Modeling Variability
(introduction to feature models)

Mathieu Acher
Maître de Conférences
mathieu.acher@irisa.fr

Material

https://github.com/FAMILIAR-project/HackOurLanguages-SIF

Plan

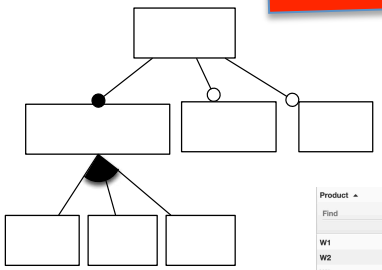
- Challenges and Overview
  - Developping billions of software product is hard but now a common practice
- Implementing Variability
  - Revisit of existing techniques and curriculum
- Specificity of Product Line Engineering
  - Process, methods
- **Feature Models**
  - **Defacto standard for modeling product lines and variability**
  - **Syntax, semantics, automated reasoning, synthesis**

Contract

- The idea of software product lines and variability
  - You will be able to recognize this class of systems
  - Aware of the complexity, the specific development process, and existing techniques
- **Feature modeling**
  - **A widely used formalism for modeling product lines and configurable systems in a broad sense**
- Composing/Decomposing feature models with a domain-specific language
- Reverse engineering variability models

Modeling Variability

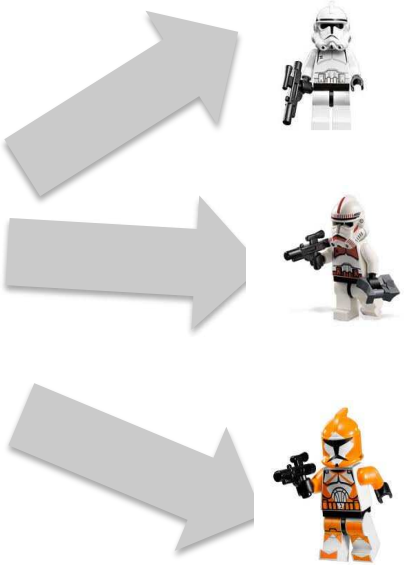
Modeling and Reverse Engineering Variability



Product	License	Price	Language Support	Language	WYSIWIG
	Find				
W1	Commercial	10	Yes	Java	Yes
W2	NoLimit	20	No		Yes
W3	NoLimit	10	No		Yes
W4	GPL	0	Yes	Python	Yes
W5	GPL	0	Yes	Perl	Yes
W6	GPL	10	Yes	Perl	Yes
W7	GPL	0	Yes	PHP	No
W8	GPL	10	Yes	PHP	Yes

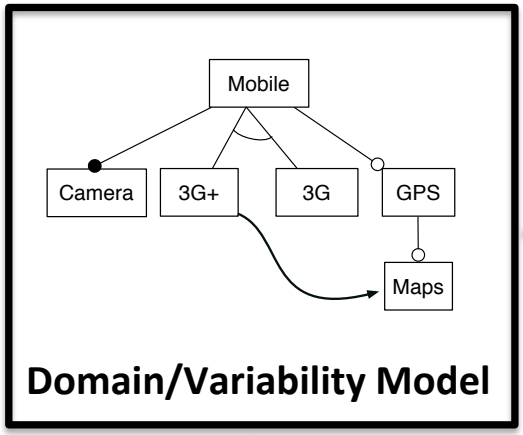
Feature models
or Product Matrices

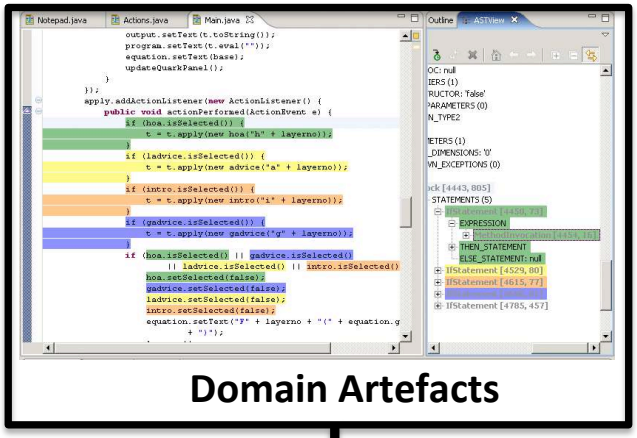
(product lines)



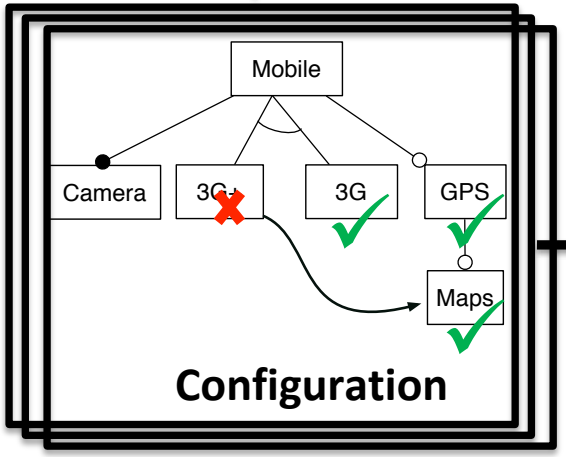
Variants of code (e.g., Java or C)
Variants of user interfaces
Variants of video sequences
Variants of models (e.g., UML or SysML)
Variants of « things » (3D models)
…

Domain Engineering



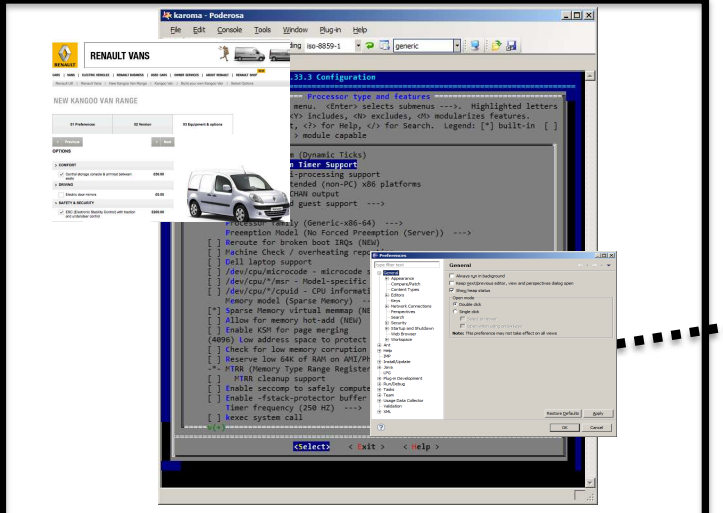


Application Engineering



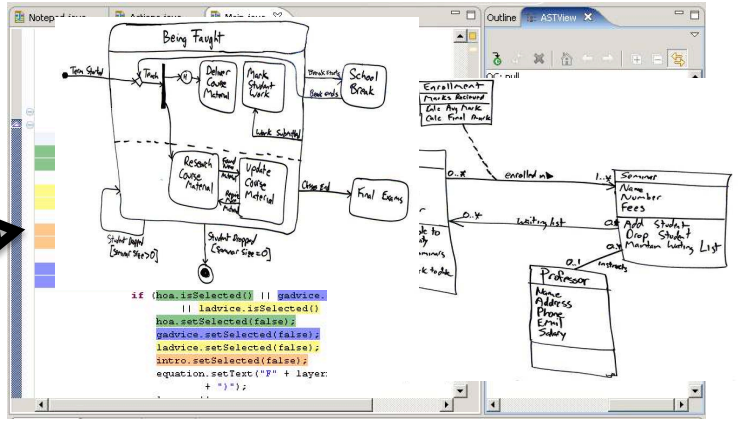
*« the investments required to develop the reusable artifacts during **domain engineering**, are outweighed by the benefits of deriving the individual products during **application engineering** »*

Jan Bosch et al. (2004)

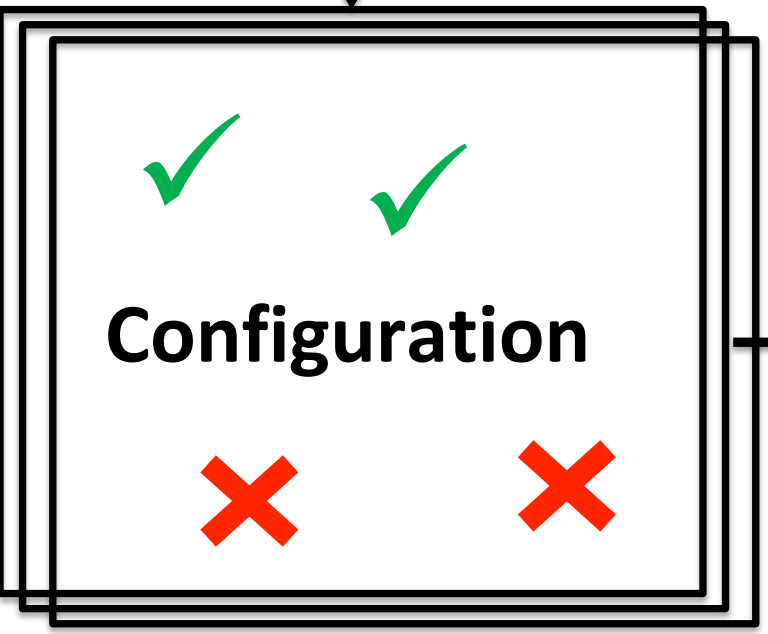


Variability Model

mapping



Base Artefacts (e.g., models)

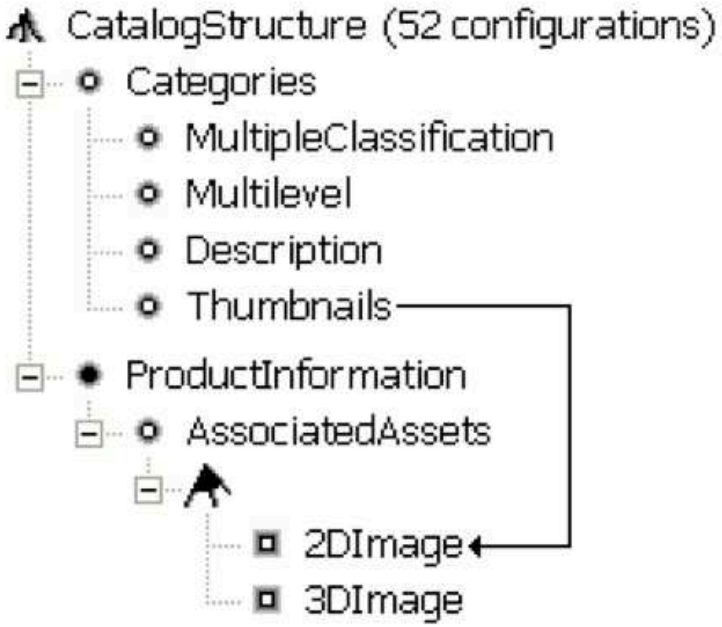


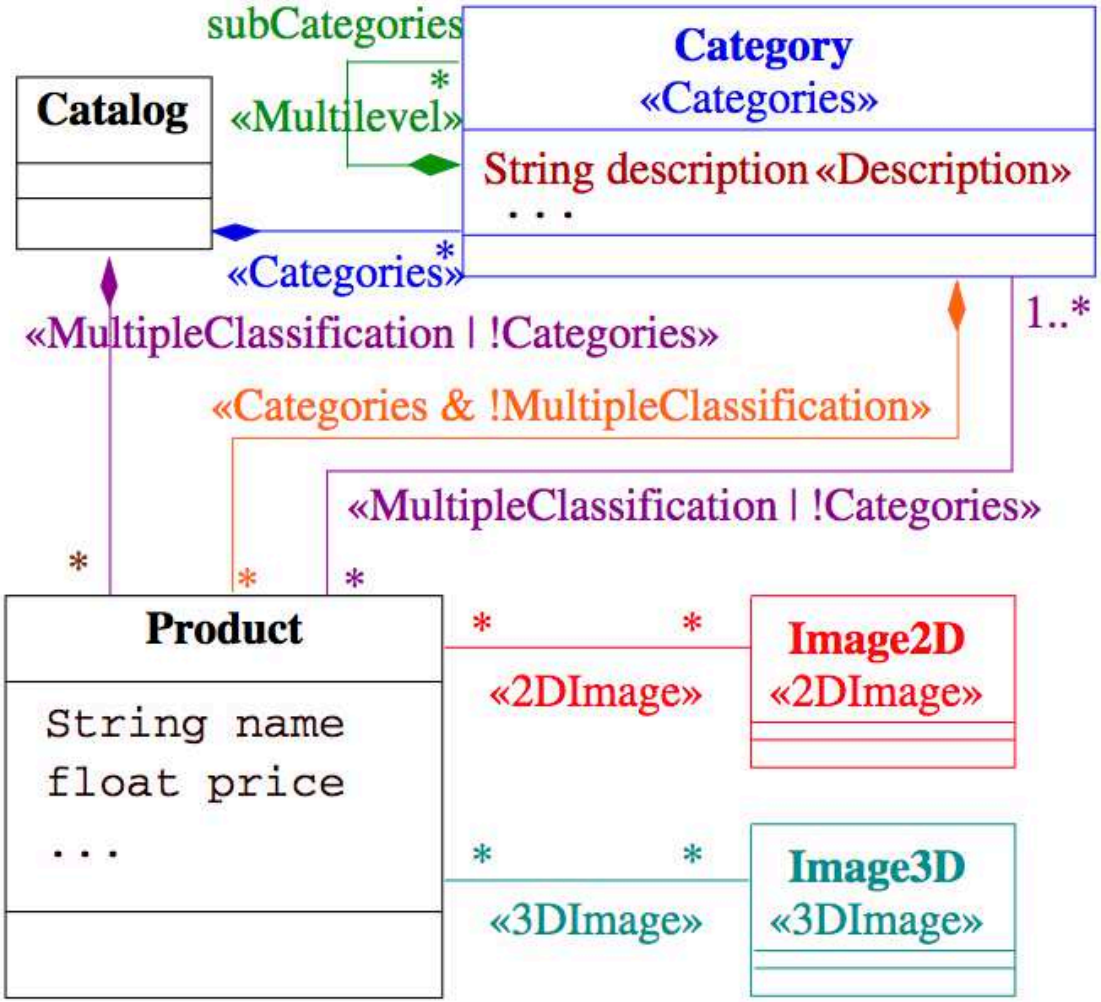
Configuration

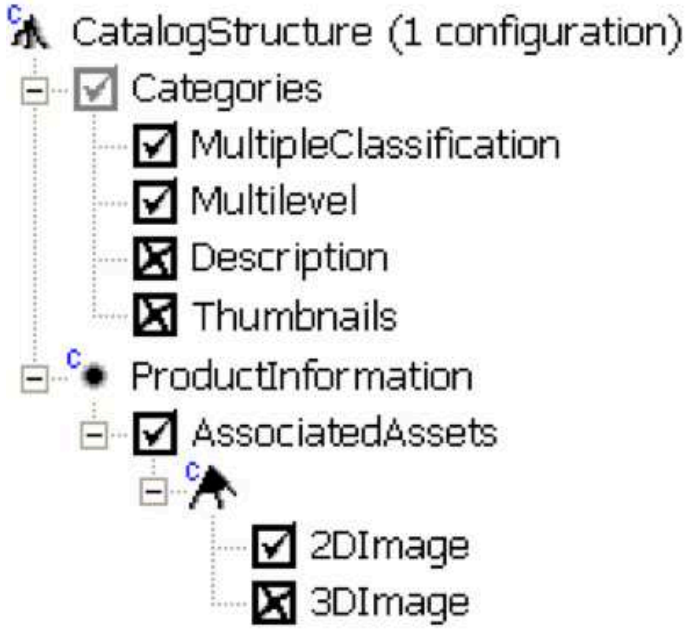
Software Generator
(derivation engine)

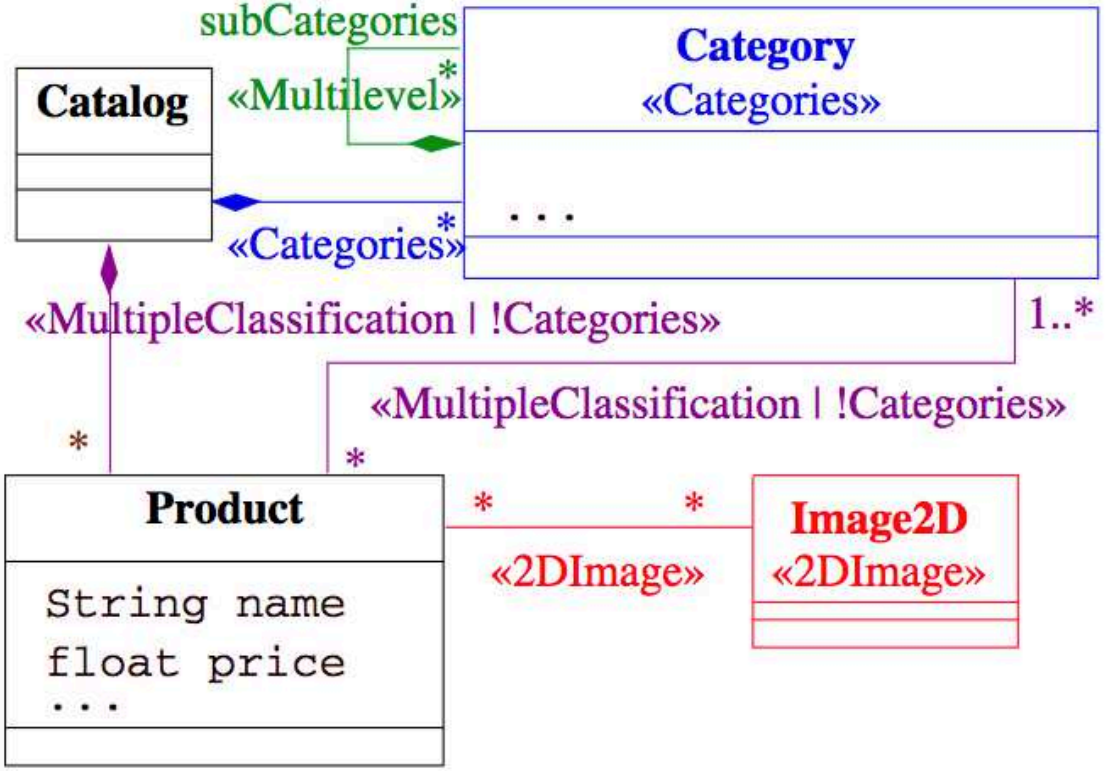
Quizz: what are the constraints over WORLD and BYE?

```
#include <stdio.h>

#ifdef WORLD
char * msg = "Hello World\n";
#endif
#ifdef BYE
char * msg = "Bye bye!\n";
#endif

main() {
  printf(msg);
}
```

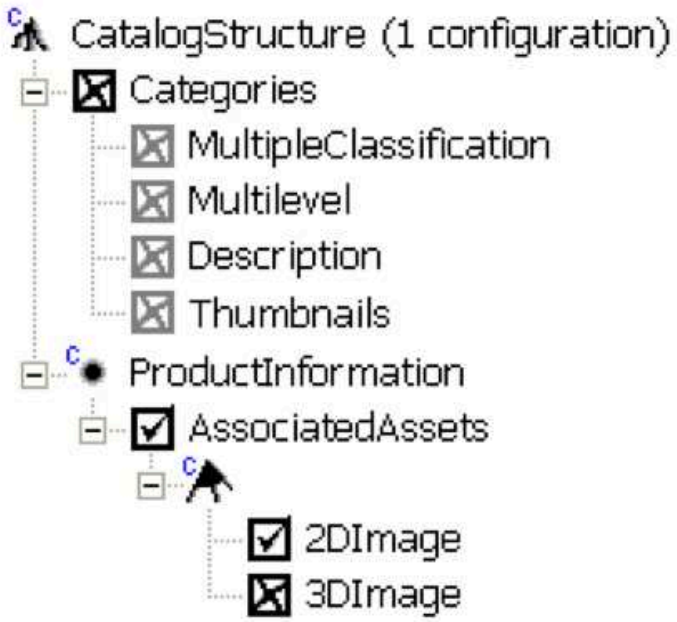

CatalogStructure (52 configurations)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage

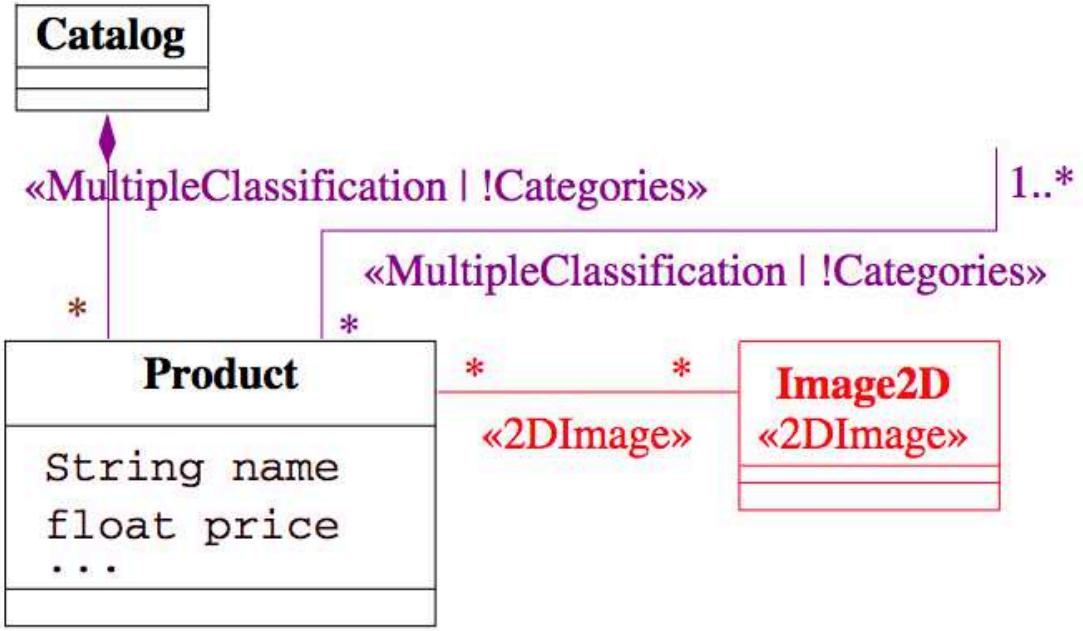

subCategories
Catalog
«Multilevel»
Category
«Categories»
String description «Description»
. . .
«Categories»
«MultipleClassification | !Categories»
«Categories & !MultipleClassification»
«MultipleClassification | !Categories»
1..*
Product
String name
float price
. . .
«2DImage»
Image2D
«2DImage»
«3DImage»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

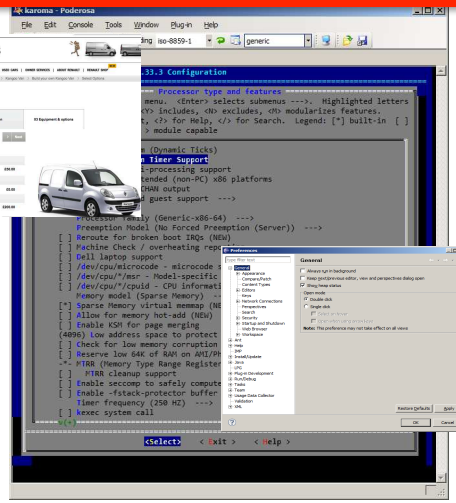

CatalogStructure (1 configuration)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage


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

Ooops

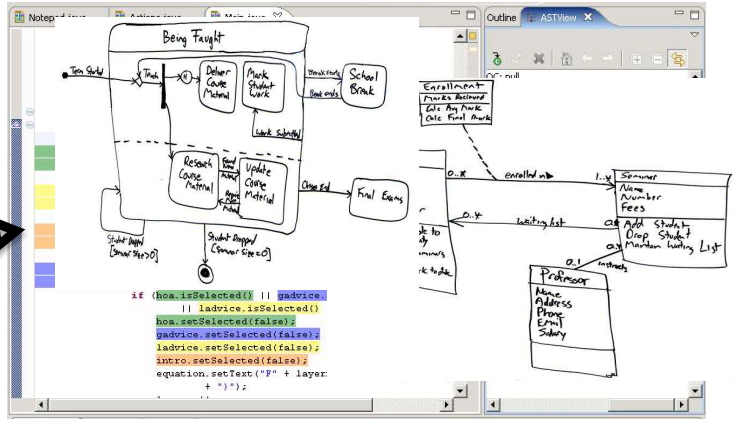

CatalogStructure (1 configuration)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage


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

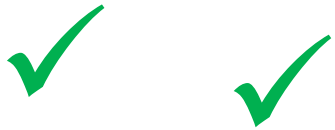


Variability Model

mapping



Base Artefacts (e.g., models)



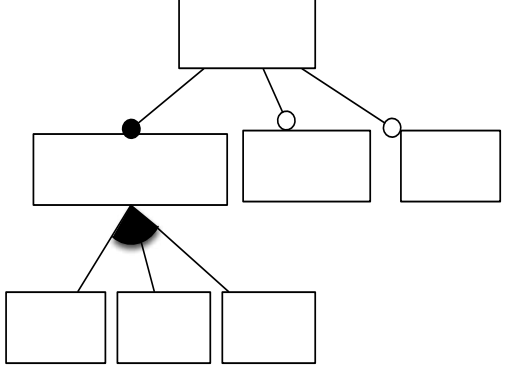
Configuration

Software Generator
(derivation engine)


Unused flexibility

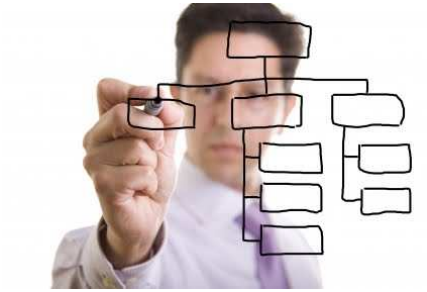

Illegal variant

Feature Model



not, and, or, implies

Communicative

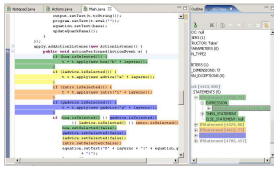


Analytic

Generative

Feature Models

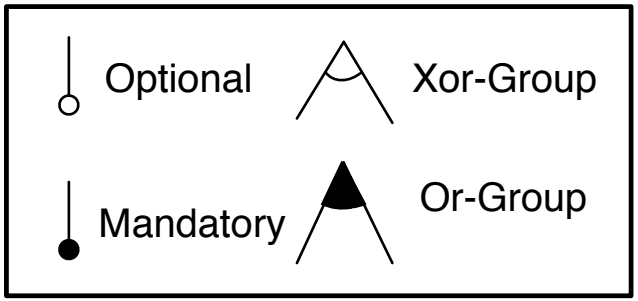
(defacto standard for modeling variability)



Hierarchy: rooted tree

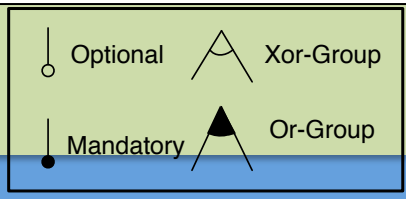
Variability:

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



SPLC12Scripts Model | SPLC12Scripts.config

- CarEquipment (valid, 1 possible configurations)
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights



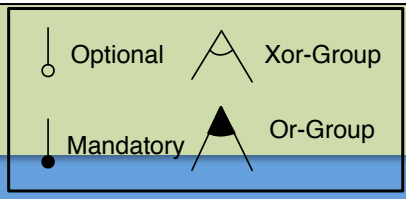
Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

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

SPLC12Scripts Model | SPLC12Scripts.config

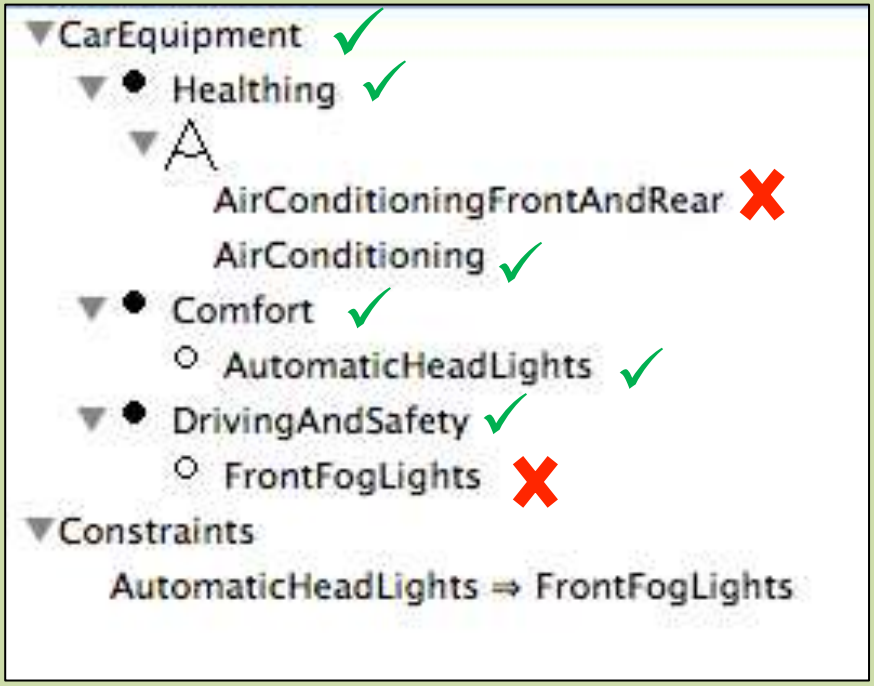
- CarEquipment (valid, 1 possible configurations)
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights

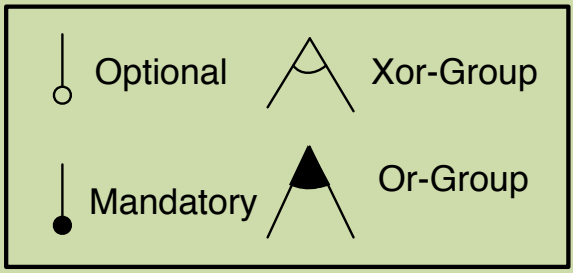


Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning}

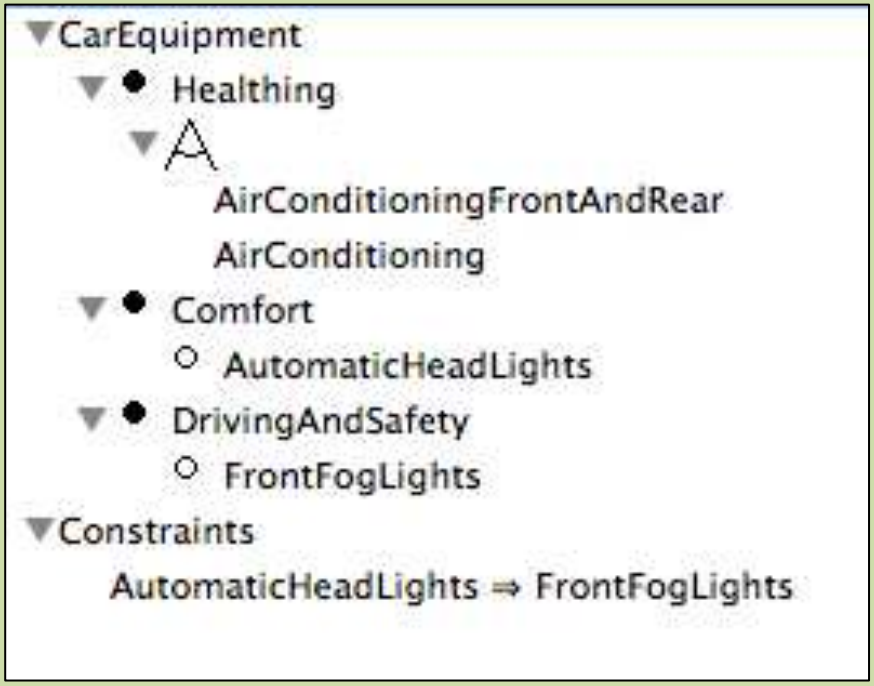


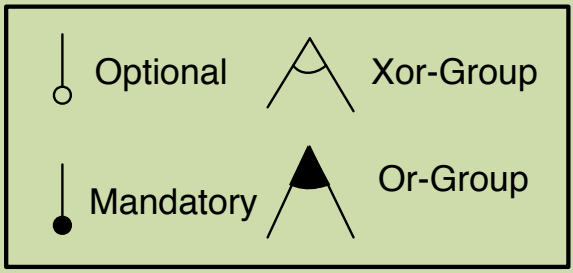


Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

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



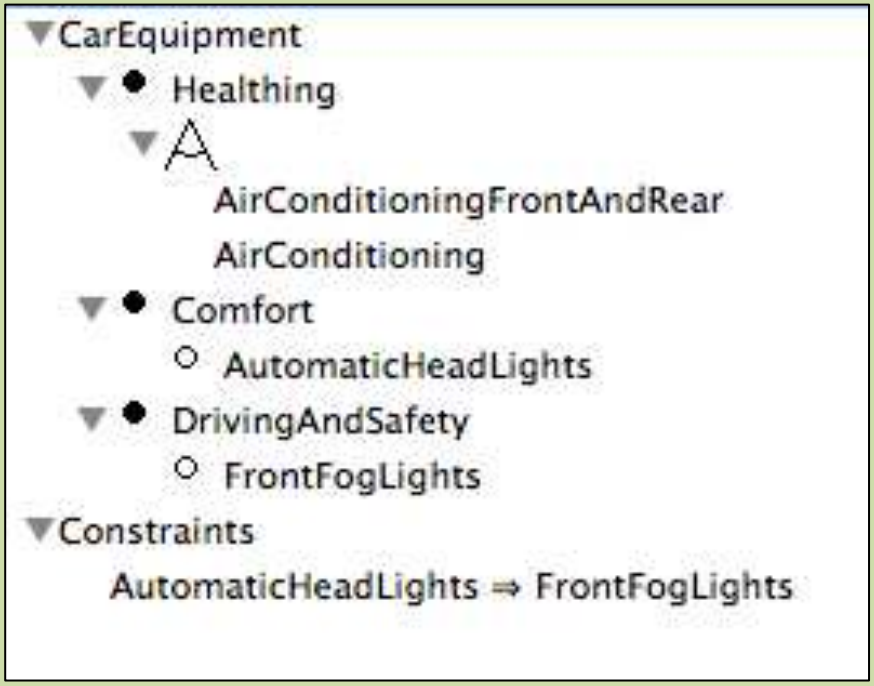


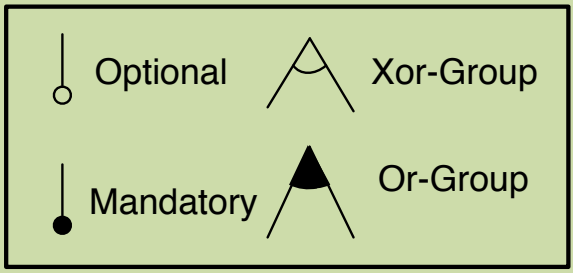
Hierarchy + Variability
=
set of valid configurations

{CarEquipment, Comfort, DrivingAndSafety, Healthing}

X

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

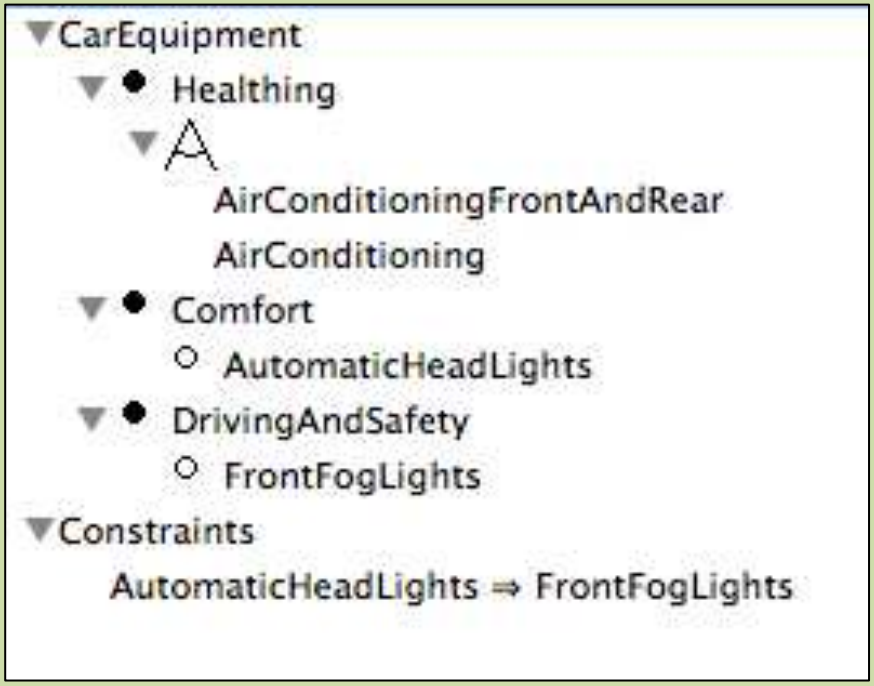


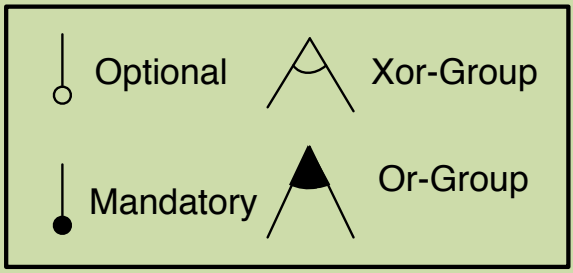


Hierarchy + Variability
=
set of valid configurations

Configuration set (from a basic feature model of car)

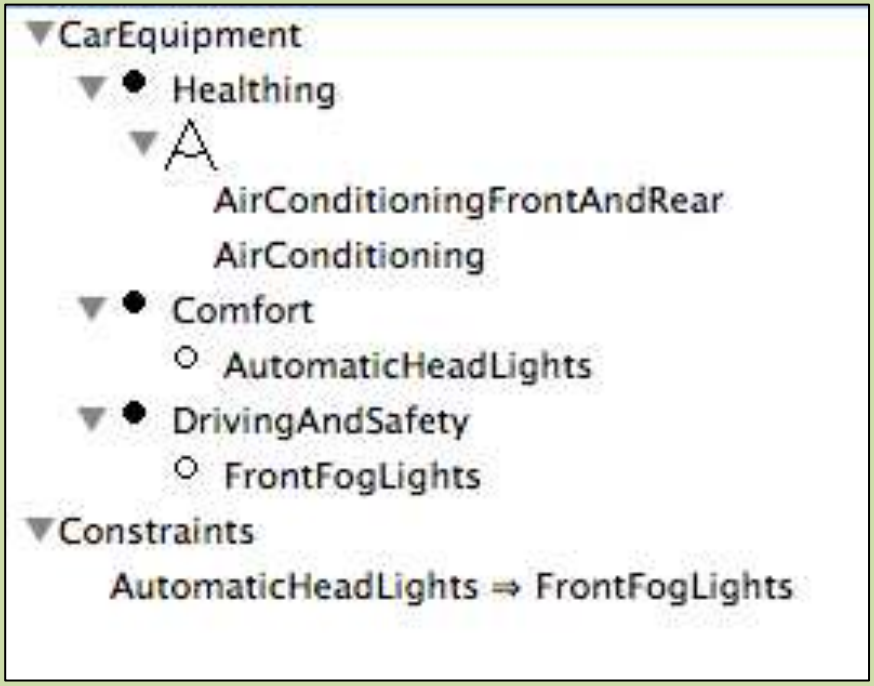
	CarEquipment	Comfort	DrivingAndSafety	Healting	AirConditioning	FrontFogLights	AutomaticHeadLights	AirConditioningFrontAndRear
Car2	yes	yes	yes	yes	yes	yes	yes	no
Car6	yes	yes	yes	yes	no	yes	no	yes
Car1	yes	yes	yes	yes	yes	yes	no	no
Car4	yes	yes	yes	yes	no	no	no	yes
Car5	yes	yes	yes	yes	yes	no	no	no
Car3	yes	yes	yes	yes	no	yes	yes	yes

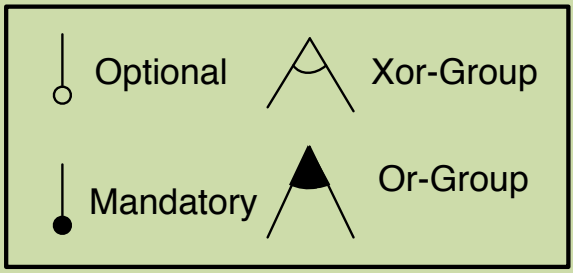




Hierarchy + Variability
=
set of valid configurations

Product	CarEquipment	Comfort	DrivingAndSafety	Healting	AirConditioning	FrontFogLights	AutomaticHeadLights	AirConditioningFrontAndRear
Find	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No
Car1	yes	yes	yes	yes	yes	yes	no	no
Car2	yes	yes	yes	yes	yes	yes	yes	no
Car3	yes	yes	yes	yes	no	yes	yes	yes
Car4	yes	yes	yes	yes	no	no	no	yes
Car5	yes	yes	yes	yes	yes	no	no	no
Car6	yes	yes	yes	yes	no	yes	no	yes



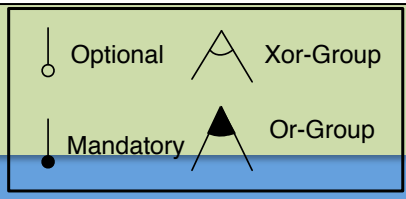


Hierarchy + Variability
=
set of valid configurations

Product					AirConditioning	FrontFogLights	AutomaticHeadLights	AirConditioningFrontAndRear
Find					Yes No	Yes No	Yes No	Yes No
Car1					yes	yes	no	no
Car2					yes	yes	yes	no
Car3					no	yes	yes	yes
Car4					no	no	no	yes
Car5					yes	no	no	no
Car6					no	yes	no	yes

SPLC12Scripts Model | SPLC12Scripts.config

- CarEquipment (valid, 1 possible configurations)
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights

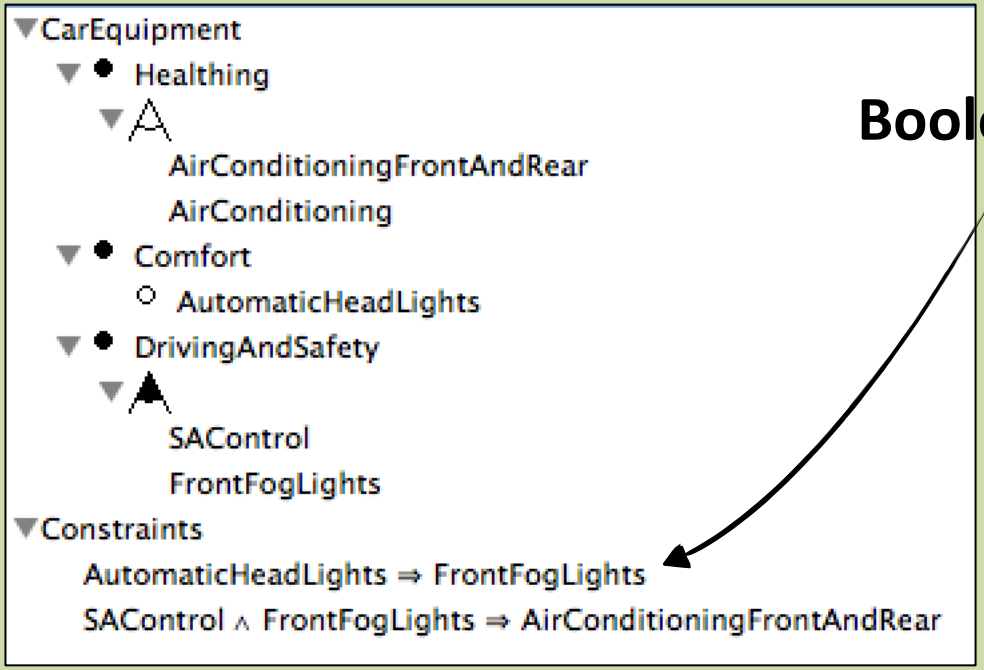


Hierarchy + Variability
=
set of valid configurations

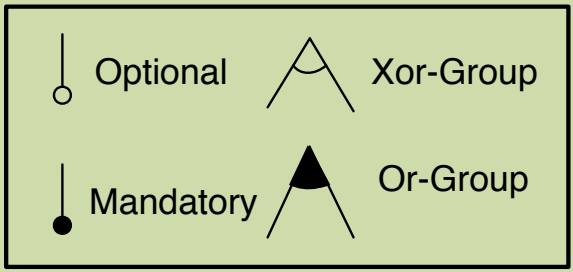
configuration = set of features selected

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning}

Product	AirConditioning	FrontFogLights	AutomaticHeadLights	AirConditioningFrontAndRear
Find	Yes ☑ No ☐	Yes ☐ No ☑	Yes ☐ No ☑	Yes ☐ No ☑
Car5	yes	no	no	no



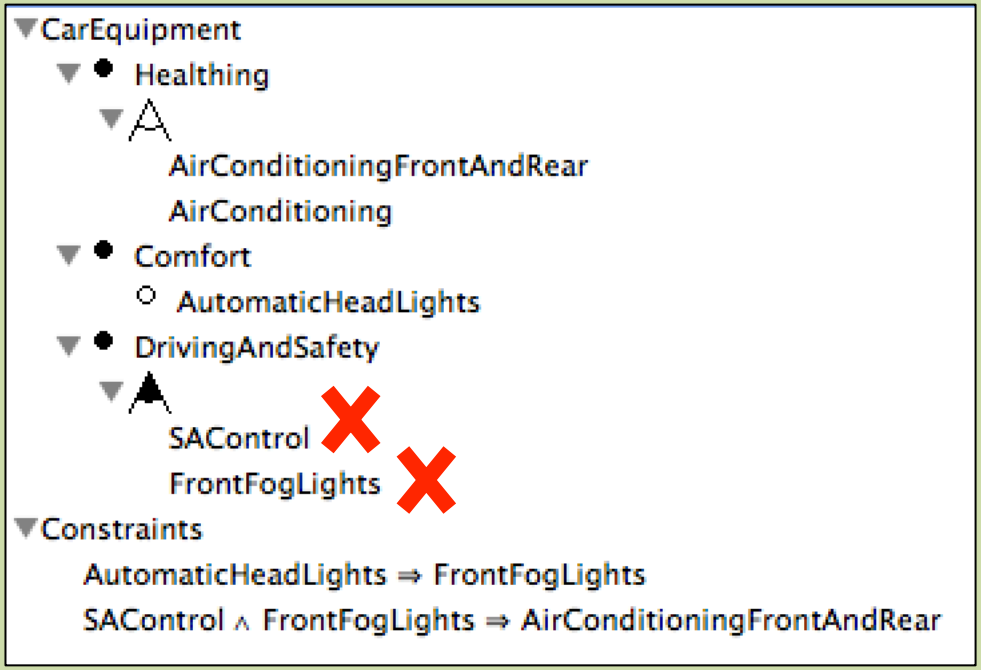
Boolean logic: ^, v, not, implies

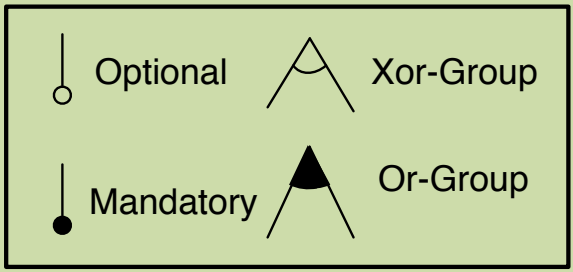


Hierarchy + Variability

=

set of valid configurations



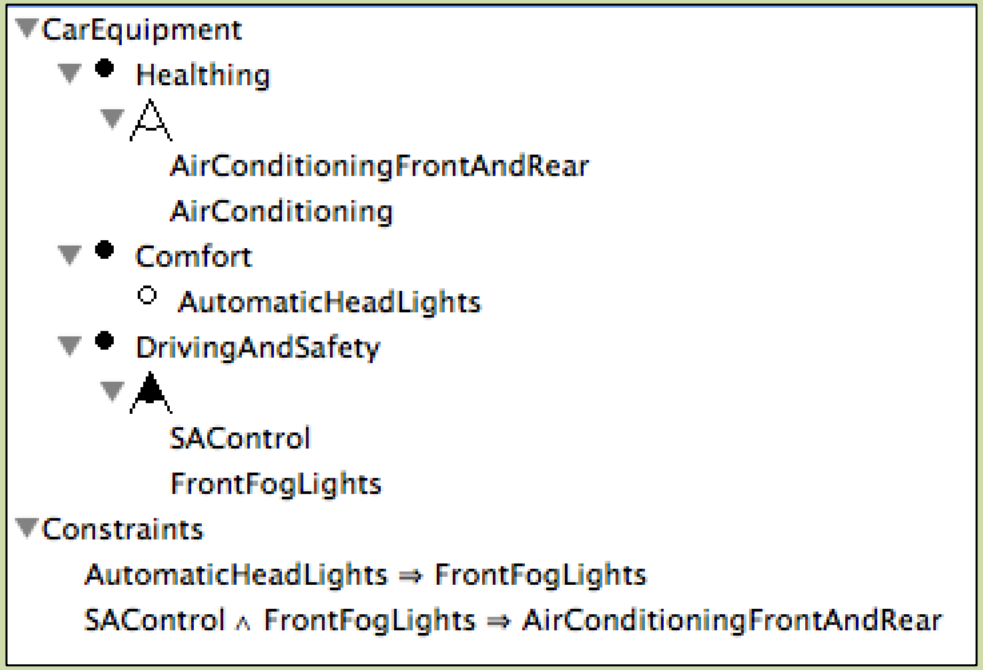


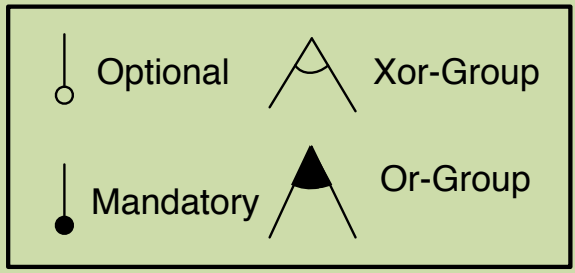
Hierarchy + Variability

=

set of valid configurations

Or-group: at least one!



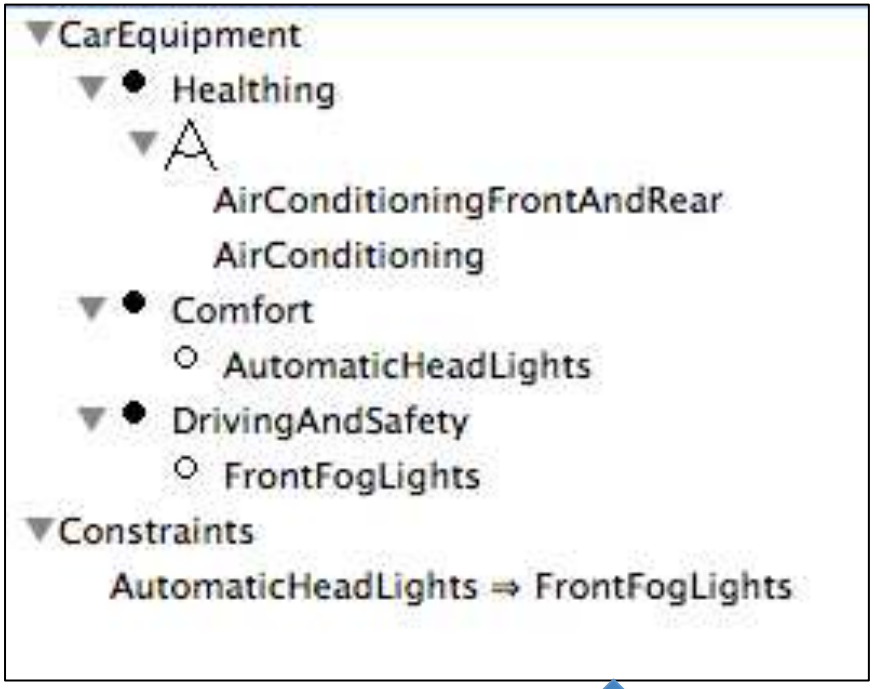


Hierarchy + Variability
=
set of valid configurations

{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}



(Boolean)
Feature Models

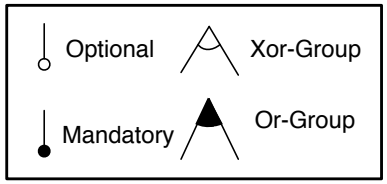
(Boolean)
Formula φ

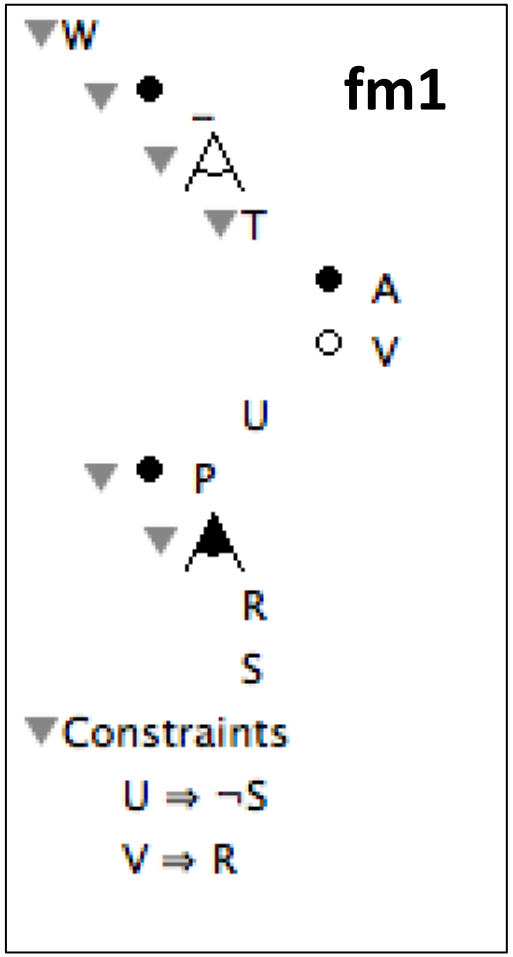
Product	CarEquipment	Comfort	DrivingAndSafety	Healting	AirConditioning	FrontFogLights	AutomaticHeadLights	AirConditioningFrontAndRear
Find	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No
Car1	yes	yes	yes	yes	yes	yes	no	no
Car2	yes	yes	yes	yes	yes	yes	yes	no
Car3	yes	yes	yes	yes	no	yes	yes	yes
Car4	yes	yes	yes	yes	no	no	no	yes
Car5	yes	yes	yes	yes	yes	no	no	no
Car6	yes	yes	yes	yes	no	yes	no	yes

(Boolean)
Product Comparison Matrix

(Boolean) Feature Models

Hierarchy + Variability = set of valid configurations

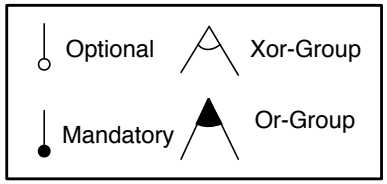


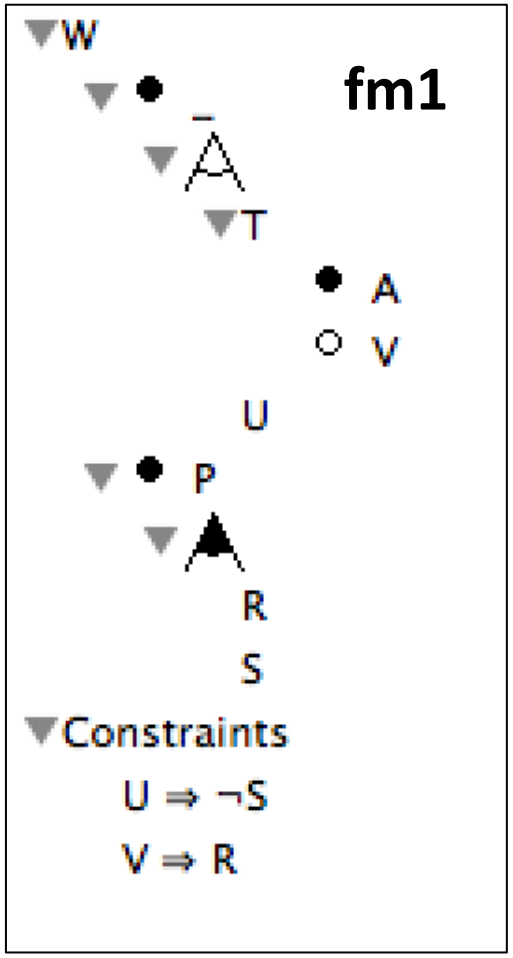


$$[\![fm1]\!] = \{$$
$$\{W, P, R, S, T, A, V\},$$
$$\{W, P, S, T, A\},$$
$$\{W, P, R, T, A\},$$
$$\{W, P, R, U\},$$
$$\{W, P, R, T, V, A\},$$
$$\{W, P, R, S, T, A\},$$
$$\}$$

(Boolean) Feature Models

~ Boolean formula



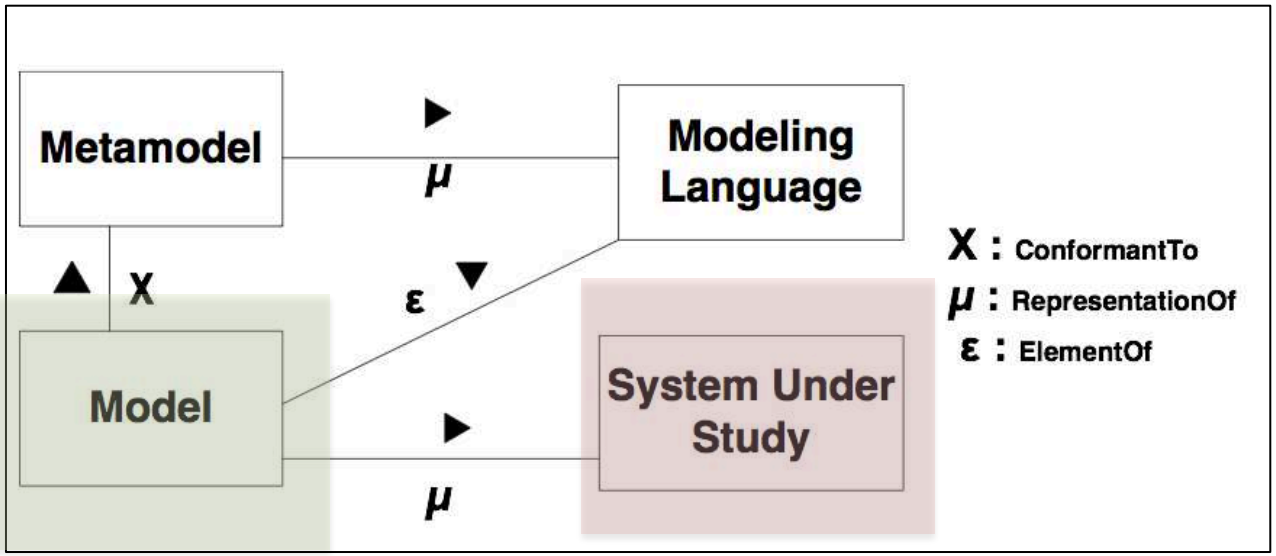


$$
\begin{aligned}
&\phi_{fm_1} = W \;//\; \text{root} \\
&\wedge\, W \Leftrightarrow P \;//\; \text{mandatory} \\
&//\; \text{Or-group} \\
&\wedge\, P \Rightarrow R \vee S \\
&\wedge\, R \Rightarrow P \wedge\, S \Rightarrow P \\
&\wedge\, V \Rightarrow T \;//\; \text{optional} \\
&\wedge\, A \Leftrightarrow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

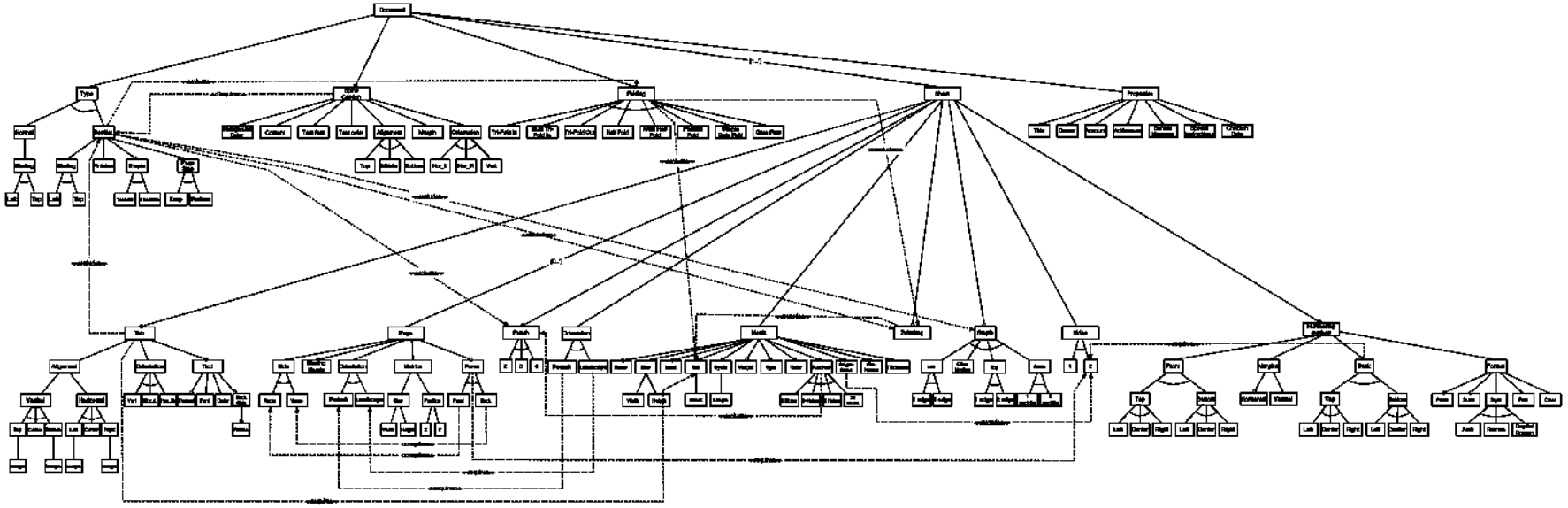
Languages:

How to specify feature models?

Feature Models



- CarEquipment
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights
- Constraints
  - AutomaticHeadLights ⇒ FrontFogLights



FAMiliAR

(FeAture Model scrIpt Language for manIpulation and Automatic Reasoning)

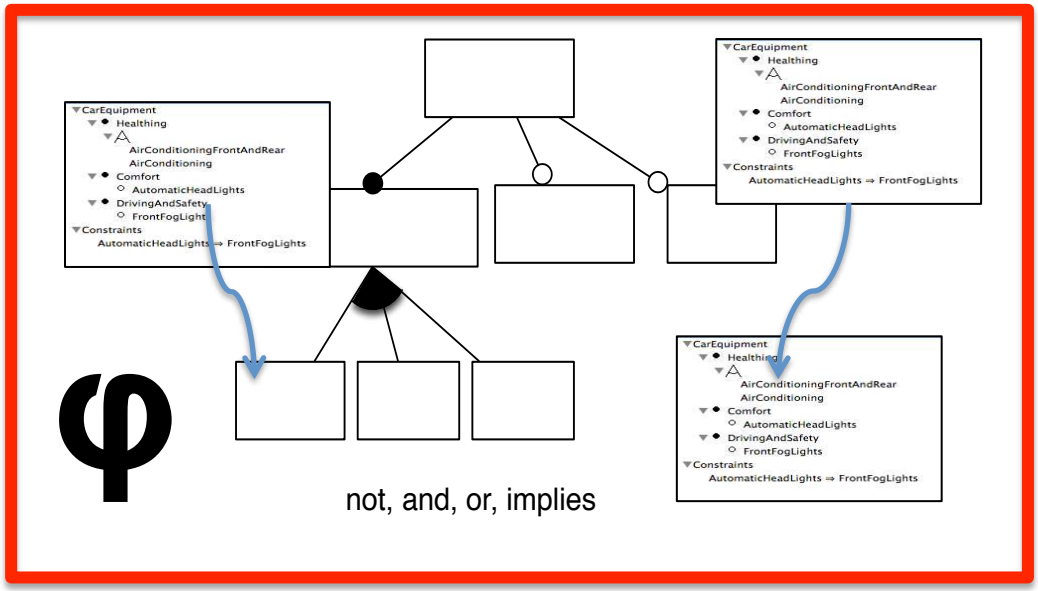
http://familiar-project.github.com/

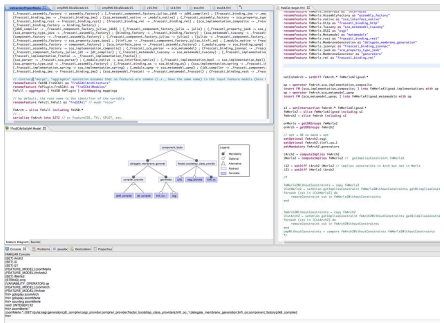
COMMON VARIABILITY LANGUAGE

S.P.L.O.T. Software Product Lines Online Tools

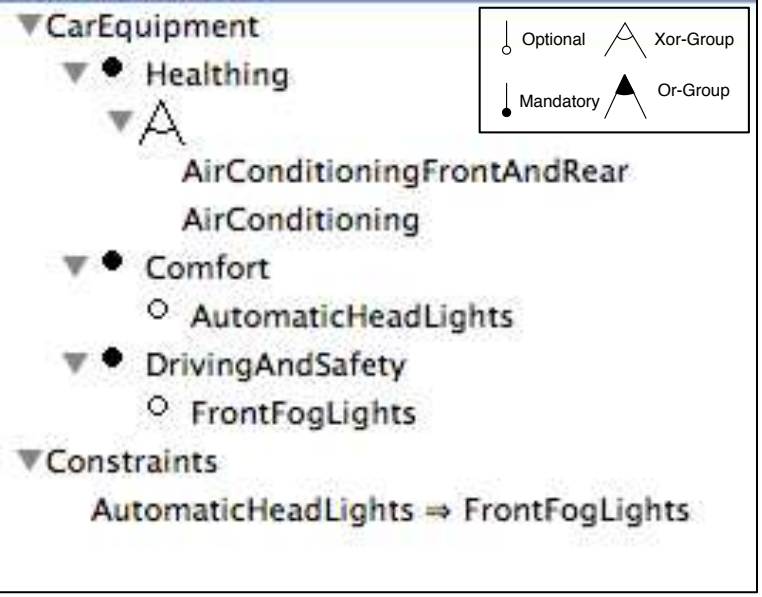
IDE Feature

TVL
DIMACS





Mathieu Acher, Philippe Collet, Philippe Lahire, Robert B. France « A Domain-Specific Language for Large-Scale Management of Feature Models » Science of Computer Programming (SCP), 2013



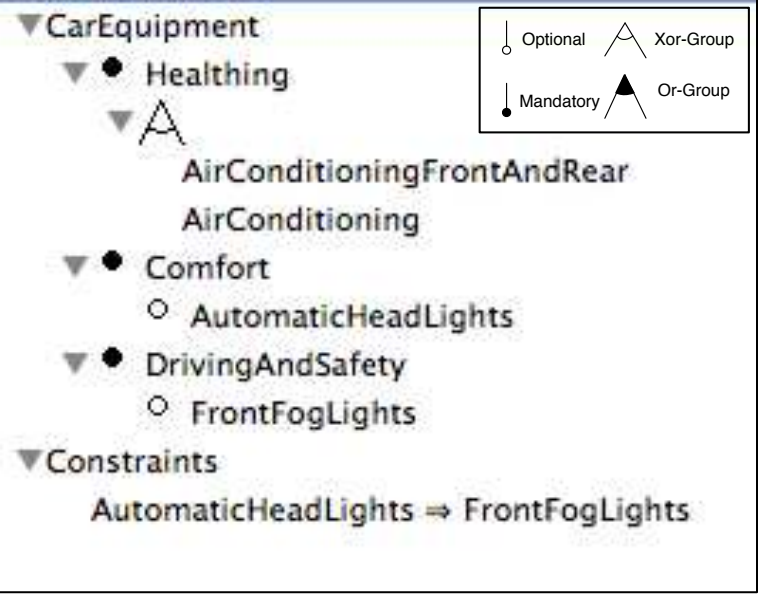

```
fml> convert fmCarEquipment into SPLOT
res1: (STRING) <feature_model name="fmCarEquipment">
<meta>
<data name="description"/>
<data name="creator"/>
<data name="address"/>
<data name="email"/>
<data name="phone"/>
<data name="website"/>
<data name="organization"/>
<data name="department"/>
<data name="date"/>
<data name="reference"/>
</meta>
<feature_tree>
:r CarEquipment(_r0)
        :m Healthing(_r1)
                :g [1,1]
                        : AirConditioningFrontAndRear(_r2)
                        : AirConditioning(_r3)
        :m DrivingAndSafety(_r4)
                :o FrontFogLights(_r5)
        :m Comfort(_r6)
                :o AutomaticHeadLights(_r7)
</feature_tree>
<constraints>
C0:~_r7 or _r5
</constraints>
</feature_model>
```

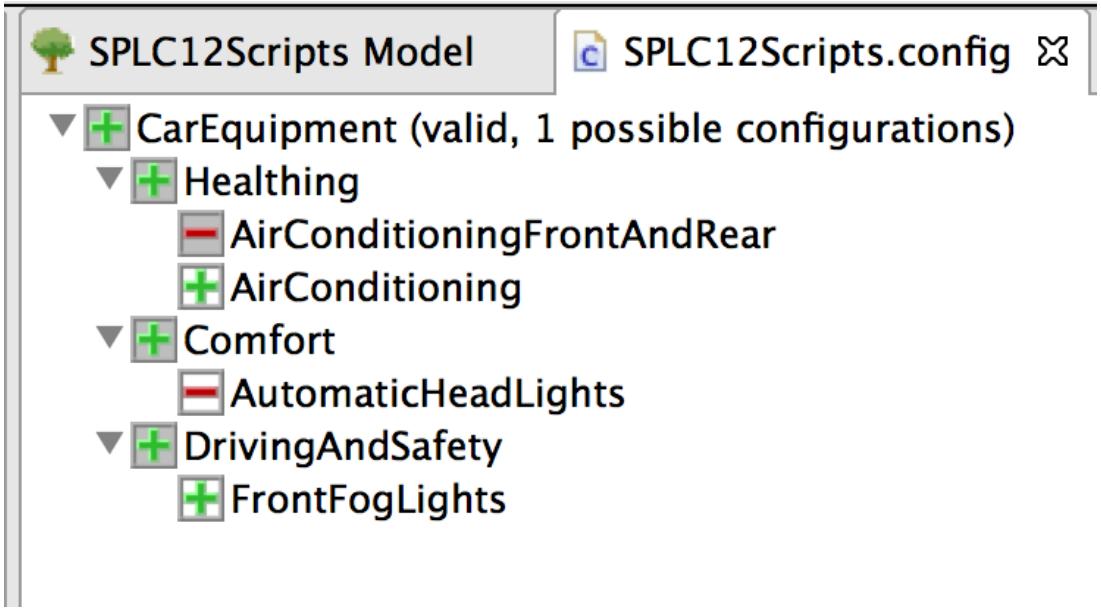
S.P.L.O.T. Software Product Lines Online Tools

IDE eature

```
fmCarEquipment = FM (CarEquipment : Healthing DrivingAndSafety Comfort ; // 3 mandatory features
                     Healthing : (AirConditioning|AirConditioningFrontAndRear) ; // Xor
                     DrivingAndSafety : [FrontFogLights] ; // optional
                     Comfort : [AutomaticHeadLights] ; // optional
                     // cross-tree constraints
                     AutomaticHeadLights -> FrontFogLights ; )
```

```
MacBook-Pro-de-Mathieu-3:Documents macher1$ java -Xmx1024M -jar FML-basic-1.1.jar FMLTestRepository/carEquipTuto.fml
FAMILIAR (for FeAture Model scrIpt Language for manIpulation and Automatic Reasoning)  version 1.1 (beta)
http://familiar-project.github.com/
fml> ls
(FEATURE_MODEL) fmCarEquipment
fml>
```

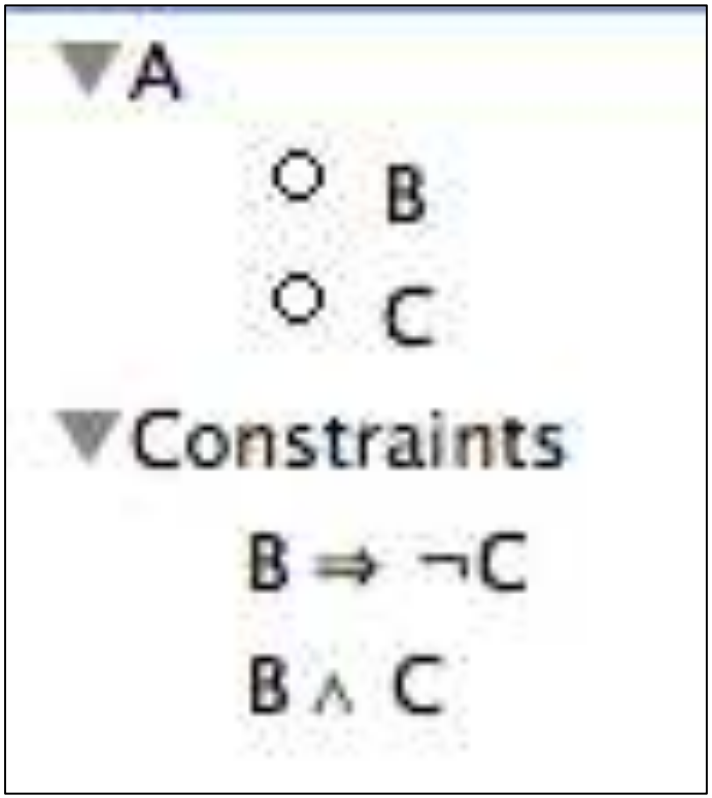




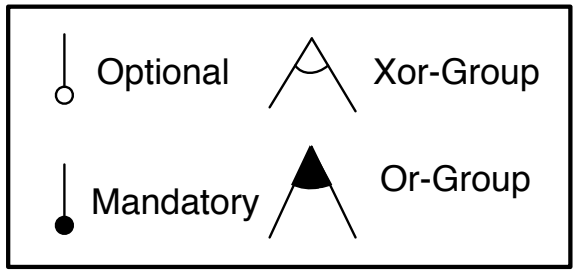

```
fmCarEquipment = FM (CarEquipment : Healthing DrivingAndSafety Comfort ; // 3 mandatory features
                                    Healthing : (AirConditioning|AirConditioningFrontAndRear) ; // Xor
                                    DrivingAndSafety : [FrontFogLights] ; // optional
                                    Comfort : [AutomaticHeadLights] ; // optional
                                    // cross-tree constraints
                                    AutomaticHeadLights -> FrontFogLights ; )
```

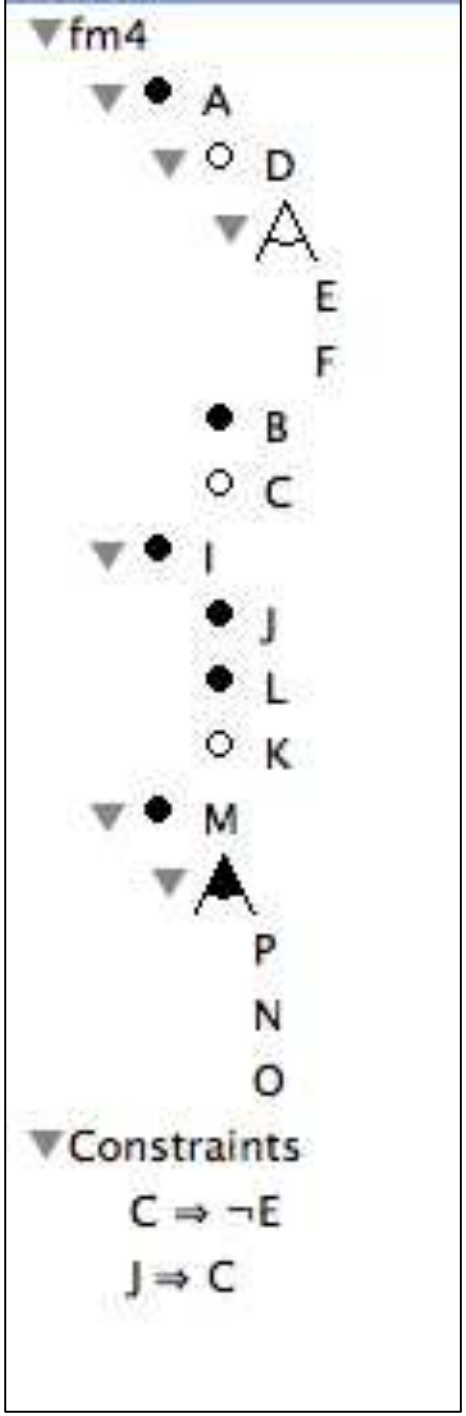
```
fml> c1 = configuration fmCarEquipment
c1: (CONFIGURATION) selected: [Healthing, CarEquipment, DrivingAndSafety, Comfort]       deselected: []
fml> select AirConditioning FrontFogLights in c1
res2: (BOOLEAN) true
fml> deselect AutomaticHeadLights in c1
res3: (BOOLEAN) true
fml> selectedF c1
res4: (SET) {Comfort;CarEquipment;Healthing;AirConditioning;DrivingAndSafety;FrontFogLights}
```

I want to analyze and play with my specification!



Empty set of configurations





Dead feature

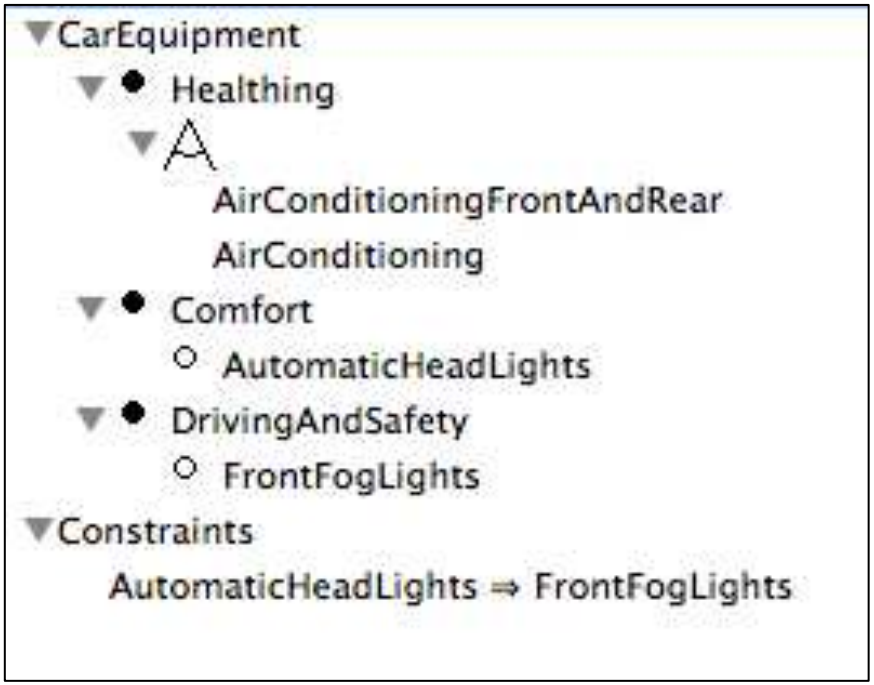
False optional feature

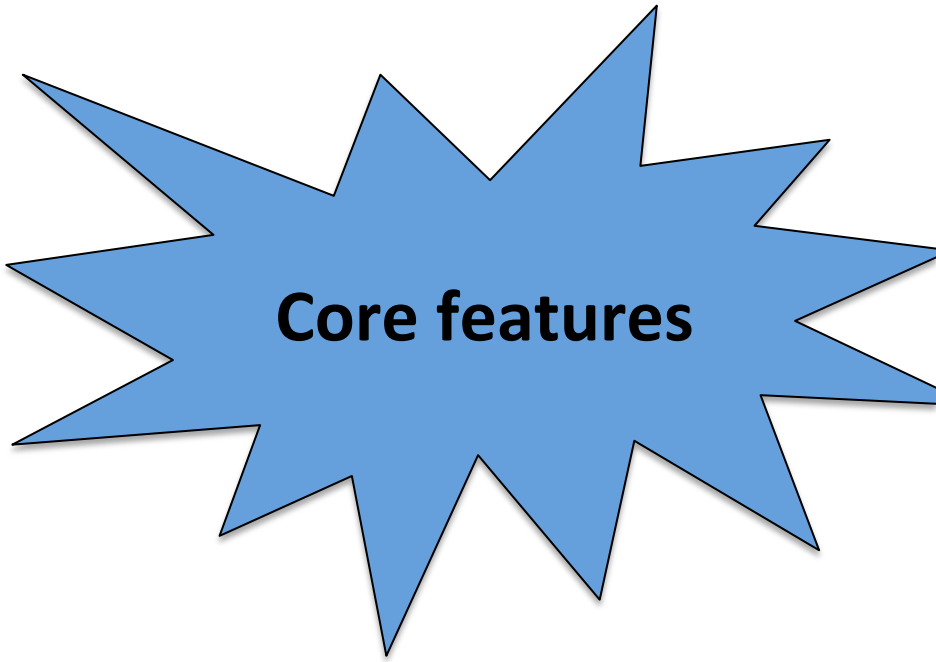
Optional

Xor-Group

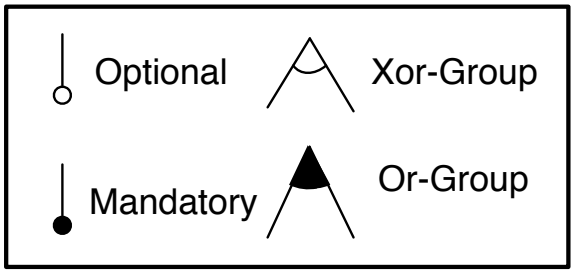
Mandatory

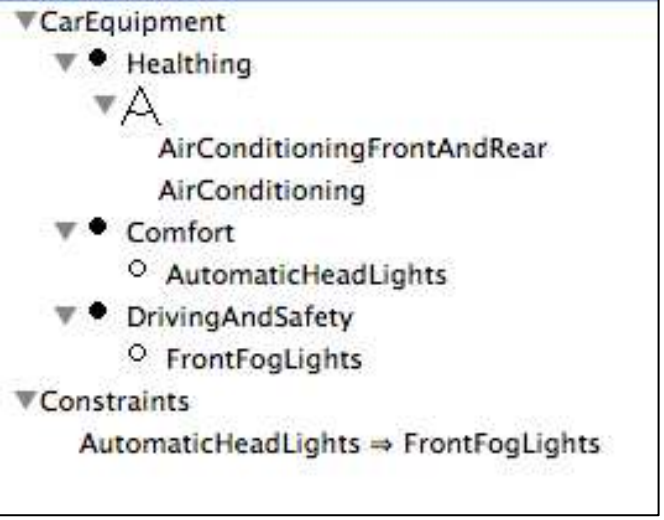
Or-Group

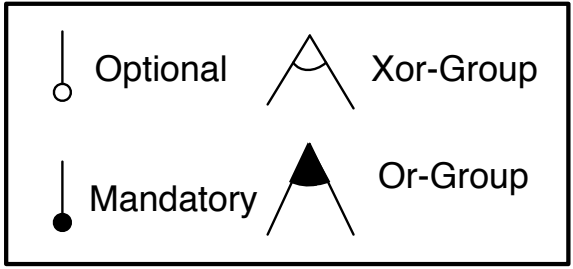


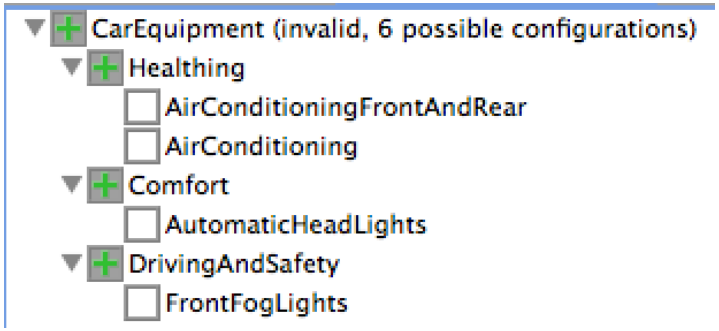


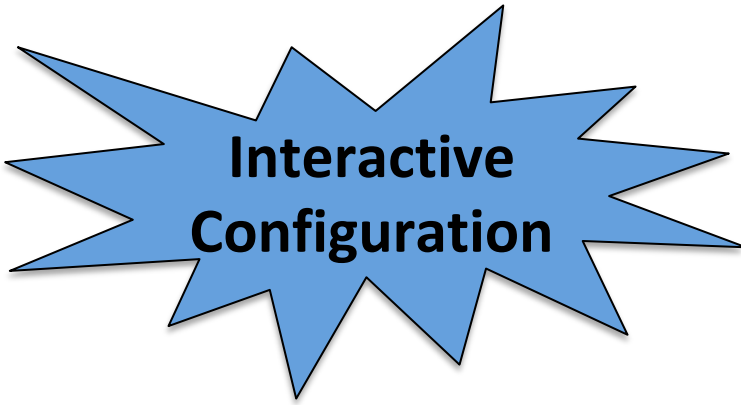
{CarEquipment, Comfort, DrivingAndSafety, Healthing}

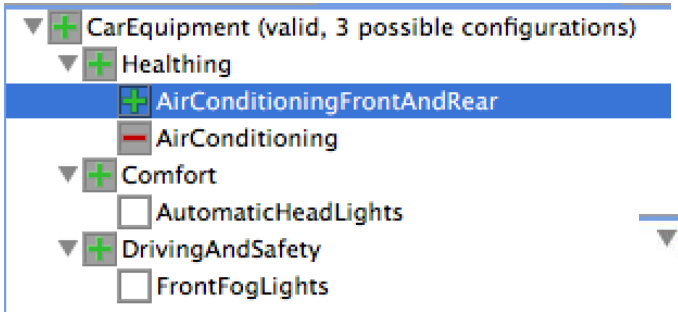


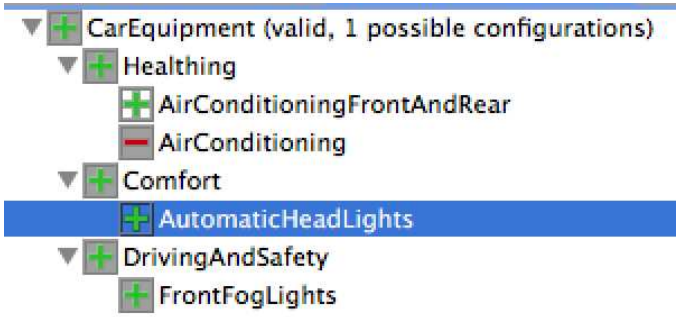

CarEquipment
Healthing
AirConditioningFrontAndRear
AirConditioning
Comfort
AutomaticHeadLights
DrivingAndSafety
FrontFogLights
Constraints
AutomaticHeadLights ⇒ FrontFogLights


Optional
Xor-Group
Mandatory
Or-Group


CarEquipment (invalid, 6 possible configurations)
Healthing
AirConditioningFrontAndRear
AirConditioning
Comfort
AutomaticHeadLights
DrivingAndSafety
FrontFogLights


Interactive
Configuration


CarEquipment (valid, 3 possible configurations)
Healthing
AirConditioningFrontAndRear
AirConditioning
Comfort
AutomaticHeadLights
DrivingAndSafety
FrontFogLights


CarEquipment (valid, 1 possible configurations)
Healthing
AirConditioningFrontAndRear
AirConditioning
Comfort
AutomaticHeadLights
DrivingAndSafety
FrontFogLights

Feature Models and Automated Reasoning

Benavides et al. survey, 2010

	Batory [5]	Czarnecki *et al.* [30]	Gheyi *et al.* [37]	Mannion *et al.* [51, 52]	Mendonca *et al.* [57]	Mendonca *et al.* [56]	Sun *et al.* [74]	Thüm *et al.* [75]	van der Storm [86, 87]	Zhang *et al.* [102, 101]	Zhang *et al.* [103]	Yan *et al.* [100]	Benavides *et al.* [10, 11, 12]	Benavides *et al.* [15]	Djebbi *et al.* [34]	Trinidad *et al.* [78, 76]	White *et al.* [99]	White *et al.* [97]	Abo Zaid *et al.* [1]	Fan *et al.* [35]	Wang *et al.* [92, 93]	Benavides *et al.* [14]	Benavides *et al.* [16]	Segura [70]	Bachmeyer *et al.* [4]	Cao *et al.* [20]	Fernandez *et al.* [36]	Hemakumar [41]	Cheyi *et al.* [38]	Kang *et al.* [43]	Mendonca *et al.* [55]	Osman *et al.* [59, 60]	Salinesi *et al.* [66]	Van den Broek *et al.* [84]	Van Deursen *et al.* [88]	Von der Massen *et al.* [90]	Von der Massen *et al.* [91]	White *et al.* [98, 96]	Batory *et al.* [7]	Schobbens *et al.* [42, 68, 69]	Trinidad *et al.* [80]	Von der Massen *et al.* [89]
	PL												CP						DL			Multi			Others														No support			
Void feature model	+	+		+		+	+		+	+	+	+	+	+	+	+				+	+	+	+	+		+			+	⊕		+	+	+					~	~	
#Products		+		⊕									+	+	+							+	+	+			+					+		+	⊕		+			~	~	
Dead features		~				+				+	+	+				+			+		+		⊕						+	⊕			+	+				~	~		~
Valid product	+	+	+	+			+									+	+				+		⊕		+					⊕			+	+	+			~	~		
All products	+		+	⊕			+						+										+			+								⊕					~	~		
Explanations	+	~					+									+			+		+									⊕		+		+	⊕				~	~		
Refactoring			+				⊕	+													+							+											~	~		
Optimization													⊕		+																	+							~	~		
Commonality													⊕			+							+				+												~	~		
Filter		+											⊕		+																			+					~	~		
Valid partial configuration	+	+							+																					⊕								~	~		
Atomic sets					+					⊕														+							+											
False optional features										⊕	+					+																								~	⊖	
Corrective explanations																	+															+								⊖	
Dependency analysis																														⊕	+										
ECR					⊕	+																																				
Generalization			⊕					+																																		
Core features						+																																				
Variability factor													⊕																											~		
Arbitrary edit								+																																		
Conditional dead features																												+														
Homogeneity																											+															
LCA					+																																					
Muti-step configuration																		+																								
Roots features					+																																					
Specialization								+																																		
Degree of orthogonality		~																																								
Redundancies																																									~	
Variant features																																								~	
Wrong cardinalities																																								~	
Feature model notation	B	C	B	B	B	B	B	B	B	B	C	B	B	C	C	B	B	B	B	B	B	B	C	B	B	B	C	B	B	B	C	B	B	B	B	B	B	C	C	B
Extended feature model													+		+		+	+	+										+		+						+	+		+		
Formalization			+		+	+	+	+		+	+		+			+		+		+					+		+		+		+				+			+			

+ Supported ~ No support ⊕ Supported(first reference) ⊖ No support (first reference) B Basic feature model C Cardinality–based feature models

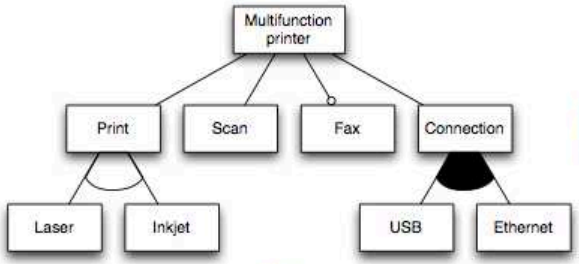
Decision problems and complexity

- Validity of a feature model
- Validity of a configuration
- Computation of dead and core features
- Counting of the number of valid configurations
- Equivalence between two feature models
- Satisfiability (SAT) problem
  - NP-complete

How to automate analysis of your feature models?

Binary Decision Diagram (BDD)
SAT solver

Typical implementations

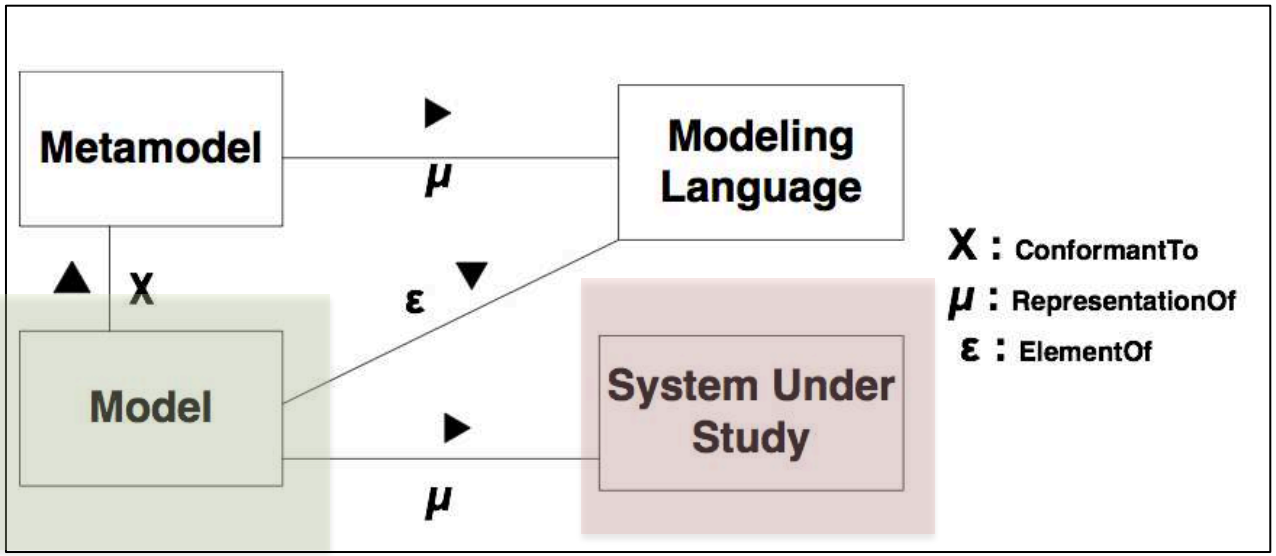


result

logics

solvers

Feature Models

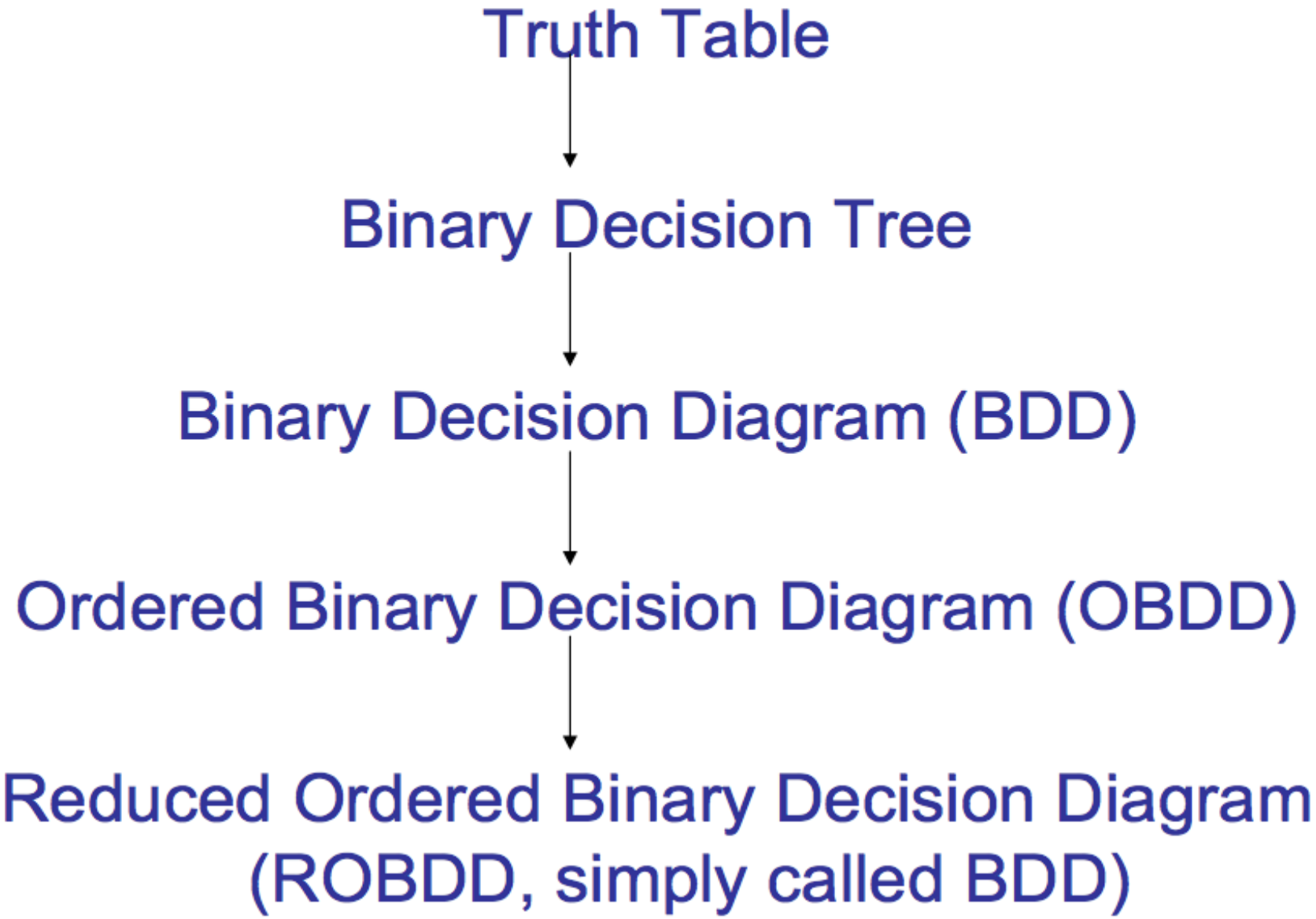


- CarEquipment
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights
- Constraints
  - AutomaticHeadLights ⇒ FrontFogLights

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

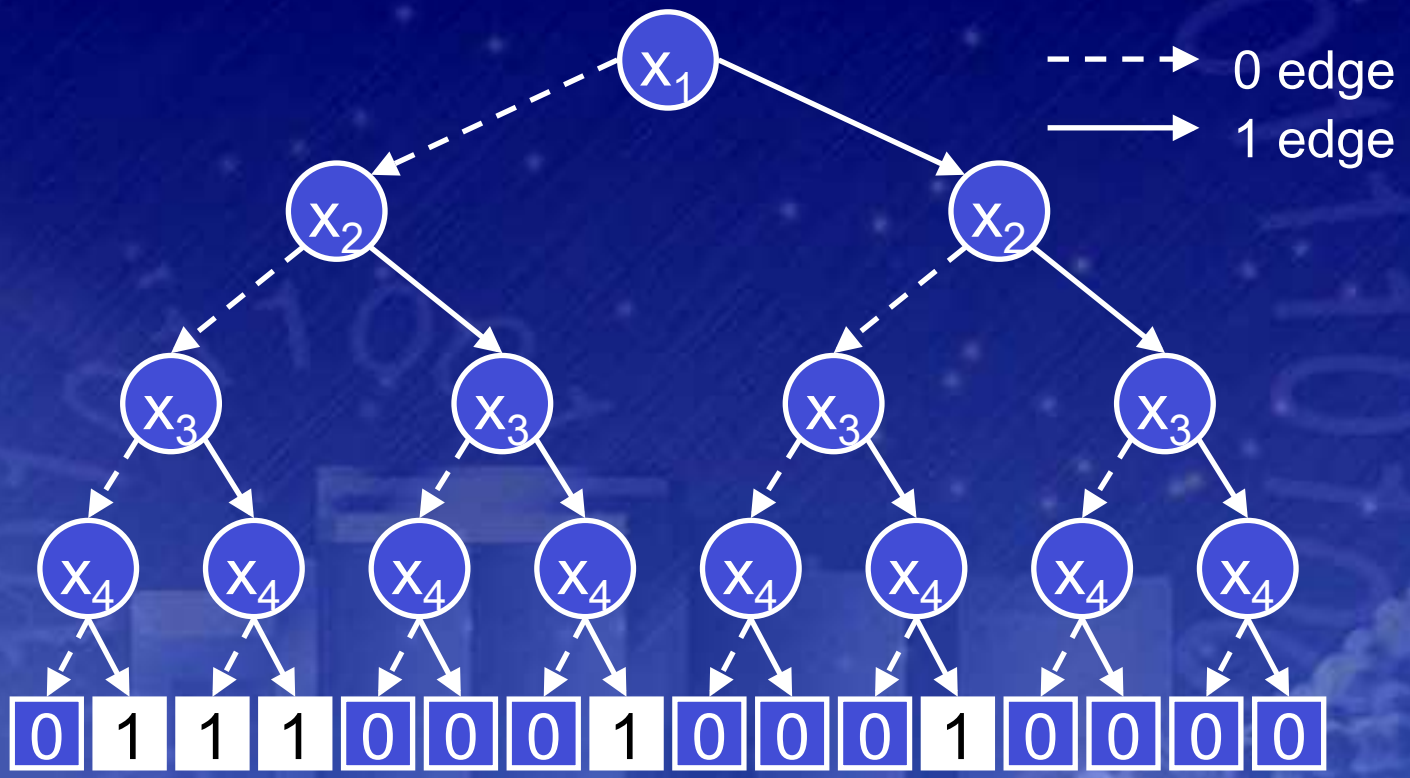
BDDs from Truth Tables



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

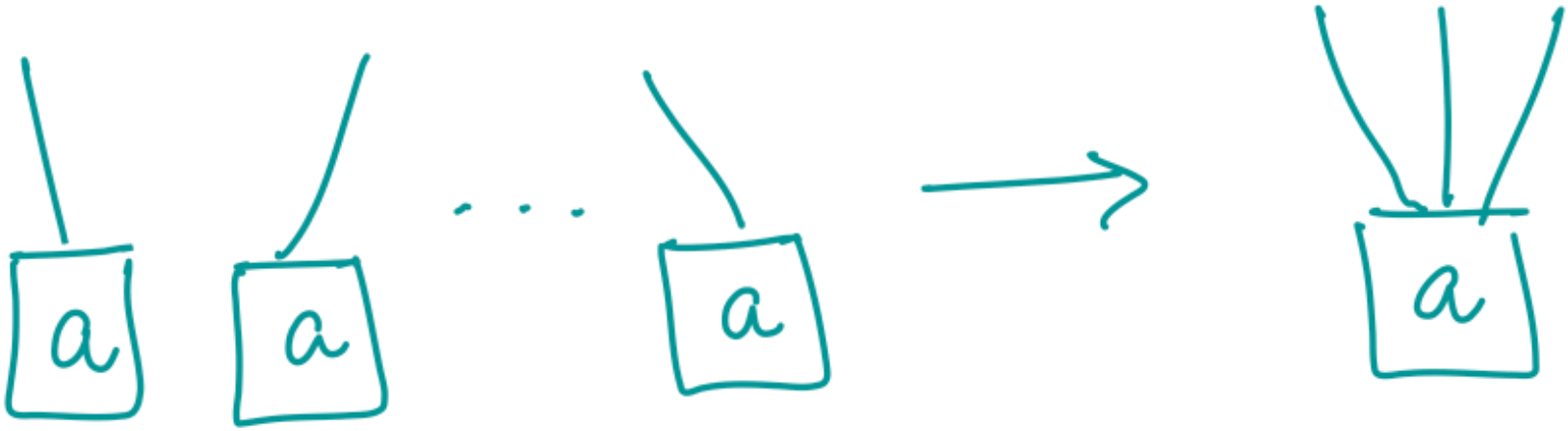


Reduction

- Identify Redundancies

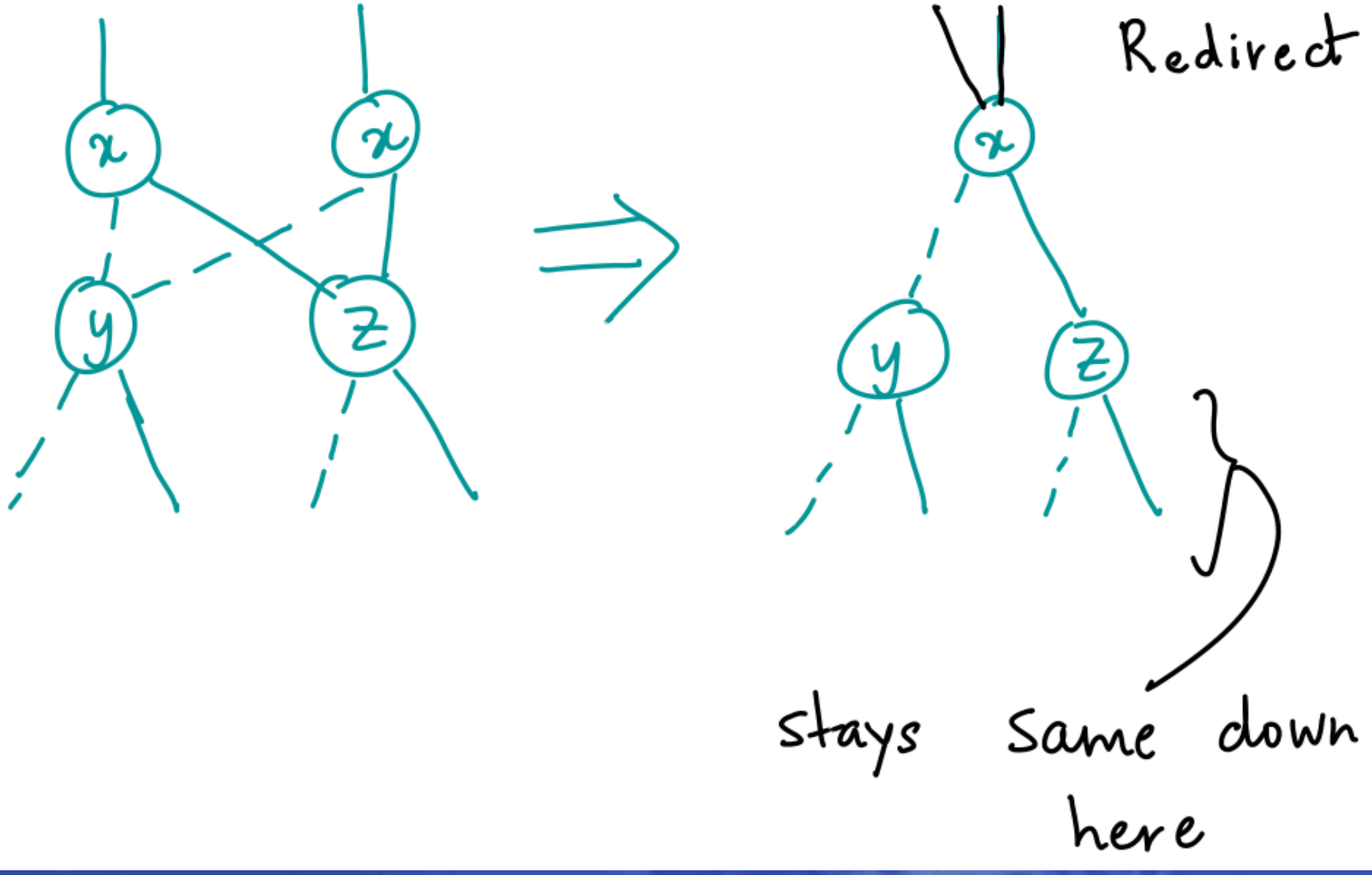
- 3 Rules
  - Merge equivalent leaves
  - Merge isomorphic nodes
  - Eliminate redundant tests

Merge equivalent leaves

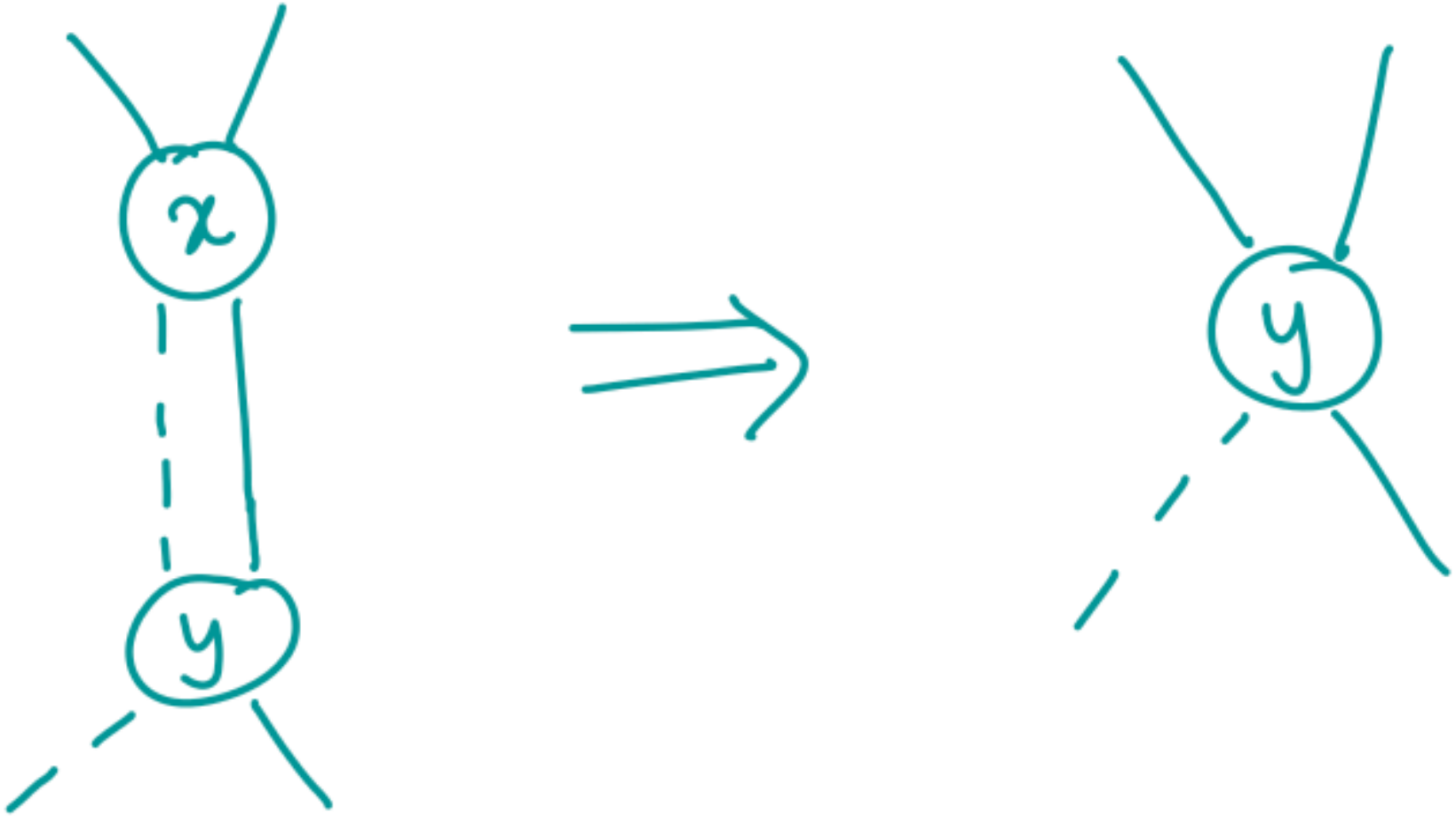


"a" is either 0 or 1

Merge isomorphic nodes

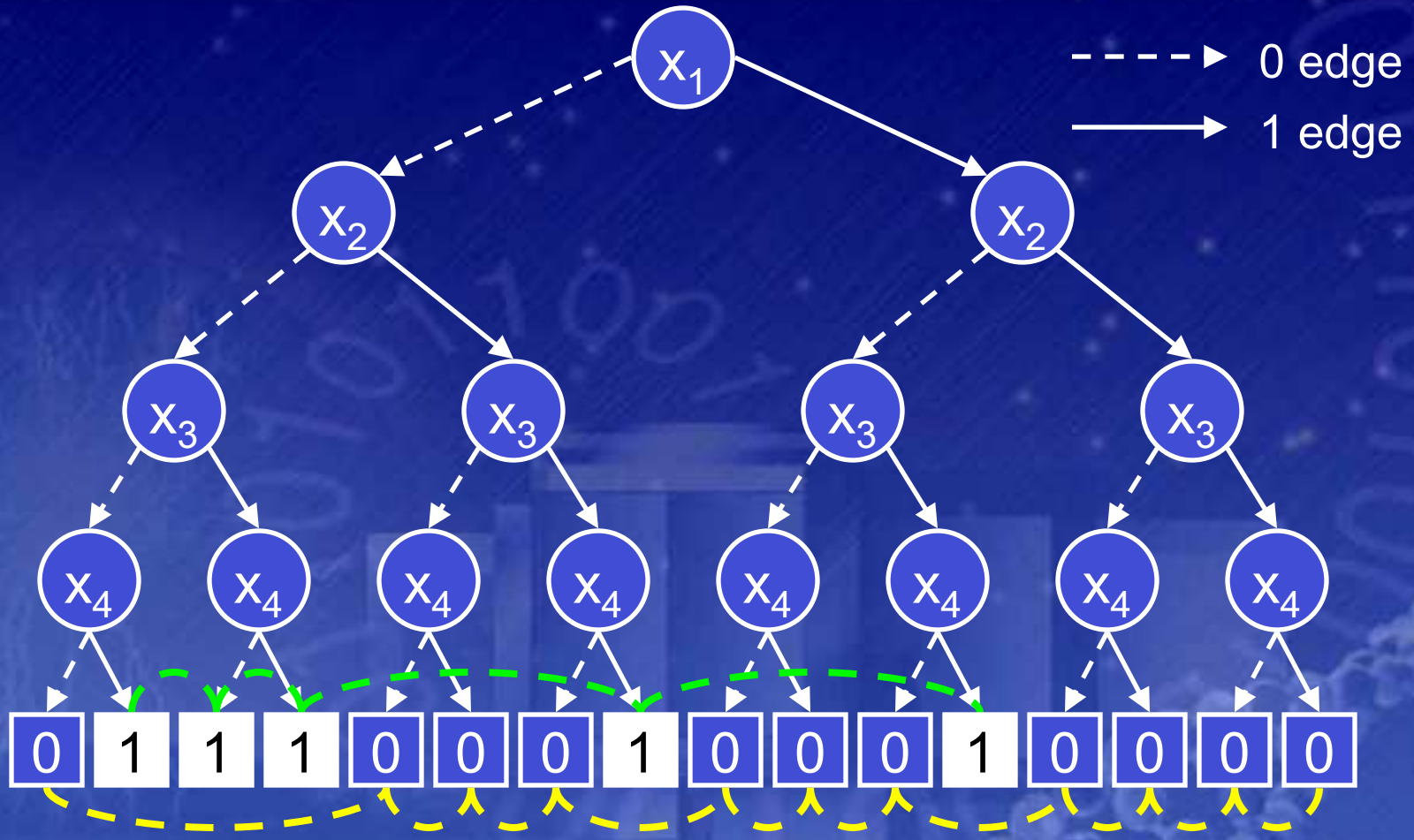


Eliminate redundant tests



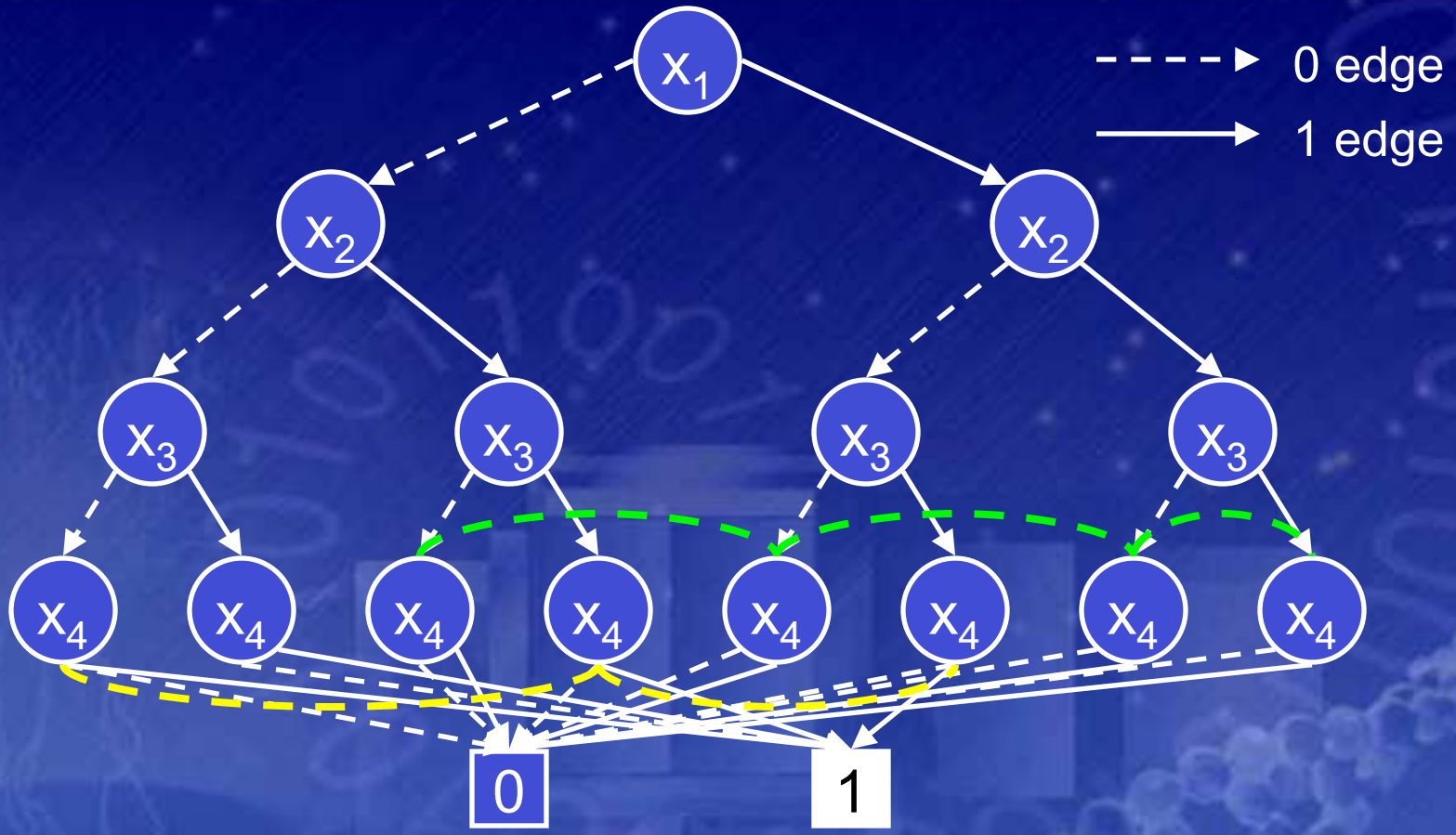
Binary Decision Diagrams

- Collapse redundant nodes.



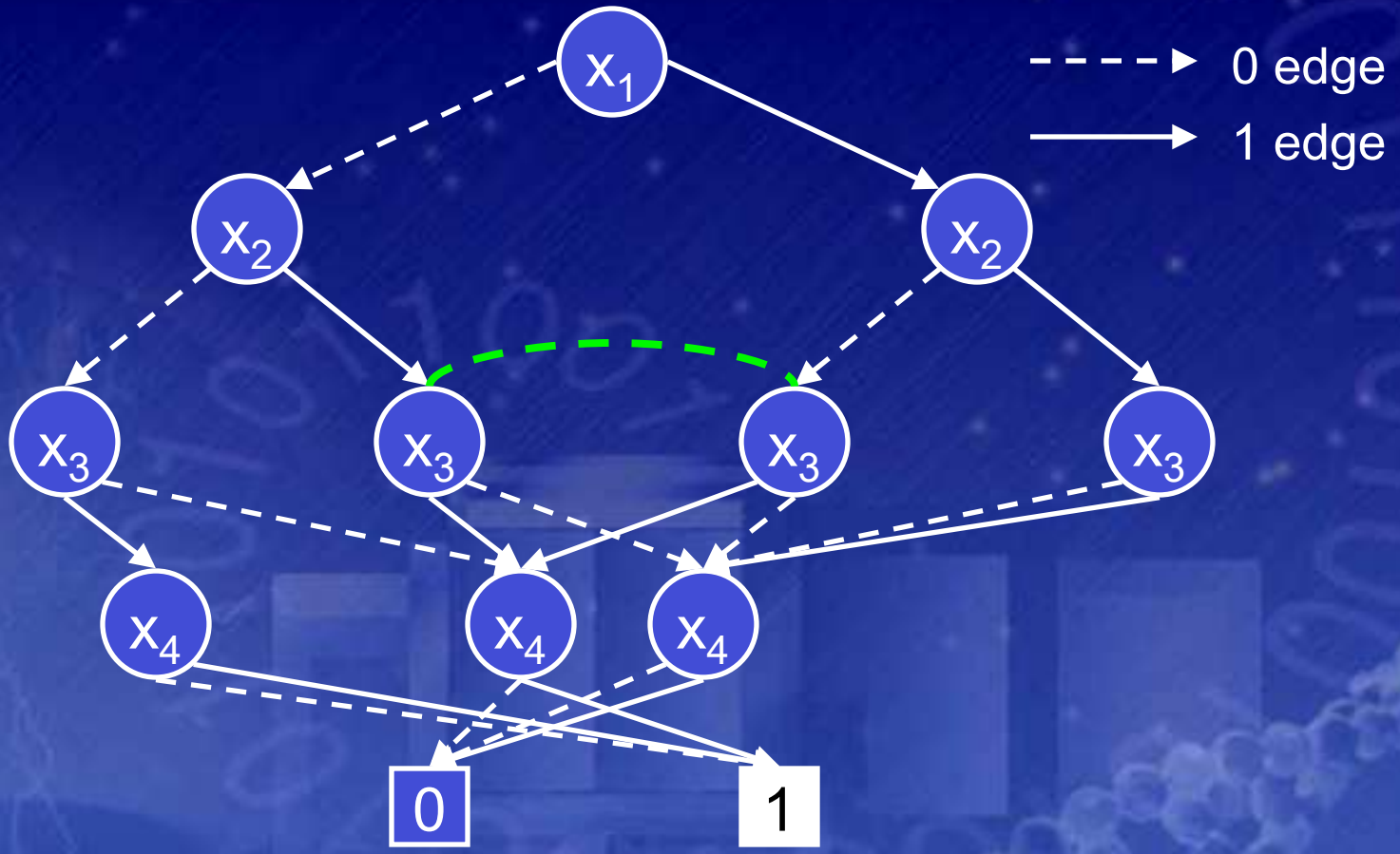
Binary Decision Diagrams

- Collapse redundant nodes.



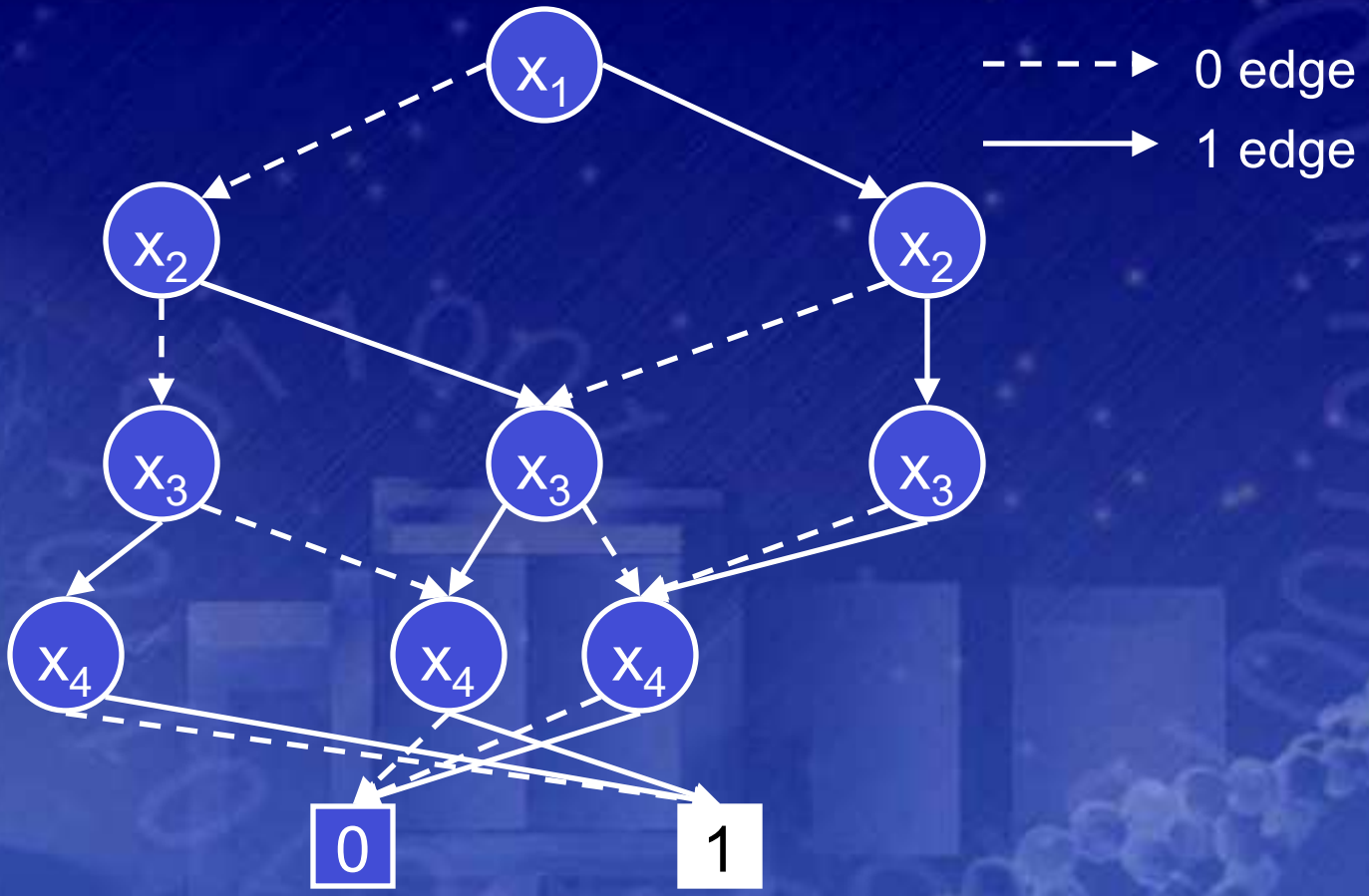
Binary Decision Diagrams

- Collapse redundant nodes.



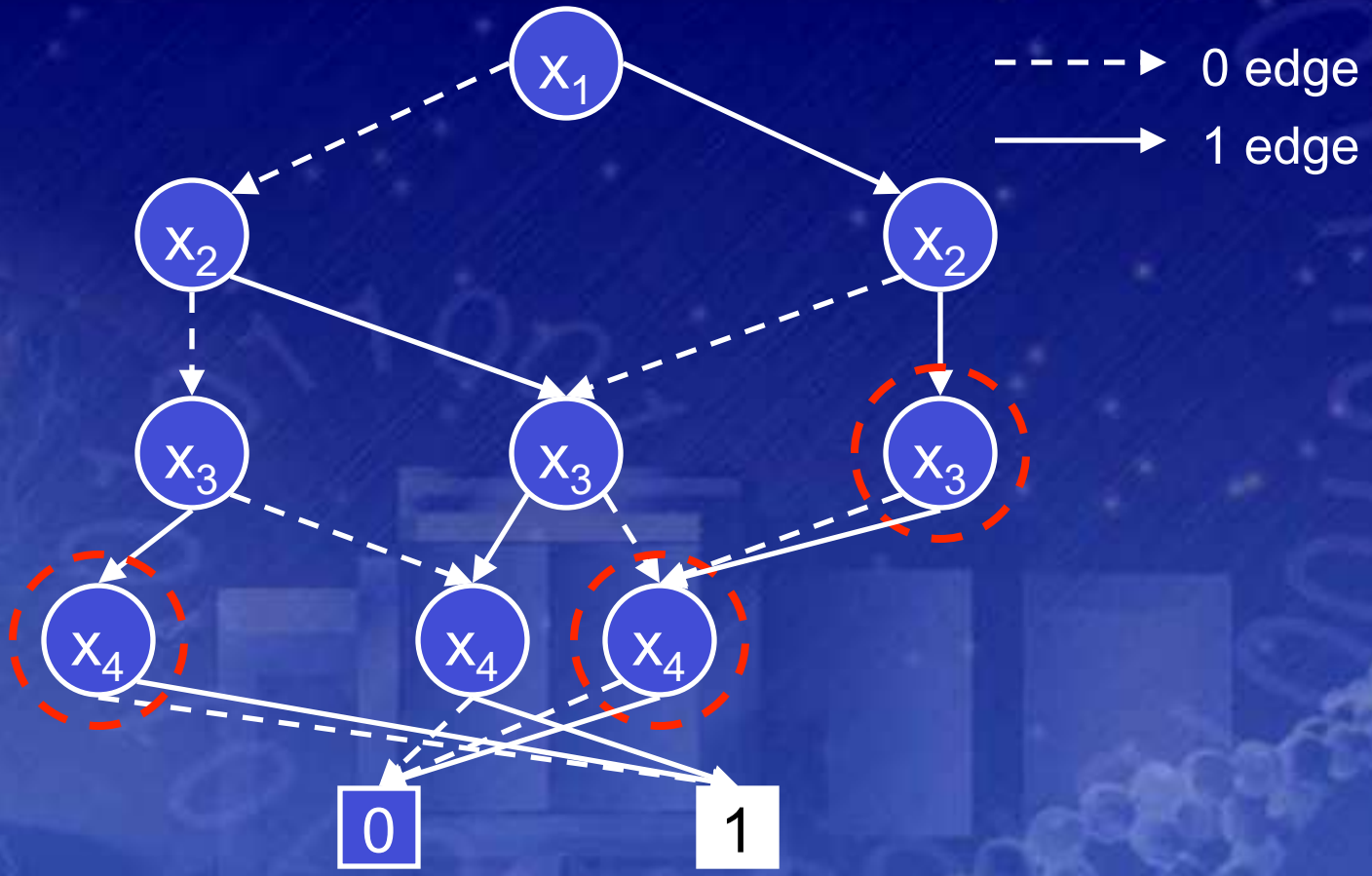
Binary Decision Diagrams

- Collapse redundant nodes.



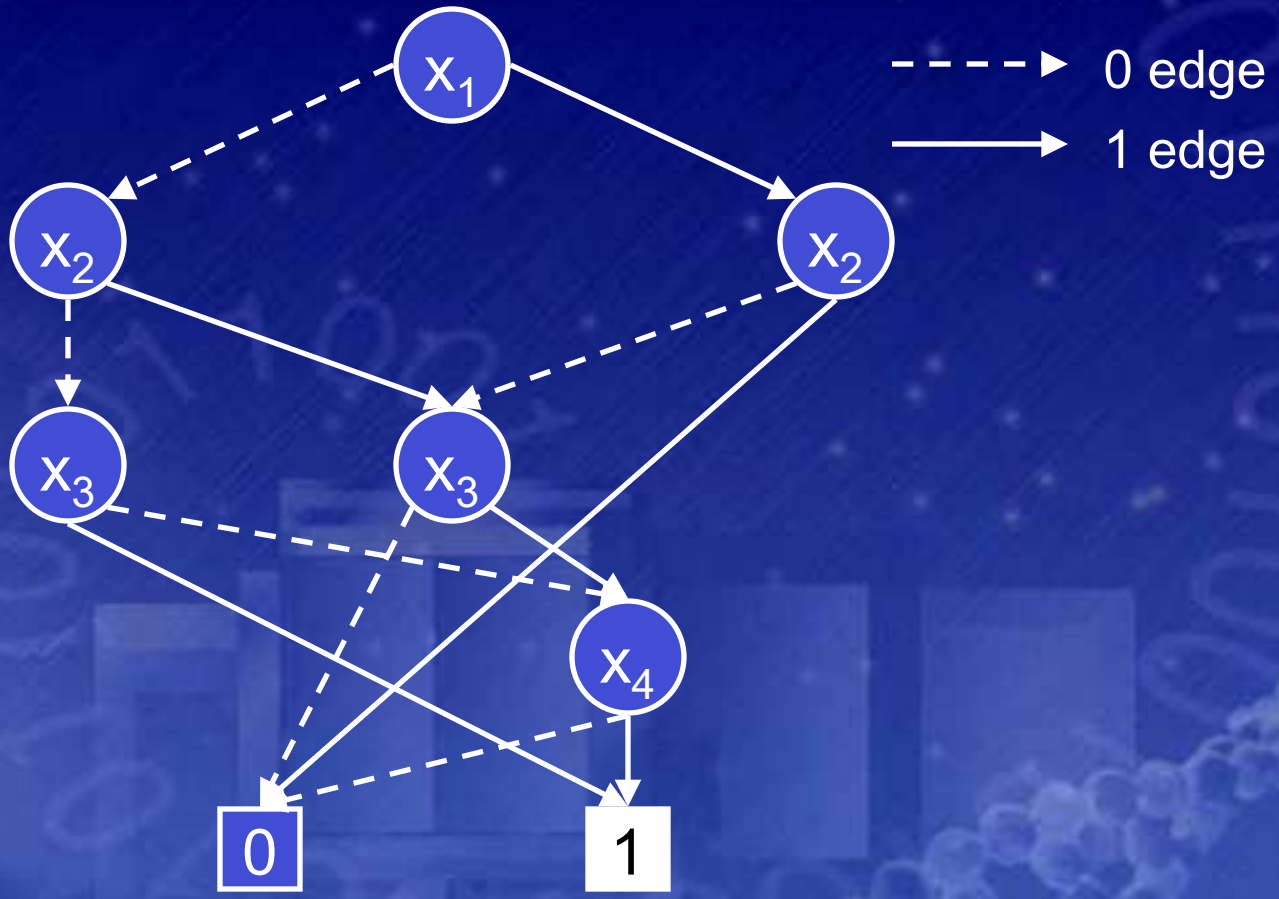
Binary Decision Diagrams

- Eliminate unnecessary nodes.



Binary Decision Diagrams

- Eliminate unnecessary nodes.

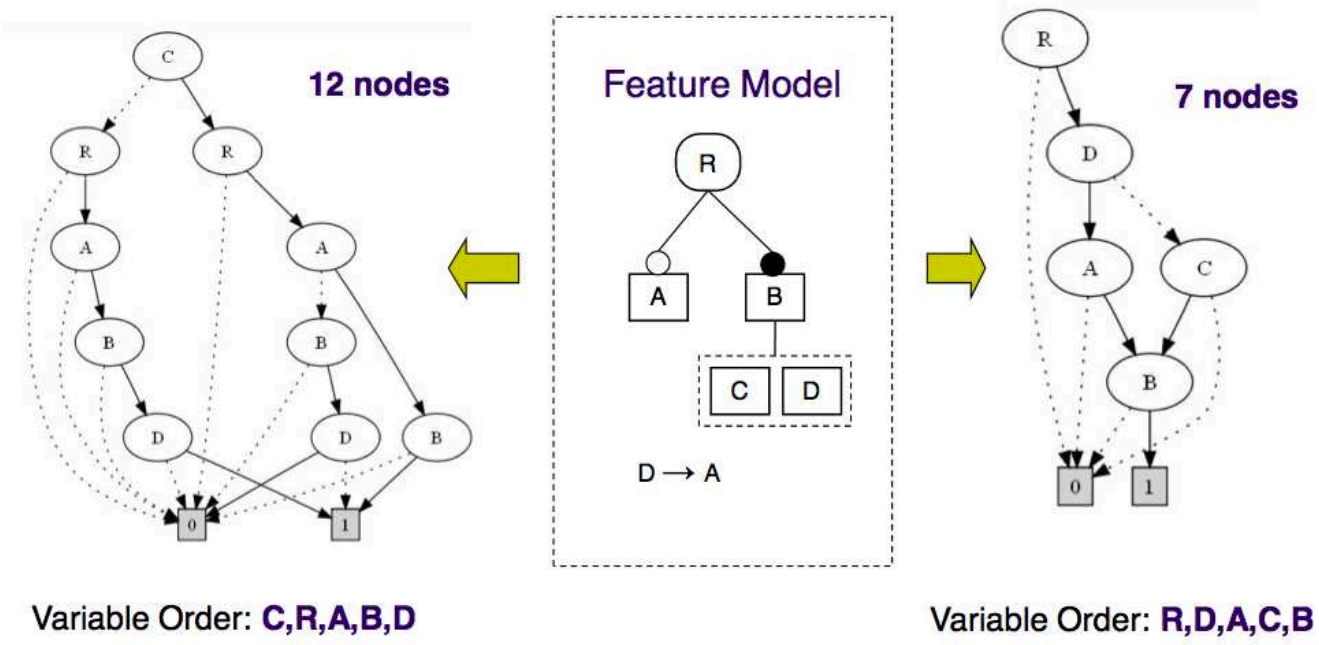


Binary Decision Diagrams (BDDs)

- Very efficient structure for most of the satisfiability operations
- Polynomial in time for checking satisfiability and determining equivalence between two BDDs
- Graph trasversal
- So great?

Binary Decision Diagrams (BDDs): Theoretical Problem

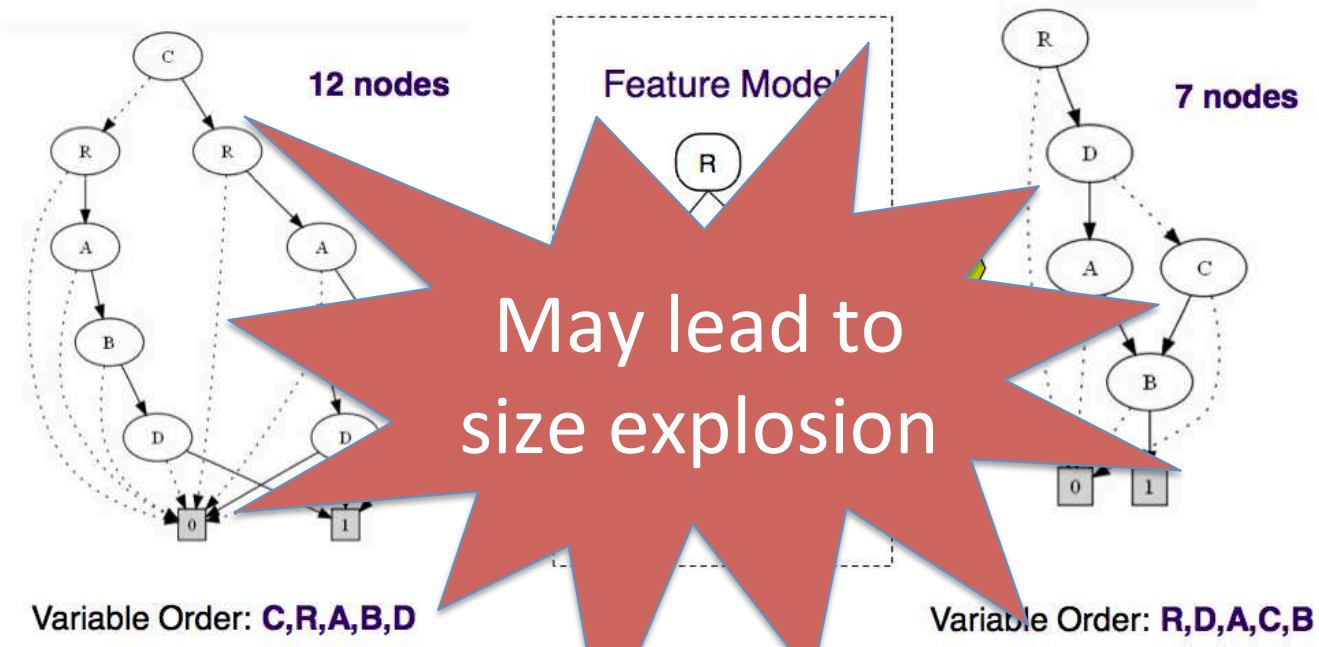
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**



[Mendonca, slide]

Binary Decision Diagrams (BDDs): Theoretical Problem

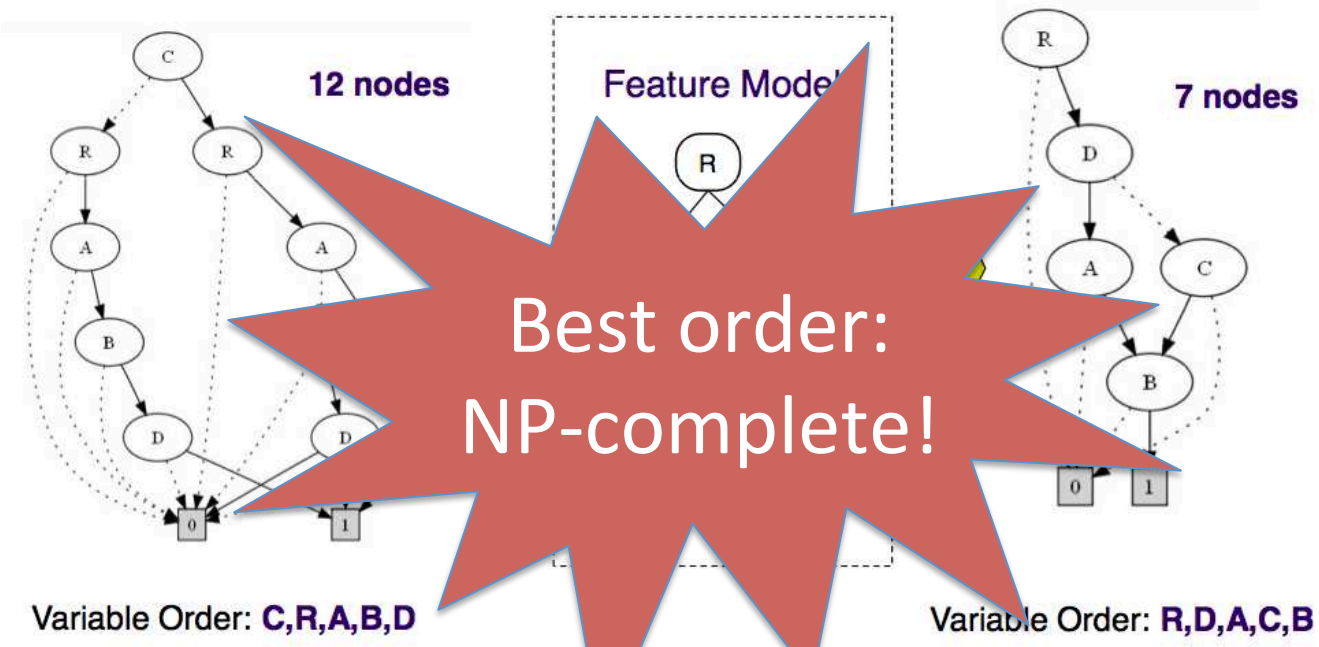
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**



[Mendonca, slide]

Binary Decision Diagrams (BDDs): Theoretical Problem

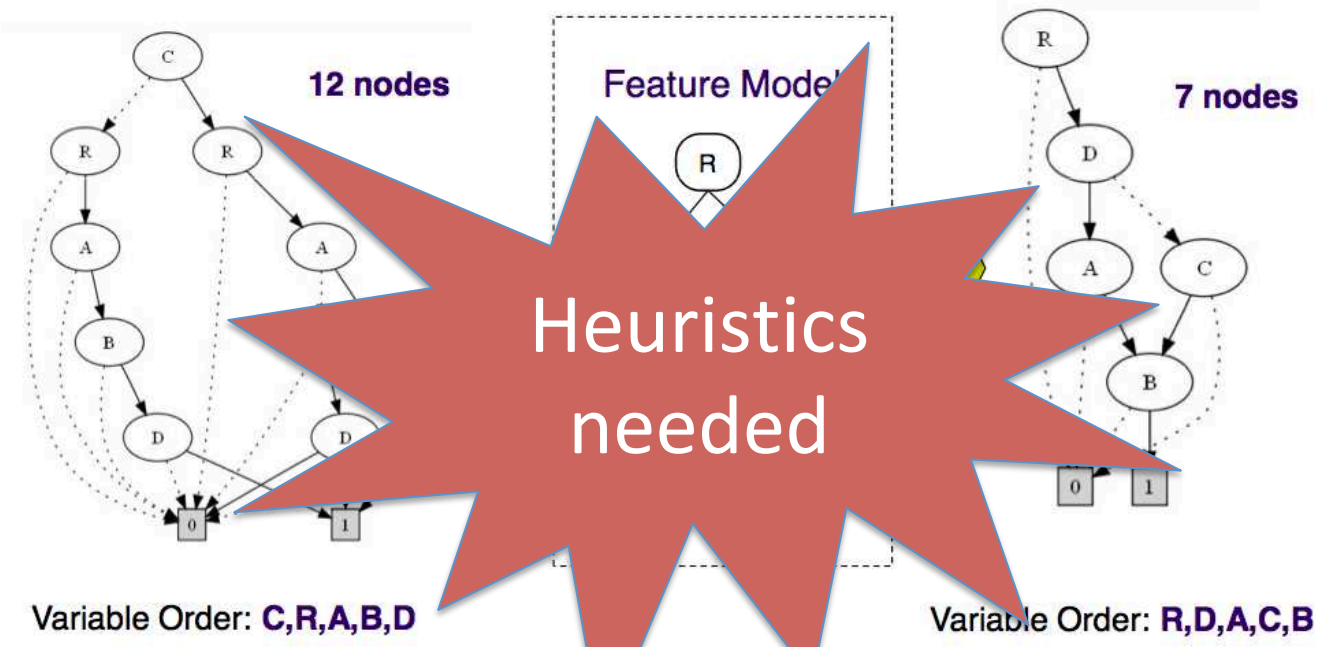
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**



[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**



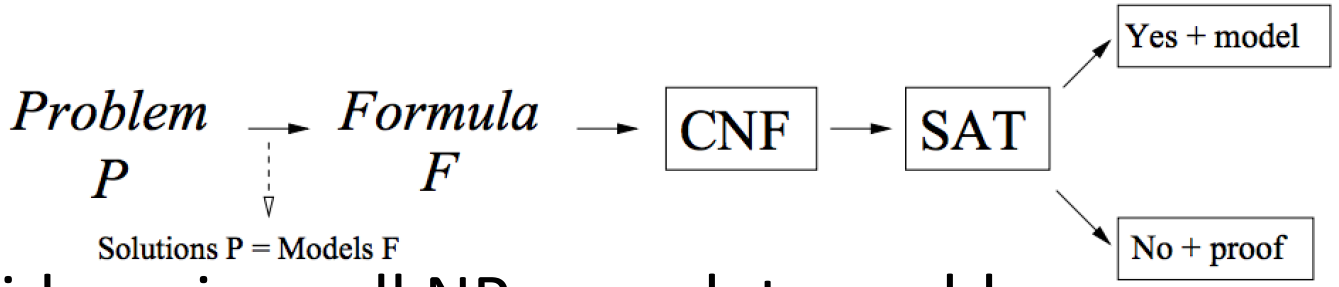
[Mendonca, slide]

How to automate analysis of your feature models?

Let us try with SAT solvers

Satisfiability (SAT) solver

- A "SAT solver" is a program that automatically decides whether a propositional logic formula is satisfiable.
  - If it is satisfiable, a SAT solver will produce an example of a truth assignment that satisfies the formula.



- Basic idea: since all NP-complete problems are mutually reducible:
  - Write one really good solver for NP-complete problems (in fact, get lots of people to do it. Hold competitions.)
  - Translate your NP-complete problems to that problem.

SAT solver and CNF

- All current fast SAT solvers work on CNF
- Terminology:
  - A literal is a propositional variable or its negation (e.g., p or ¬q).
  - A clause is a disjunction of literals (e.g., (p ∨ ¬q ∨ r)). Since ∨ is associative, we can represent clauses as lists of literals.
- A formula is in conjunctive normal form (CNF) if it is a conjunction of clauses
  - e.g., (p ∨ q ∨ ¬r) ∧ (¬p ∨ s ∨ t ∨ ¬u)

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;

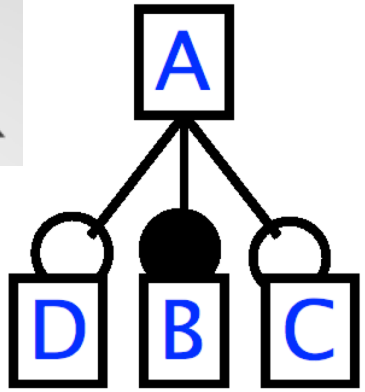
satisfy(ϕ) {
  if every clause of ϕ has a true literal, return T;
  if BCP() == F, return F;
  assign appropriate values to all pure literals;
  choose an $x \in V$ that is unassigned in A,
    and choose $v \in \{T, F\}$.
  $A(x) = v$;
  if satisfy(ϕ) return T;
  $A(x) = \neg v$;
  if satisfy(ϕ) return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

How to automate analysis of your feature models?

(back to our example)

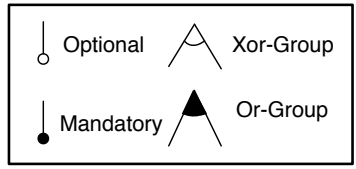
A ^
A ⇔ B ^
C => A ^
D => A

FAMiliAR



φ

FM




```
fml> fm1 = FM ("output/fm1.tvl")
root A {
    group [ 3..3 ] {
        opt D {
        },
        B {
        },
        opt C {
        }
    }
}
fm1: (FEATURE_MODEL) A: [D] B [C] ;
```

```
fm1bis = FM ("foo3.dimacs")
fm1bisbis = FM ("foo3.constraints")
```

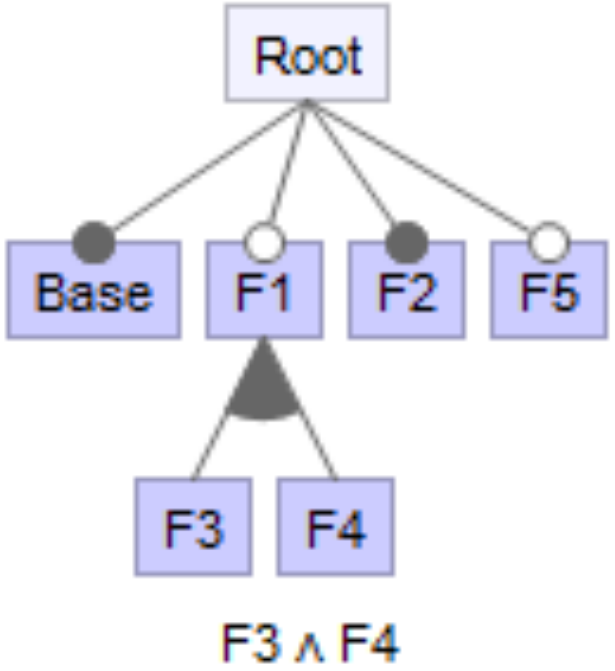
```
fml> c1 = cores fm1
c1: (SET) {B;A}
fml> c1bis = cores fm1bis
```

```
fml> compare fm1 fm1bis
res7: (STRING) REFACTORING
fml> compare fm1bis fm1bisbis
res8: (STRING) REFACTORING
```

```
fml> c1 eq c1bisbis
res6: (BOOLEAN) true
```

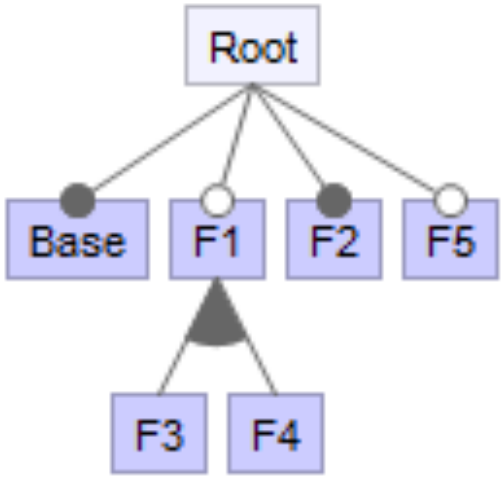
Consistency

- SAT-Solver
  - SAT(FM)



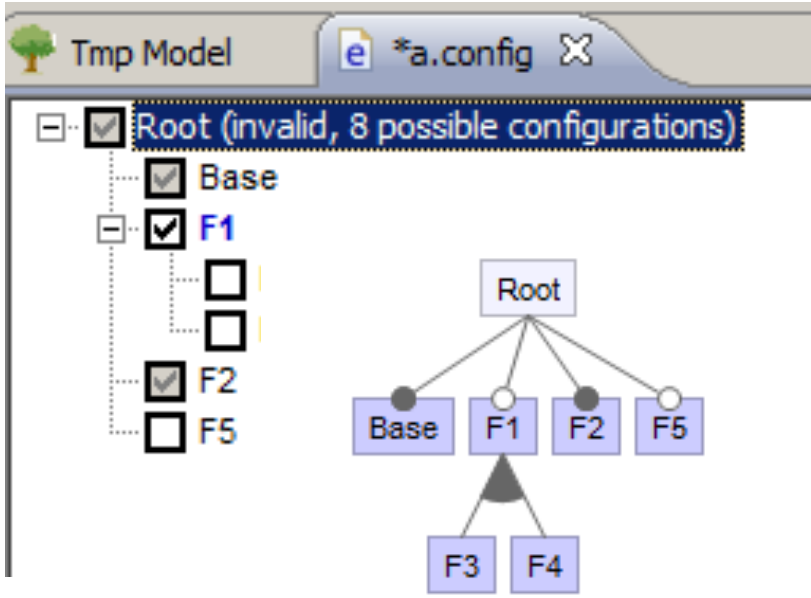
Core and dead features

- Dead : SAT(FM ^ F)
- Core: SAT(FM ^ not(F))

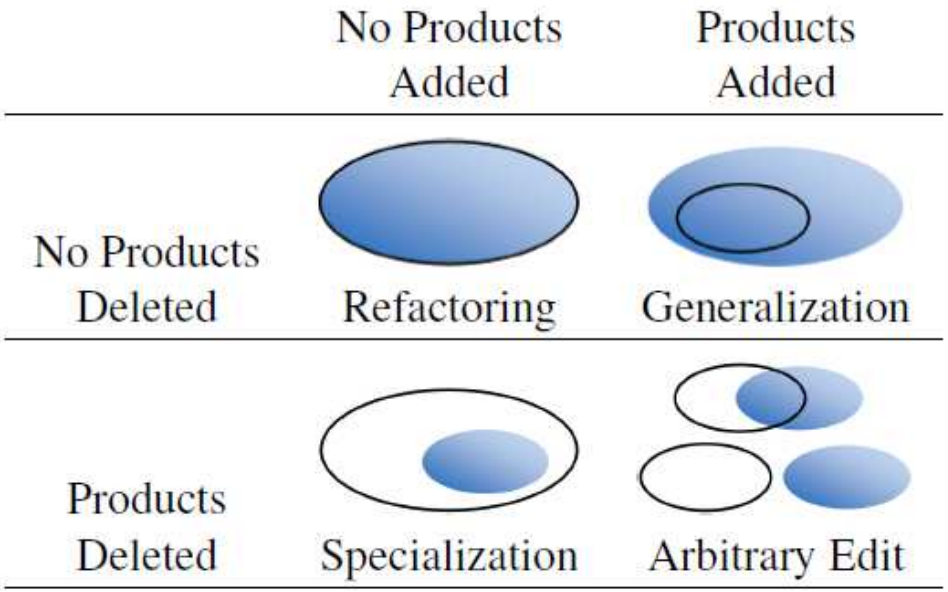


Partial configuration

- SAT(FM ^ PK ^ F)
- SAT(FM ^ PK ^ not(F))



Relationship between feature models



- Refactoring
  - Tautology: (FM1 <=> FM2)
    = not SAT(not (FM1 <=> FM2))

From Logics to Feature Models (and back again)

syntax, semantics

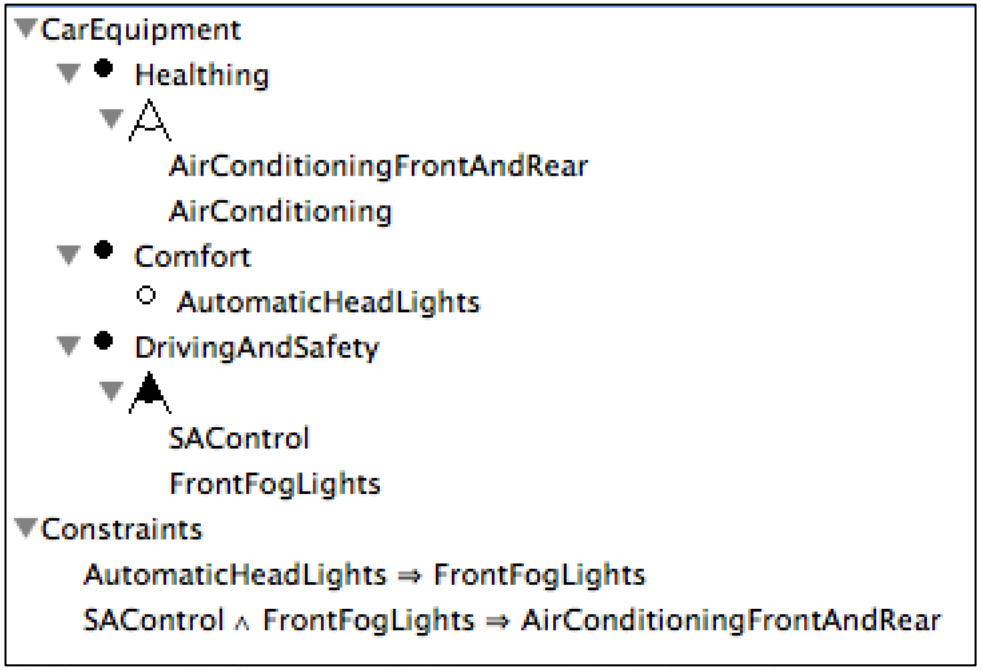
Formal semantics of a language

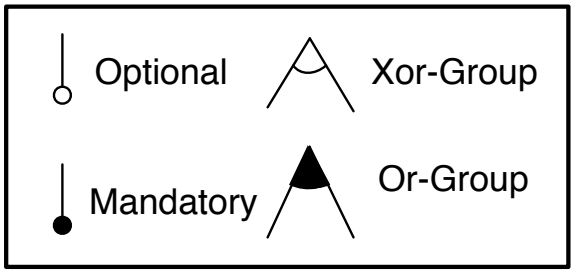
– **formal syntax** (L) – clearcut syntactic rules defining all legal diagrams, a.k.a. syntactic domain

– **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

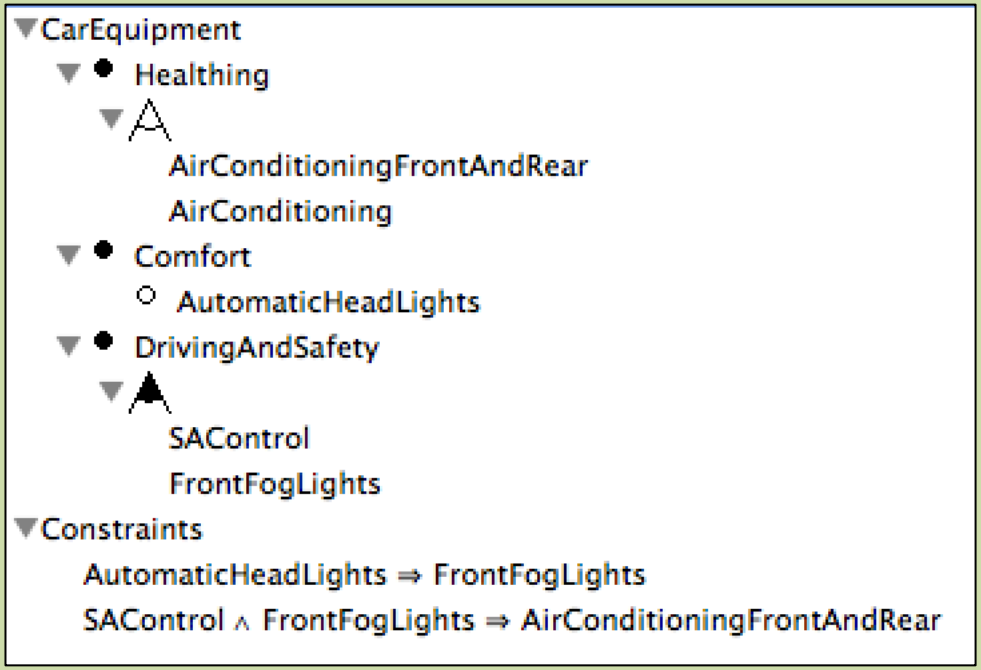
[Harel & Rumpe, IEEE Computer, 2004]

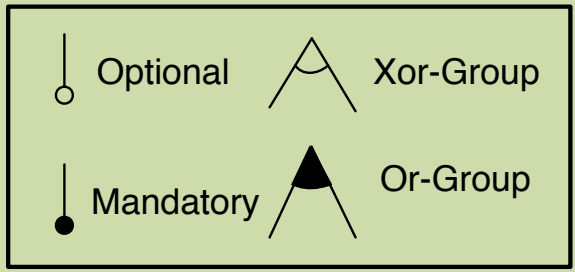




Definition 2 (Feature Diagram) *A feature diagram* $FD = \langle G, E_{MAND}, G_{XOR}, G_{OR}, I, EX \rangle$ *is defined as follows:* $G = (\mathcal{F}, E, r)$ *is a rooted, labeled tree where* $\mathcal{F}$ *is a finite set of features,* $E \subseteq \mathcal{F} \times \mathcal{F}$ *is a finite set 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* ψ *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}

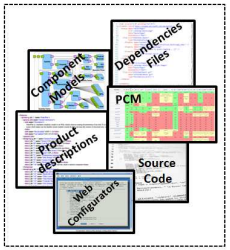
X

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

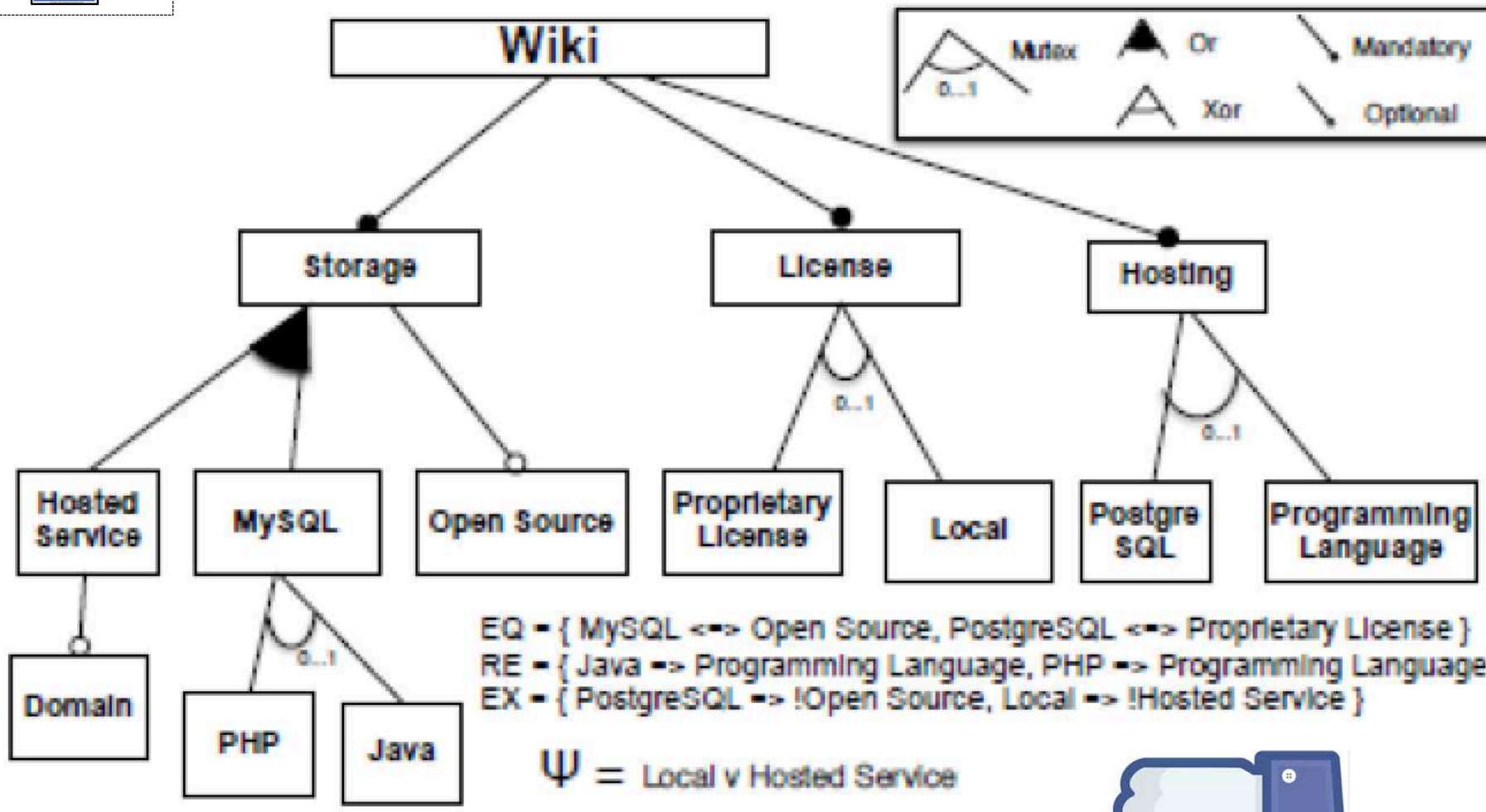
Quizz

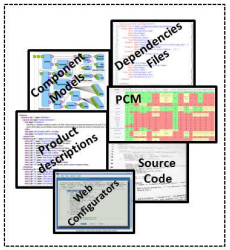
1) Give two feature models with the same configuration semantics but with different syntax

2) Does it matter ?



Importance of ontological semantics (1)





Importance of ontological semantics (2)

Wiki Wizard

Choose your Wiki

Storage

Do you want to choose Open Source as your storage system?

- (•) Yes, I want.
- ( ) No.

Should your hosted service provide a Domain ?

- (•) Yes.
- ( ) No.

Choose your MySQL database:

- (•) JAVA
- ( ) PHP
- ( ) None.

License

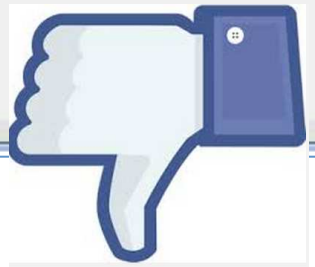
Select your License:

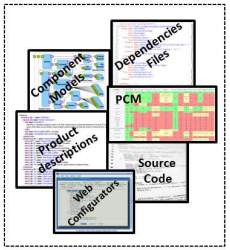
- ( ) Proprietary License
- (•) Local
- ( ) None

Hosting

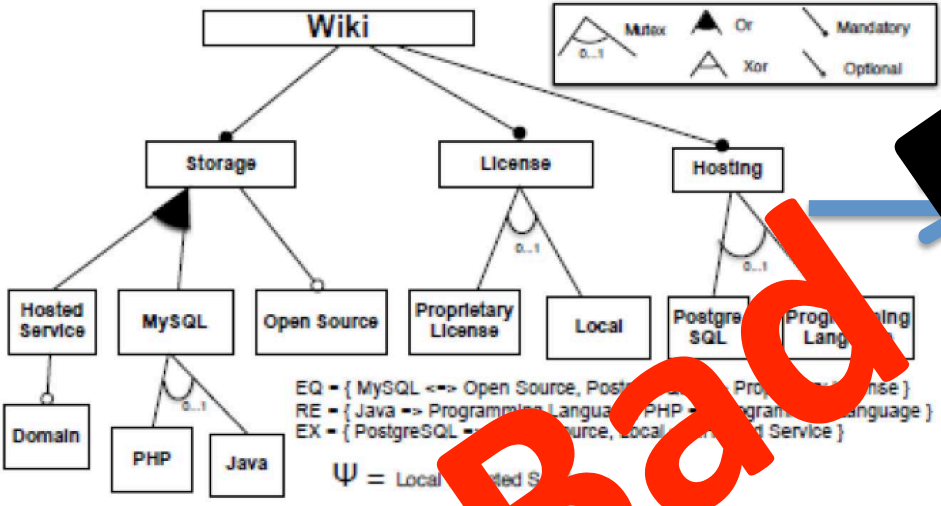
What would you like to choose?

- ( ) PostgreSQL
- (•) Programming Language
- ( ) None.





Importance of ontological semantics (3)

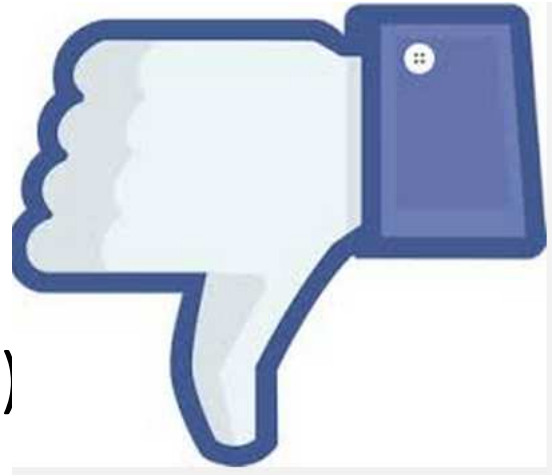


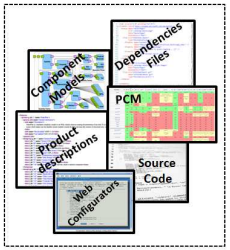
Bad FM

Communication

Comprehension

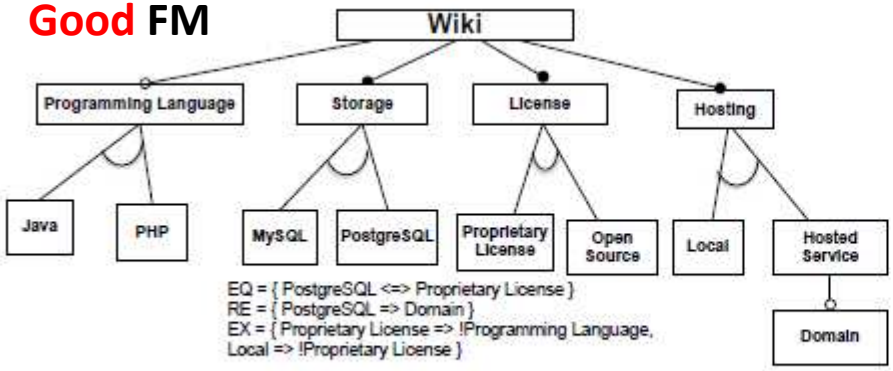
Forward engineering (e.g., generation)



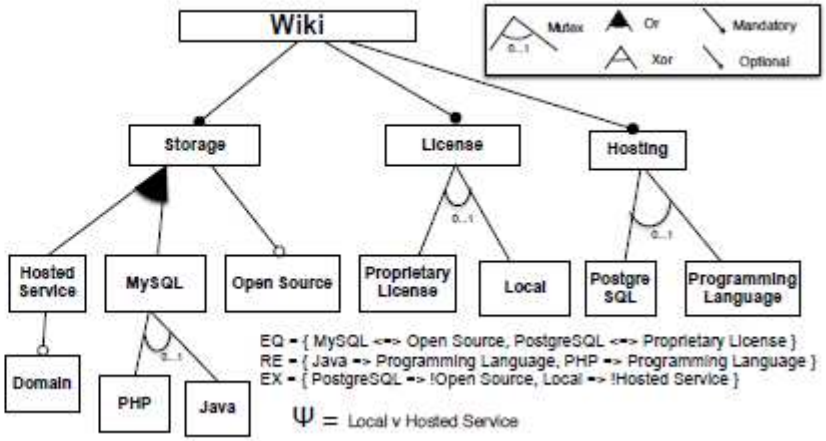


Importance of ontological semantics (4)

Good FM

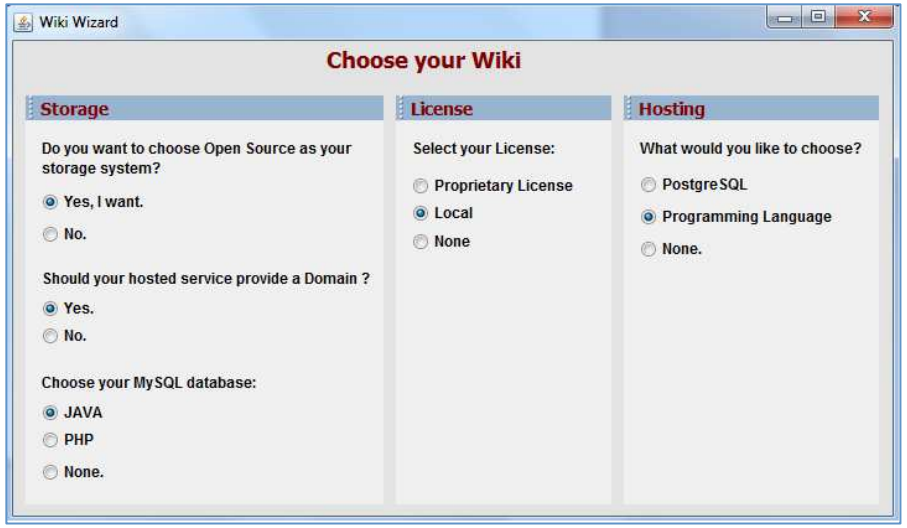


Bad FM



Good configuration interface

Bad configuration interface



Two product configurators generated from two FMs with the same configuration semantics but different ontological semantics.

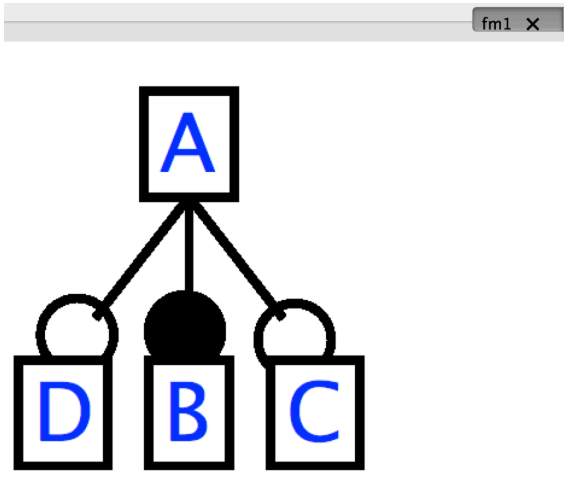
A ^
A ⇔ B ^
C => A ^
D => A

Feature Model Synthesis Problem

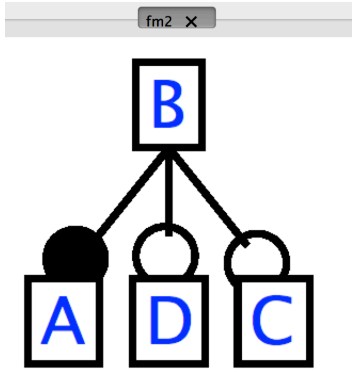
[Czarnecki et al., SPLC'07]
[She et al., ICSE'11]
[Andersen et al., SPLC'12]

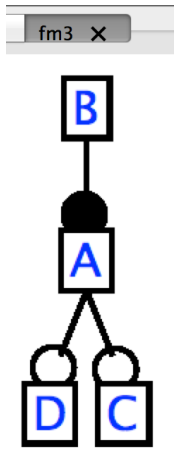
φ

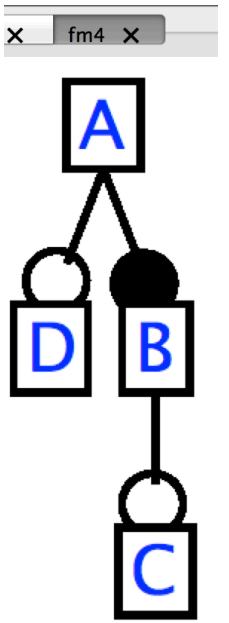
FAMiliAR



FM







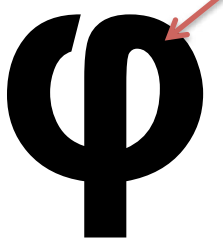

```
fm2 = FM (B : A [C] [D] ; )
fm3 = FM (B : A ; A : [C] [D] ; )
fm4 = FM (A : B [D] ; B : [C] ; )
fm5 = FM (A : B [C] ; B : [D] ; )

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

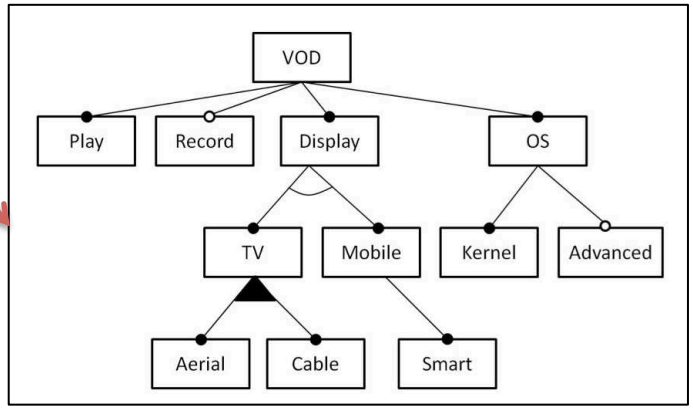
#1 Reverse Engineering Scenarios

- [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 ;
```

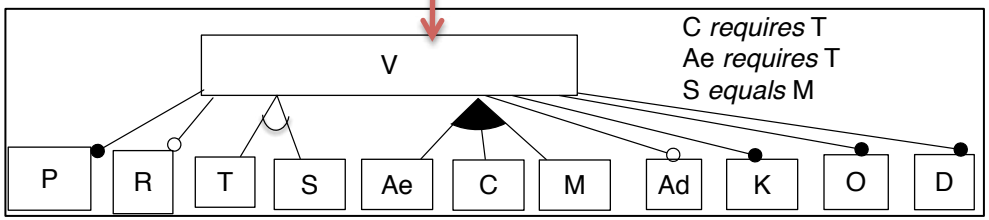


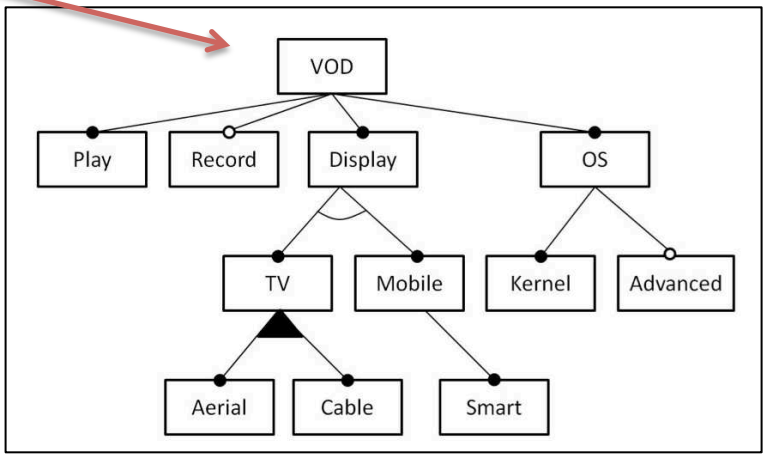


#2 Refactoring

- [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

Quizz (Class Diagram)

1) Give two class diagrams with the same semantics but with different syntax

   – (preliminary) what is the semantic domain of CDs?

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

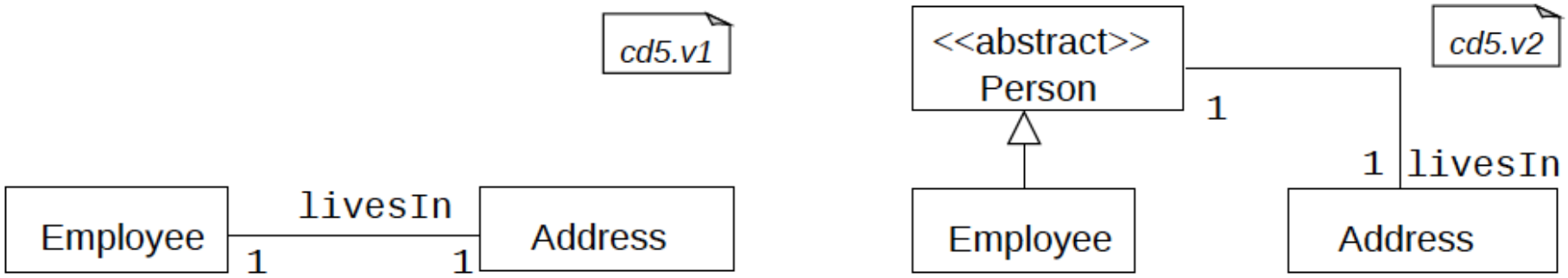


Fig. 4. $cd5.v1$ and its revised version $cd5.v2$. The two versions have equal semantics.

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

Quizz (back to Feature Model)

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* ψ *is a propositional formula over the set of features* $\mathcal{F}$.

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

ie [[fd]] = s

In other words: is the formalism of feature diagram expressive enough wrt Boolean logic?

Feature Diagram ?

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

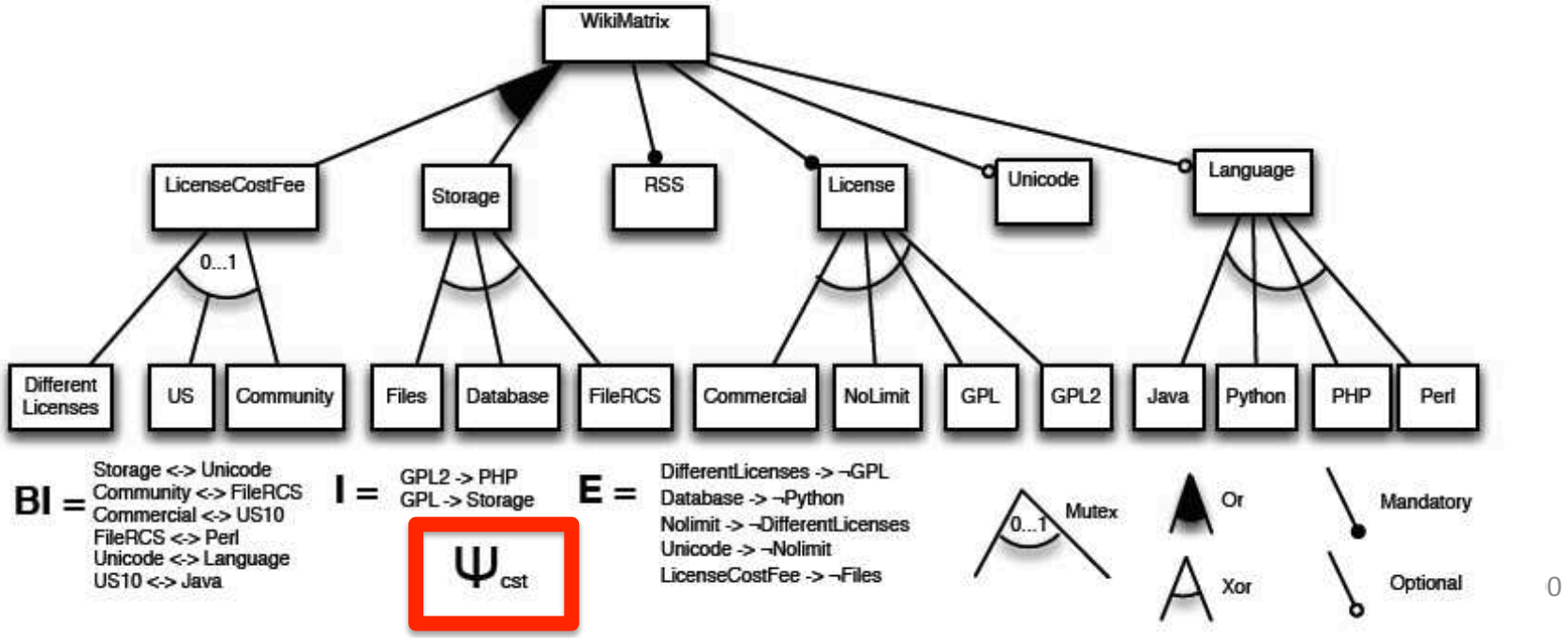
fm1 = FM (A : [B] [C] [D] ^
// B, C and D are optional features of A

((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



Quizz

Feature Model (bis)

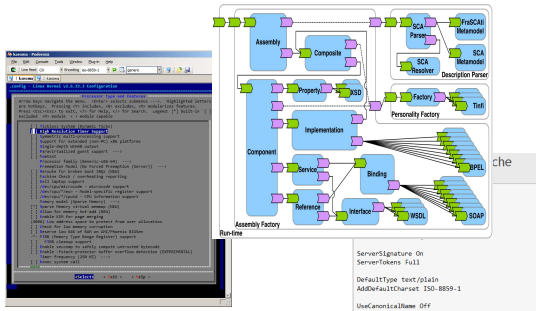
s = {{A}, {B}}

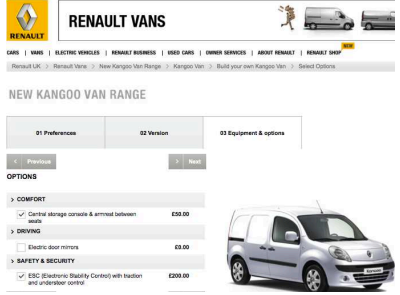
fd = ?

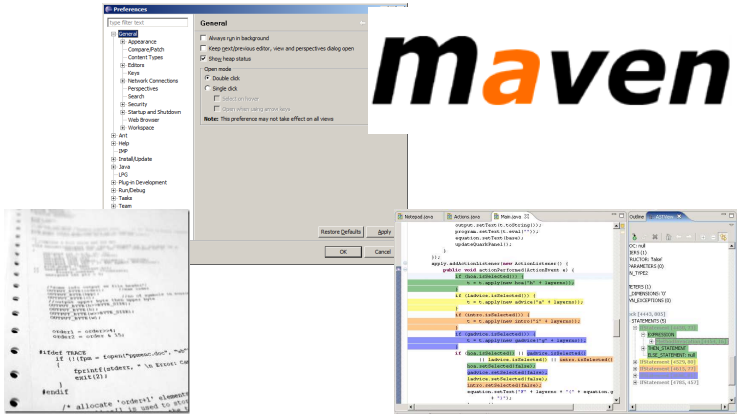
Feature Model: Key Insights

- 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

Synthesising Feature Models (from semantics to syntacs)





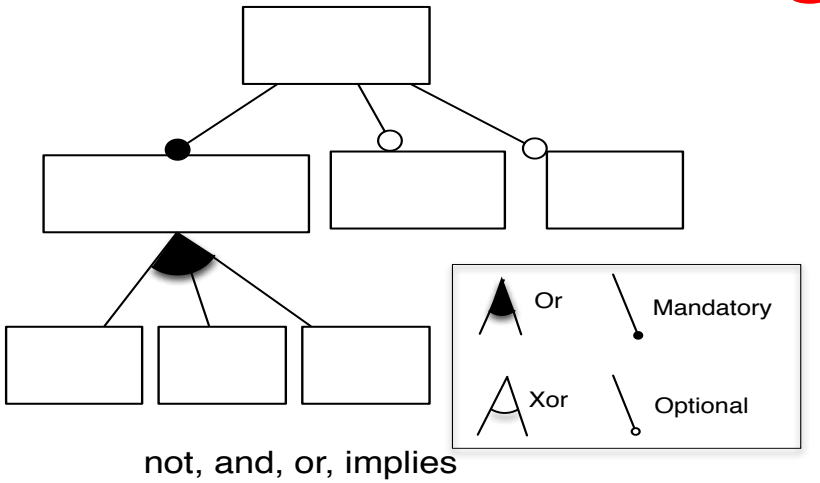


maven

Mining/Extracting

Encoding/Formalizing

Synthesising



not, and, or, implies

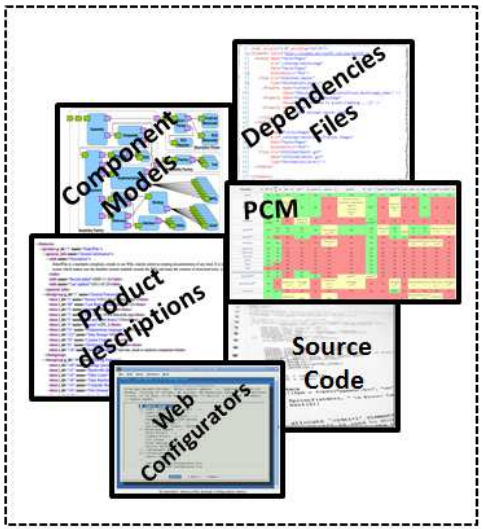
FAMiliAR

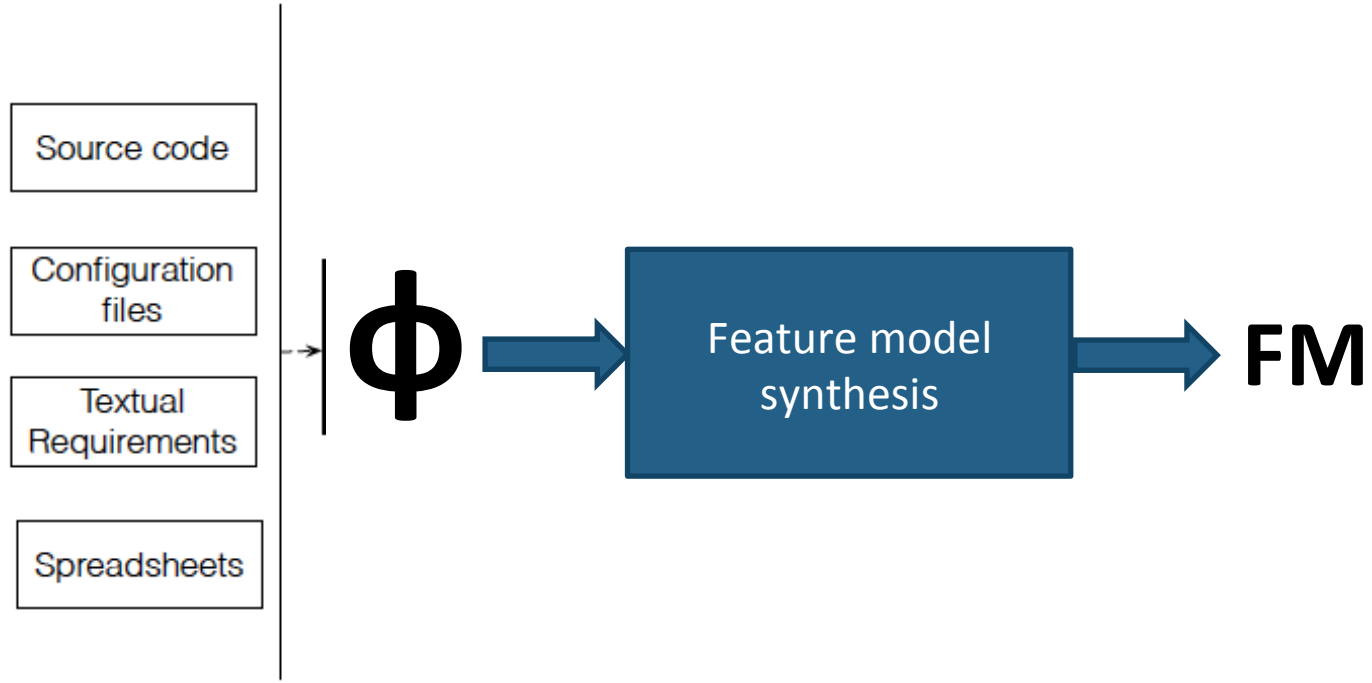
Variability Models (feature models)

- CarEquipment
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights
- Constraints
  - AutomaticHeadLights ⇒ FrontFogLights

φ

Product	CarEquipment	Comfort	DrivingAndSafety	Healting	AirConditioning
Find	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No
Car1	yes	yes	yes	yes	yes
Car2	yes	yes	yes	yes	yes
Car3	yes	yes	yes	yes	no
Car4	yes	yes	yes	yes	no
Car5	yes	yes	yes	yes	yes
Car6	yes	yes	yes	yes	no





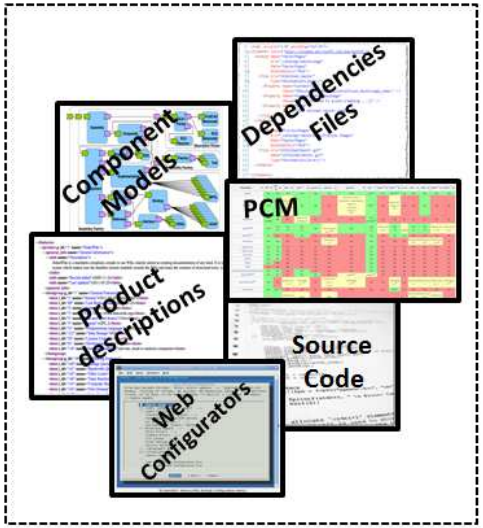
Feature model synthesis problem

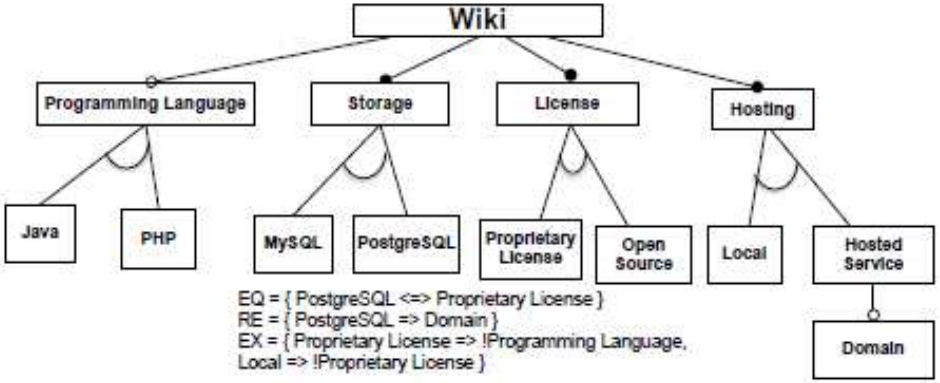
Input: ϕ, a propositional formula representing the **dependencies** over a set of features *F*.

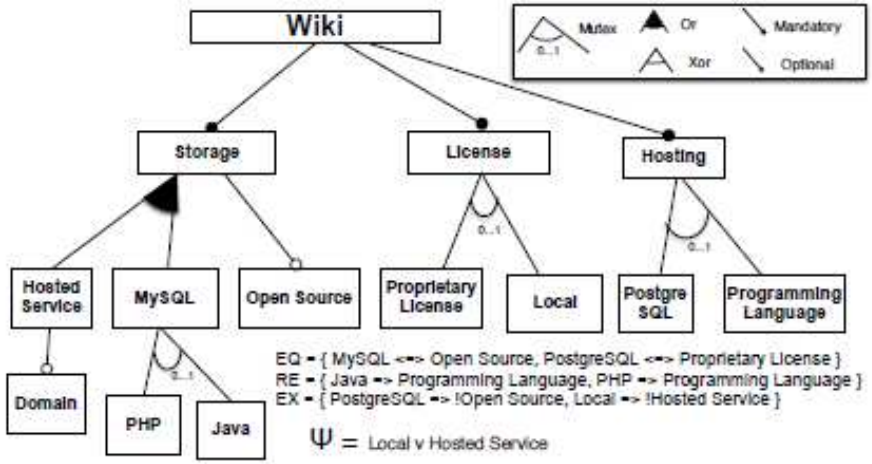
Output: a maximal feature model with a **sound configuration semantics**

For a given configuration set,

many (maximal) feature diagrams

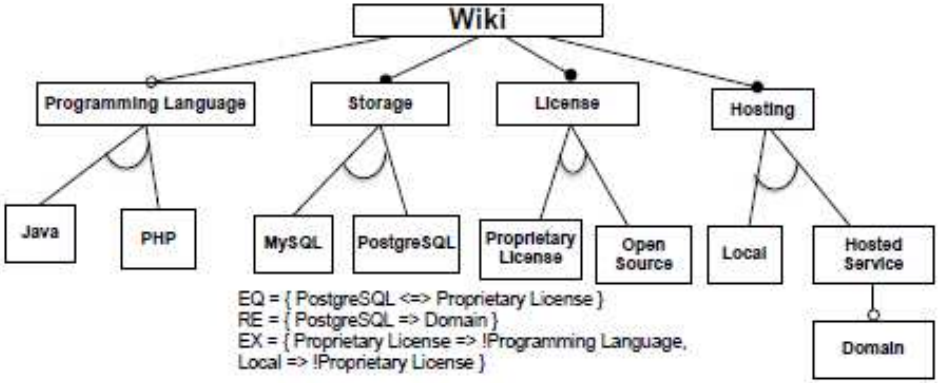


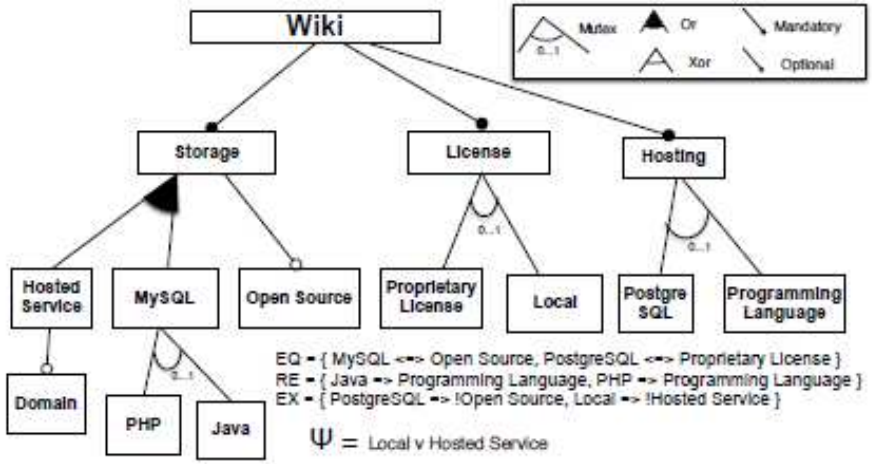




Maximal feature diagrams

~ “as much logical information as possible is represented in the diagram itself, without resorting to the constraints”



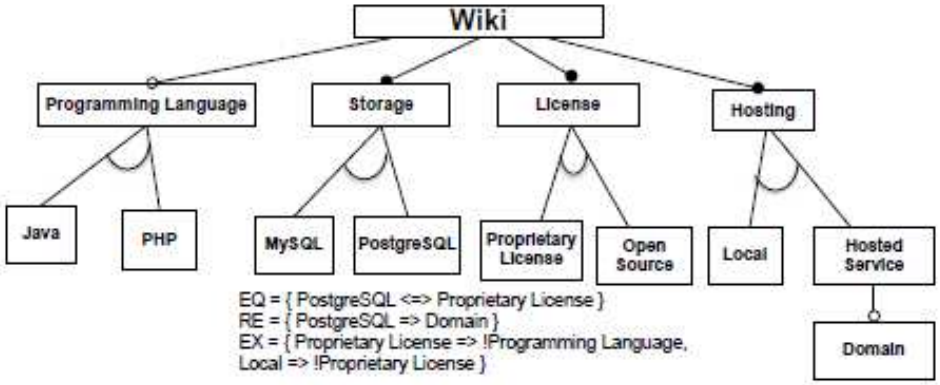


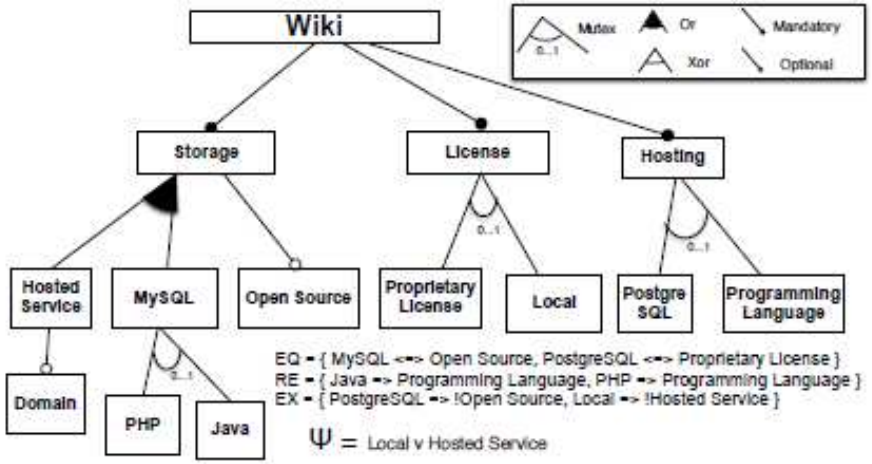
Maximal feature diagrams

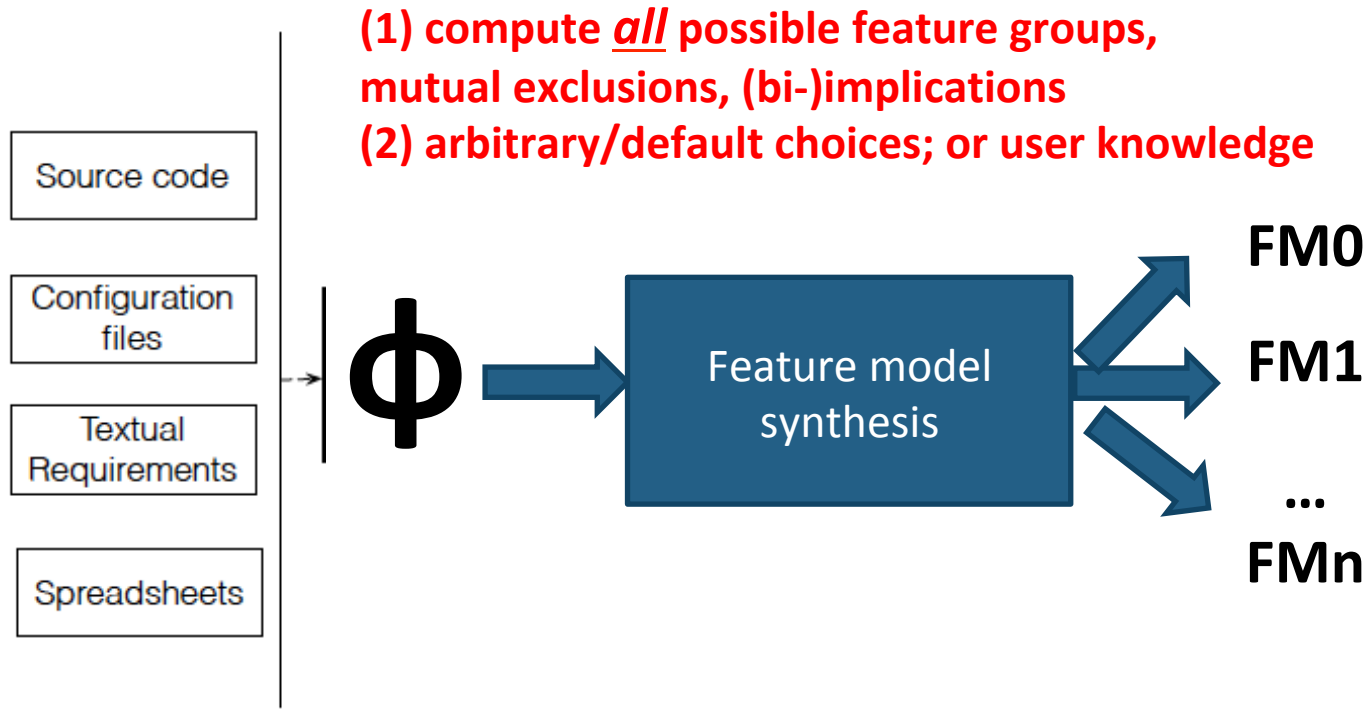
~ “as much logical information as possible is represented in the diagram itself, without resorting to the constraints”

Quizz:

give a non maximal feature diagram







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**

$(\varphi$: 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

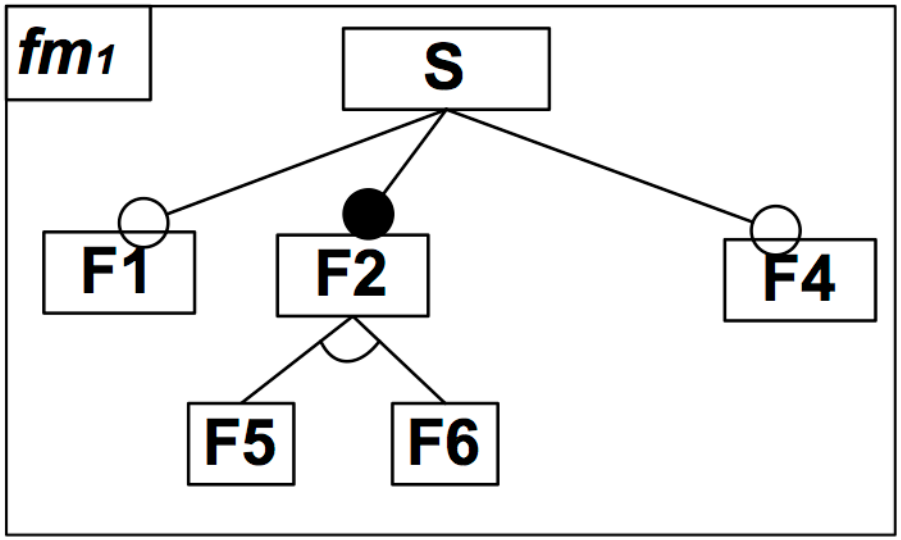
13 $p' \in p$ and $\forall f_i.\ f'_i \in f_i \wedge (f_i, p) \in E'\}$

▷ 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)



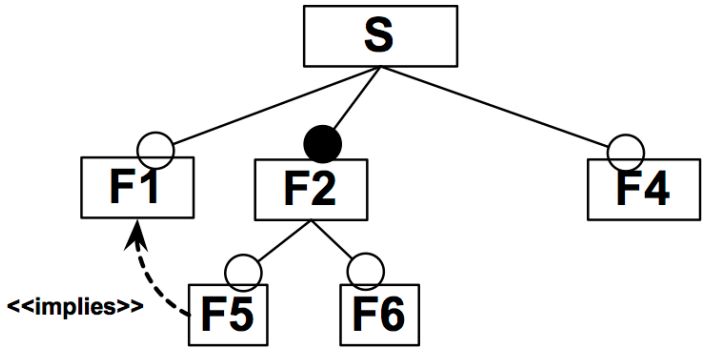

```
fml> computeImplies fm1
computeImplies fm1 ===> {(F1 -> F2);(F5 -> S);(S -> F2);(F6 -> F2);(F4 -> S);(F1 -> S);(F5 -> F2);(F4 -> F2);(F2 -> S);(F6 -
fml> cliques fm1
cliques fm1 ===> {{S;F2}}
fml> computeXORGroups fm1
computeXORGroups fm1 ===> {[F6, F5] -> S (XOR);[F6, F5] -> F2 (XOR)}
fml> computeMUTEXGroups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

Quizz

There and Back Again

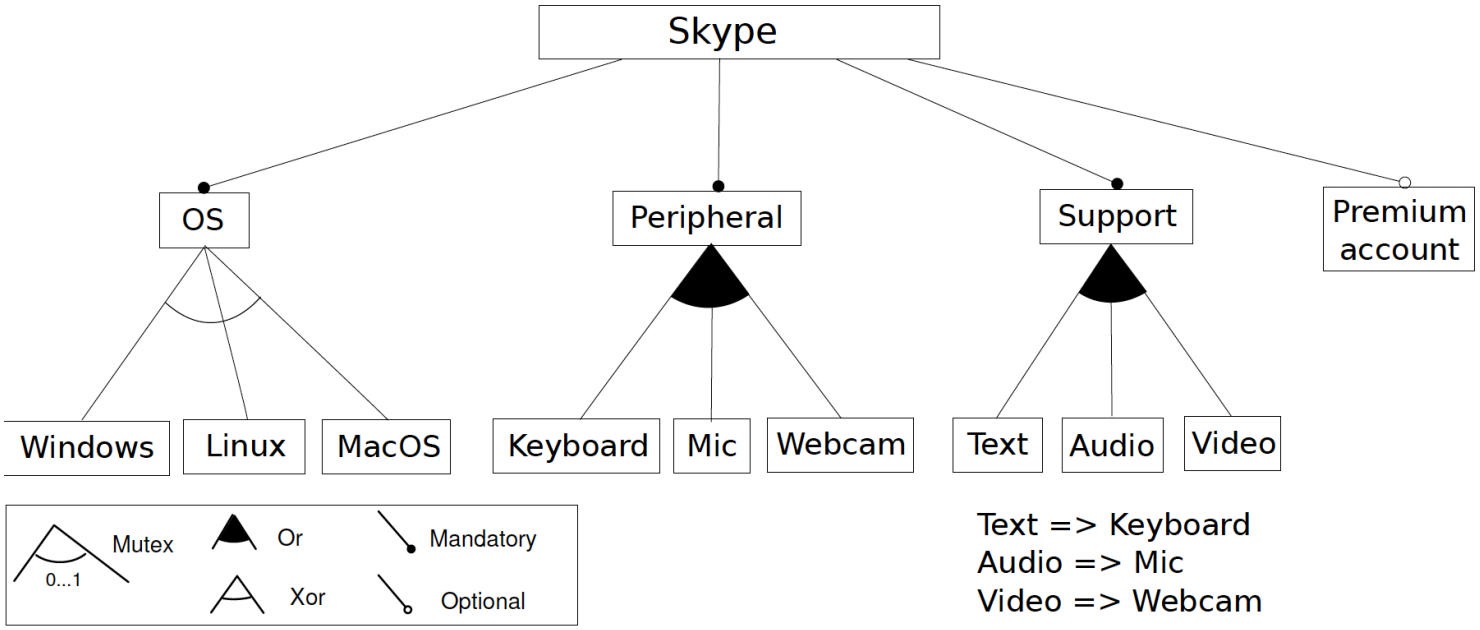
1) Translate this feature model to a propositional formula

2) Compute the synthesis « structures »

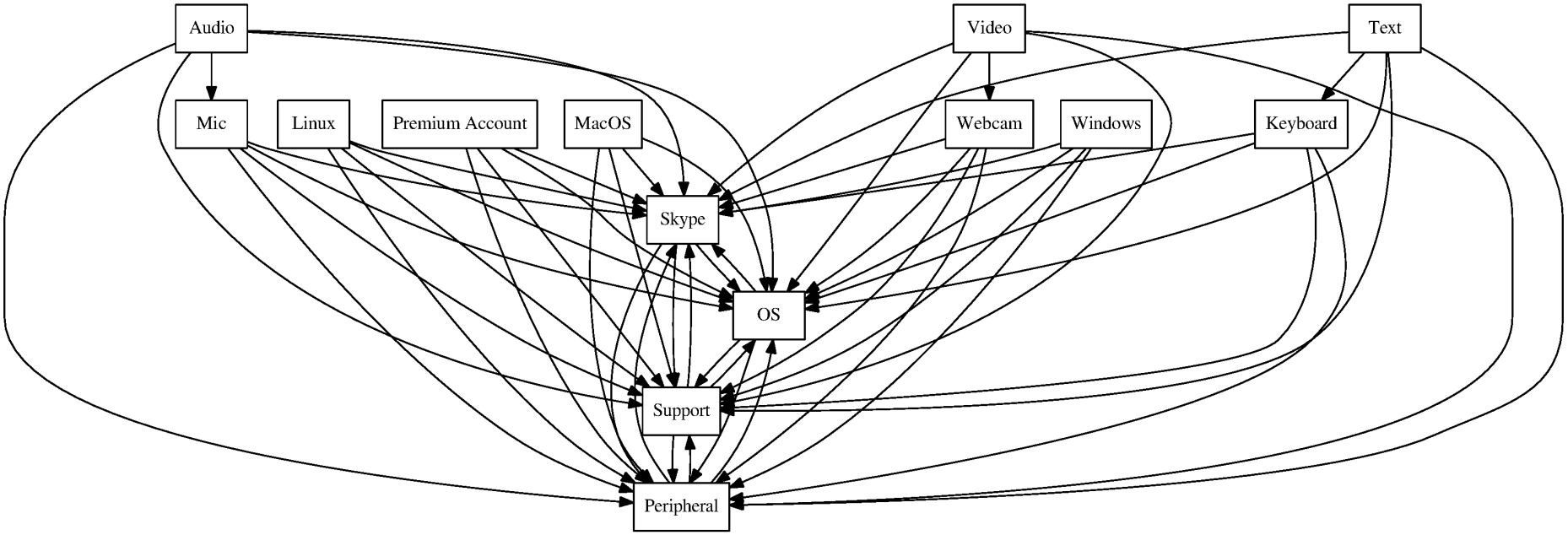


Key: Binary Implication Graph

Sound and complete representation of possible hierarchies



Sound and complete representation of possible hierarchies

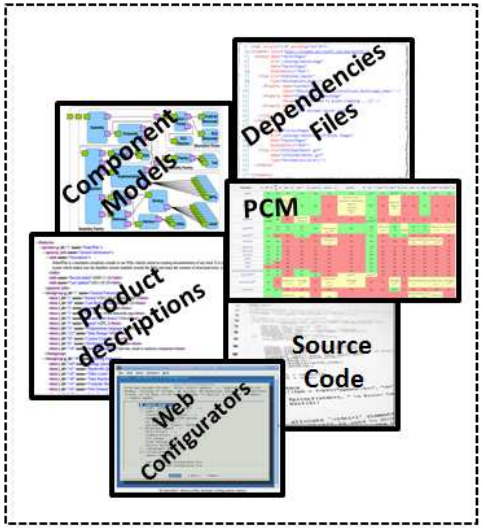


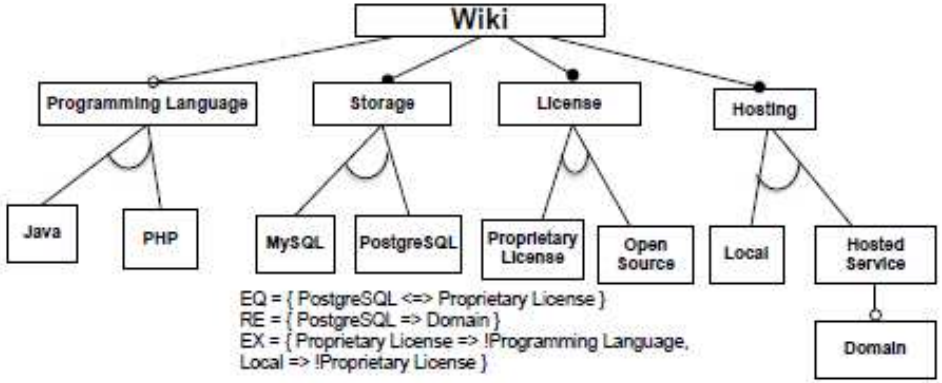
For a given configuration set,

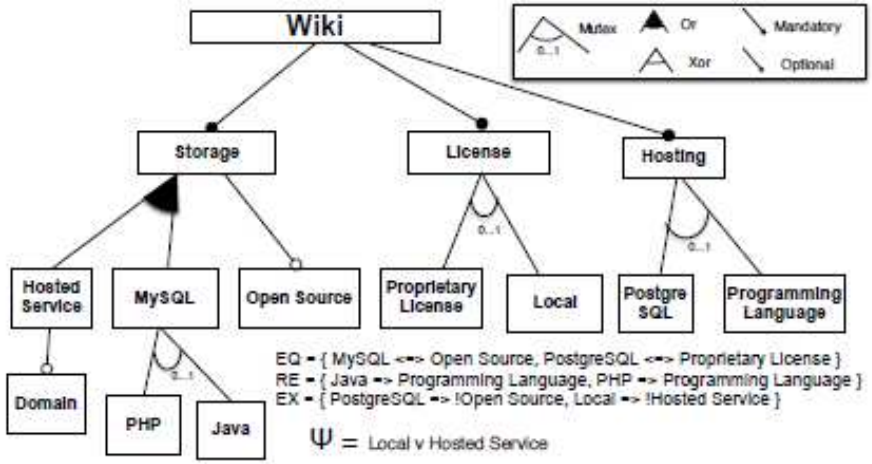
many (maximal) feature diagrams

with different ontological semantics

[She et al. ICSE'11, Andersen et al. SPLC'12, Acher et al. VaMoS'13]



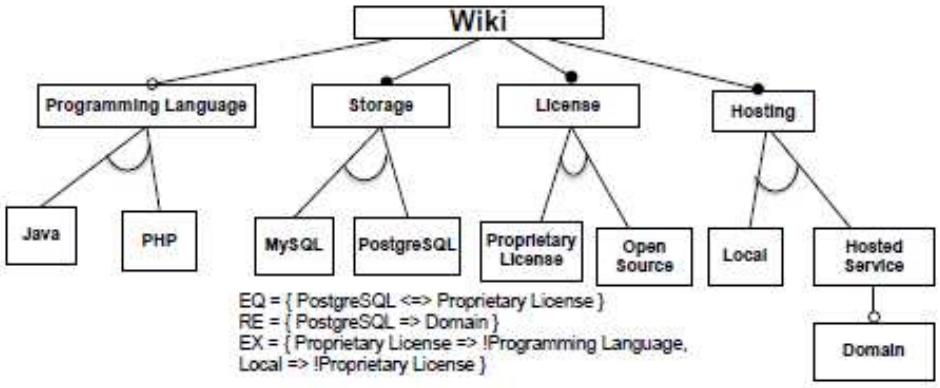


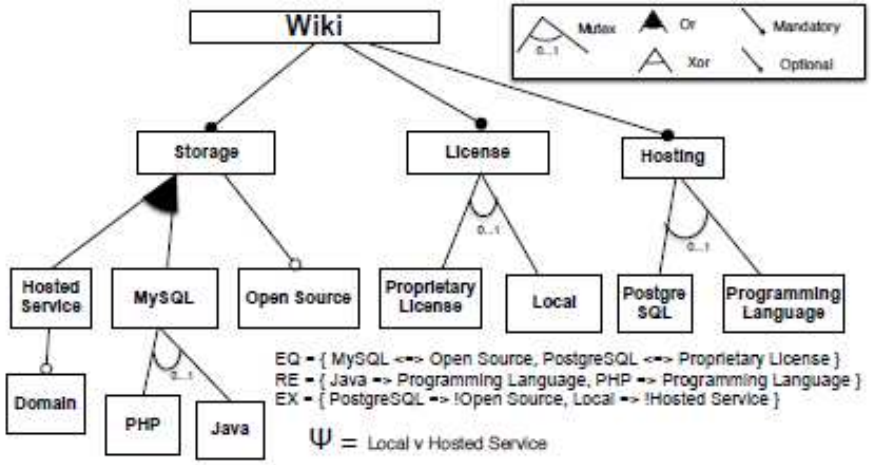


For a given configuration set, many (maximal) feature diagrams with different ontological semantics [She et al. ICSE'11, Andersen et al. SPLC'12, Acher et al. VaMoS'13]

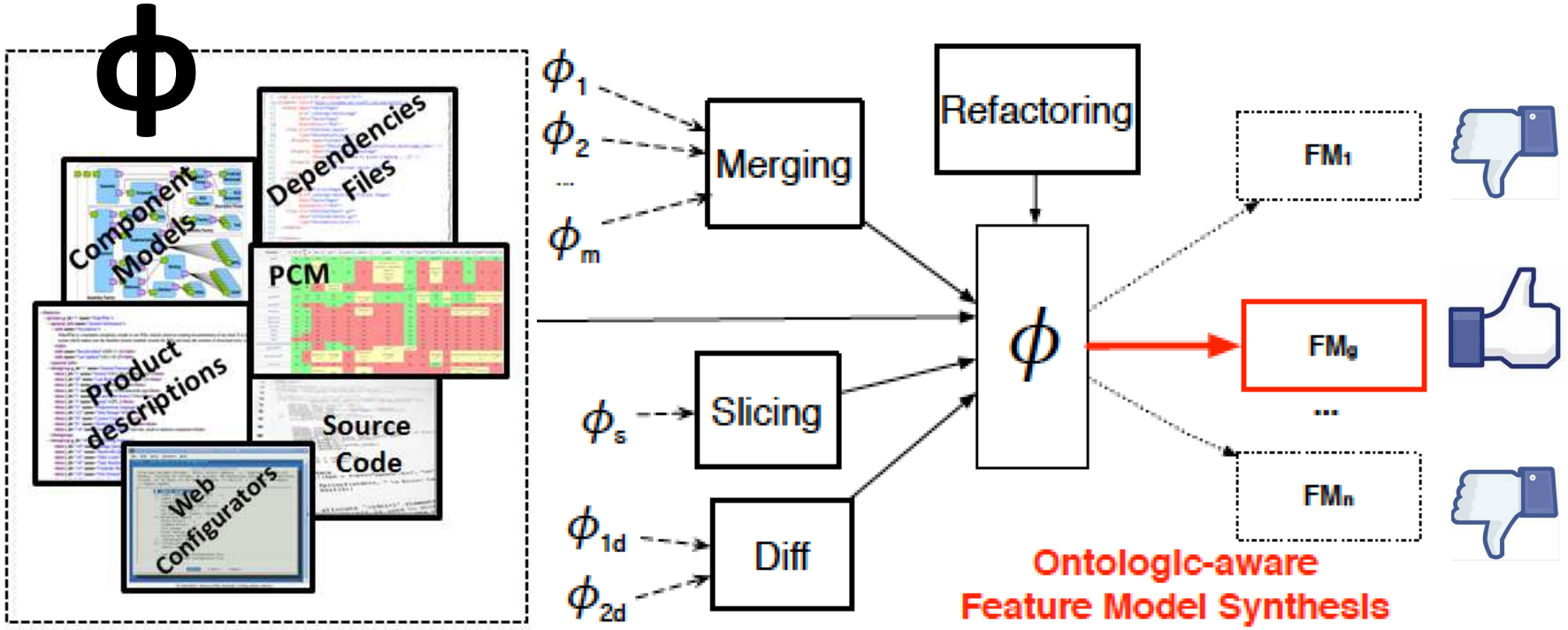
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
PostgreSQL	✓	×	×	×	×	×	×	×	×	×
MySQL	×	✓	✓	✓	✓	✓	✓	✓	✓	✓
License	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Domain	✓	×	×	×	×	✓	×	✓	✓	×
Proprietary License	✓	×	×	×	×	×	×	×	×	×
Local	×	×	✓	×	✓	×	×	×	×	✓
Programming Language	×	✓	×	×	✓	✓	✓	✓	×	✓
Java	×	✓	×	×	×	✓	×	×	×	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PHP	×	×	×	×	✓	×	✓	✓	×	×
Open Source	×	✓	✓	✓	✓	✓	✓	✓	✓	✓
Wiki	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Hosting	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Hosted Service	✓	✓	×	✓	×	✓	✓	✓	✓	×

(a) Product comparison matrix (✓ feature is in the product ; × feature is not in the product)



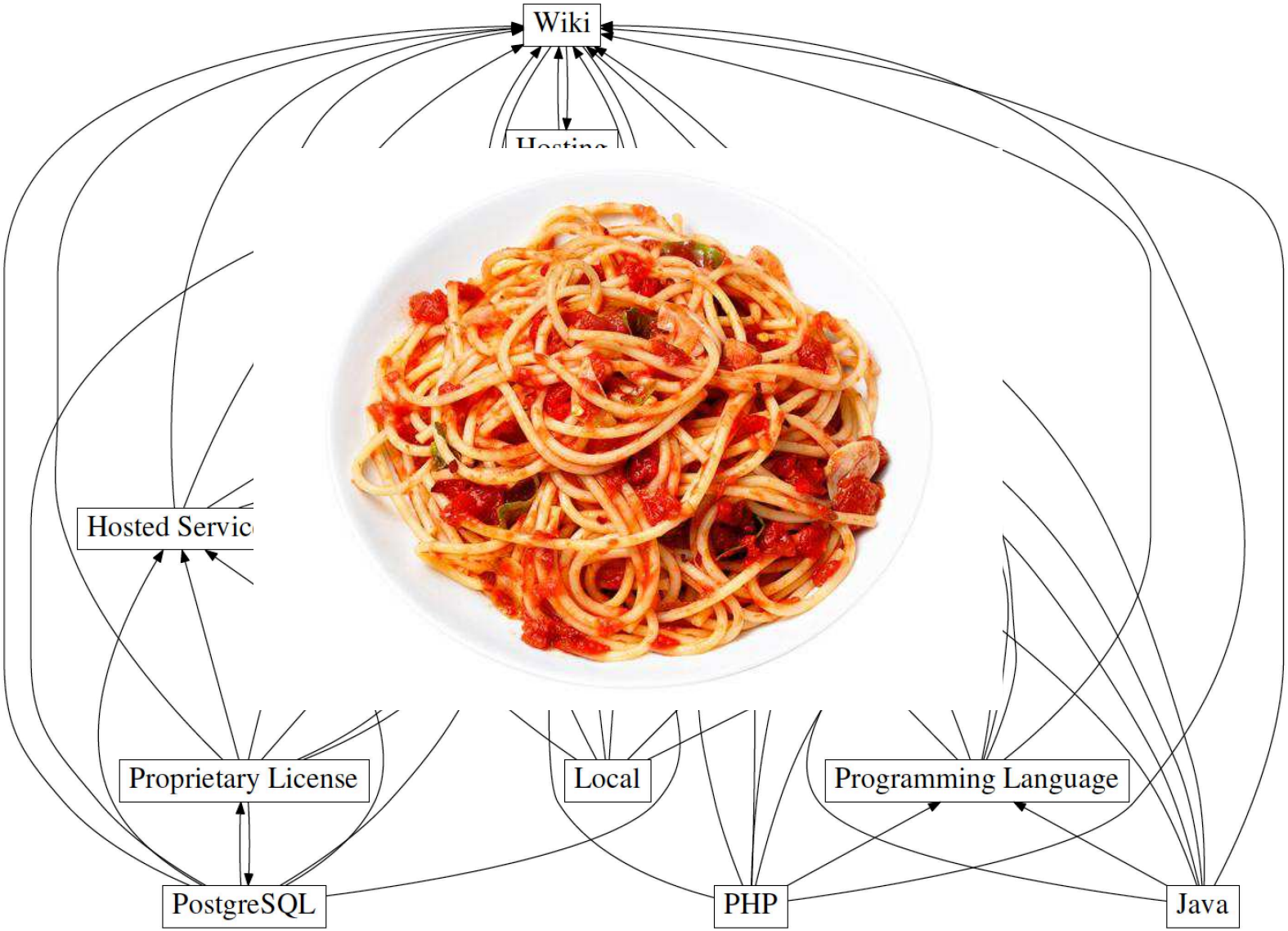


Most of the existing approaches neglect either configuration or ontological semantics. We want both!



Fundamental Problem

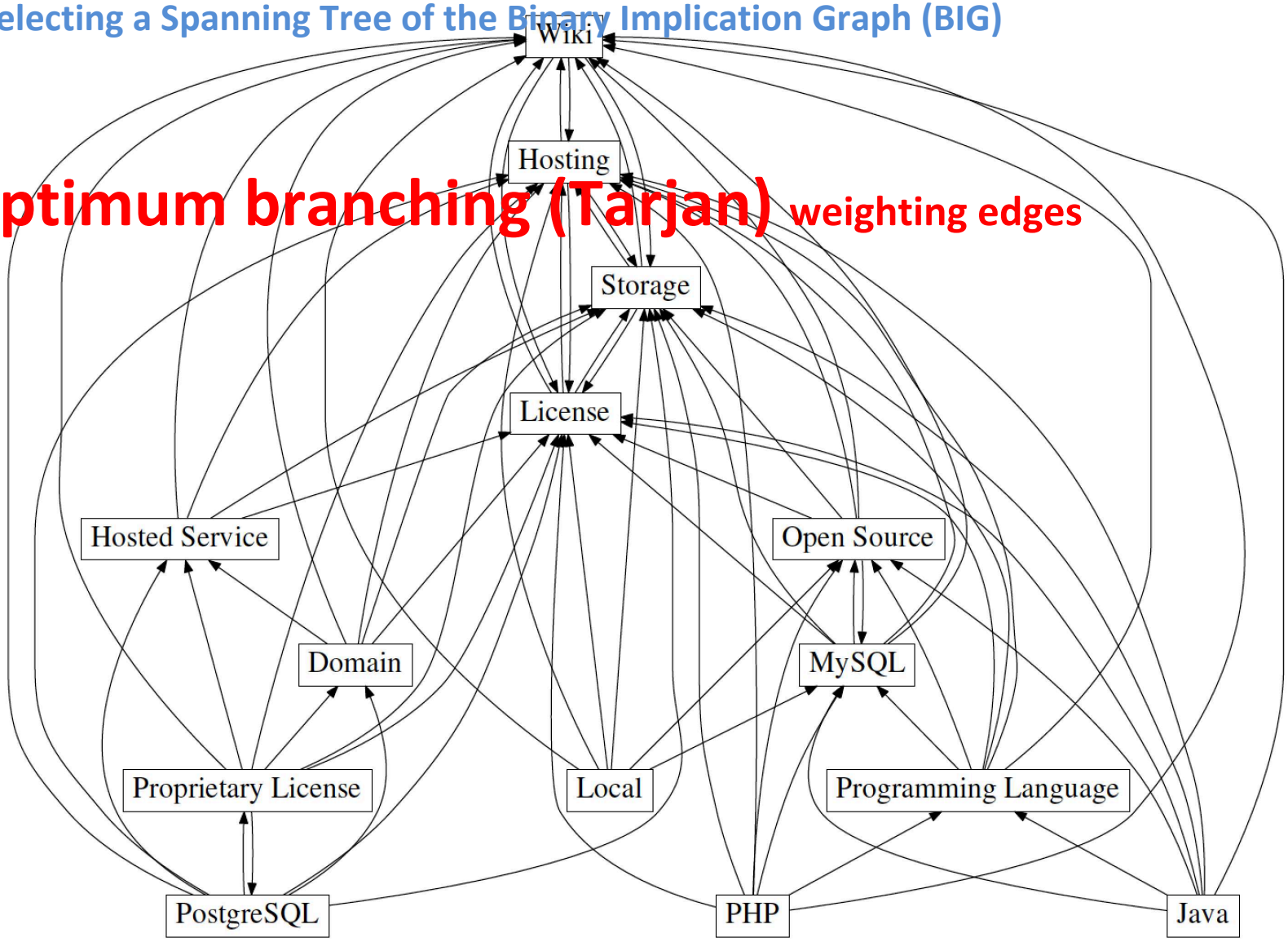
Selecting a Spanning Tree of the Binary Implication Graph (BIG)



Fundamental Problem

Selecting a Spanning Tree of the Binary Implication Graph (BIG)

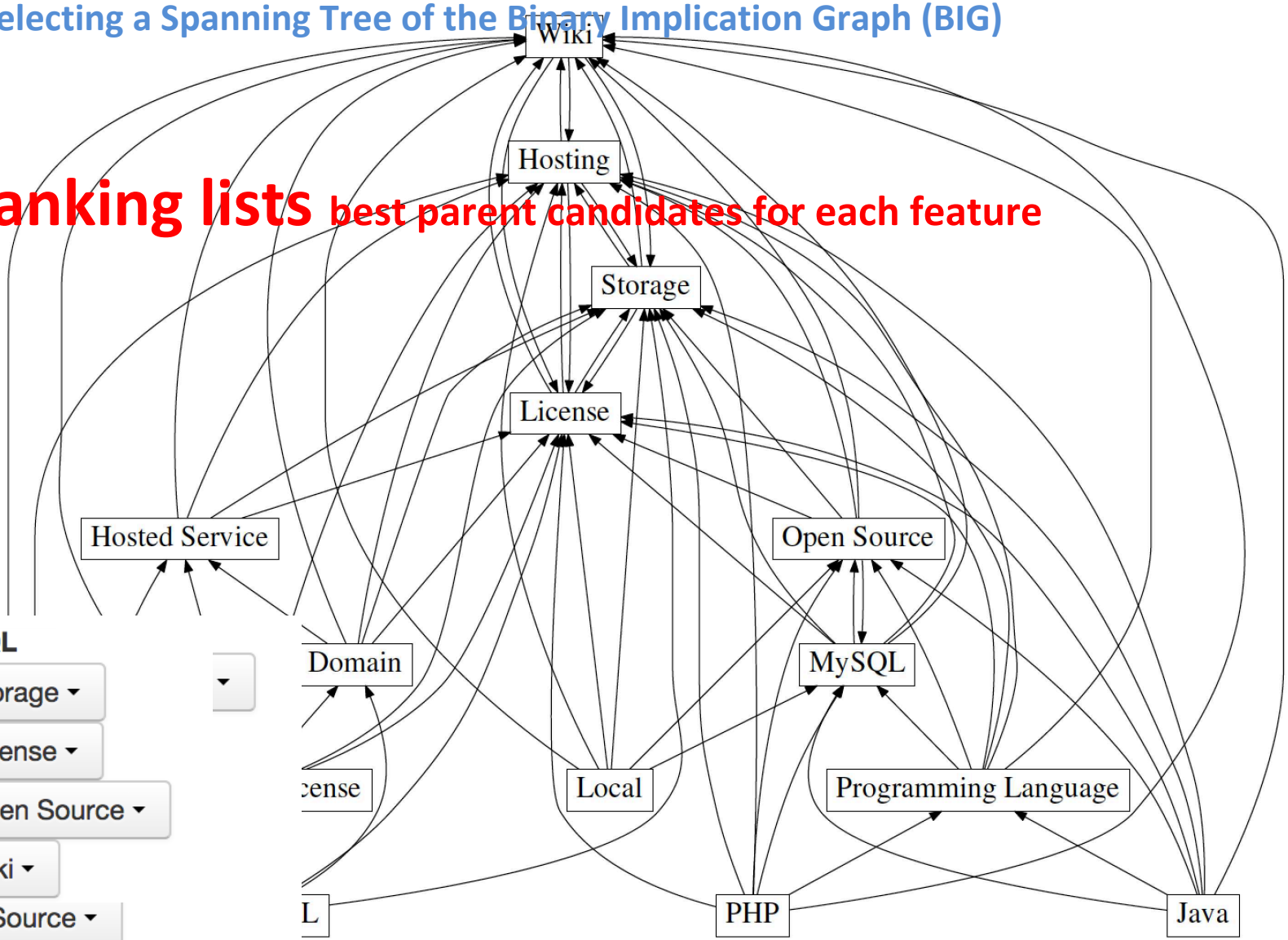
#0 Optimum branching (Tarjan) weighting edges



Fundamental Problem

Selecting a Spanning Tree of the Binary Implication Graph (BIG)

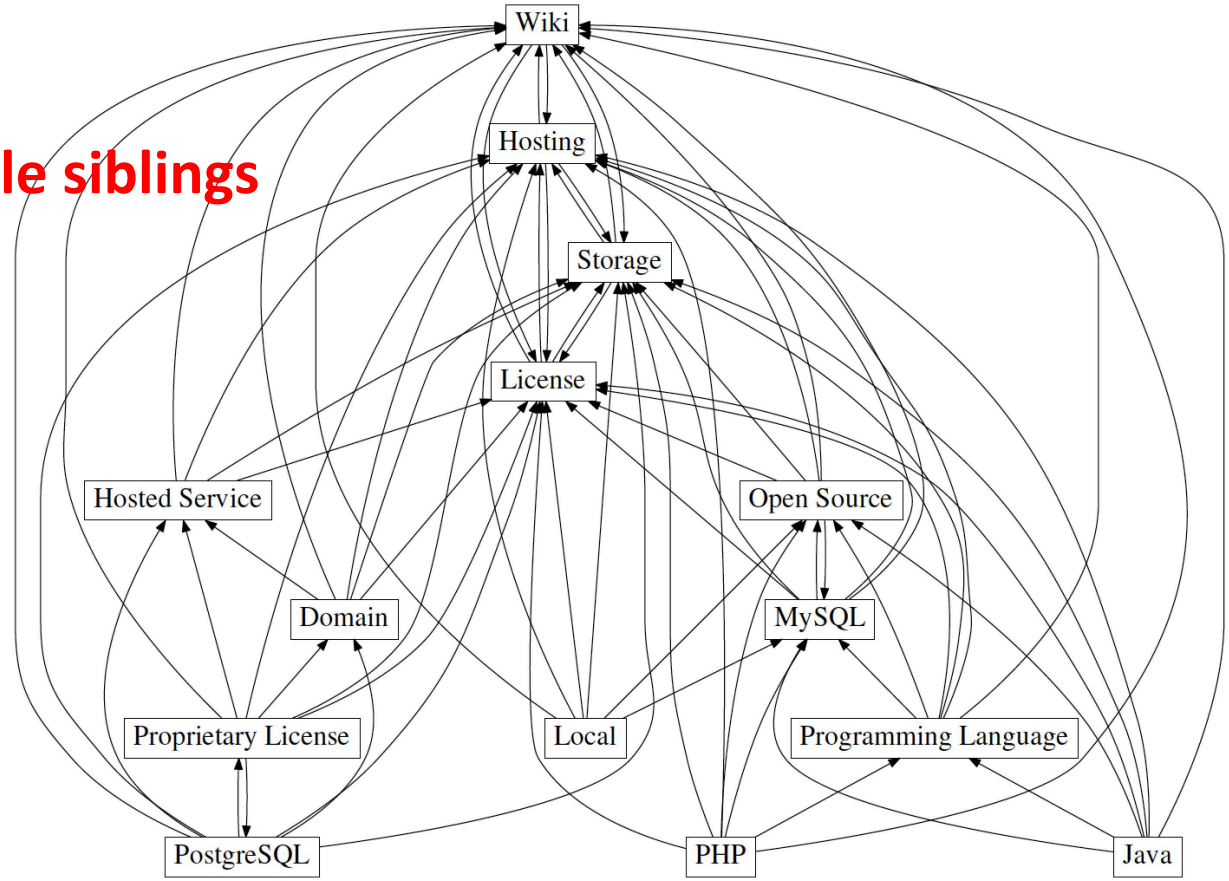
#1 Ranking lists best parent candidates for each feature



Fundamental Problem

Selecting a Spanning Tree of the Binary Implication Graph (BIG)

#2 Clusters ~possible siblings



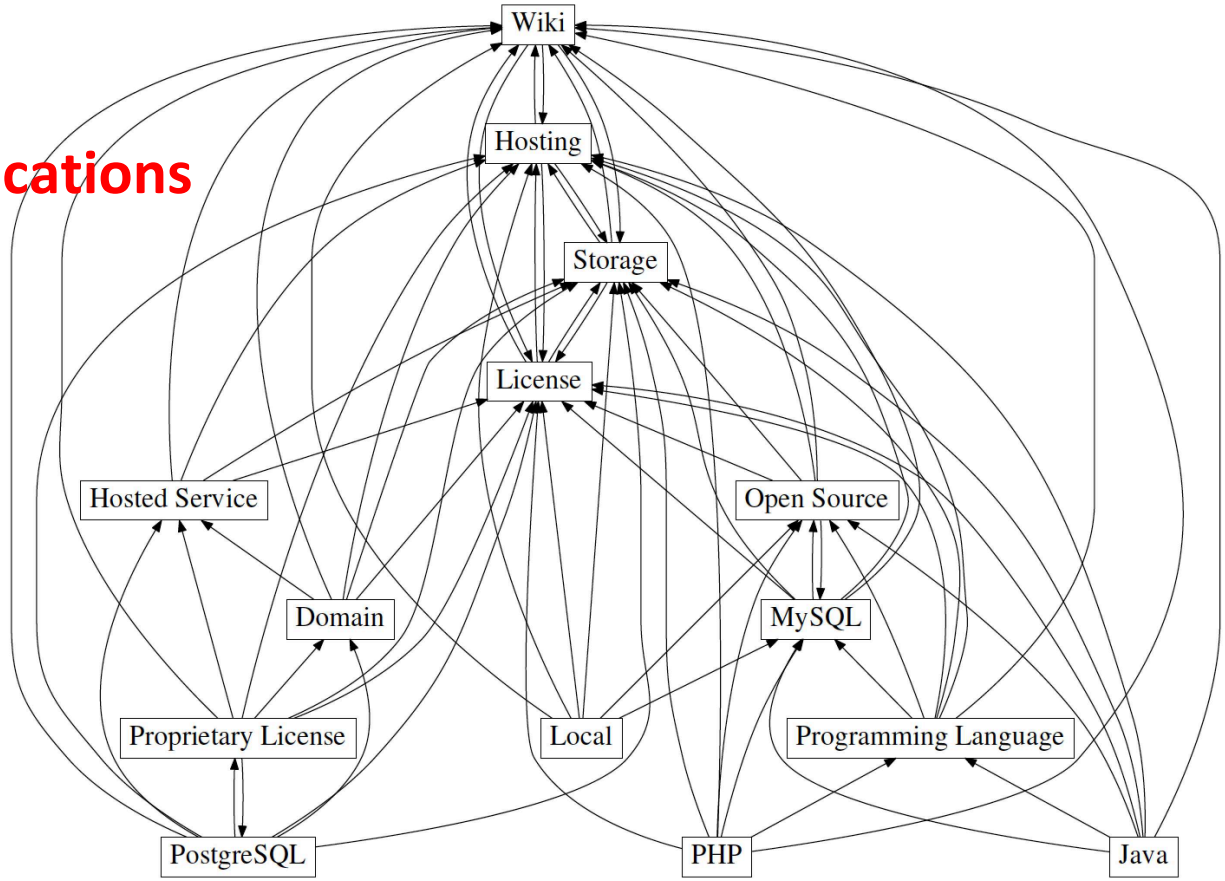
Clusters

- License
- Proprietary License
- PostgreSQL
- MySQL

Fundamental Problem

Selecting a Spanning Tree of the Binary Implication Graph (BIG)

#3 Cliques ~bi-implications

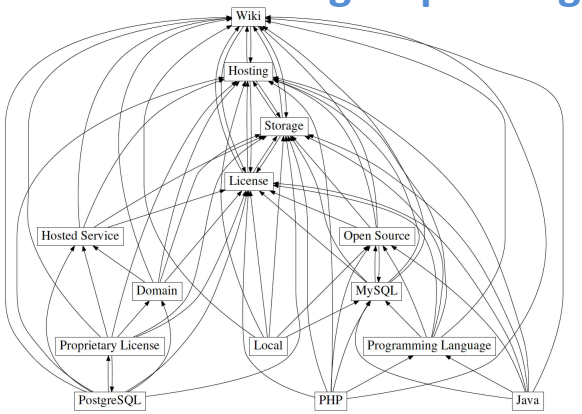


Cliques

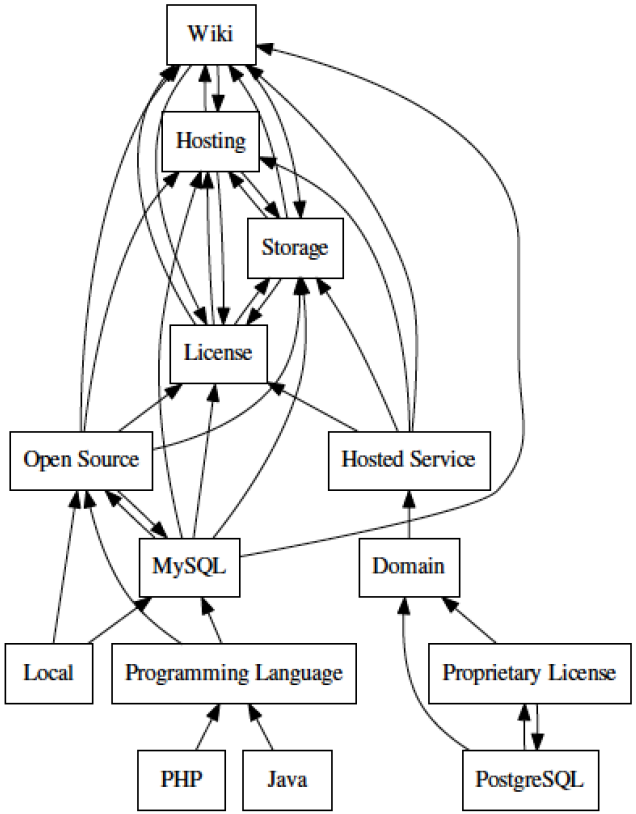
- Storage
- License
- Wiki
- PostgreSQL
- Proprietary License
- MySQL
- Open Source

Fundamental Problem

Selecting a Spanning Tree of the Binary Implication Graph (BIG)

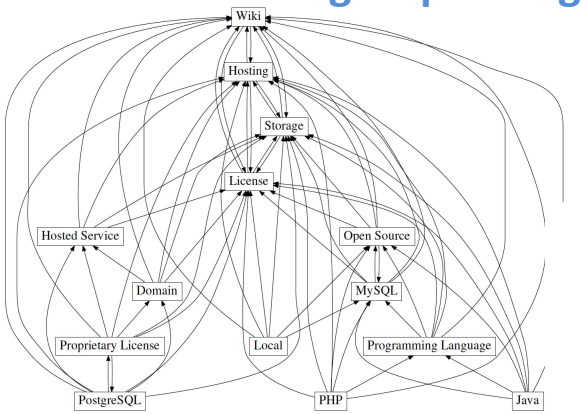


#4 small BIG



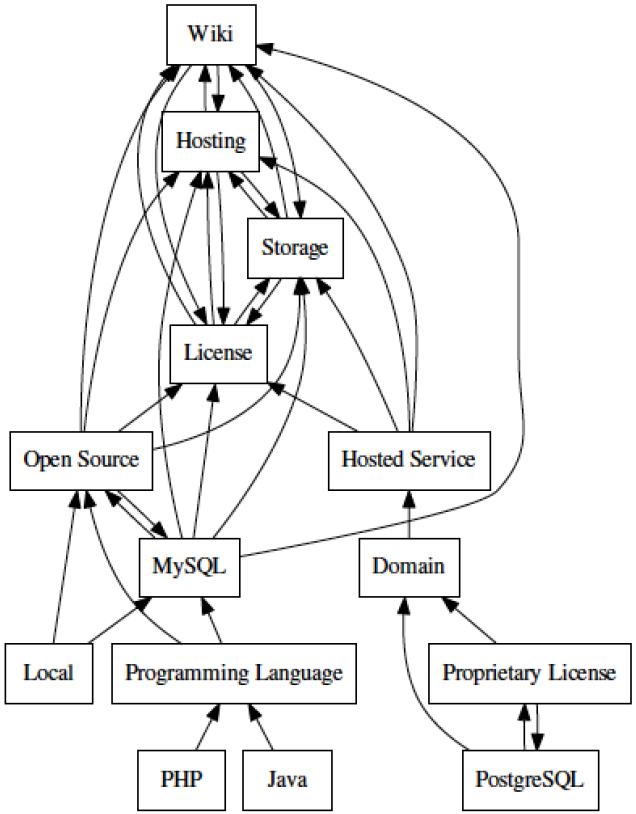
Fundamental Problem

Selecting a Spanning Tree of the Binary Implication Graph (BIG)



#4 reduced BIG

incomplete but dramatically reduce the problem



Ontological Heuristics

- For optimum branching, computing ranking lists and clusters
  - ~ « closedness » of features based on their names

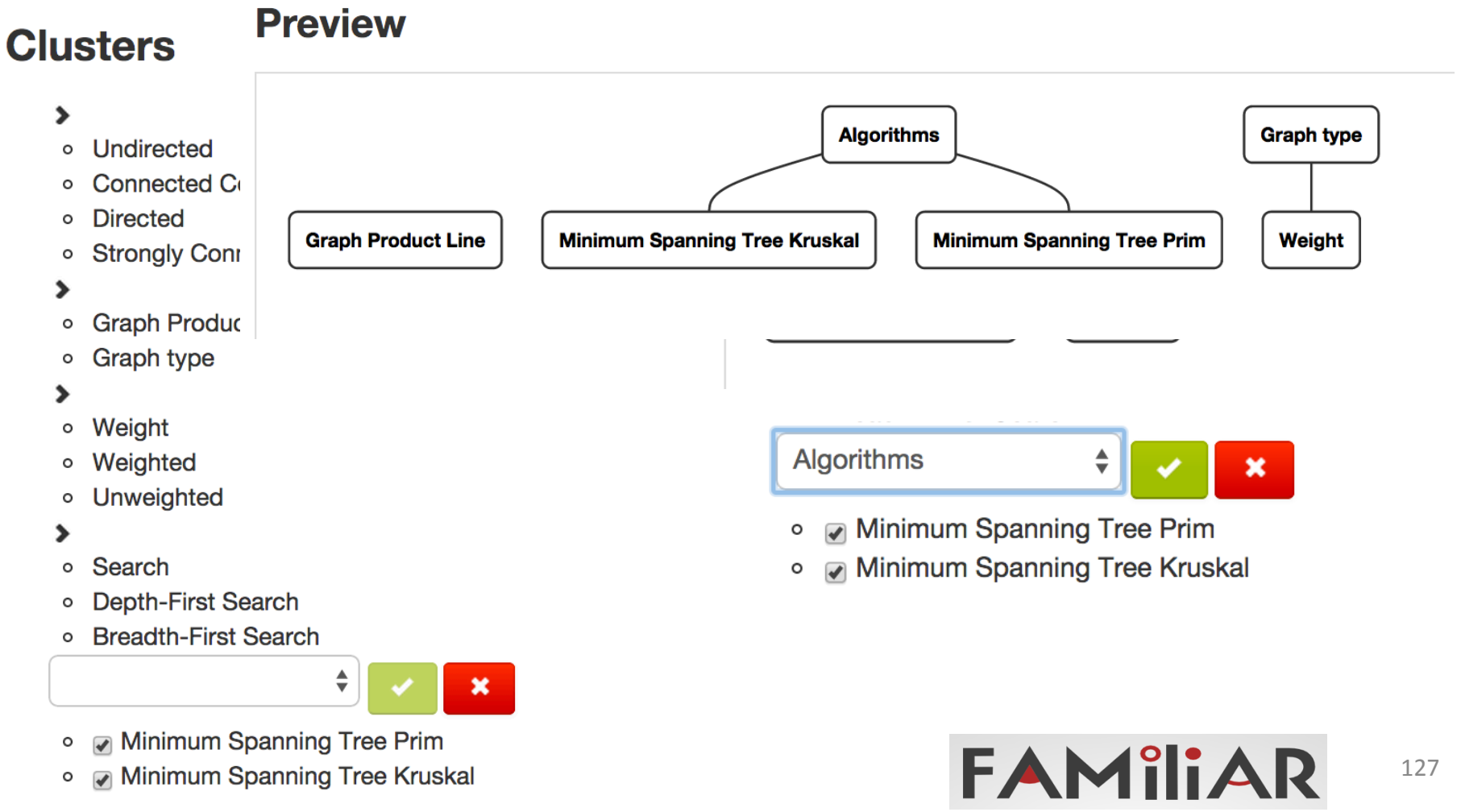
- Syntactical heuristics
  - Smith-Waterman (SW) and Levenshtein (L)
- Wordnet
  - PathLength (PL) and Wu&Palmer (WP)
- Wikipedia Miner offers an API to browse Wikipedia's articles and compute their relatedness
  - Wiktionary (Wikt)

40 GB!

Milne, D.N., Witten, I.H.: An open-source toolkit for mining wikipedia. Artif. Intell. 194, 222{239 (2013)

Guillaume Bécan, Mathieu Acher, Benoit Baudry, and Sana Ben Nasr. Breathing Ontological Knowledge Into Feature Model Synthesis: An Empirical Study (2015). In Empirical Software Engineering (ESE)

WebFML: Synthesizing Feature Models Everywhere (SPLC'14)



She et al.
Reverse Engineering Feature Models ICSE'11 (Linux/eCos/FreeBSD case study)

Configuring FreeBSD:

```
# IPI_PREEMPTION relies on the PREEMPTION option

# Mandatory:
Device apic                # I/O apic

# Optional:
options MPTABLE_FORCE_HTT #enable HTT CPUs ...
options IPI_PREEMPTION
```

Code:

```
MODULE_DEPEND(at91_twi, iicbus, …);
#ifdef A … #endif
```

Features and dependencies are scattered through code and documentation.

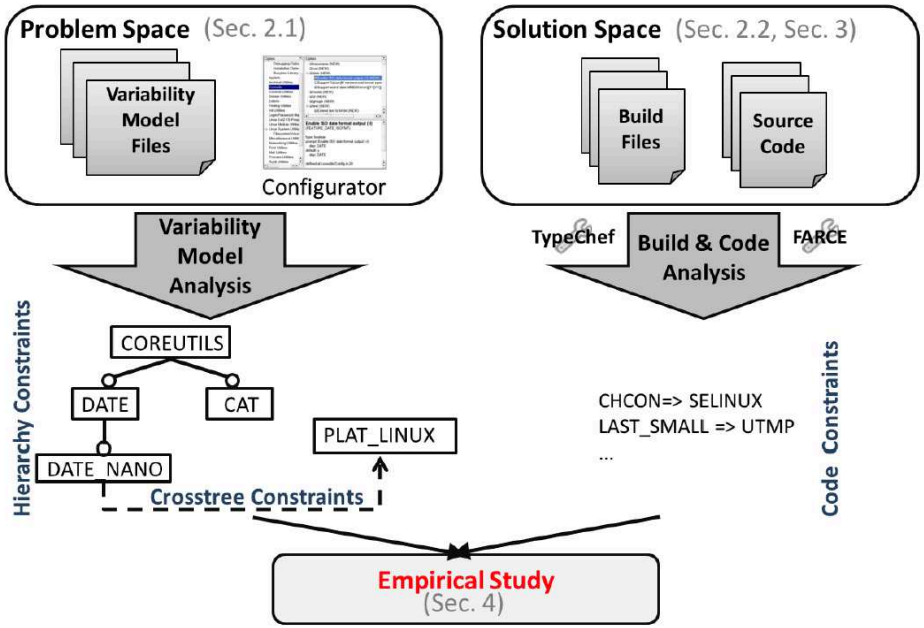
Mining Configuration Constraints: Static Analyses and Empirical Results

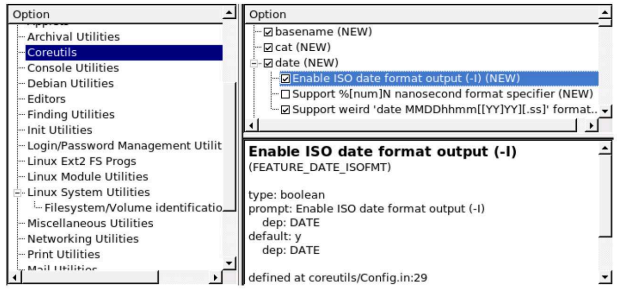
Sarah Nadi
University of Waterloo, Canada

Thorsten Berger
IT University of Copenhagen, Denmark

Christian Kästner
Carnegie Mellon University, USA

Krzysztof Czarnecki
University of Waterloo, Canada

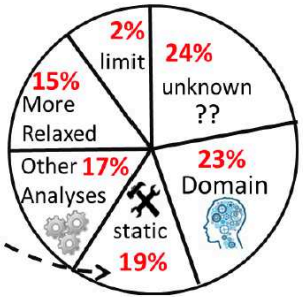




Obj. 1
Accuracy & Scalability
Spec. 1: **93%** accurate
Spec. 2: **77%** accurate

Obj. 2
Recoverability
19 %

Obj. 3
Constraint Classification




```
1 #ifndef Y
2 void foo() { ... }
3 #endif
4
5 #ifdef X
6 void bar() { foo(); }
7 #endif
```

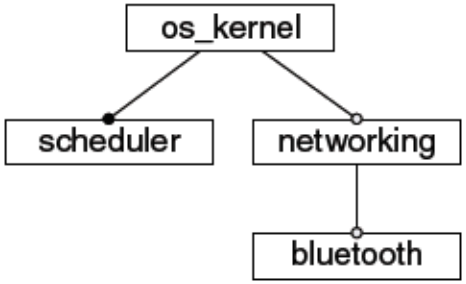
```
1 #if defined(Z)&&defined(X)
2 ...
3 #ifdef W
4 ...
5 #endif
6 ...
7 #endif
```

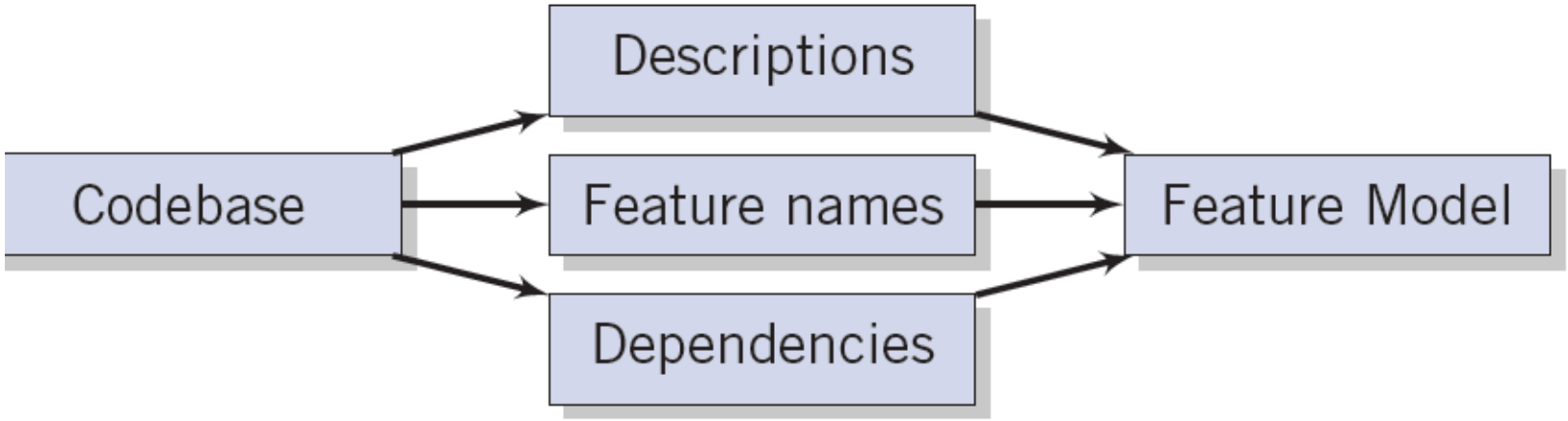
She et al. Reverse Engineering Feature Models ICSE'11

```
#ifdef A
   #ifndef B
      #error ...
    #endif
#endif
```

scheduler ↔ os_kernel
networking → os_kernel
bluetooth → networking

bluetooth is a network driver.





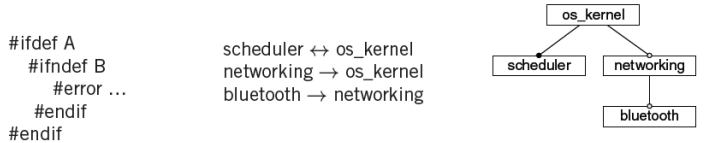
Leverage both names and descriptions for additional domain knowledge.

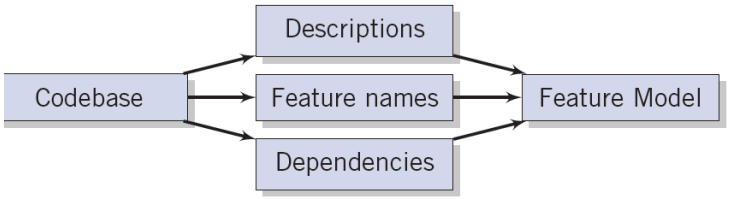
She et al. Reverse Engineering Feature Models ICSE'11

```
#ifdef A
  #ifndef B
    #error ...
  #endif
#endif
```

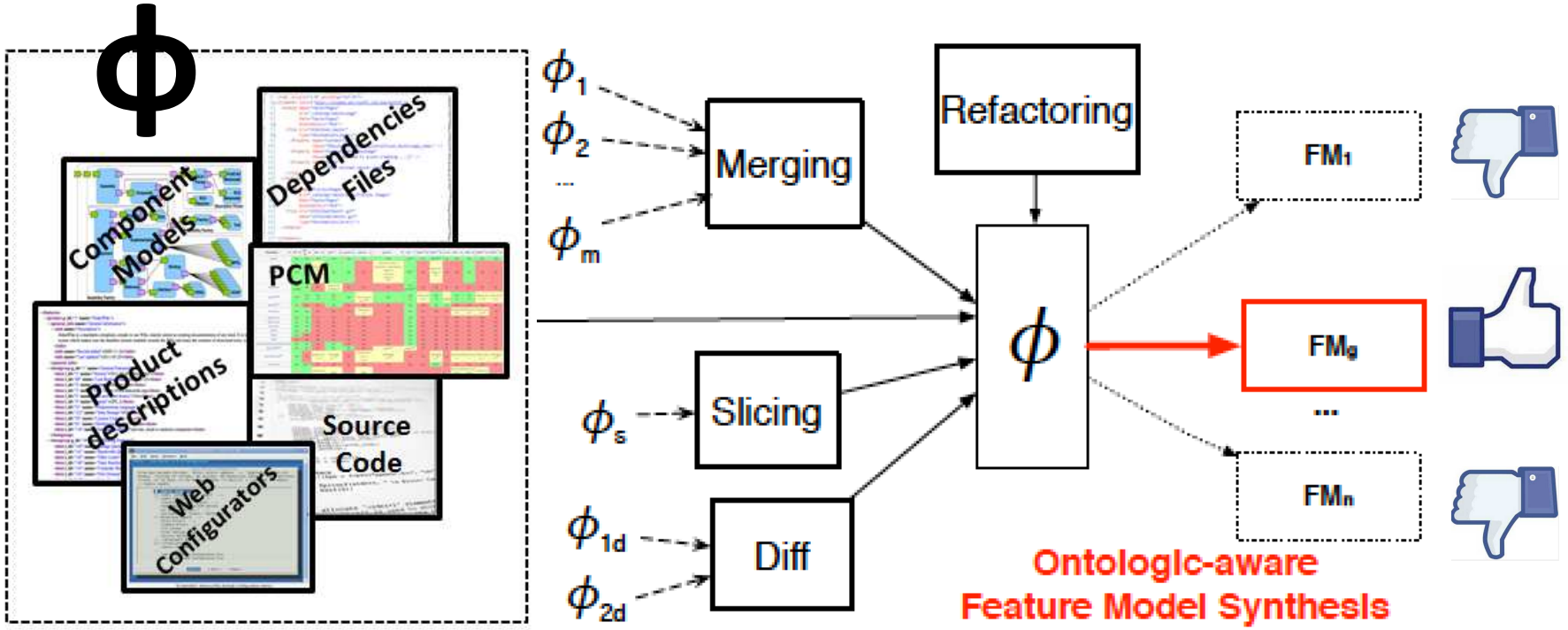
scheduler ↔ os_kernel
networking → os_kernel
bluetooth → networking

bluetooth is a network driver.





Leverage both names and descriptions for additional domain knowledge.



Ontological heuristics (based on feature descriptions)

Feature similarity.

Feature names and descriptions

os_kernel	Operating system.
scheduler	I/O scheduling.
networking	Networking drivers.
ethernet	Type of local area networking.

Selecting a parent for:

bluetooth, a network driver.

Ontological heuristics (based on feature descriptions)

Feature similarity.

Feature names and descriptions

os_kernel	Operating system.
scheduler	I/O scheduling.
networking	Networking **drivers**.
ethernet	Type of local area networking.

Selecting a parent for:

bluetooth, a network **driver**.

Ontological heuristics (based on feature descriptions)

Feature similarity.

Feature names and descriptions

os_kernel	Operating system.
scheduler	I/O scheduling.
networking	**Networking** drivers.
ethernet	Type of local area **networking**.

Selecting a parent for:

bluetooth, a **network** driver.

Ontological heuristics (based on feature descriptions)

Feature similarity.

Ranked list

Feature names and descriptions

1. networking Networking drivers.
2. ethernet Type of local area networking.
3. os_kernel Operating system.
4. scheduler I/O scheduling.

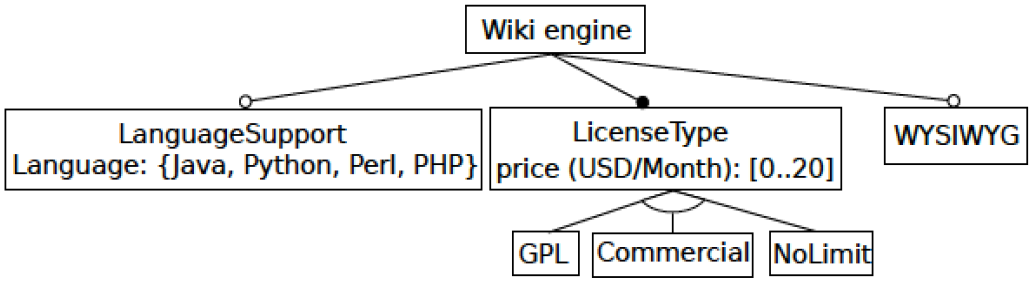
Selecting a parent for:

bluetooth, a network driver.

Empirical results

How many features have their reference parents ranked in the top 5 of our RIFs?

- Linux: 76% of features, eCos: 79% of features.
- Ignoring root features, 90% for Linux and 81% for eCos.
- For incomplete descriptions, At least 50% of words needed for good results (roughly 10 words in Linux).



Commercial <=> LicenseType.price = 10
Commercial <=> Java
GPL => LicenseType.price <= 10
NoLimit <=> ¬LanguageSupport
GPL => LanguageSupport
NoLimit => LicenseType.price >= 10
¬PHP => WYSIWYG
Python => LicenseType.price = 0

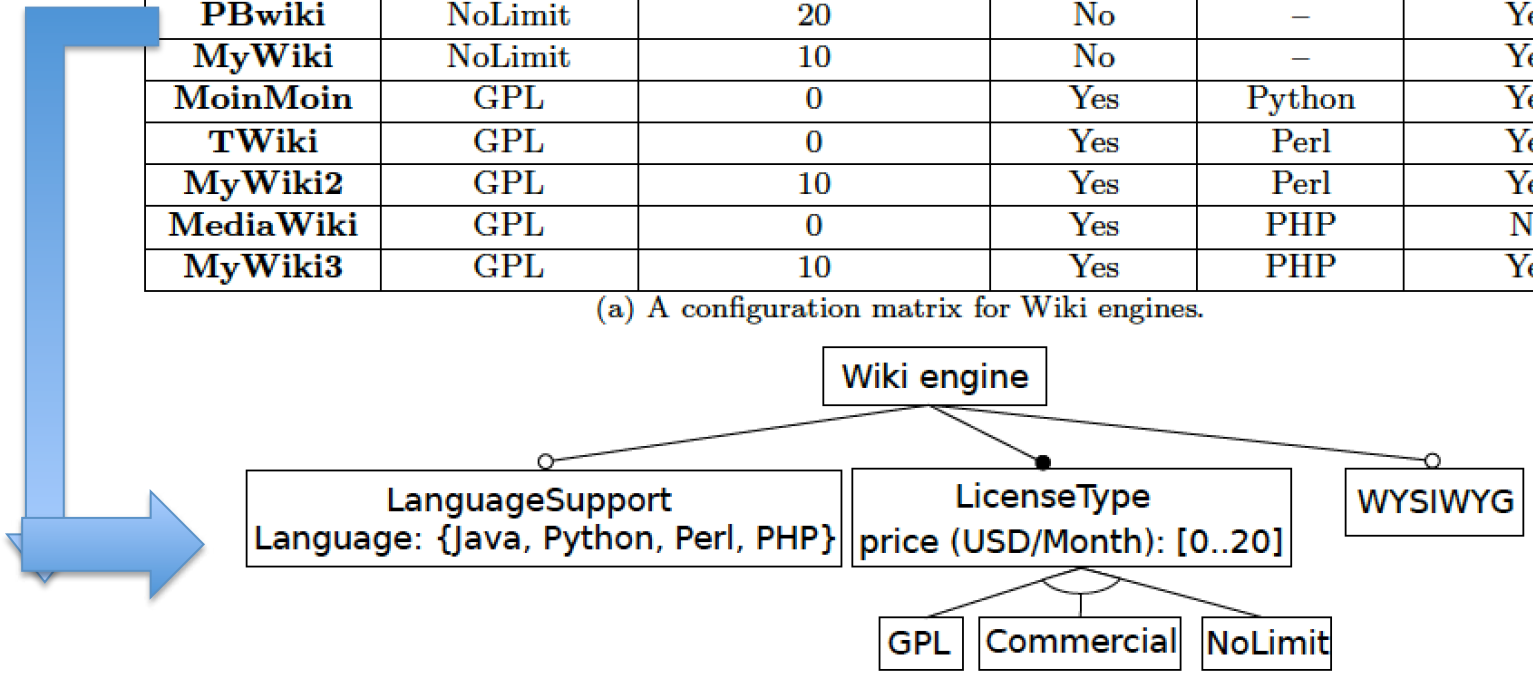
$\Phi = \neg WYSIWYG \iff PHP \wedge LicenseType.price = 0$

Product	License	Price	Language Support	Language	WYSIWIG
W1	Commercial	10	Yes	Java	Yes
W2	NoLimit	20	No		Yes
W3	NoLimit	10	No		Yes
W4	GPL	0	Yes	Python	Yes
W5	GPL	0	Yes	Perl	Yes
W6	GPL	10	Yes	Perl	Yes
W7	GPL	0	Yes	PHP	No
W8	GPL	10	Yes	PHP	Yes

Guillaume Bécan, Razieh Behjati, Arnaud Gotlieb, and Mathieu Acher. Synthesis of Attributed Feature Models From Product Descriptions (2015). In 19th International Software Product Line Conference (SPLC'15) (research track, long paper)

Identifier	LicenseType	LicenseType.price	Language Support	Language	WYSIWYG
Confluence	Commercial	10	Yes	Java	Yes
PBwiki	NoLimit	20	No	–	Yes
MyWiki	NoLimit	10	No	–	Yes
MoinMoin	GPL	0	Yes	Python	Yes
TWiki	GPL	0	Yes	Perl	Yes
MyWiki2	GPL	10	Yes	Perl	Yes
MediaWiki	GPL	0	Yes	PHP	No
MyWiki3	GPL	10	Yes	PHP	Yes

(a) A configuration matrix for Wiki engines.



Commercial <=> LicenseType.price = 10
Commercial <=> Java
GPL => LicenseType.price <= 10
NoLimit <=> ¬LanguageSupport
GPL => LanguageSupport
NoLimit => LicenseType.price >= 10
¬PHP => WYSIWYG
Python => LicenseType.price = 0

$\Phi = \neg WYSIWYG <=> PHP \wedge LicenseType.price = 0$

(b) An attributed feature model representing the configuration matrix in Figure 2(a)

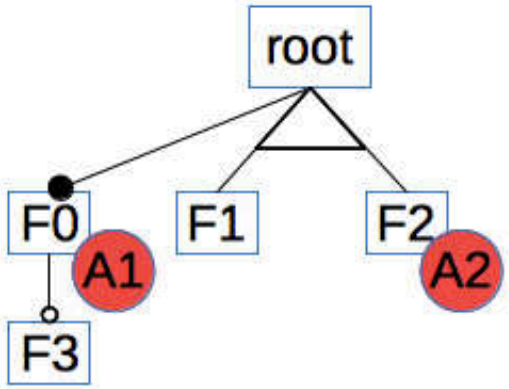
Guillaume Bécan, Razieh Behjati, Arnaud Gotlieb, and Mathieu Acher. Synthesis of Attributed Feature Models From Product Descriptions (2015). In 19th International Software Product Line Conference (SPLC'15) (research track, long paper)

Upload a Matrix | Features Attributes | Hierarchy | Feature Groups | Constraints | AFM

Next

#	root Feature Null value	F0 Feature Null value	F1 Feature Null value	F2 Feature Null value	F3 Feature Null value	A1 Attribute 0	A2 Attribute 0
0	Yes	Yes	Yes	No	Yes	3	0
1	Yes	Yes	No	Yes	Yes	2	2
2	Yes	Yes	Yes	No	No	2	0
3	Yes	Yes	No	Yes	No	0	8

root	F0	F1	F2	F3	A1	A2
Yes	Yes	Yes	No	Yes	3	0
Yes	Yes	No	Yes	Yes	2	2
Yes	Yes	Yes	No	No	2	0
Yes	Yes	No	Yes	No	0	8

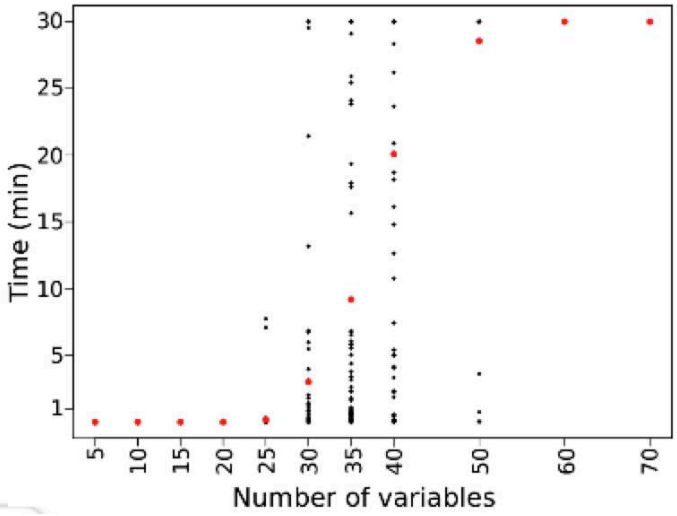


A2 < 8 => A1 >= 2
A1 > 0 => A2 <= 2

Scalability

Random dataset

- Generator of configuration matrices
  - Number of variables (features + attributes)
  - Number of configurations
  - Maximum domain size (number of distinct values in a column)
- Execution time of or-group computation



 = default heuristics only

- 1000 configurations
- max domain size of 10

Timeout always reached with more than 60 variables

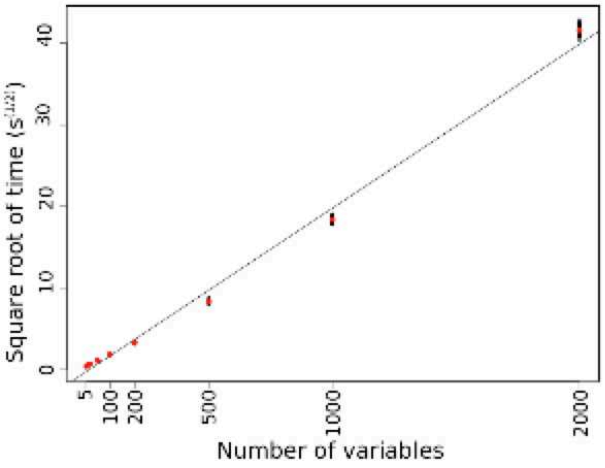
Or groups do not scale !

Scalability

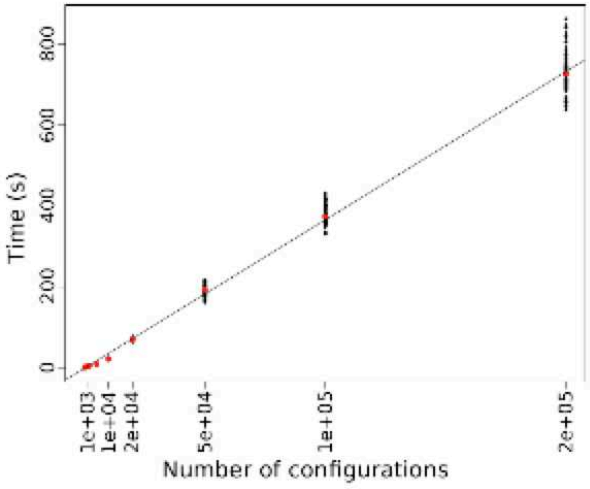
Random dataset

- Execution time (no or-groups)

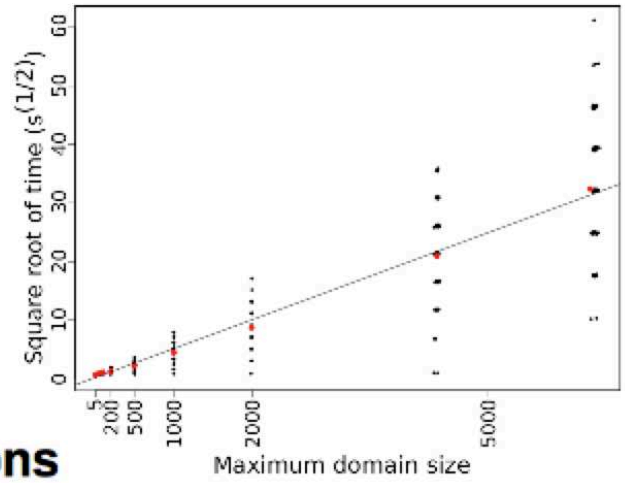
= default heuristics only



- **Up to 2,000 variables**
- 1,000 configurations
- Max domain size of 10



- 100 variables
- **Up to 200,000 configurations**
- Max domain size of 10



- 10 variables
- 10,000 configurations
- **Up to 6000 distinct values**

On all experiments:
Average: 2.6 min
Max: 62 min

Scalability

Best Buy dataset

- 242 matrices
- < 25% of empty cells
- Interpretation of empty cells

= default heuristics only

Media Card Reader	Yes	Yes	Yes
Number Of Ethernet Ports	1	1	
Number Of HDMI Outputs	1	1	1
Number Of USB Port(s)	3	3	3
Number Of VGA Ports	1		
Operating System	Windows 8.1	Windows 8.1	Windows 8.1

Execution time of 2.1s for the most challenging matrix:

- 77 variables
- 185 configurations
- Maximum domain size of 185

Execution time is similar to the random dataset

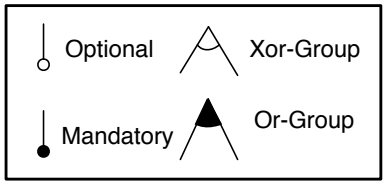
Composing and Decomposing Feature Models

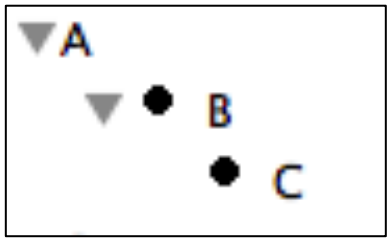
(advanced topics)

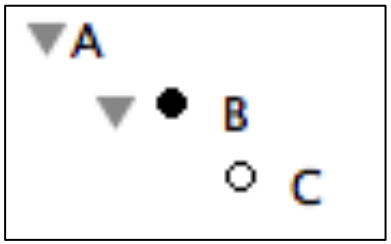
mergeNonPC.fml

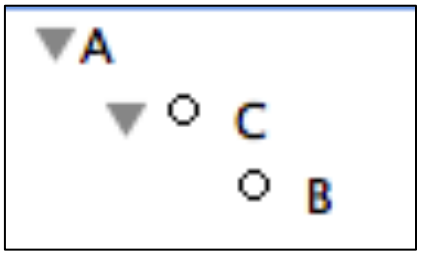
Merging operation (2)

```
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}}

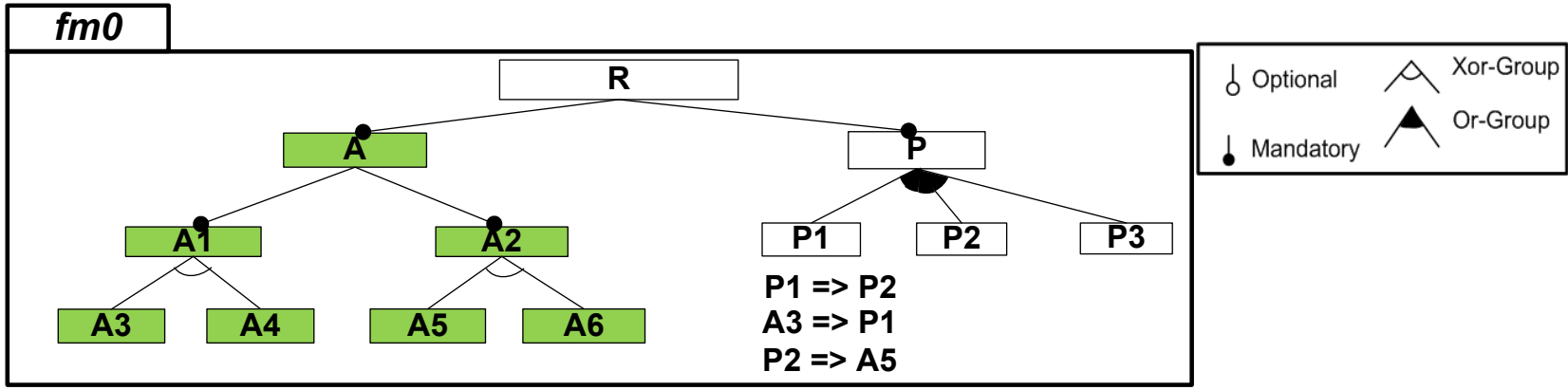
?

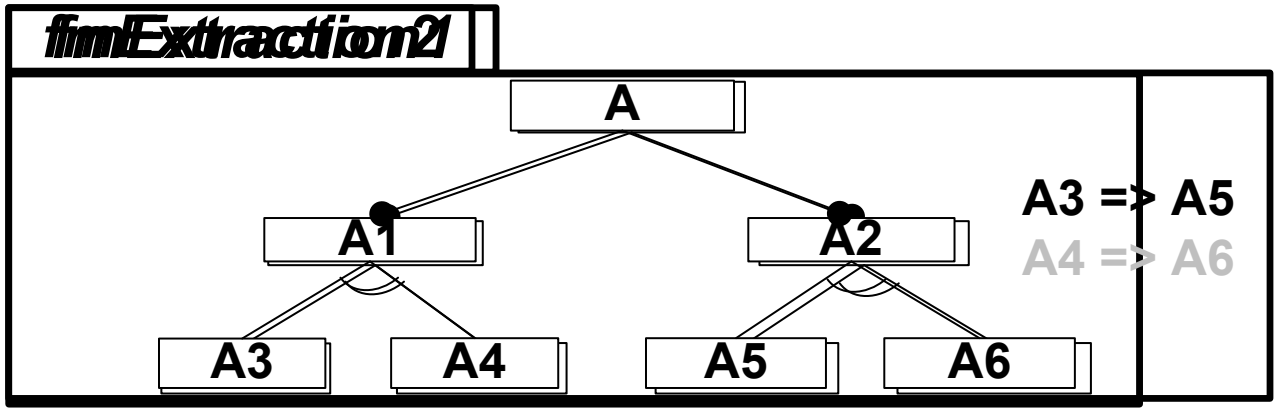
Building “views” of a feature model

- Problem: given a feature model, how to decompose it into smaller feature models?

- Semantics?
  - What’s the hierarchy
  - What’s the set of configurations?

A first try



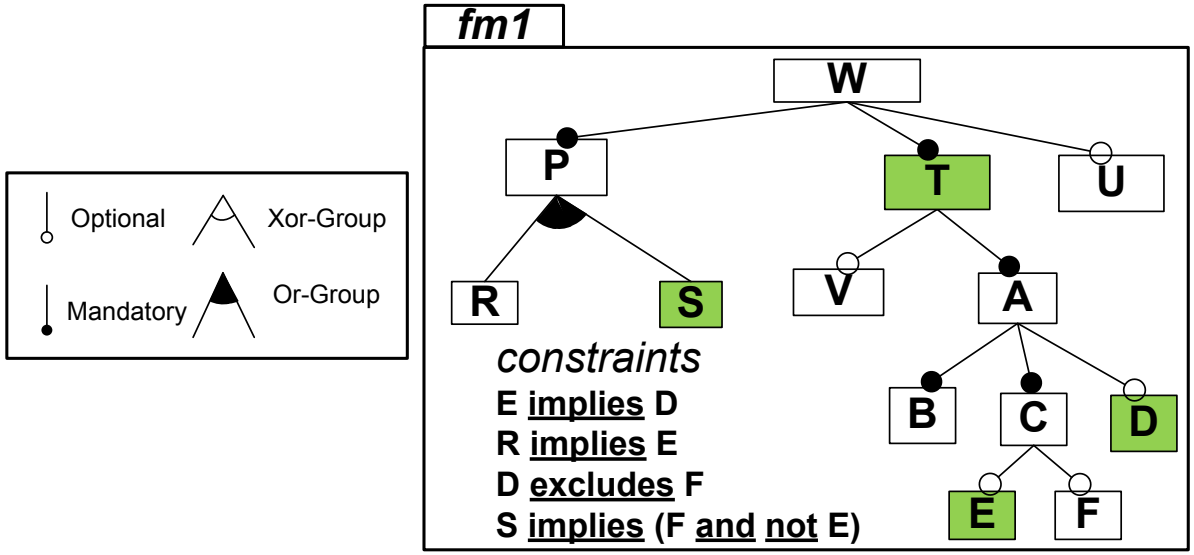


Problem: You can select **A3** without **A5**

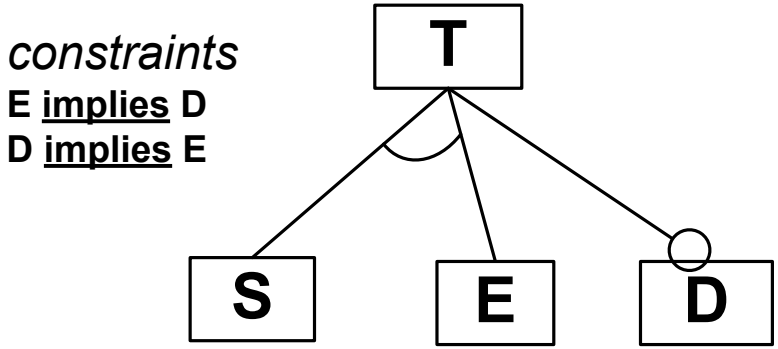
Hierarchy and Configuration matter!

Slicing Operator

slicing criterion : an arbitrary set of features, relevant for a feature model user

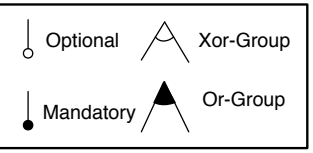


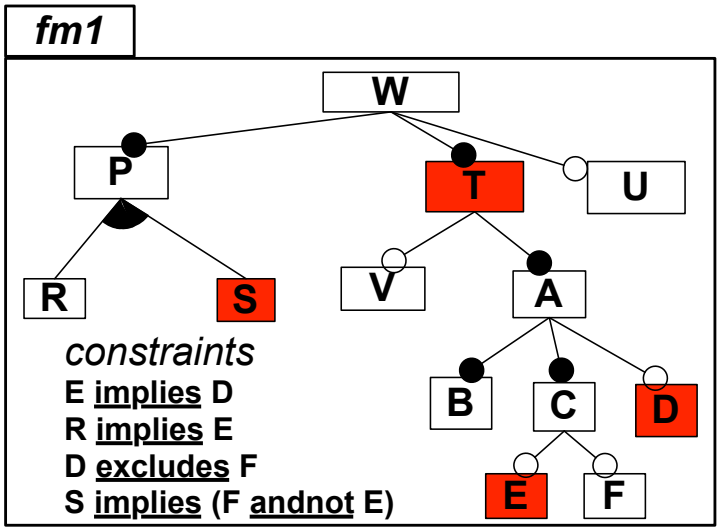
slice : a new feature model, representing a projected set of configurations



Slicing operator: going into details

projected set of configurations



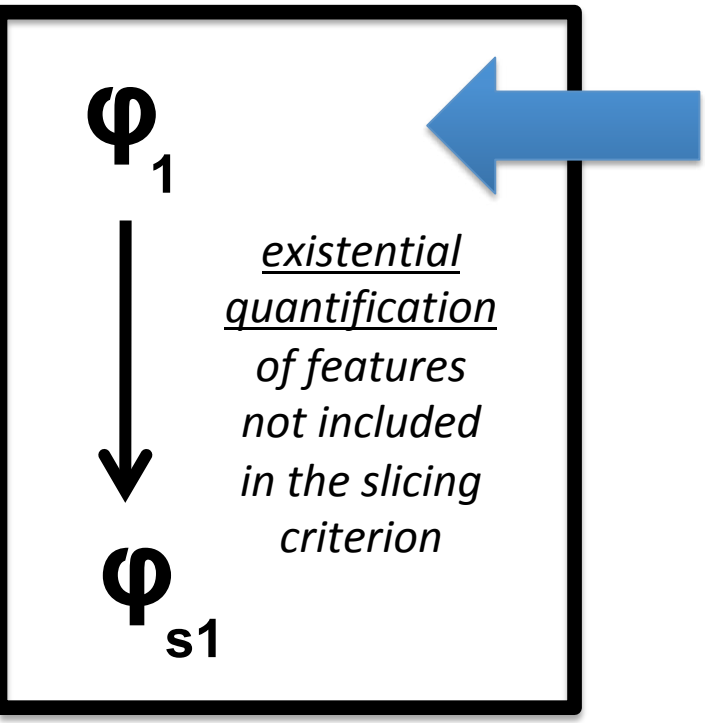


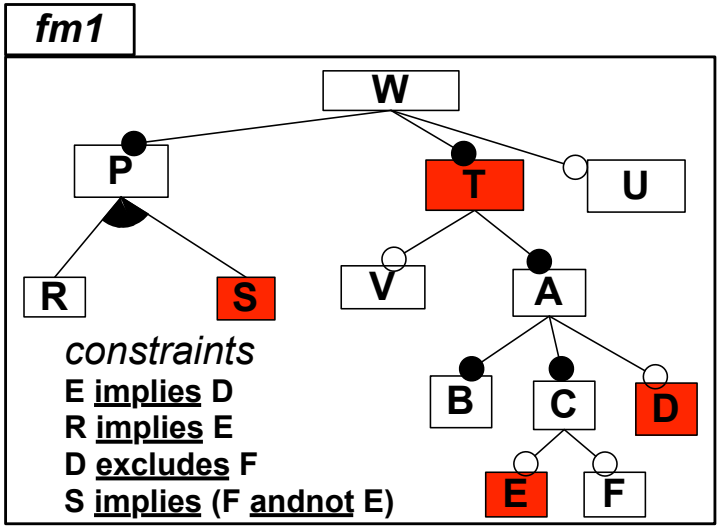
fm1 = {
{A,B,C,D,E,P,R,T,U,W},
{A,B,C,P,S,T,U,W},
{A,B,C,D,E,P,R,T,W},
{A,B,C,P,S,T,V,W},
{A,B,C,P,S,T,U,V,W},
{A,B,C,P,S,T,W},
{A,B,C,D,E,P,R,T,V,W},
}

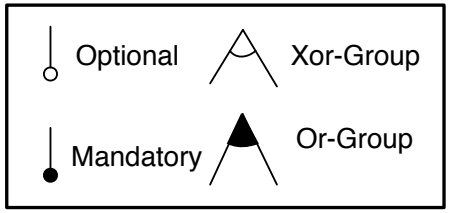
fm1p = {
{D,E,T},
{S,T},
{D,E,T},
{S,T},
{S,T},
{S,T},
{D,E,T}
}

Slicing operator: going into details

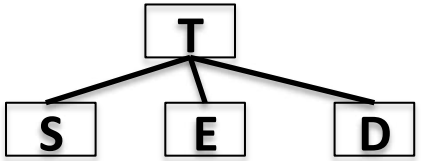
synthesizing the corresponding feature model



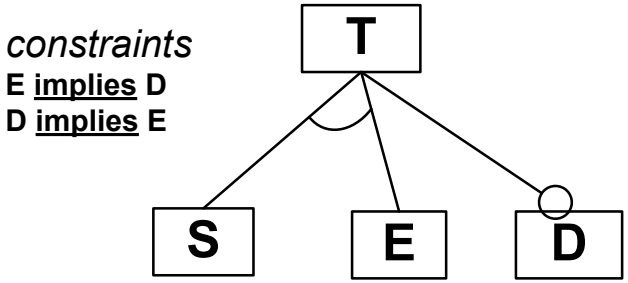




+



fm1p = {
{D,E,T},
{S,T}
}

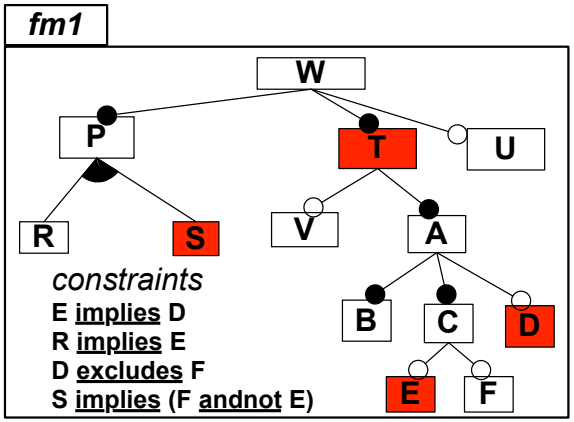


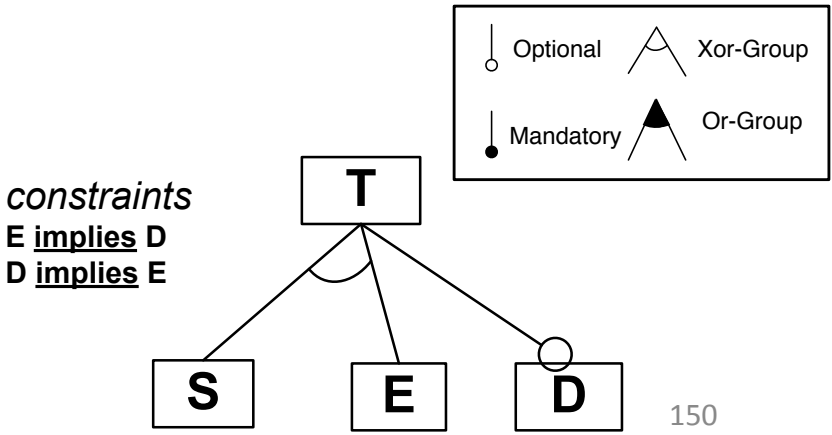
Mathieu Acher, Philippe Collet, Philippe Lahire, Robert B. France « Separation of Concerns in Feature Modeling: Support and Applications » AOSD'12

slicingOp2.fml

Slicing operator with FAMILIAR (1)

```
fm1 = FM (W : P T [U] ; T : [V] A ;
                          A : B C [D] ;
                           C : [E] [F] ;
                           P : (R|S)+ ;
                           E implies D ; R implies E ;
                           S implies (F and !E) ; D implies !F ; )

fm2 = slice fm1 including { S T E D }
fm2bis = slice fm1 excluding { W P R V A B C F U }

cmp = compare fm2 fm2bis
assert (cmp eq REFACTORING)
```



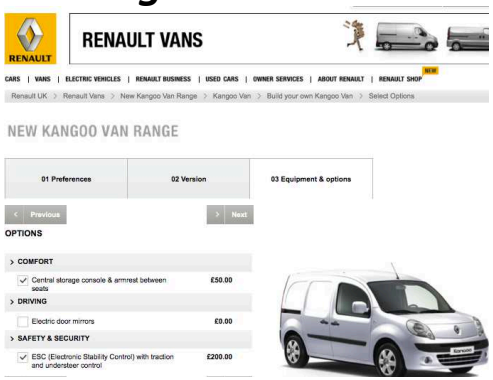


slicingOp.fml

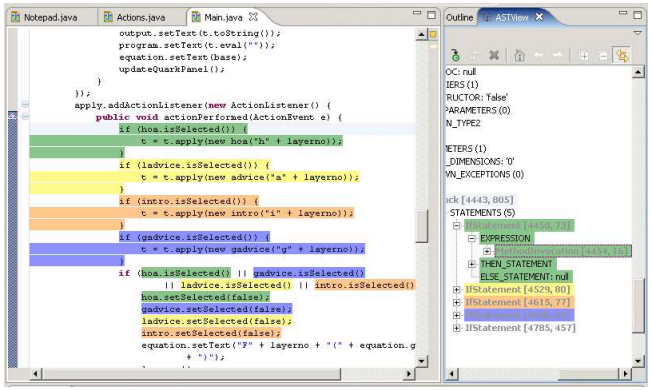
Slicing with FAMILIAR (2)

```
fm1 = FM (W : P T [U] ; T : [V] A ;
                          A : B C [D] ;
                           C : [E] [F] ;
                           P : (R|S)+ ;
                           E implies D ; R implies E ;

                           S implies (F and !E) ; D implies !F ; )

fm2 = slice fm1 including fm1.A.* ++ { fm1.A }

fm3 = slice fm1 including fm1.P.* ++ { fm1.P }
//fm3bis = slice fm1 including { fm1.P fm1.R fm1.S } // equivalent to fm3

fm4 = slice fm1 including { fm1.E fm1.D fm1.F }

fts5 = { fm1.P fm1.W } ++ fm1.P.*
fm5 = slice fm1 including fts5
```

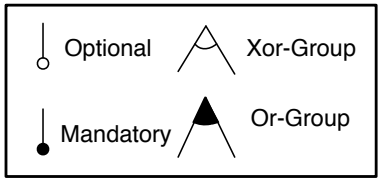
From marketing, customers, product management

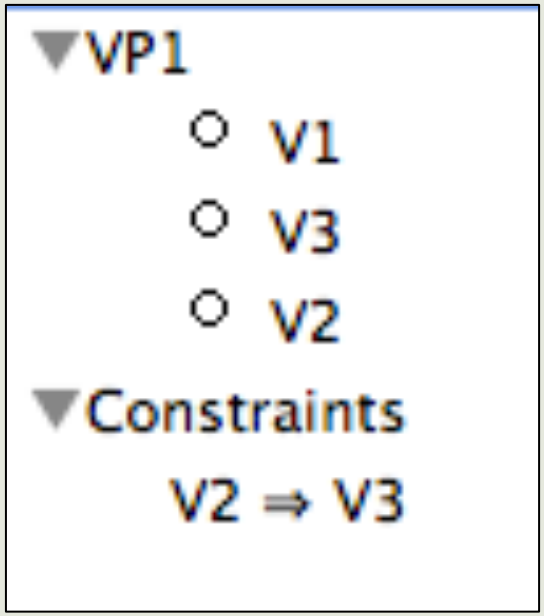


From existing software assets (technical variability)



Metzger, Heymans et al. "Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis" (RE'07)



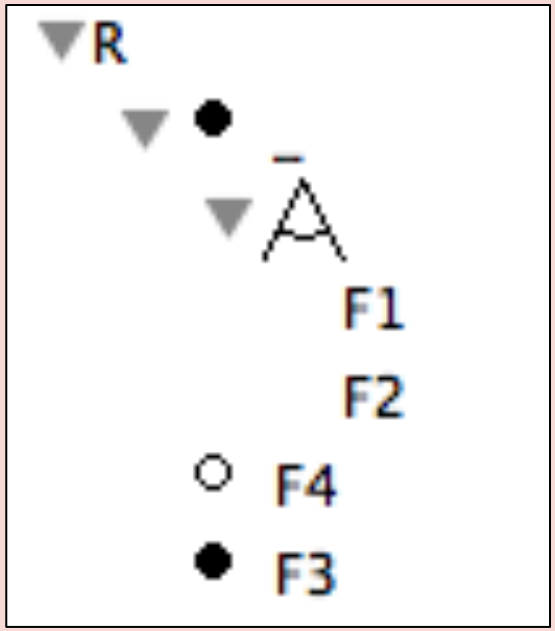


From marketing, customers, product management

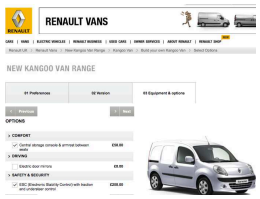
usefulness

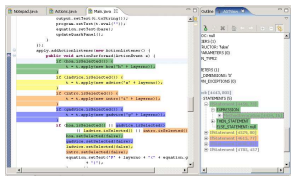
V1 ⇔ f1
V2 ⇔ f2
V3 ⇔ f3

realizability

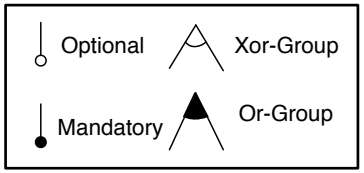


From existing software assets

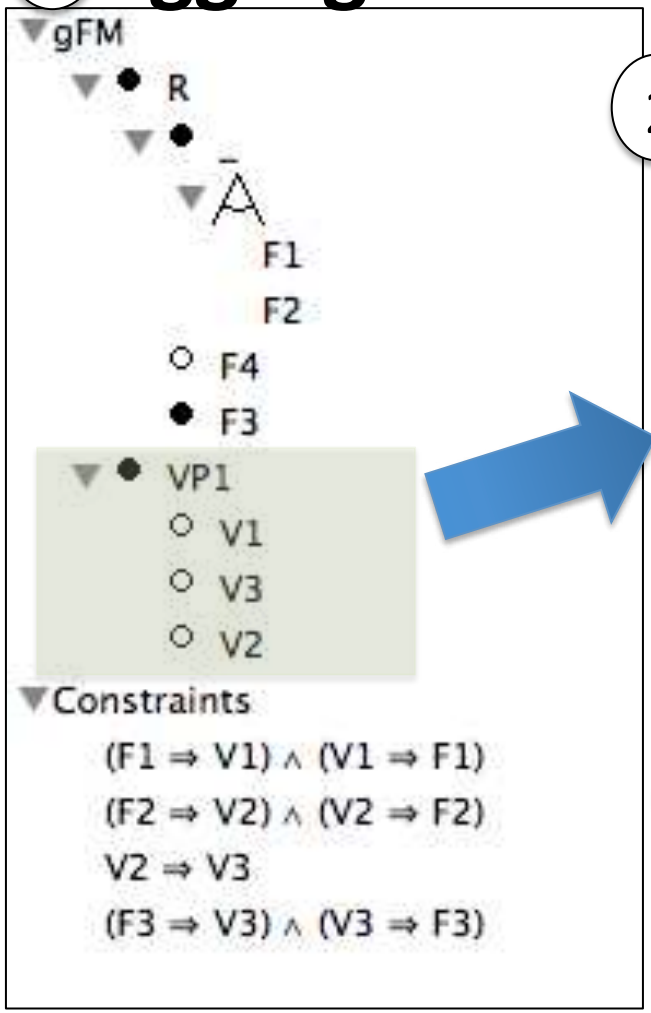




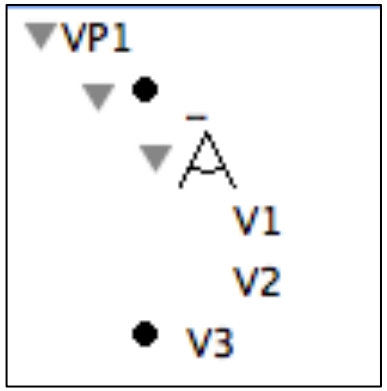
Realizability checking



1 **aggregate**



2 **slice** (“realizable part”)



φ

3 **compare**

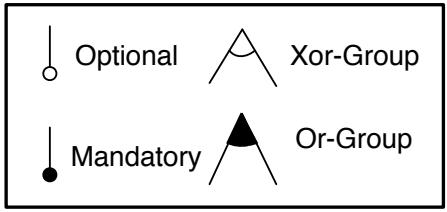
{{V1,V3,V2,VP1},
{V1,VP1},
{V3,VP1},
{VP1}}

4 **merge diff**

(“unrealizable products”)

Mathieu Acher, Philippe Collet, Philippe Lahire, Robert B. France « Separation of Concerns in Feature Modeling: Support and Applications » AOSD’12

Revisiting Merge: Aggregate + Slice



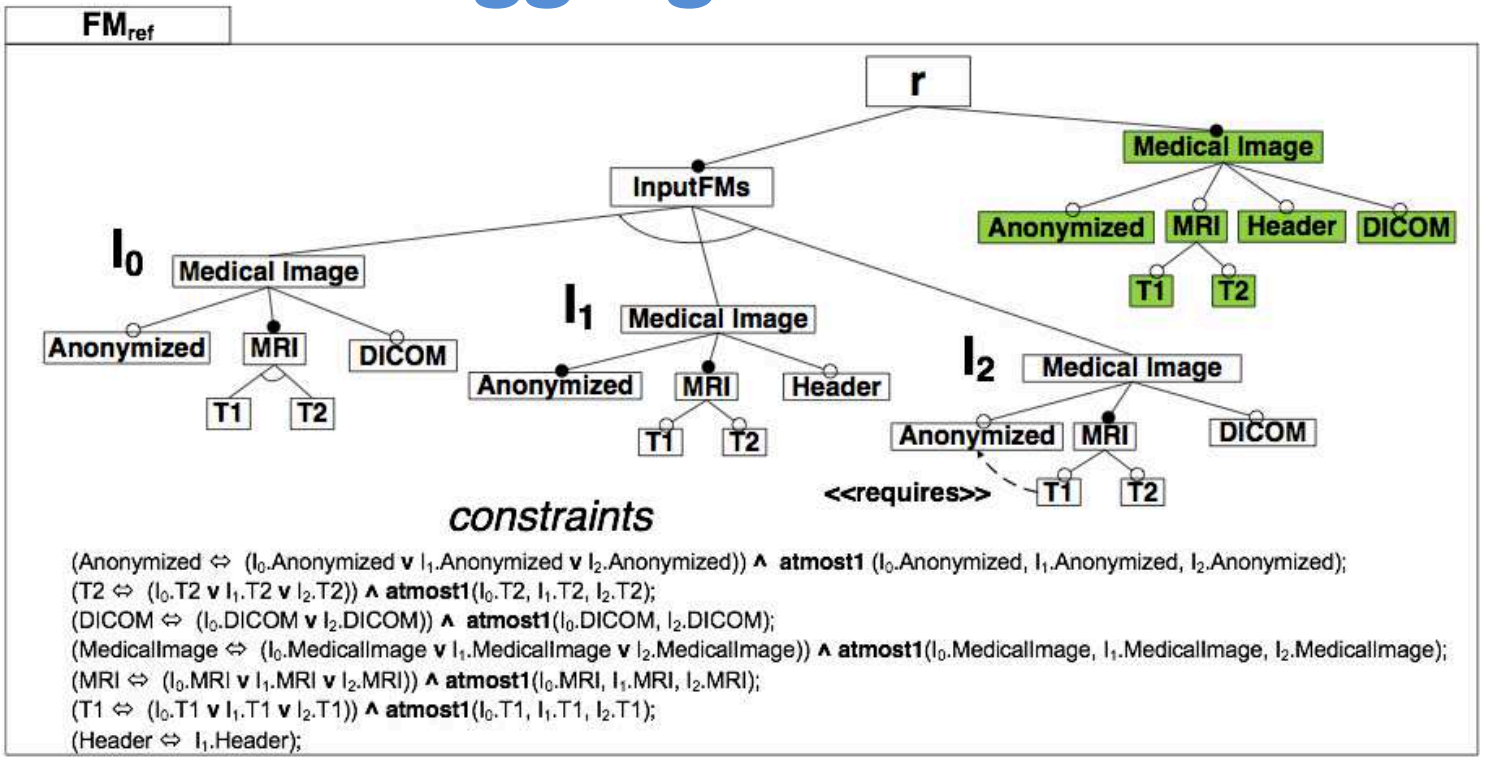
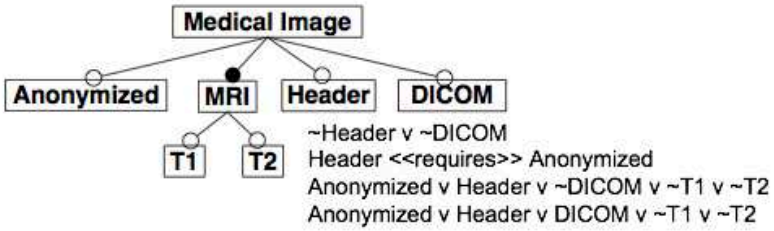


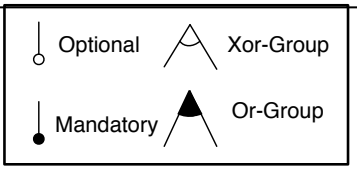
Figure 7.10: Merge of three feature models, I_0, I_1 and I_2 using the slicing operator



mergeWithAggregateMI.fml

Revisiting Aggregate, Merge and Slice:

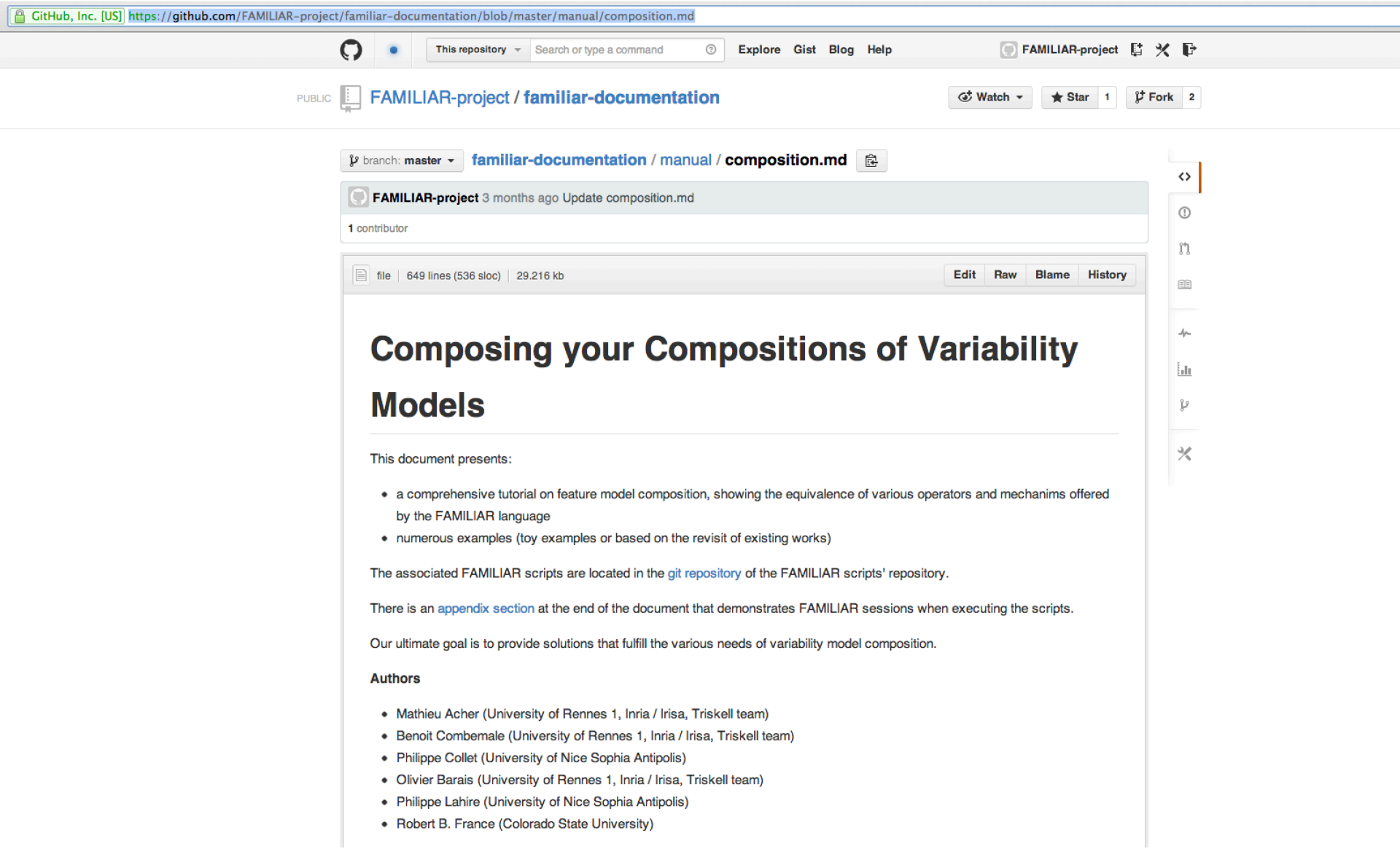
```
fmsupp1 = FM (MedicalImage : [Anonymized] MRI [DICOM] ; MRI : (T1
fmsupp2 = FM (MedicalImage : Anonymized MRI [Header] ; MRI : [T1]
fmsupp3 = FM (MedicalImage : [Anonymized] MRI [DICOM] ; MRI : [T1

fmSupp = merge sunion fmsupp*
fmSuppAgg = aggregateMerge sunion fmsupp*

fmSuppAggSlice = slice fmSuppAgg including fmSupp.*
```




```
fml> compare fmSuppAggSlice fmSupp
res4: (STRING) REFACTORING
```

- MergeCST
  - MedicalImage
    - DICOM
    - Anonymized
    - Header
    - MRI
      - T1
      - T2
  - InputFMs
    - MedicalImage1000
      - DICOM1000
      - Anonymized1000
      - MRI1000
        - T11000
        - T21000
    - MedicalImage1001000
      - Anonymized1001000
      - MRI1001000
        - T21001000
        - T11001000
      - DICOM1001000
    - MedicalImage2001000
      - MRI2001000
        - T12001000
        - T22001000
      - Header2001000
      - Anonymized2001000
- Constraints
  - (Anonymized1000 ∨ Anonymized1001000 ∨ Anonyr
  - (MedicalImage2001000 ∨ MedicalImage1000 ∨ Medi
  - (DICOM1001000 ∨ DICOM1000 ⇒ DICOM) ∧ (DICOM
  - (T21000 ∨ T21001000 ∨ T22001000 ⇒ T2) ∧ (T2 ⇒
  - T11001000 ⇒ Anonymized1001000
  - (Header2001000 ⇒ Header) ∧ (Header ⇒ Header200
  - (T12001000 ∨ T11000 ∨ T11001000 ⇒ T1) ∧ (T1 ⇒
  - (MRI1001000 ∨ MRI2001000 ∨ MRI1000 ⇒ MRI) ∧ (M

Mathieu Acher, Benoit Combemale, Philippe Collet, Olivier Barais, Philippe Lahire, Robert B. France « Composing your Compositions of Variability Models » MODELS'13



Mathieu Acher, Benoit Combemale, Philippe Collet, Olivier Barais, Philippe Lahire, Robert B. France « Composing your Compositions of Variability Models » MODELS'13

Conclusion

- Implementing variability (code clones, preprocessor, generative programming, …, DSLs)
- Modeling variability
  - Syntax and semantics of feature models; synthesis (from semantics to syntax)
  - Reasoning and operations for feature models
- Reverse engineering variability
  - Code base, textual inputs, tabular data, configuration files, “variants”