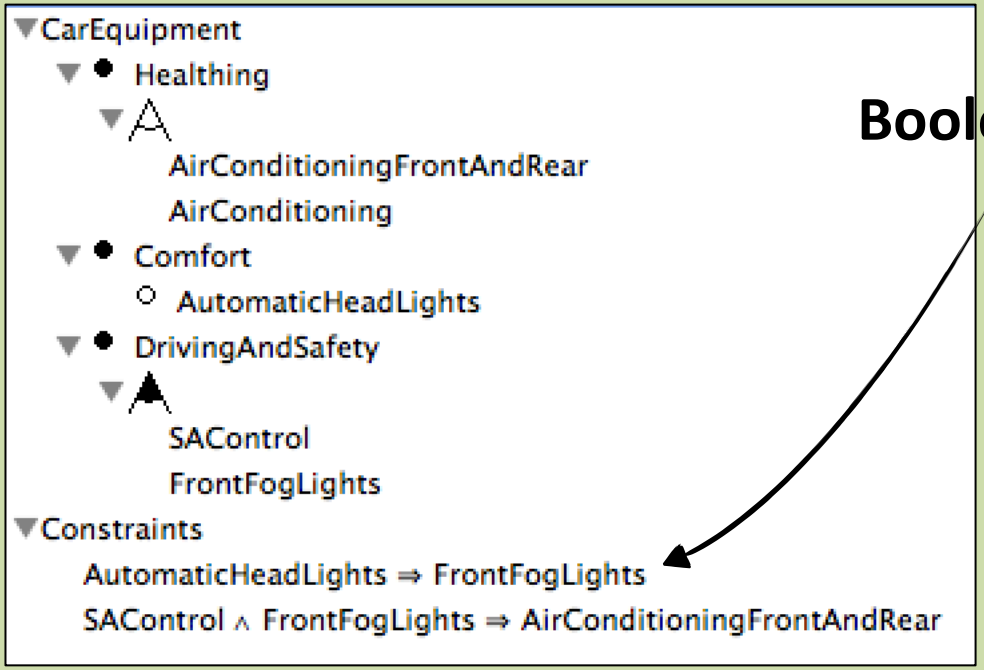


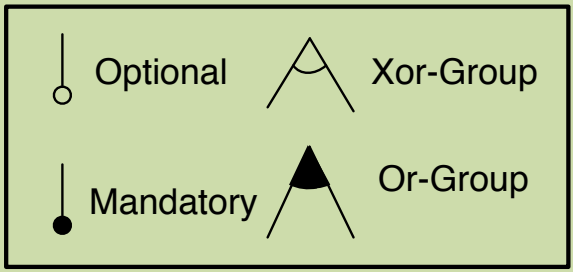

# Hierarchy + Variability
# =
# set of valid configurations

**configuration = set of features selected**

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning, AutomaticHeadLights}



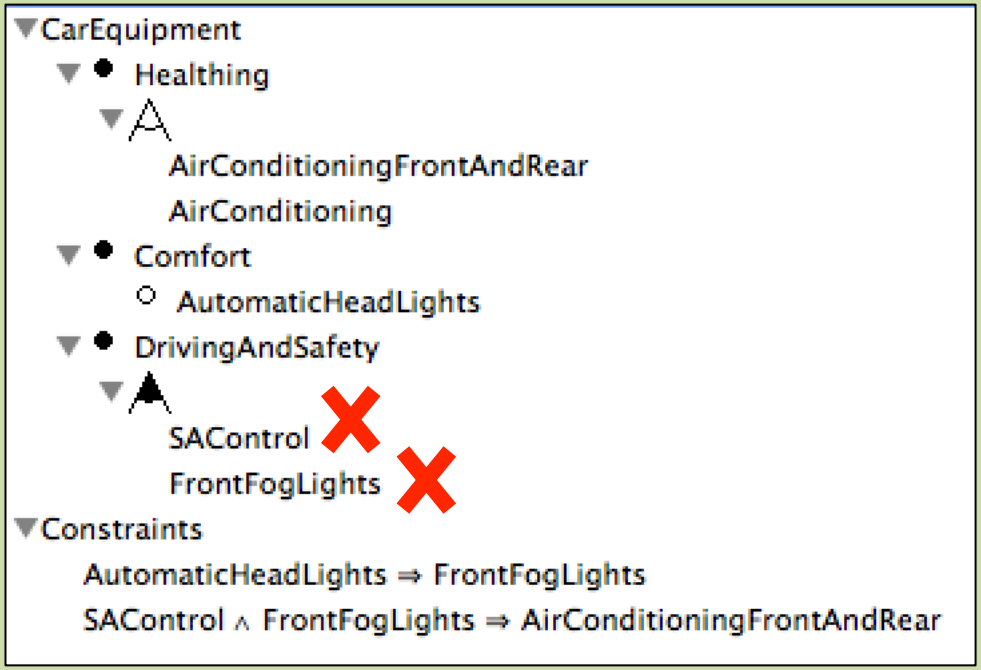

**Boolean logic: ^, v, not, implies**

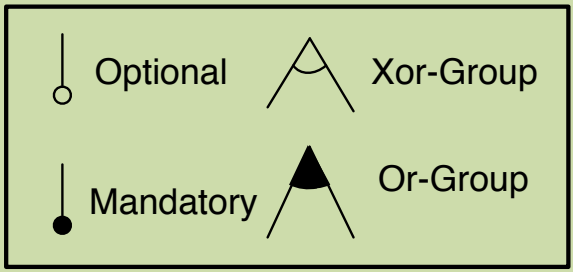



**Hierarchy + Variability**

**=**

**set of valid configurations**



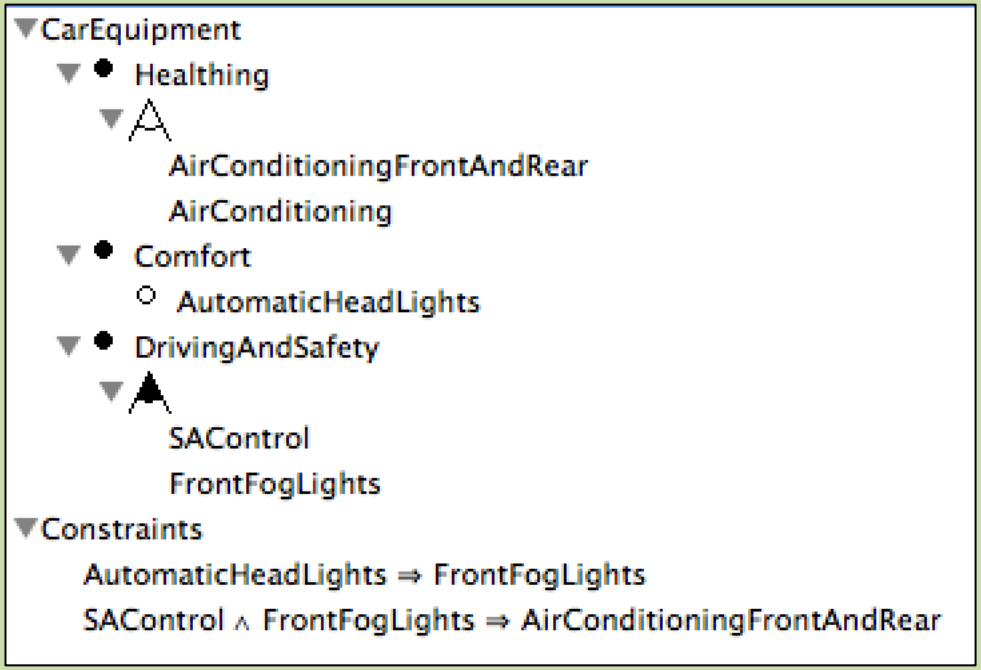



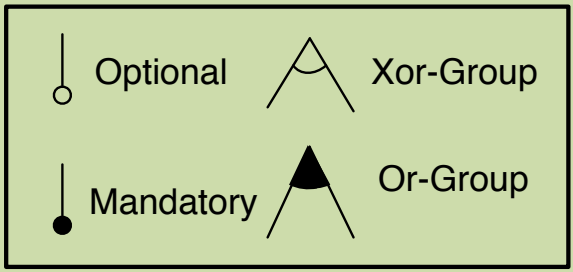

**Hierarchy + Variability**

**=**

**set of valid configurations**

*Or-group: at least one!*




# Hierarchy + Variability
# =
# set of valid configurations

{CarEquipment, Comfort, DrivingAndSafety, Healthing}

X

{AirConditioningFrontAndRear, FrontFogLights, SAControl}
{AirConditioningFrontAndRear, SAControl}
{AutomaticHeadLights, AirConditioning, FrontFogLights}
{AirConditioningFrontAndRear, SAControl, AutomaticHeadLights, FrontFogLights}
{FrontFogLights, AirConditioning}
{AutomaticHeadLights, AirConditioningFrontAndRear, FrontFogLights}
{FrontFogLights, AirConditioningFrontAndRear}
{SAControl, AirConditioning}

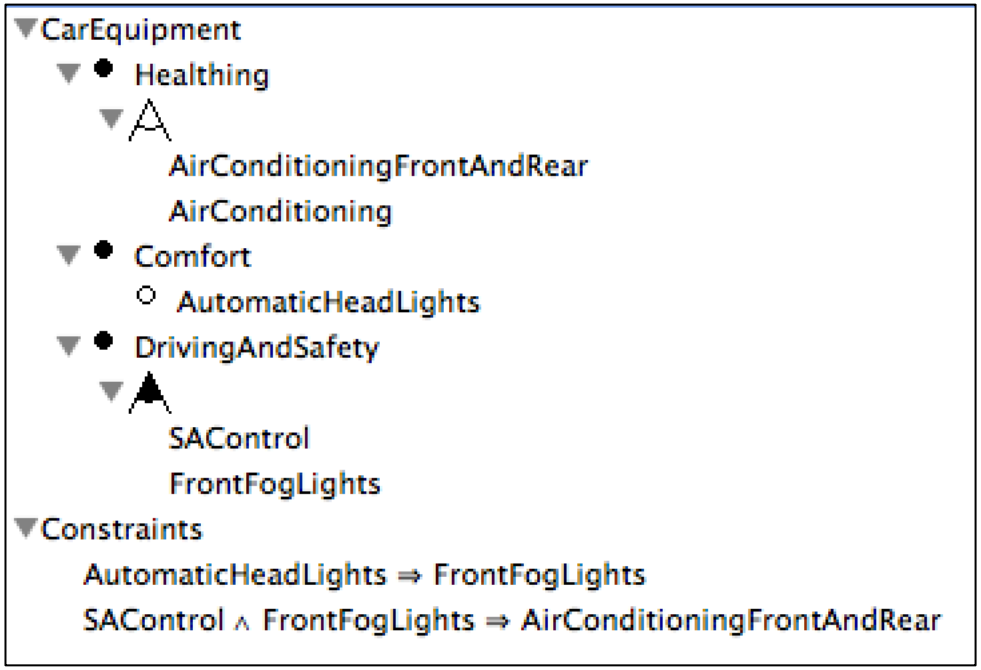
# **Formal semantics** of a language

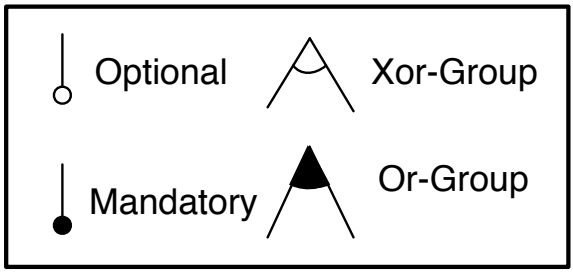
– **formal syntax** (L) – clearcut syntactic rules defining all legal diagrams, a.k.a. syntactic domain

– **semantic domain** (S) – a mathematical abstraction of the real-world concepts to be modelled

– **semantic function** (M: L ⟶ S) – clearcut semantic rules defining the meaning of all legal diagrams

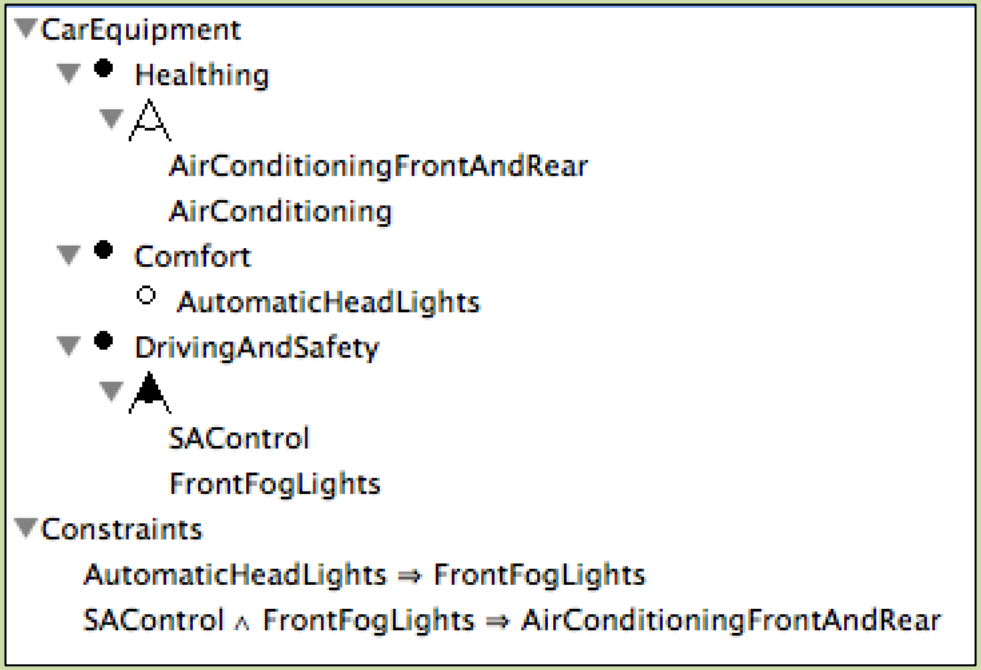
[Harel & Rumpe, IEEE Computer, 2004]

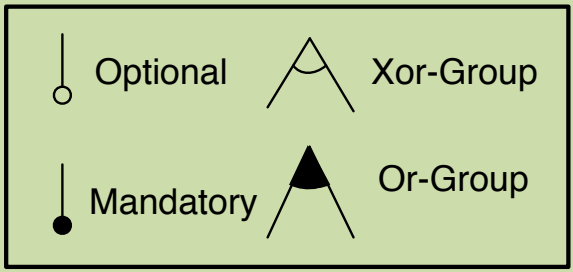



**Definition 2 (Feature Diagram)** *A feature diagram* $FD = \langle G, E_{MAND}, G_{XOR}, G_{OR}, I, EX \rangle$ *is defined as follows:* $G = (\mathcal{F}, E, r)$ *is a rooted, labeled tree where* $\mathcal{F}$ *is a finite set of features,* $E \subseteq \mathcal{F} \times \mathcal{F}$ *is a finite set of edges and* $r \in \mathcal{F}$ *is the root feature ;* $E_{MAND} \subseteq E$ *is a set of edges that define mandatory features with their parents ;* $G_{XOR} \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ *and* $G_{OR} \subseteq \mathcal{P}(\mathcal{F}) \times \mathcal{F}$ *define feature groups and are sets of pairs of child features together with their common parent feature ;* $I$ *a set of implies constraints whose form is* $A \Rightarrow B$, $EX$ *is a set of excludes constraints whose form is* $A \Rightarrow \neg B$ $(A \in \mathcal{F}$ *and* $B \in \mathcal{F})$.

**Definition 3 (Feature Model)** *An FM is a tuple* $\langle FD, \psi \rangle$ *where* $FD$ *is a feature diagram and* $\psi$ *is a propositional formula over the set of features* $\mathcal{F}$.




**Definition 1 (Configuration Semantics)** *A configuration of an FM fr*
*defined as a set of selected features.* $[\![fm_1]\!]$ *denotes the set of valid configura*
*of* $fm_1$ *and is a set of sets of features.*

{CarEquipment, Comfort, DrivingAndSafety, Healthing}

X

{AirConditioningFrontAndRear, FrontFogLights, SAControl}
{AirConditioningFrontAndRear, SAControl}
{AutomaticHeadLights, AirConditioning, FrontFogLights}
{AirConditioningFrontAndRear, SAControl, AutomaticHeadLights, FrontFogLights
{FrontFogLights, AirConditioning}
{AutomaticHeadLights, AirConditioningFrontAndRear, FrontFogLights}
{FrontFogLights, AirConditioningFrontAndRear}
{SAControl, AirConditioning}

# Exercice #1: Feature Model

- #1.1 Give two feature models with the same configuration semantics but with different syntax

- #1.2 Does it matter ?

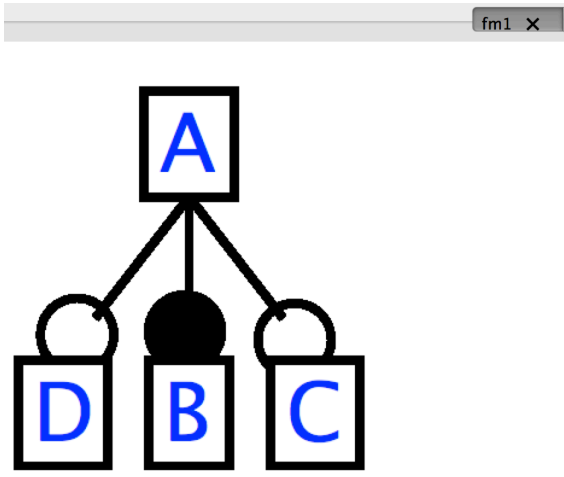
A ^
A ⇔ B ^
C => A ^
D => A

## Feature Model Synthesis Problem

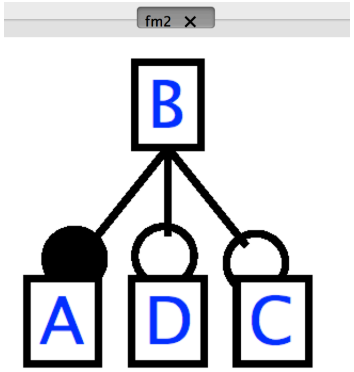
[Czarnecki et al., SPLC'07]
[She et al., ICSE'11]
[Andersen et al., SPLC'12]

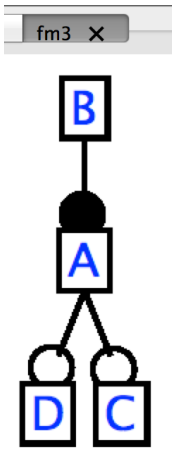
φ

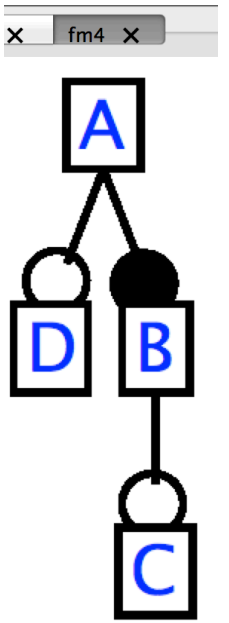
FAMiliAR




FM




```
fm2 = FM (B : A [C] [D] ; )
fm3 = FM (B : A ; A : [C] [D] ; )
fm4 = FM (A : B [D] ; B : [C] ; )
fm5 = FM (A : B [C] ; B : [D] ; )

b12 = compare fm1 fm2
b13 = compare fm1 fm3
b14 = compare fm1 fm4
b15 = compare fm1 fm5
assert (b12 eq REFACTORING)
```

# #1 Reverse Engineering Scenarios

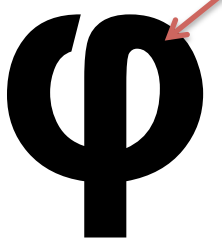
- [Haslinger et al., WCRE'11], [Acher et al., VaMoS'12]

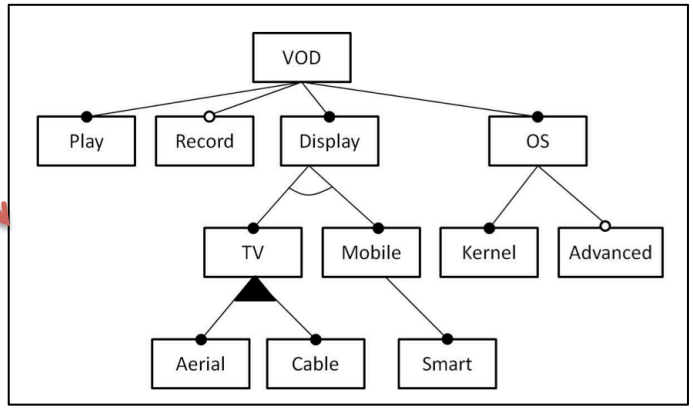
| P | V | P | R | D | O | T | M | S | K | Ad | Ae | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| P2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | ✓ | |
| P3 | ✓ | ✓ | | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| P4 | ✓ | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | |
| P5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ |
| P6 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | | ✓ |
| P7 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | ✓ | ✓ |
| P8 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | | | ✓ |
| P9 | ✓ | ✓ | | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ |
| P10 | ✓ | ✓ | | ✓ | ✓ | ✓ | | | ✓ | ✓ | | ✓ |
| P11 | ✓ | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | ✓ | ✓ |
| P12 | ✓ | ✓ | | ✓ | ✓ | ✓ | | | ✓ | | | ✓ |
| P13 | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | |
| P14 | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | | |
| P15 | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | |
| P16 | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | | | |

```
// from product descriptions to feature models
// typically something generated by VariCell (see VaMoS'12 paper or the dedicated web page)
fm_1 = FM (VOD: R Ae T D P Ad K V O ; )
fm_2 = FM (VOD: R Ae T D P K V O ; )
fm_3 = FM (VOD: Ae T D P Ad K V O ; )
fm_4 = FM (VOD: T Ae D P V K O ; )
fm_5 = FM (VOD: R T Ae D P Ad K V O C ; )
fm_6 = FM (VOD: R T D P Ad V K O C ; )
fm_7 = FM (VOD: R T Ae D P V K O C ; )
fm_8 = FM (VOD: R T D P K V O C ; )
fm_9 = FM (VOD: Ae T D P Ad V K O C ; )
fm_10 = FM (VOD: T D P Ad K V O C ; )
fm_11 = FM (VOD: Ae T D P V K O C ; )
fm_12 = FM (VOD: T D P K V O C ; )
fm_13 = FM (VOD: R S D P Ad V K O M ; )
fm_14 = FM (VOD: R S D P K V O M ; )
fm_15 = FM (VOD: S D P Ad V K O M ; )
fm_16 = FM (VOD: S D P V K O M ; )

// fmR represents the union of configurations/products
// characterized by fm_1, fm_2, ..., fm_16
fmR = merge sunion fm_*

fmR2 = ksynthesis fmR with hierarchy= VOD : V P R D O ; O : K Ad ; D : T M ; T : Ae C ; M : S ;
```

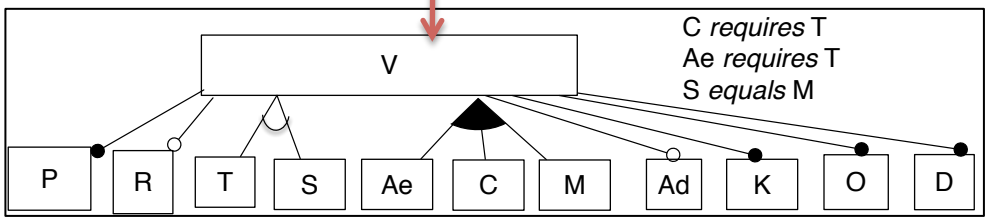


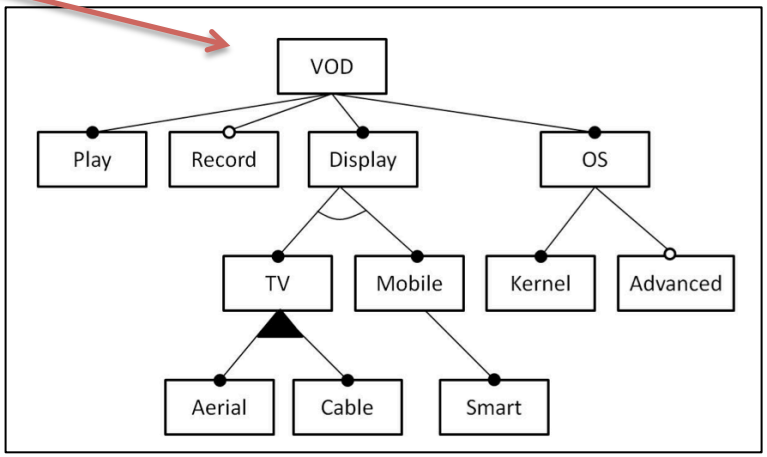


# #2 Refactoring

- [Alves et al., GPCE'06], [Thuem et al., ICSE'09]

```
// refactoring!
fmR2 = ksynthesis fmR with hierarchy= VOD : V P R D O ; O : K Ad ; D : T M ; T : Ae C ; M : S ;
```




φ

```
fml> compare fmR fmR2
res0: (STRING) REFACTORING
```

# Feature Model SemanticS

- As configuration semantics is not sufficient…

- **Ontological** semantics
  - Hierarchy
  - And feature groups

# Exercice #2: Class Diagram

- #1.1 Give two class diagrams with the same semantics but with different syntax
  - (preliminary) what is the semantic domain of CDs?

- #1.2 Does it matter ?

### 3.1 Class diagrams language

As a concrete CD language we use the class diagrams of UML/P [29], a conceptually refined and simplified variant of UML designed for low-level design and implementation. Our semantics of CDs is based on [11] and is given in terms of sets of objects and relationships between these objects. More formally, the semantics is defined using three parts: a precise definition of the syntactic domain, i.e., the syntax of the modeling language CD and its context conditions (we use MontiCore [16, 24] for this); a semantic domain - for us, a subset of the System Model (see [7, 8]) OM, consisting of all finite object models; and a mapping $sem : CD \rightarrow \mathcal{P}(OM)$, which relates each syntactically well-formed CD to a set of constructs in the semantic domain OM. For a thorough and formal account of the semantics see [8].

Note that we use a *complete* interpretation for CDs (see [29] ch. 3.4). This roughly means that 'whatever is not in the CD, should indeed not be present in the object model'. In particular, we assume that the list of attributes of each class is complete, e.g., an `employee` object with an `id` and a `salary` is not considered as part of the semantics of an `Employee` class with an `id` only.

The CD language constructs we support include generalization (inheritance), interface implementation, abstract and singleton classes, class attributes, uni- and bi-directional associations with multiplicities, enumerations, aggregation, and composition.

**Maoz et al. CDDiff: Semantic Differencing for Class Diagrams, ECOOP'11**

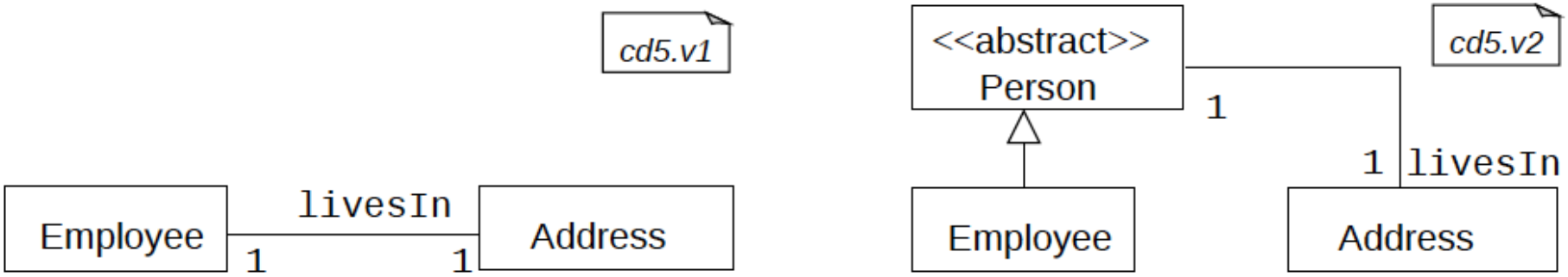



**Fig. 4.** *cd5.v1* and its revised version *cd5.v2*. The two versions have equal semantics.

**Maoz et al. CDDiff: Semantic Differencing for Class Diagrams, ECOOP'11**

# Exercice #3: Feature Model

Given a set of configurations *s*, can we characterize *s* with a feature diagram *fd* ?

ie [[fd]] = s

# Feature Diagram ?

**s = {{A},**
**{A,C,B},**
**{B,A},**
**{C,D,A},**
**{D,A},**
**{A,D,B},**
**{A,C}**
**}**

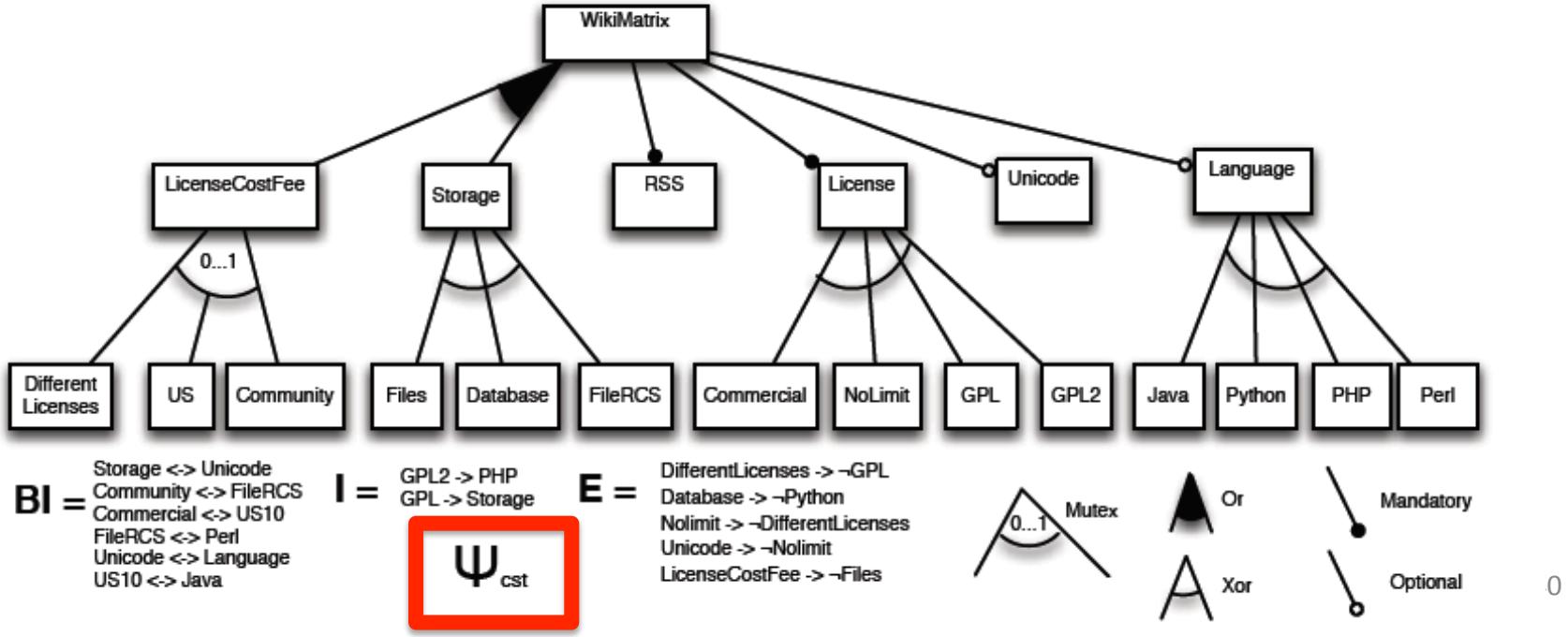
fm1 = FM (A : [B] [C] [D] ^
// B, C and D are optional features of A

**((B & C) -> !D) ^**

)

# Feature Diagram ?

| Identifier | License | Language | Storage | LicenseCostFee | RSS | Unicode |
|---|---|---|---|---|---|---|
| **Confluence** | Commercial | Java | Database | US10 | Yes | Yes |
| **PBwiki** | Nolimit | No | No | Yes | Yes | No |
| **MoinMoin** | GPL | Python | Files | No | Yes | Yes |
| **DokuWiki** | GPL2 | PHP | Files | No | Yes | Yes |
| **PmWiki** | GPL2 | PHP | Files | No | Yes | Yes |
| **DrupalWiki** | GPL2 | PHP | Database | Different Licences | Yes | Yes |
| **TWiki** | GPL | Perl | FilesRCS | Community | Yes | Yes |
| **MediaWiki** | GPL | PHP | Database | No | Yes | Yes |



# Exercice #3: Feature Model (bis)

s = {{A}, {B}}

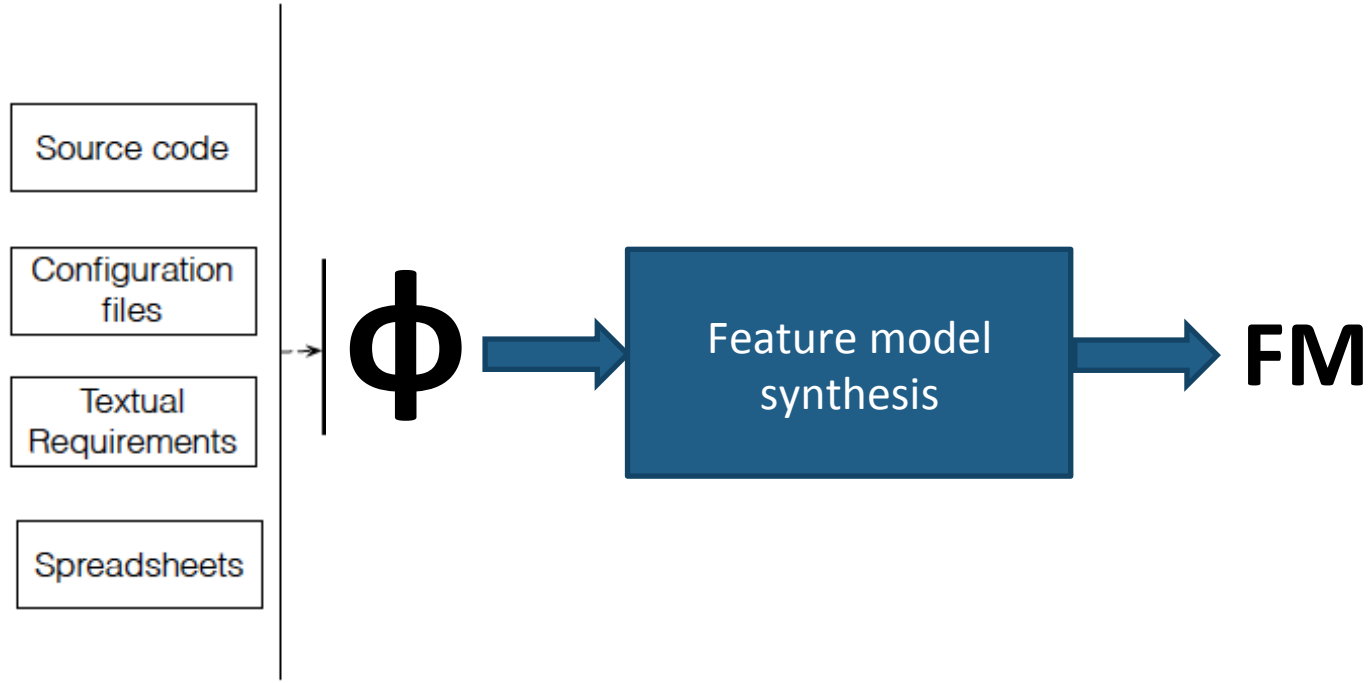
fd = ?

# Feature Model: Key Insights

- Semantics
  - Configuration and ontological
- Syntax
  - Feature diagram vs Feature Model
  - Feature diagram not expressively complete
- Feature models are a (syntactical) view of a propositional formula

# Synthesis techniques

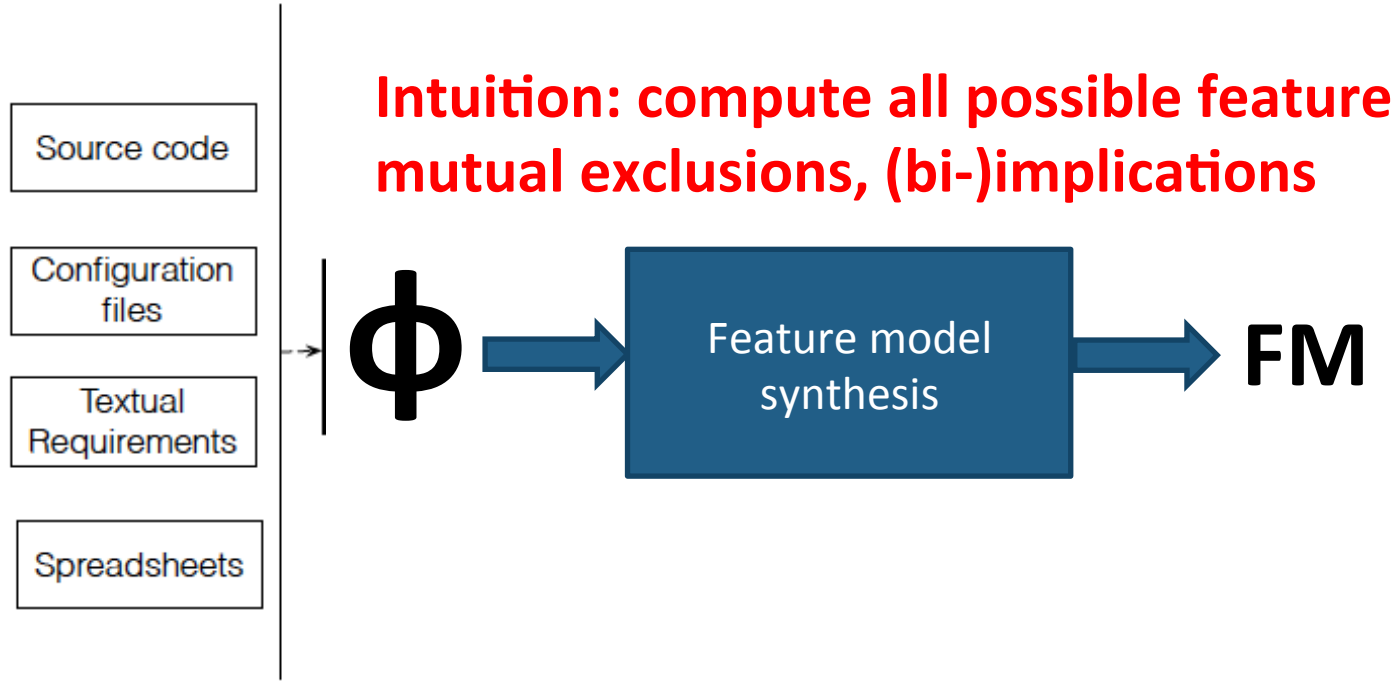
From formula to
feature models (from semantics to syntacs)




## Feature model synthesis problem

**Input:** ϕ, a propositional formula representing the **dependencies** over a set of features *F*.

**Output**: a maximal feature model with a **sound configuration semantics (**and an **appropriate ontological semantics)**




## Feature model synthesis problem

**Input:** ϕ, a propositional formula representing the **dependencies** over a set of features *F*.

**Output**: a maximal feature model with a **sound configuration semantics (**and an **appropriate ontological semantics)**

$(\varphi$ : formula over $F$ rooted in $r$, $r \in F)$

▷ Find and remove all dead features

1 $D = \{f \in F \mid \varphi \wedge r \rightarrow \neg f\}$

2 $\varphi = \varphi[d \mapsto 0]_{d \in D}$

▷ Compute the implication graph $G(V, E)$

3 $V = F \setminus D$

4 $E = \{(u, v) \in V \times V \mid \varphi \wedge u \rightarrow v\}$

▷ Compute strongly connected components

5 $V' = \{S \subseteq V \mid S$ is a SCC of $G\}$

▷ Make edges between SCCs creating a DAG

6 $E' = \{(u, v) \in V' \times V' \mid u \neq v$ and

7 $\exists u' \in u, v' \in v.\ (u', v') \in E\}$

▷ Compute the mutex graph $M(V, E_{\mathrm{x}})$

8 $E_{\mathrm{x}} = \{\{u, v\} \subseteq V' \mid \exists u' \in u, v' \in v.\ \varphi \wedge u' \rightarrow \neg v'\}$

▷ Compute mutex-groups

9 $G_{\mathrm{m}} = \{\{(f_1, p), \ldots, (f_k, p)\} \mid \{f_1, \ldots, f_k\}$ is

10 a maximal clique in $M$ and $\forall f_i.\ (f_i, p) \in E'\}$

▷ Compute or-groups

11 $G_{\mathrm{o}} = \{\{(f_1, p), \ldots, (f_k, p)\} \mid f'_1 \vee \cdots \vee f'_k$ is

12 a prime implicate of $\varphi \wedge p'$ and

13 $p' \in p$ and $\forall f_i.\ f'_i \in f_i \wedge (f_i, p) \in E'\}$

▷ Compute xor-groups

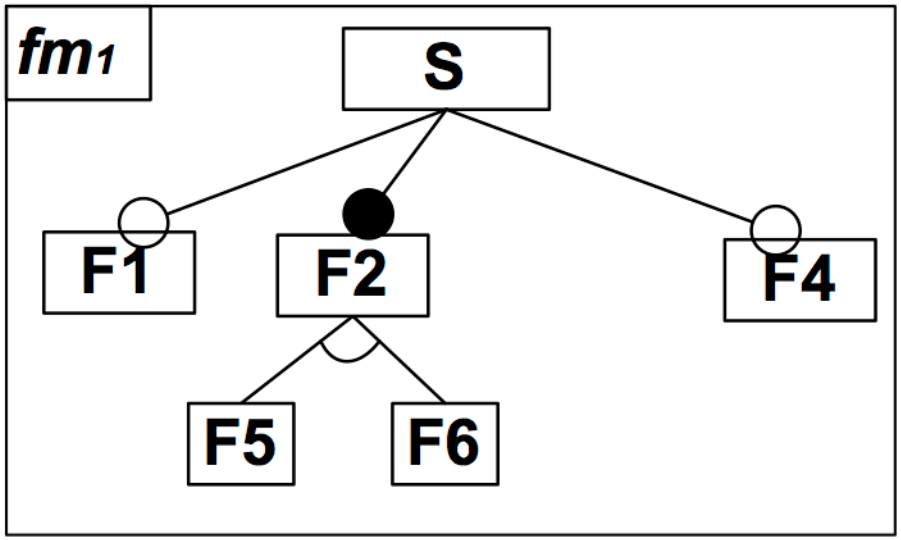
14 $G_{\mathrm{x}} = \{\{(f_1, p), \ldots, (f_k, p)\} \in G_{\mathrm{o}} \mid \forall\, i \neq j.\ (f_i, f_j) \in E_{\mathrm{x}}\}$

**Andersen et al. « Efficient Synthesis of Feature Models », SPLC'12**

(F5 -> F2) ^ (F2 -> S) ^ SYNTETIC_ROOT_FEATURE ^
(F6 -> !F5) ^ (F1 -> S) ^ (SYNTETIC_ROOT_FEATURE
-> S) ^ (F4 -> S) ^ (S -> F2) ^ (F6 -> F2) ^ ((!F2 | F6) |
(!F2 | F5)) ^ (S -> SYNTETIC_ROOT_FEATURE)

```
fml> computeImplies fm1
computeImplies fm1 ===> {(F1 -> F2);(F5 -> S);(S -> F2);(F6 -> F2);(F4 -> S);(F1 -> S);(F5 -> F2);(F4 -> F2);(F2 -> S);(F6 -
fml> cliques fm1
cliques fm1 ===> {{S;F2}}
fml> computeXORGroups fm1
computeXORGroups fm1 ===> {[F6, F5] -> S (XOR);[F6, F5] -> F2 (XOR)}
fml> computeMUTEXGroups fm1
computeMUTEXGroups fm1 ===> {}
fml> computeExcludes fm1
computeExcludes fm1 ===> {(F5 -> !F6)}
```

(F5 -> F2) ^ (F2 -> S) ^
SYNTETIC_ROOT_FEATURE ^ (F6 -> !F5) ^ (F1 ->
S) ^ (SYNTETIC_ROOT_FEATURE -> S) ^ (F4 -> S)
^ (S -> F2) ^ (F6 -> F2) ^ ((!F2 | F6) | (!F2 | F5))
^ (S -> SYNTETIC_ROOT_FEATURE) ^



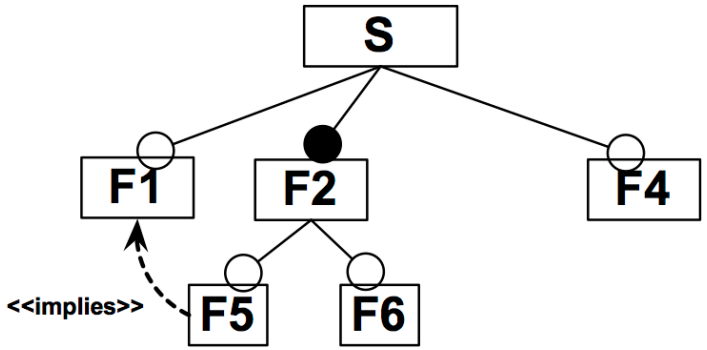

```
fml> computeImplies fm1
computeImplies fm1 ===> {(F1 -> F2);(F5 -> S);(S -> F2);(F6 -> F2);(F4 -> S);(F1 -> S);(F5 -> F2);(F4 -> F2);(F2 -> S);(F6 -
fml> cliques fm1
cliques fm1 ===> {{S;F2}}
fml> computeXORGroups fm1
computeXORGroups fm1 ===> {[F6, F5] -> S (XOR);[F6, F5] -> F2 (XOR)}
fml> computeMUTEXGroups fm1
computeMUTEXGroups fm1 ===> {}
fml> computeExcludes fm1
computeExcludes fm1 ===> {(F5 -> !F6)}
```

# Exercice #4: There and Back Again

#4.1 Translate the feature model to a proposition formula

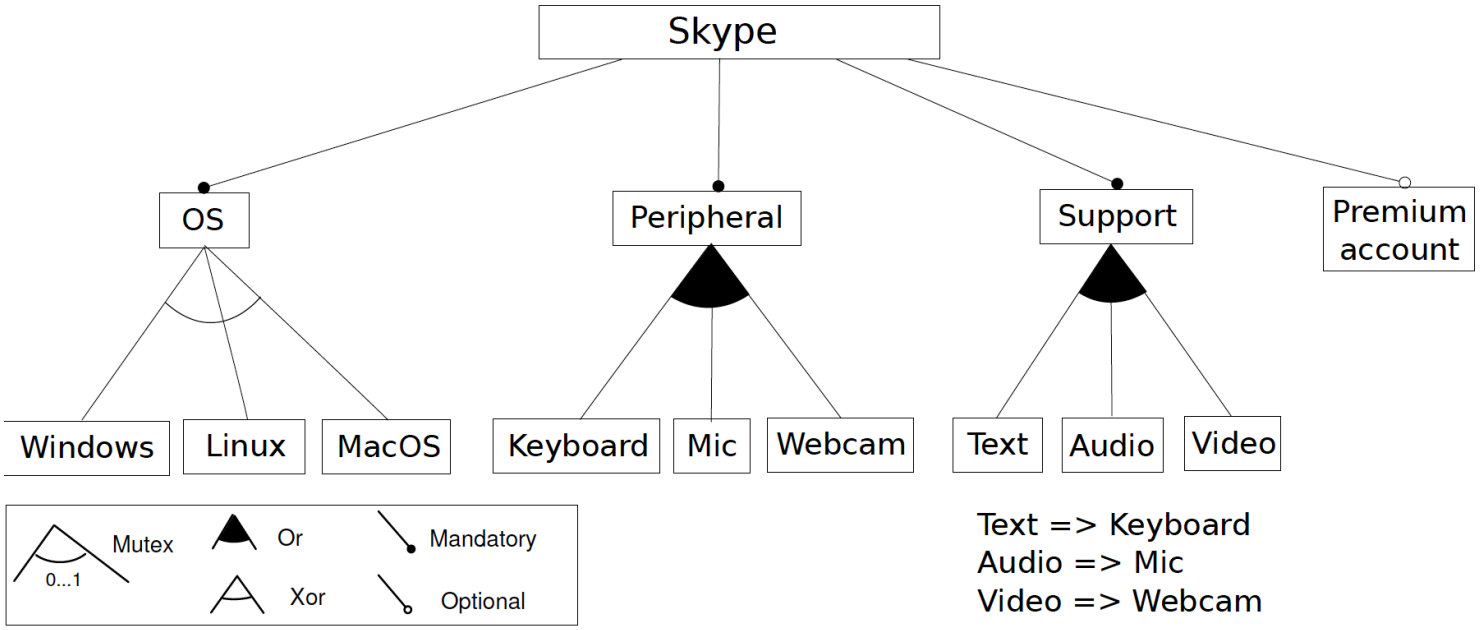
#4.2 Compute the

Synthesis « structures »

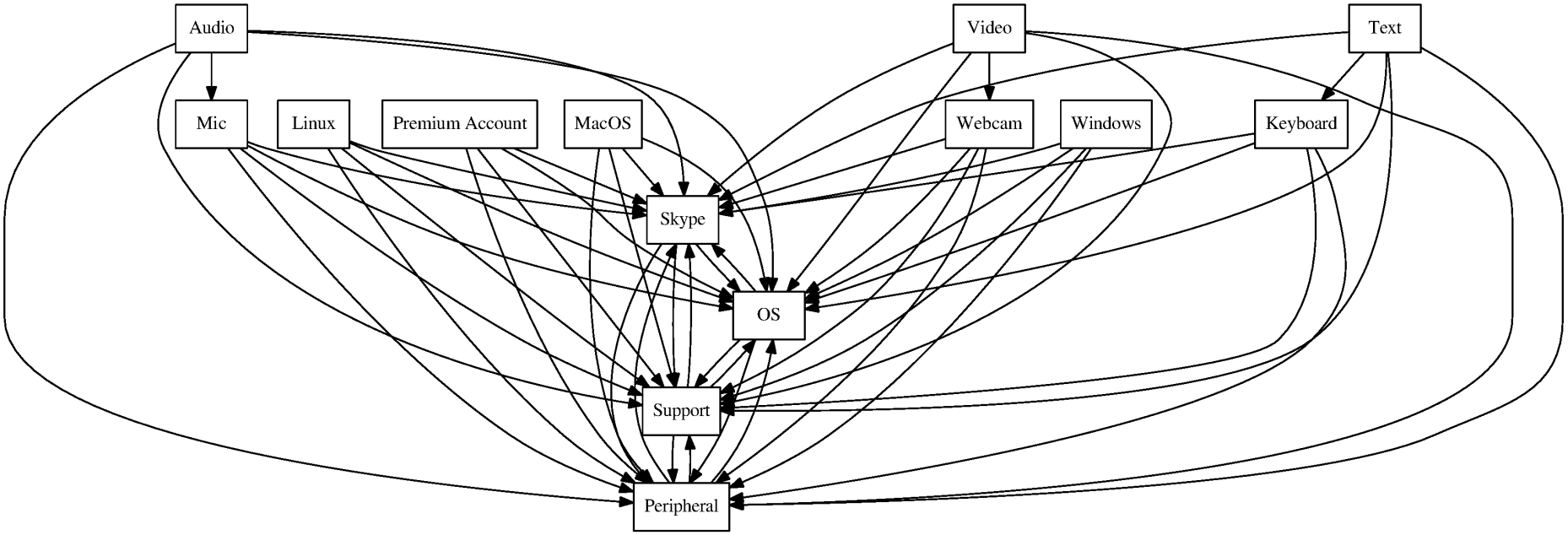


# Key: Binary Implication Graph

**Sound and complete representation of possible hierarchies**



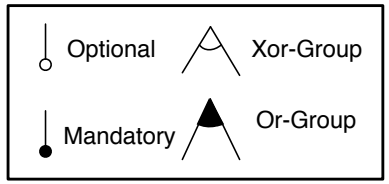
# Sound and complete representation of possible hierarchies

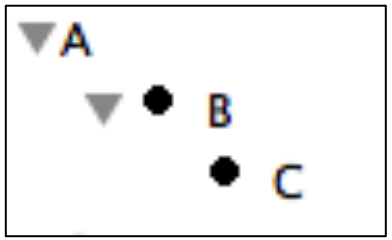


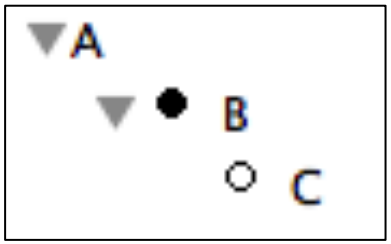
mergeNonPC.fml

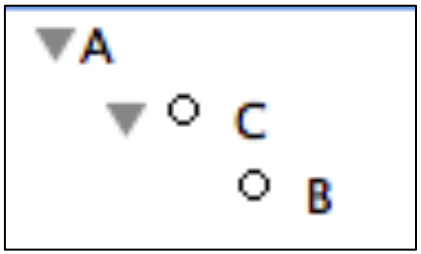
# Merging operation: back to hierarchy

```
fm1 = FM (A : B ; B : C ;)
fm2 = FM (A : B ; B : [C] ;)
fm3 = FM (A : [C] ; C : [B] ; )
fm4 = merge sunion { fm1 fm2 fm3 }
```



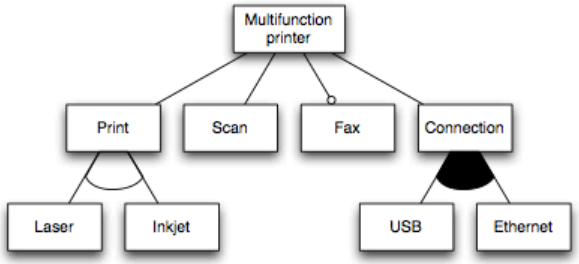




> configs fm4
> res12: (SET) {{C;A};{A;B};{A};{A;B;C}}

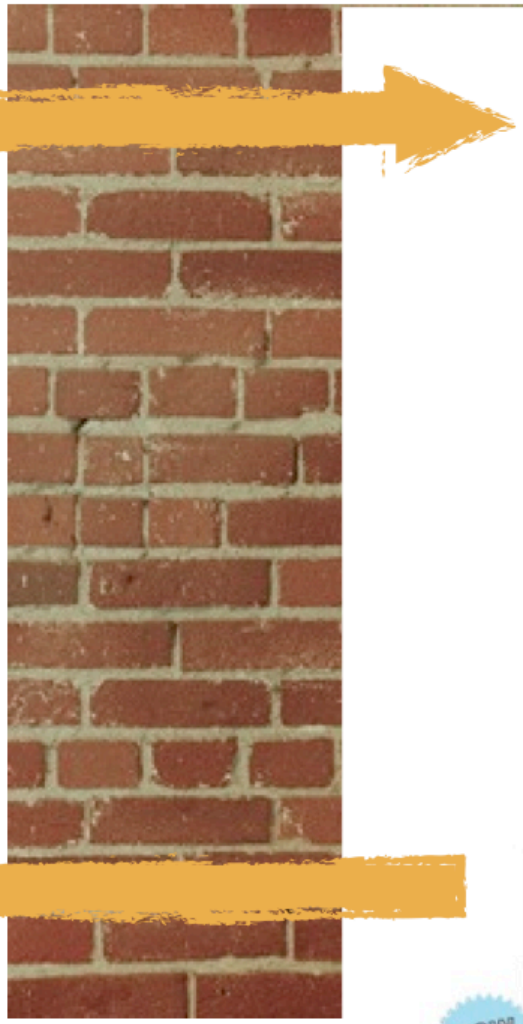
?

# Typical implementations





result



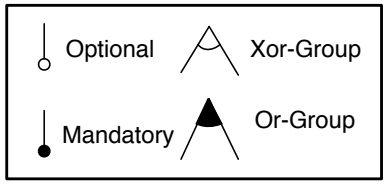
logics

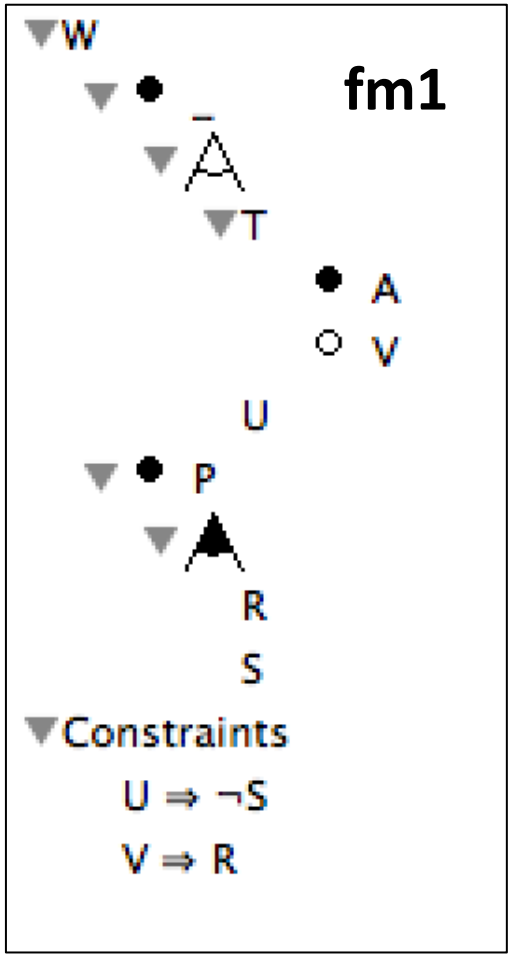
solvers



# (Boolean) Feature Models

**Hierarchy + Variability = set of valid configurations**

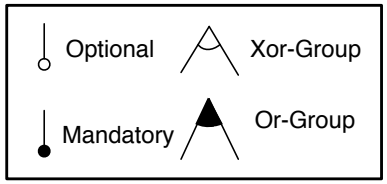


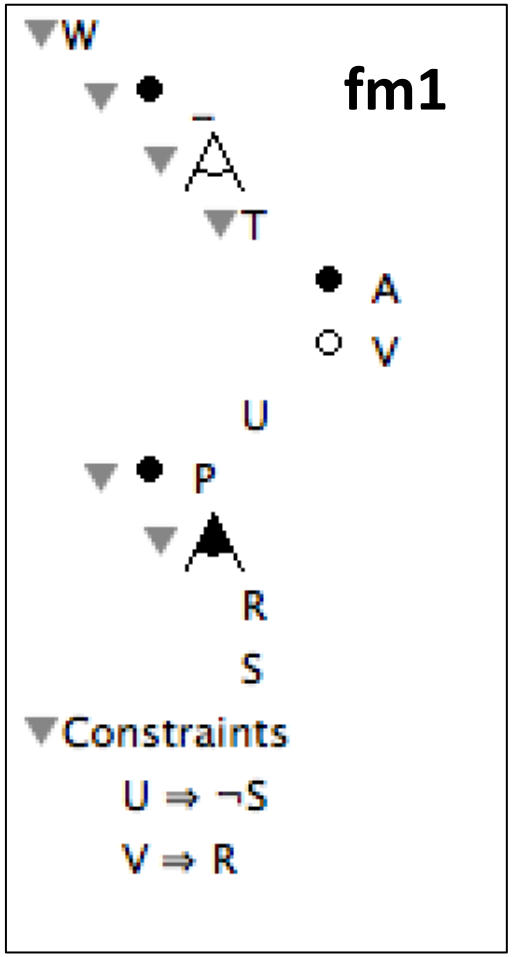




$$[\![fm1]\!] = \{$$
$$\{W, P, R, S, T, A, V\},$$
$$\{W, P, S, T, A\},$$
$$\{W, P, R, T, A\},$$
$$\{W, P, R, U\},$$
$$\{W, P, R, T, V, A\},$$
$$\{W, P, R, S, T, A\},$$
$$\}$$

# (Boolean) Feature Models

## ~ Boolean formula



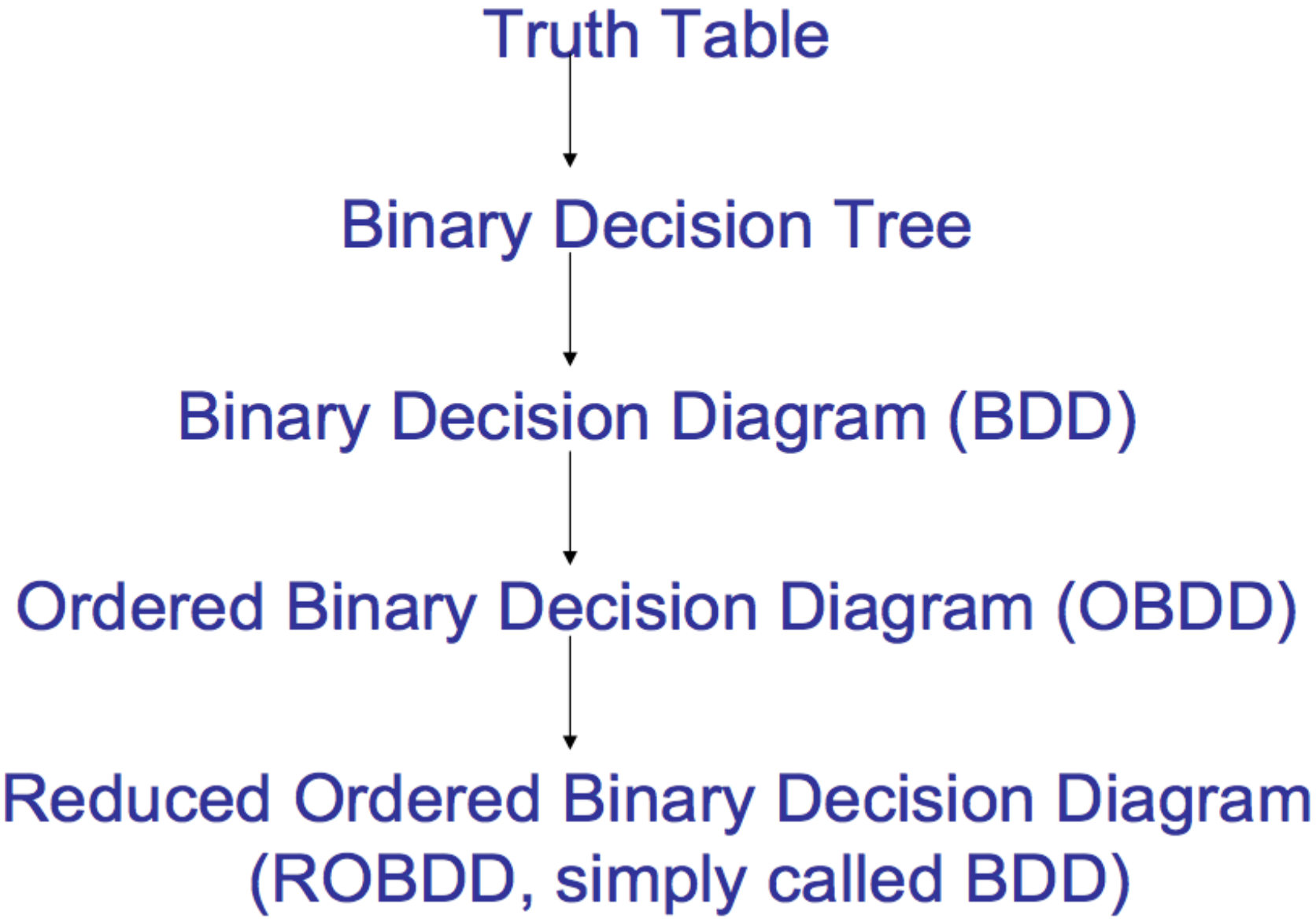




$$
\begin{aligned}
&\phi_{fm_1} = W \;//\; \text{root} \\
&\wedge\, W \Leftrightarrow P \;//\; \text{mandatory} \\
&//\; \text{Or-group} \\
&\wedge\, P \Rightarrow R \vee S \\
&\wedge\, R \Rightarrow P \wedge\, S \Rightarrow P \\
&\wedge\, V \Rightarrow T \;//\; \text{optional} \\
&\wedge\, A \Leftrightarrow T \;//\; \text{mandatory} \\
&//\; \text{Xor-group} \\
&\wedge\, T \Rightarrow W \\
&\wedge\, U \Rightarrow W \\
&\wedge\, \neg T \vee \neg U \\
&//\; \text{constraints} \\
&\wedge\, V \Rightarrow R \;//\; \text{implies} \\
&\wedge\, \neg U \Rightarrow \neg S \;//\; \text{excludes}
\end{aligned}
$$

# Truth table, boolean function

| *from* | | *to* | | |
|---|---|---|---|---|
| $x_1$ | $x_2$ | $x_3$ | $x_4$ | f |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

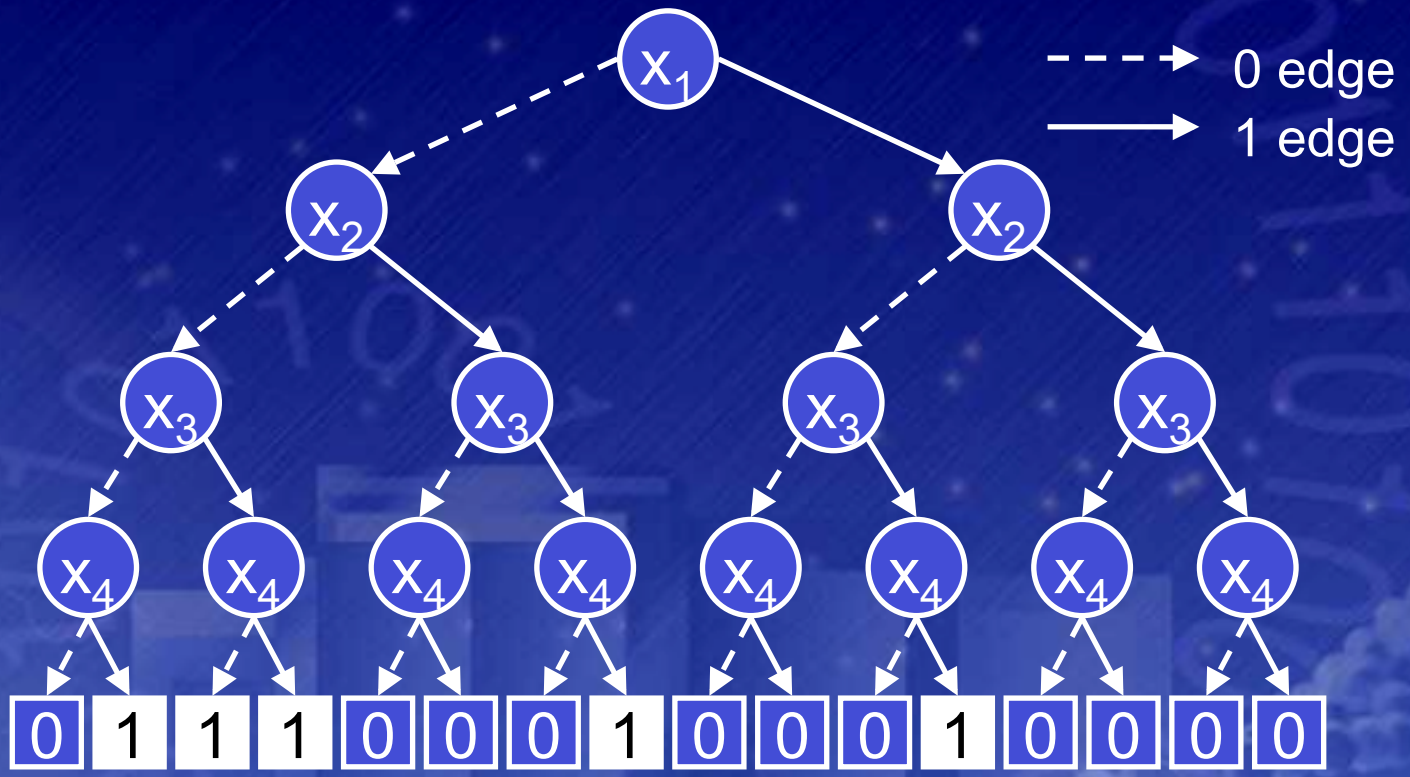
# BDDs from Truth Tables



# Binary Decision Diagrams (Bryant 1986)

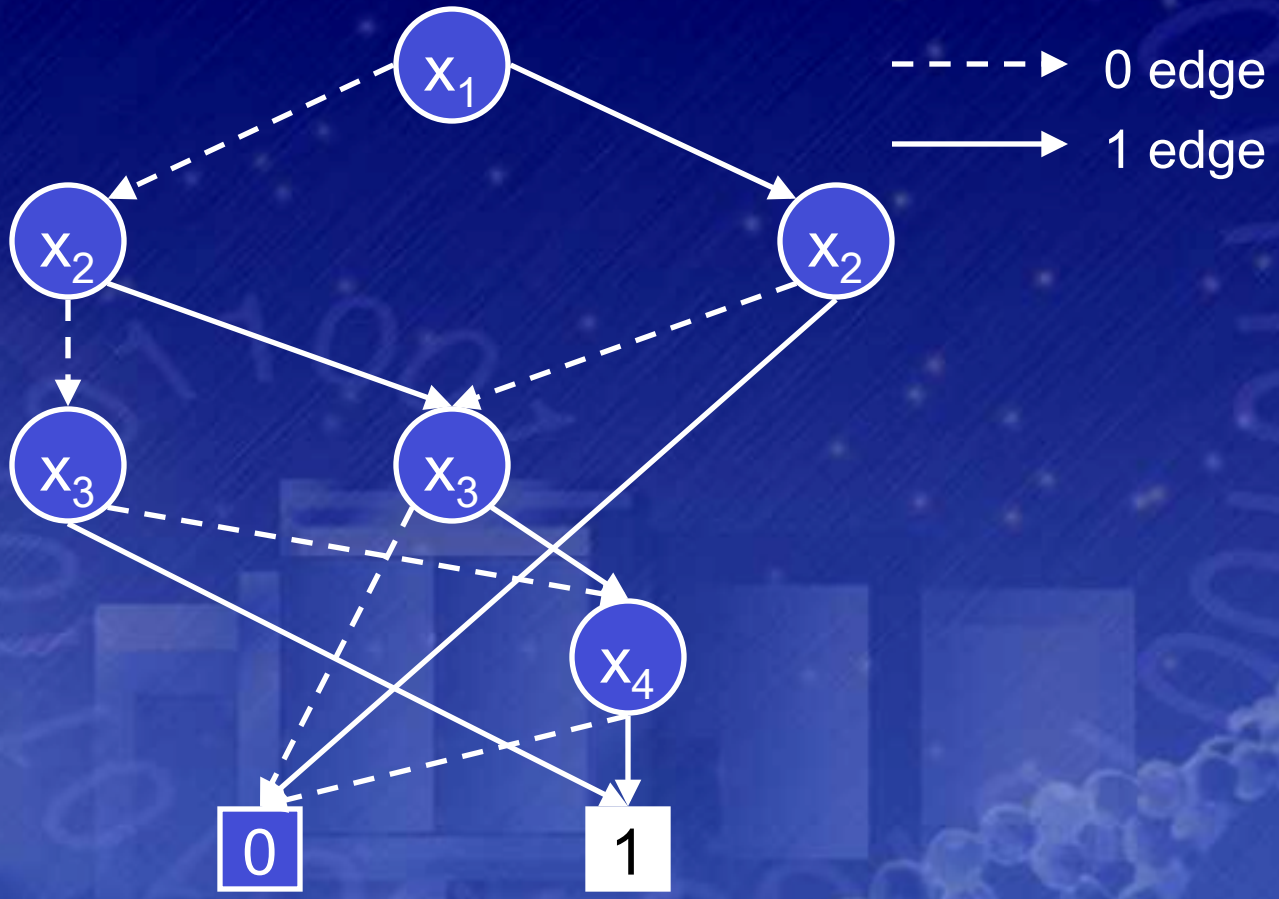
encoding of a truth table.

| *from* | | *to* | | |
|---|---|---|---|---|
| $x_1$ | $x_2$ | $x_3$ | $x_4$ | f |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |



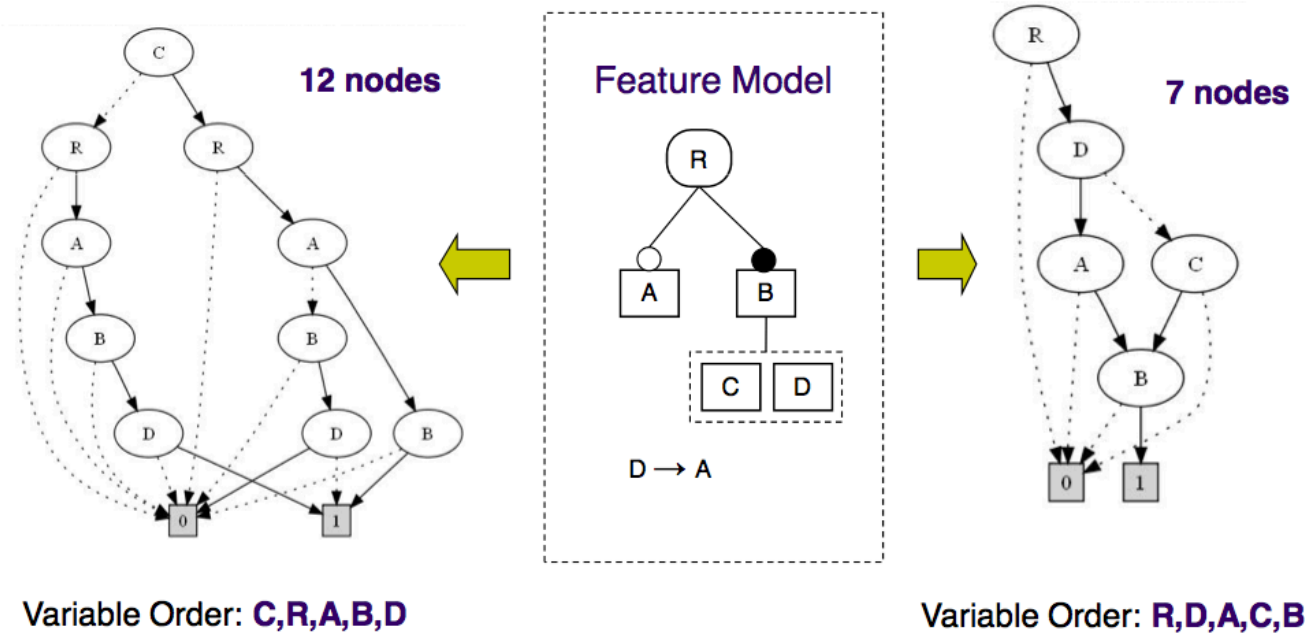
# Binary Decision Diagrams

- Eliminate unnecessary nodes.



# Binary Decision Diagrams (BDDs): Theoretical Problem

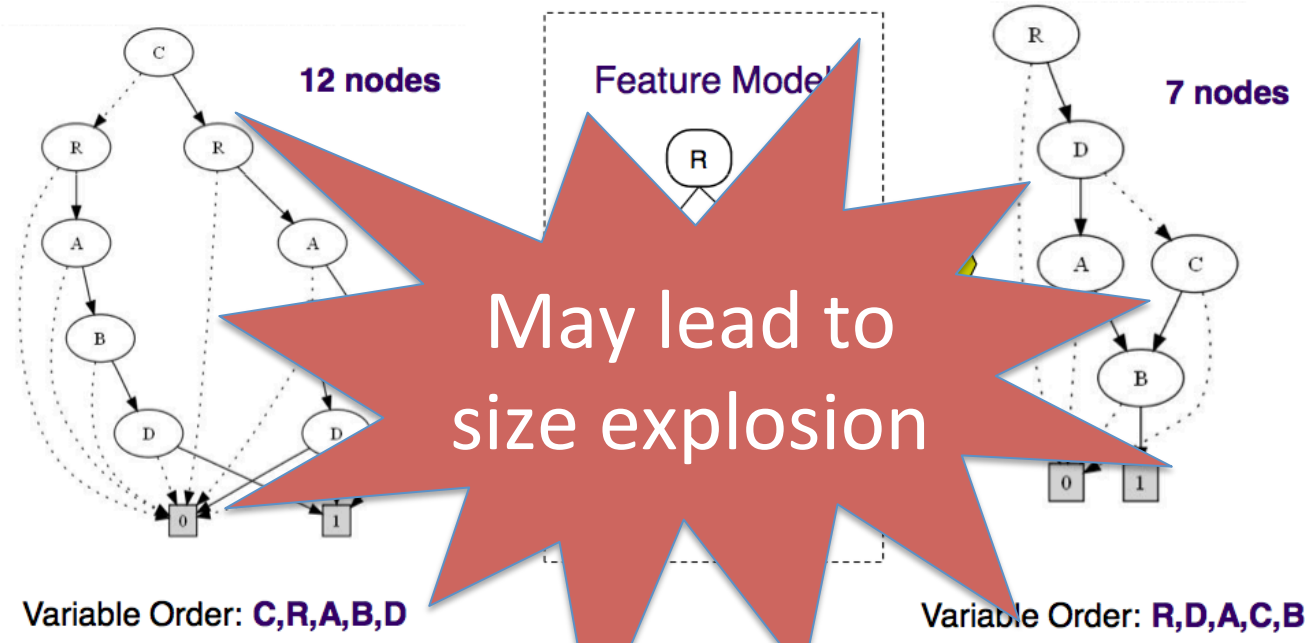
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**




*[Mendonca, slide]*

# Binary Decision Diagrams (BDDs): Theoretical Problem

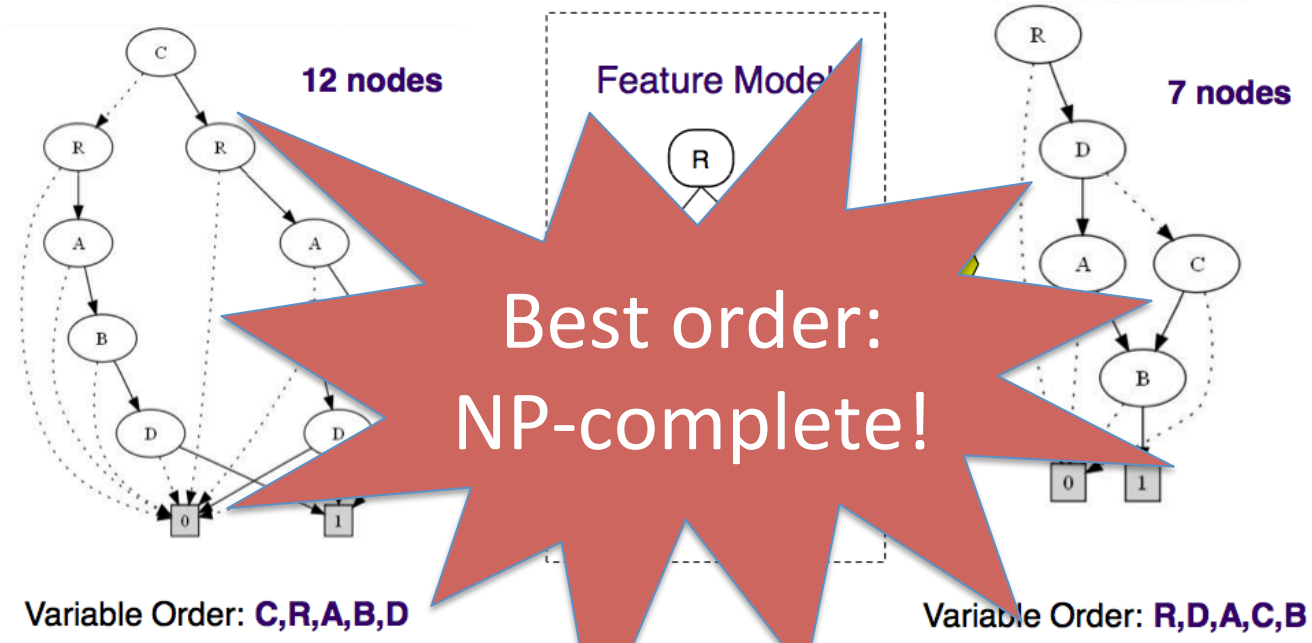
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**




*[Mendonca, slide]*

# Binary Decision Diagrams (BDDs): Theoretical Problem

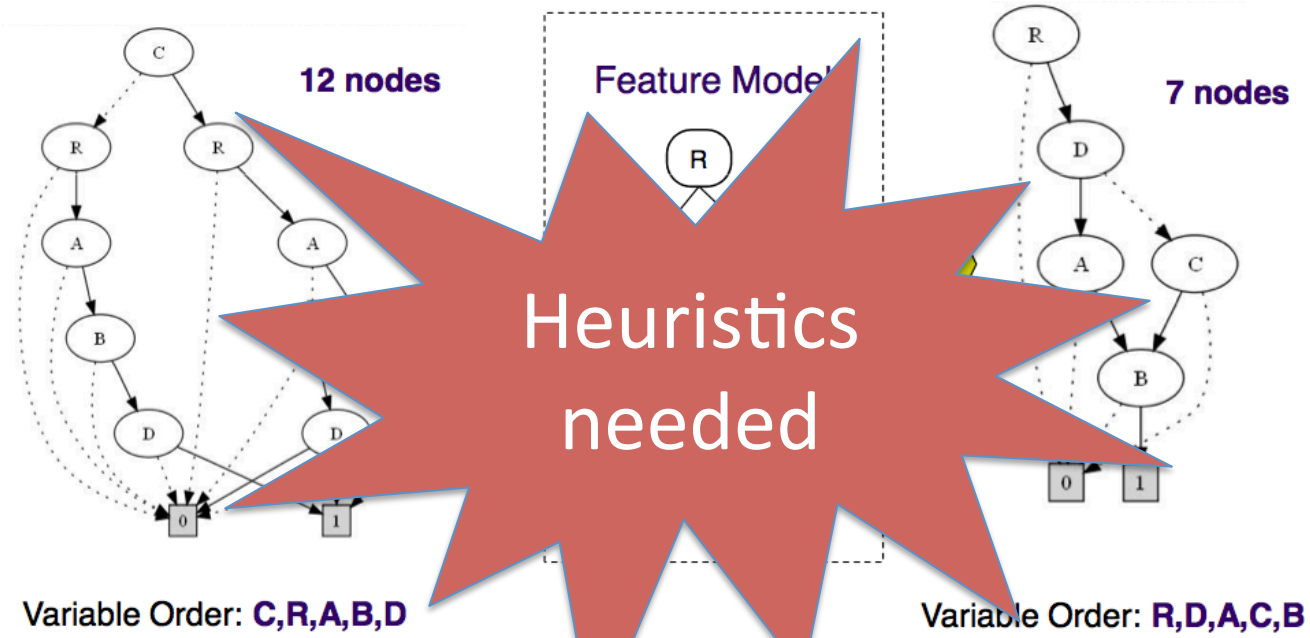
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**




*[Mendonca, slide]*

# Binary Decision Diagrams (BDDs): Practical Problem

- The size of the BDD is very sensitive to the order of the BDD variables. In practice: **BDDs cannot be build for feature models with 2000+ features**




*[Mendonca, slide]*

# SAT solver and Unit Propagation

- Whenever all the literals in a clause are false except one, the remaining literal must be true in any satisfying assignment (such a clause is called a **unit clause**).
  - Therefore, the algorithm can assign it to true immediately. After choosing a variable there are often many unit clauses.
  - Setting a literal in a unit clause often creates other unit clauses, leading to a cascade.

$$\{\neg p \vee q,\ \neg p \vee \neg q \vee r,\ p,\ \neg r\}.$$

| | |
|---|---|
| $\neg p \vee q$ | $q$ |
| $\neg p \vee \neg q \vee r$ | $\neg q \vee r$ |
| ~~$p$~~ | |
| $\neg r$ | $\neg r$ |

- A good SAT solver often spends 80-90% of its time in unit propagation.

$$\mathcal{F}_{\text{unit}} := (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$
$$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_5)$$
$$(x_1 \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)$$

$$\mathcal{F}_{\text{unit}} := (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$
$$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_5)$$
$$(x_1 \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)$$

$$\varphi = \{x_1 = 1\}$$

$$\mathcal{F}_{\text{unit}} := (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$
$$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_5)$$
$$(x_1 \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)$$

$$\varphi = \{x_1{=}1, x_2{=}1\}$$

$$\mathcal{F}_{\text{unit}} := (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$
$$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_5)$$
$$(x_1 \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)$$

$$\varphi = \{x_1{=}1, x_2{=}1, x_3{=}1\}$$

$$\mathcal{F}_{\text{unit}} := (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$
$$(\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_5)$$
$$(x_1 \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)$$

$$\varphi = \{x_1{=}1, x_2{=}1, x_3{=}1, x_4{=}1\}$$

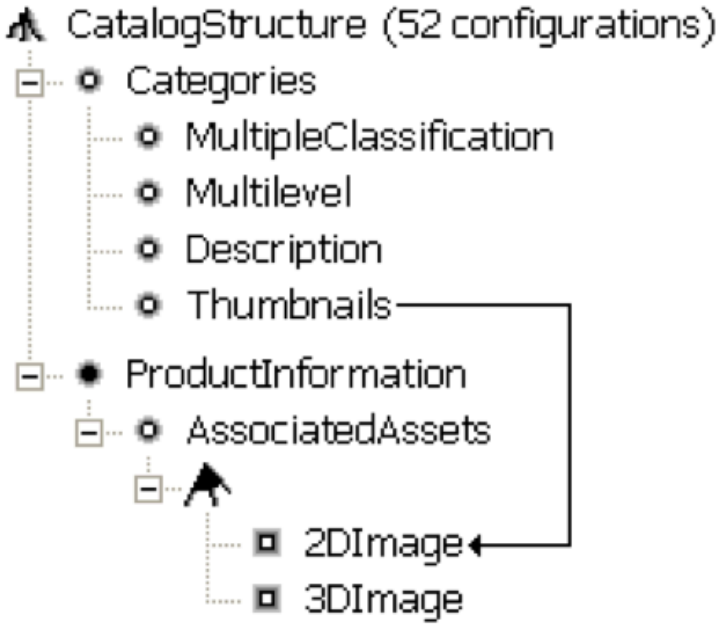
# SAT solver and Unit Propagation

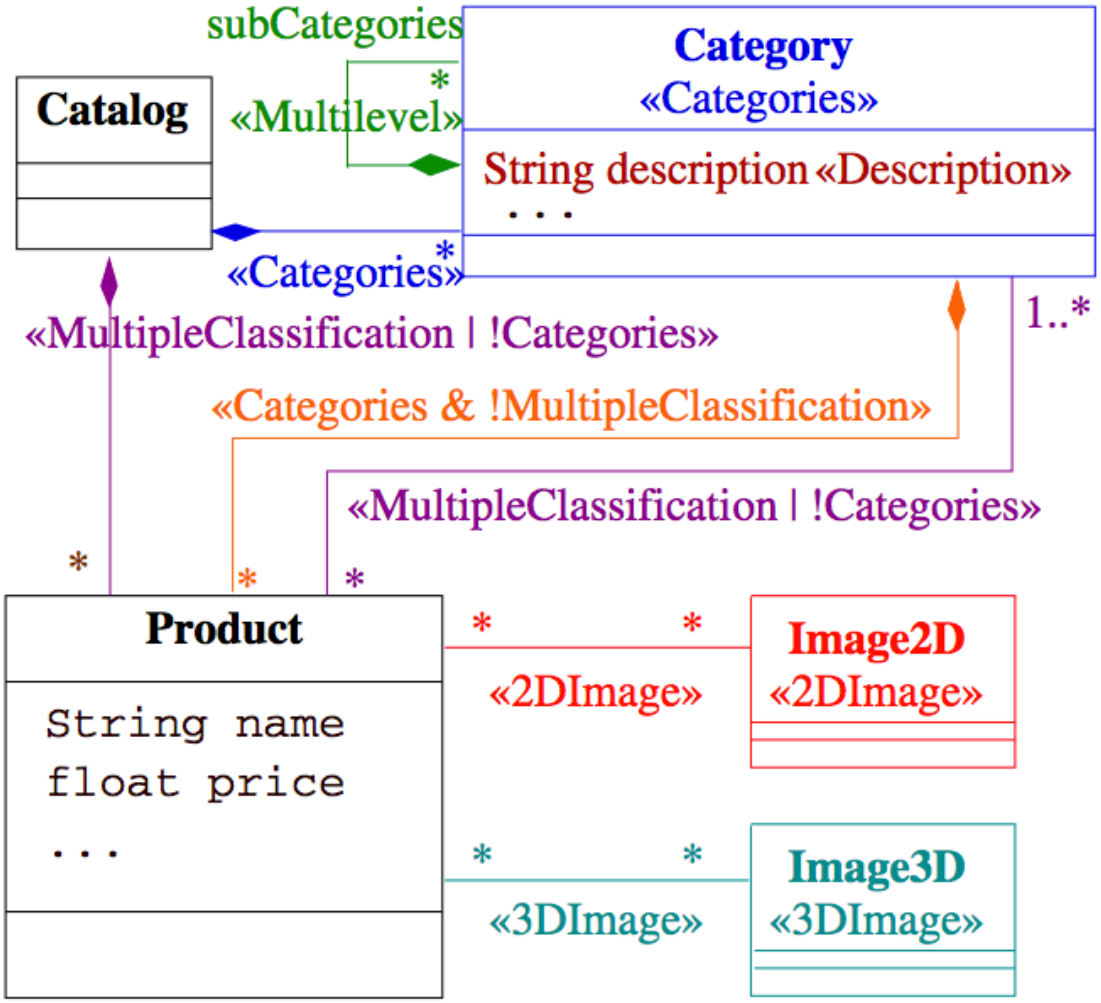
```
BCP():
   Repeatedly search for unit clauses, and
      set unassigned literal to required value.
   If a literal is assigned conflicting values, return F
      else return T;
```

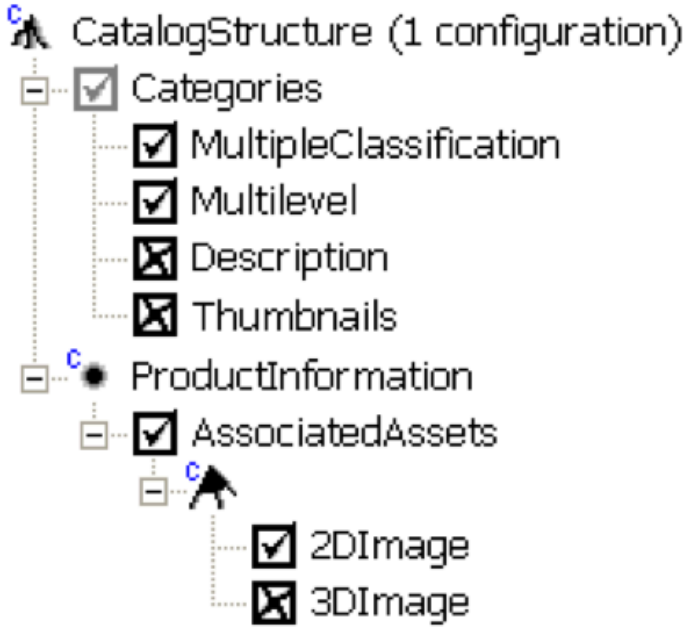
```
satisfy(φ) {
   if every clause of φ has a true literal, return T;
   if BCP() == F, return F;
   assign appropriate values to all pure literals;
   choose an x ∈ V that is unassigned in A,
      and choose v ∈ {T, F}.
   A(x) = v;
   if satisfy(φ) return T;
   A(x) = ¬v;
   if satisfy(φ) return T;
   unassign A(x); // undo assignment for backtracking.
   return F; }
```

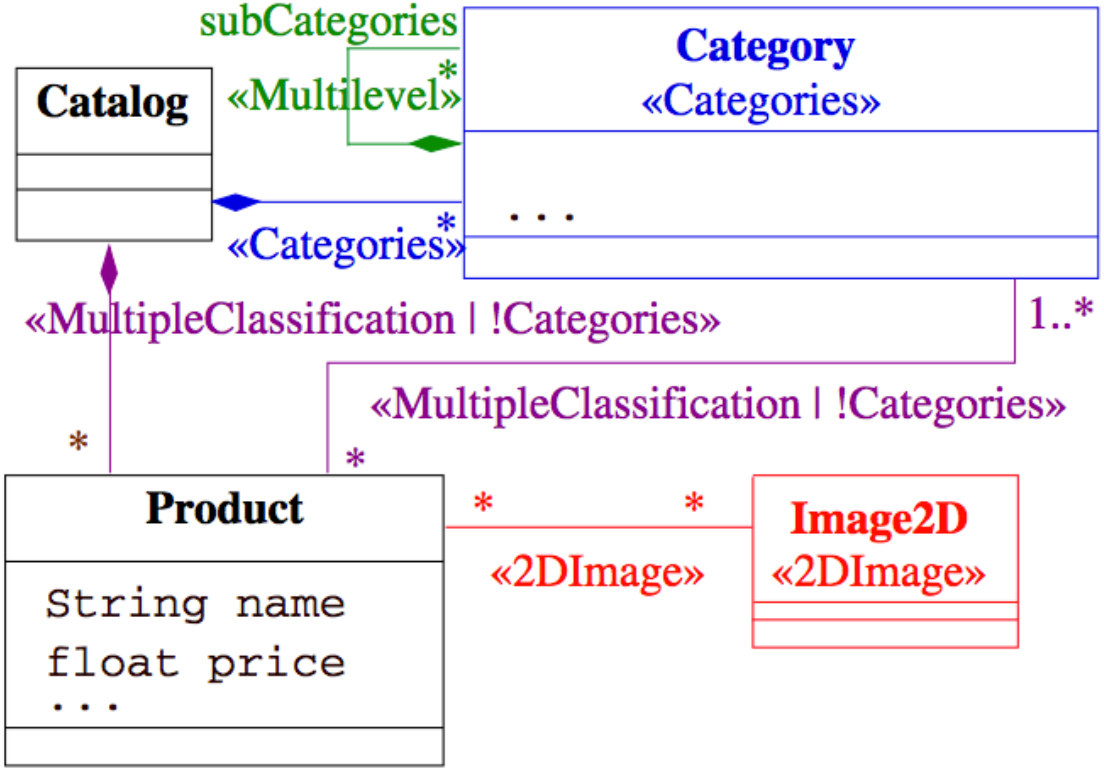
# Composition of Feature Models

# Two Open Problems


CatalogStructure (52 configurations)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage


subCategories
Catalog
«Multilevel»
Category
«Categories»
String description «Description»
. . .
«Categories»
«MultipleClassification | !Categories»
1..*
«Categories & !MultipleClassification»
«MultipleClassification | !Categories»
Product
String name
float price
. . .
«2DImage»
Image2D
«2DImage»
«3DImage»
Image3D
«3DImage»


CatalogStructure (1 configuration)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage


subCategories
Catalog
«Multilevel»
Category
«Categories»
...
«Categories»
«MultipleClassification | !Categories»
1..*
«MultipleClassification | !Categories»
Product
String name
float price
...
«2DImage»
Image2D
«2DImage»

# #Open Problem 1

- How to compose a model-based product line
  - ie <FM1, m1, BM1> with <FM2, m2, BM2>

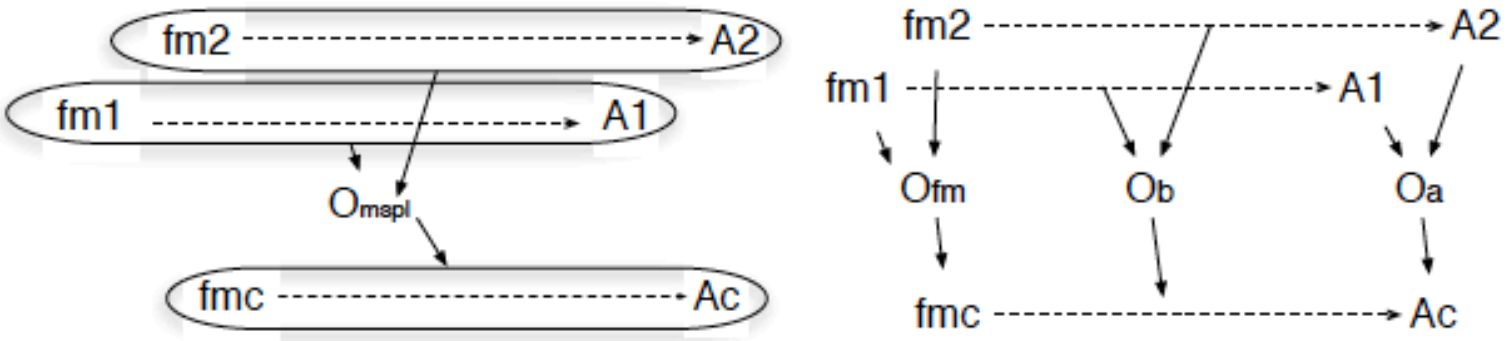



Fig. 9: Composing model-based SPL (left-part) is mirroring the semantics of the composition-based operators on the bindings and the assets (right-part)

# Exercice

- Give an example of composition with class diagrams

- Think about the possible result

# #Open Problem 2

- Homework

- Composition of **attributed** feature models