Modeling Variability

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

UNIVERSITÉ DE RENNES 1

istic Informatique Électronique

Material

http://mathieuacher.com/teaching/MDE/MRI1516/

Plan

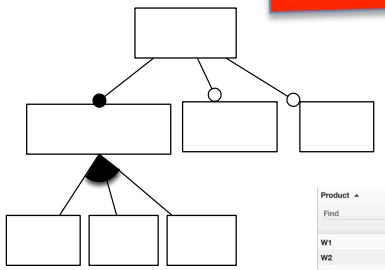
- Challenges and Overview
  - Developping billions of software product is hard but now a common practice
- Implementing Variability
  - Revisit of existing techniques and curriculum
- Specificity of Product Line Engineering
  - Process, methods
- **Feature Models**
  - **Defacto standard for modeling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

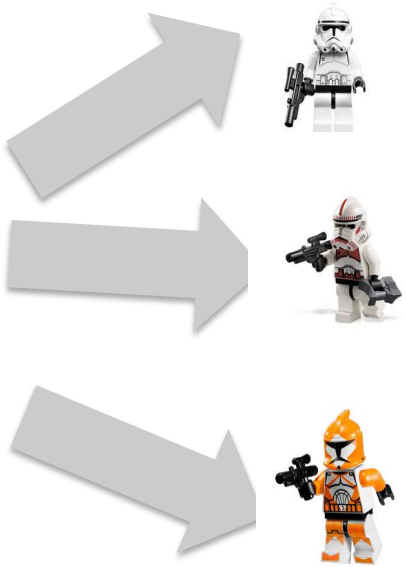


not, and, or, implies

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

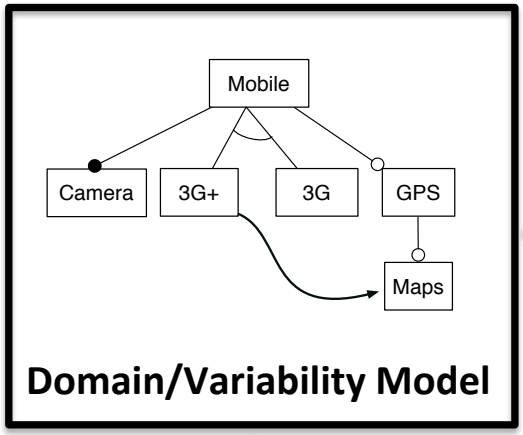
Feature models
or Product Matrices

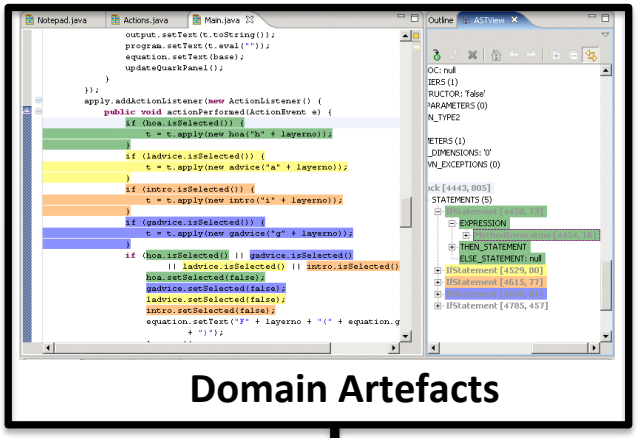
(product lines)



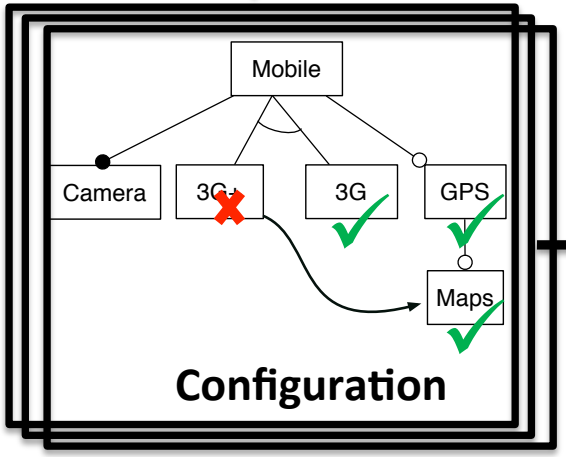
Variants of code (e.g., Java or C)
Variants of user interfaces
Variants of video sequences
Variants of models (e.g., UML or SysML)
Variants of « things » (3D models)
…

Domain Engineering



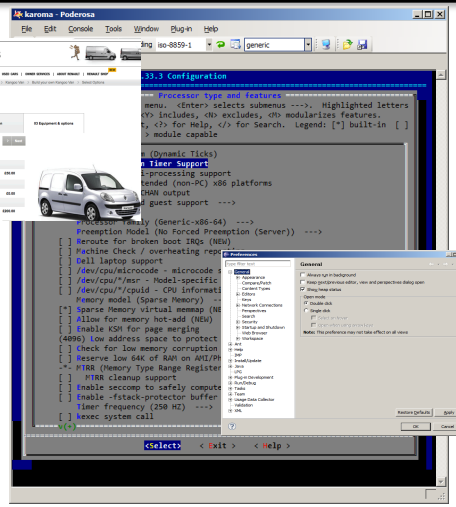


Application Engineering



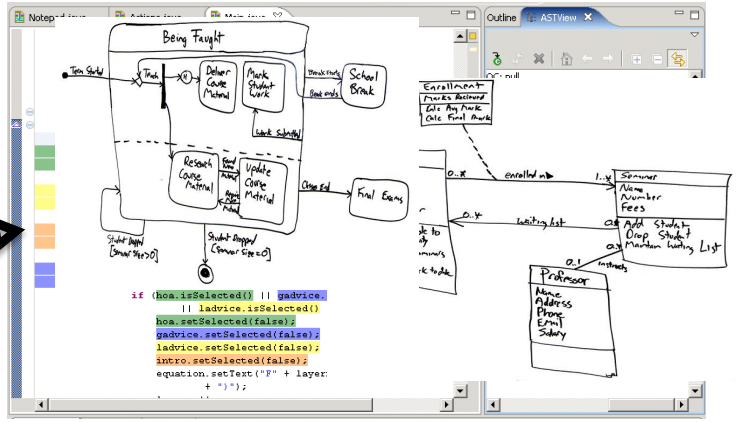
*« the investments required to develop the reusable artifacts during **domain engineering**, are outweighed by the benefits of deriving the individual products during **application engineering** »*

Jan Bosch et al. (2004)

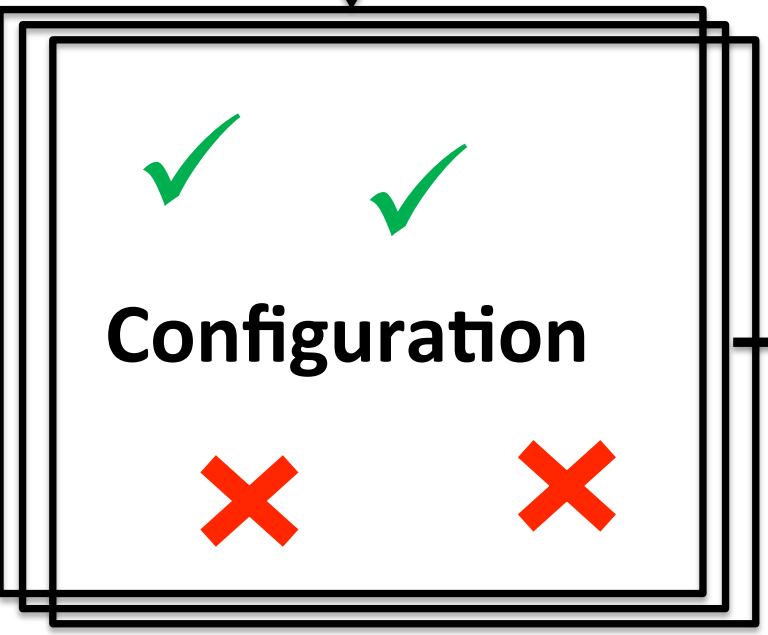


Variability Model

mapping



Base Artefacts (e.g., models)

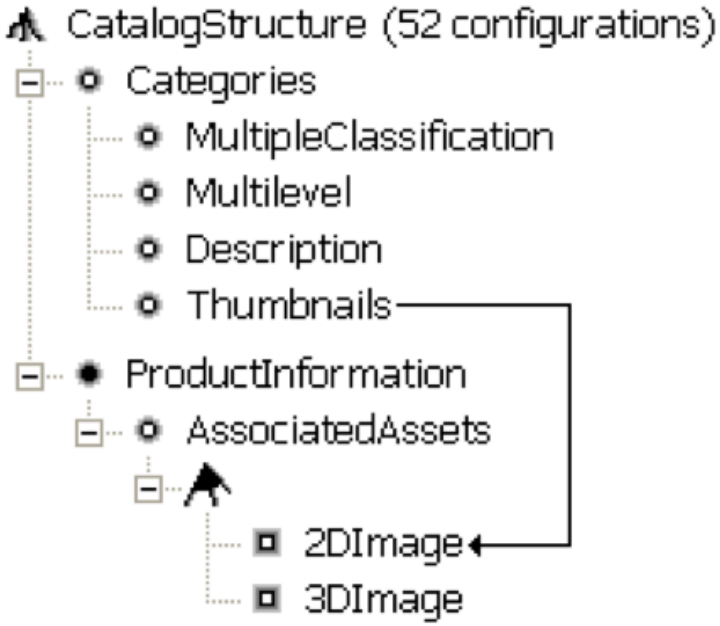


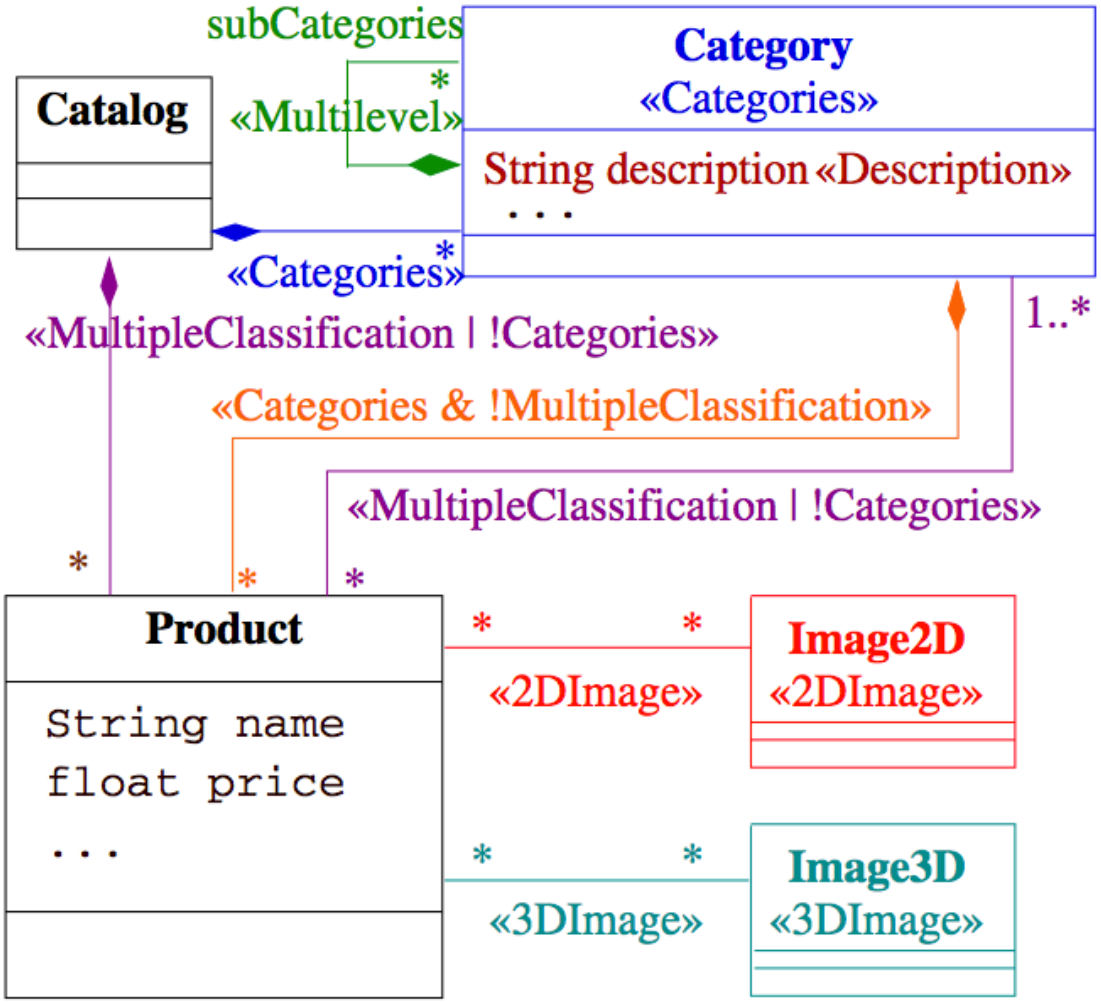
Configuration

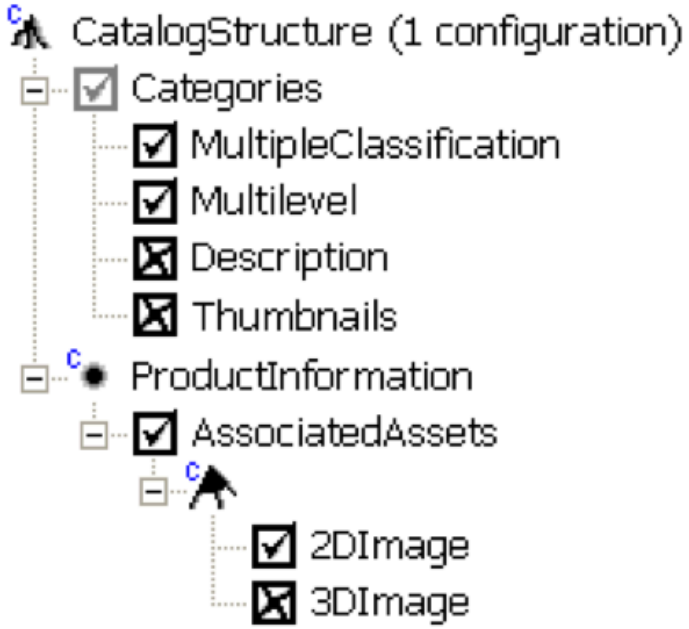
Software Generator
(derivation engine)

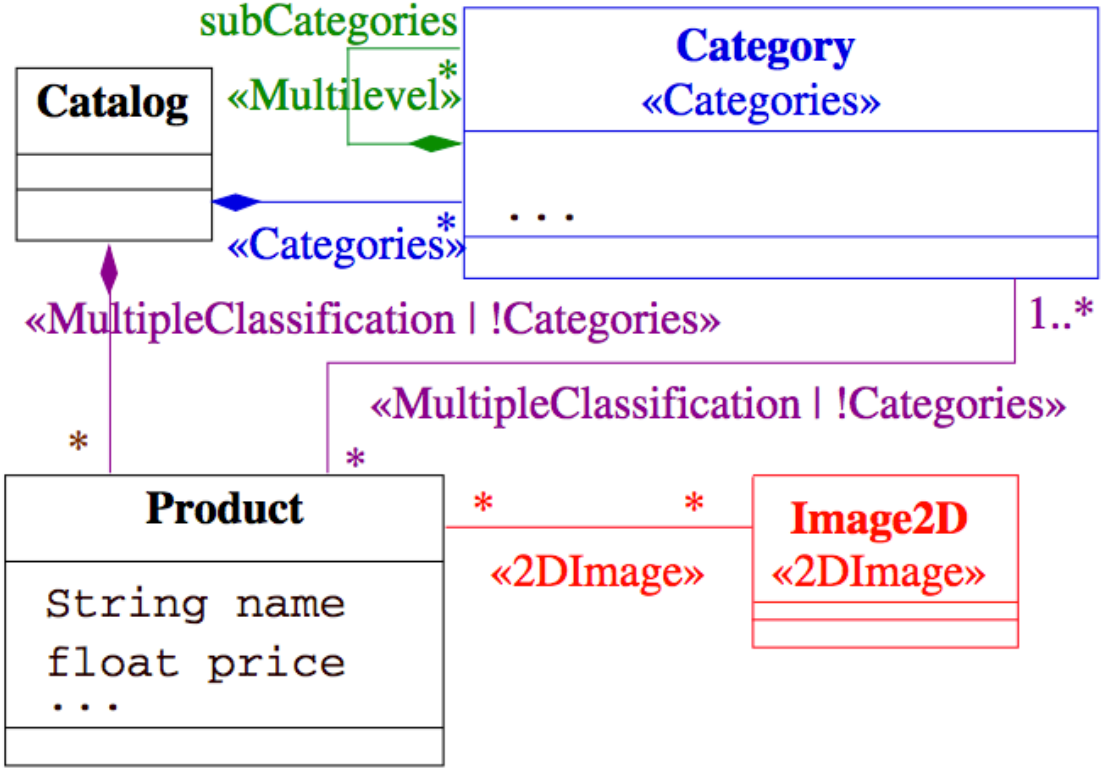
Quizz: what are the constraints over WORLD and BYE?

```
#include <stdio.h>

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

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

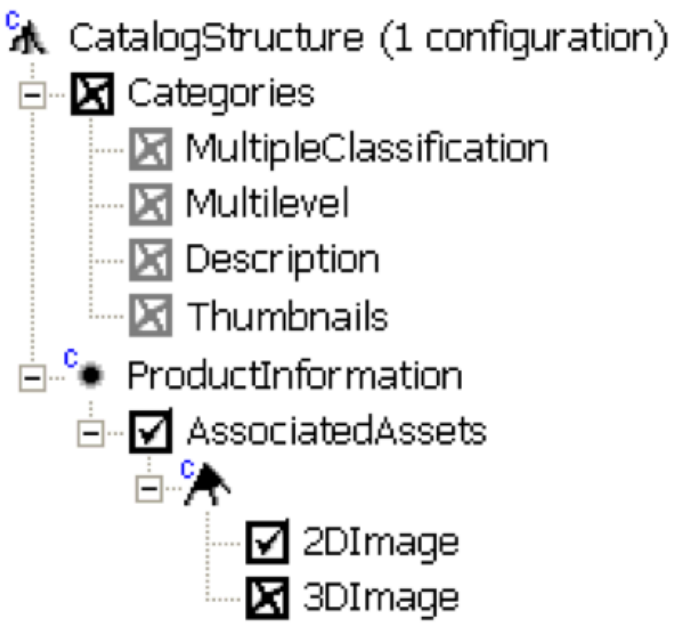

CatalogStructure (52 configurations)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage

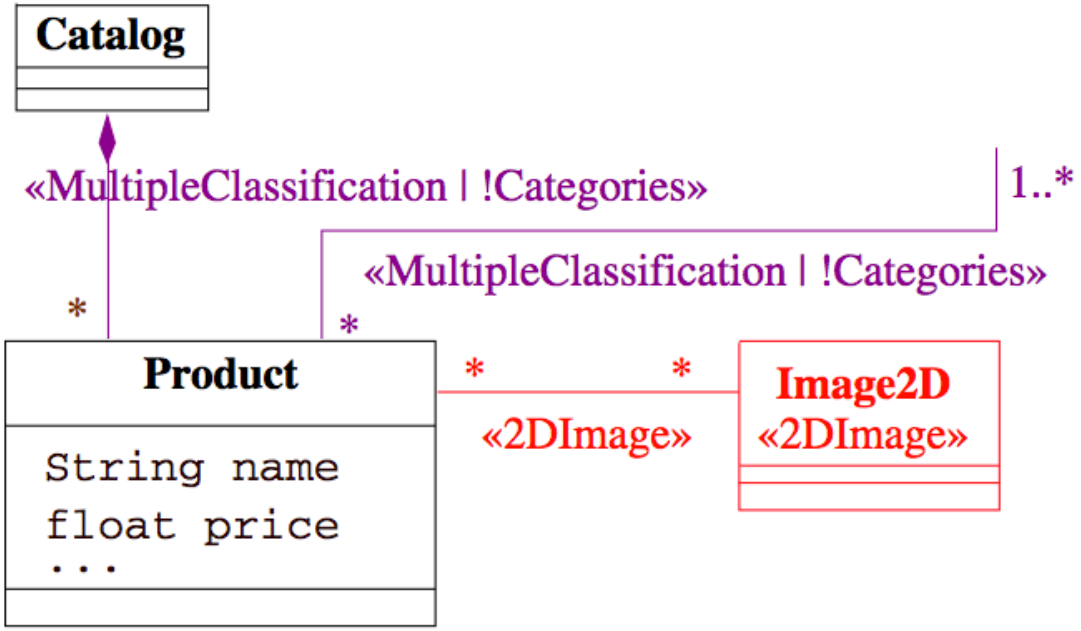

subCategories
Catalog
«Multilevel»
Category
«Categories»
String description «Description»
. . .
«Categories»
«MultipleClassification | !Categories»
«Categories & !MultipleClassification»
«MultipleClassification | !Categories»
1..*
Product
String name
float price
. . .
«2DImage»
Image2D
«2DImage»
«3DImage»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

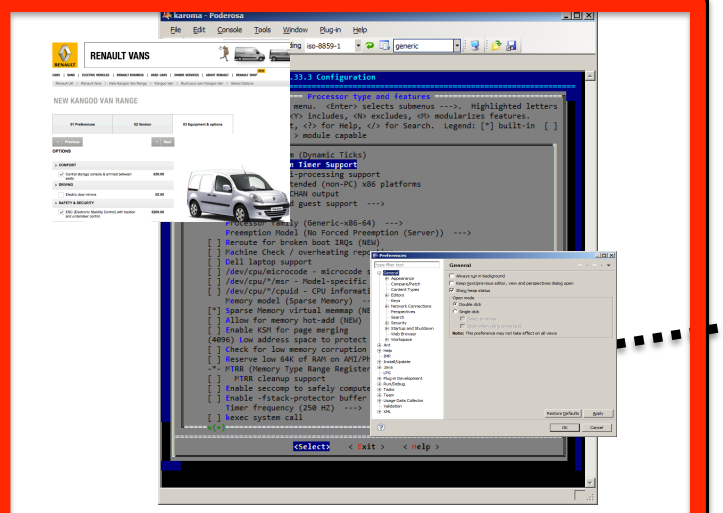

CatalogStructure (1 configuration)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage


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

Ooops

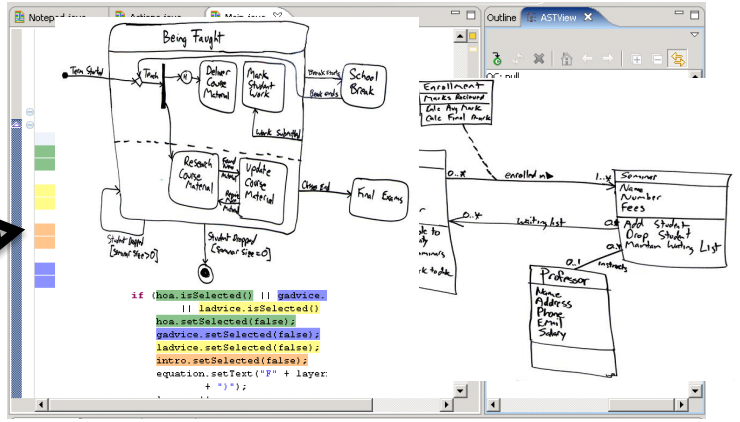

CatalogStructure (1 configuration)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage


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



Variability Model

mapping



Base Artefacts (e.g., models)

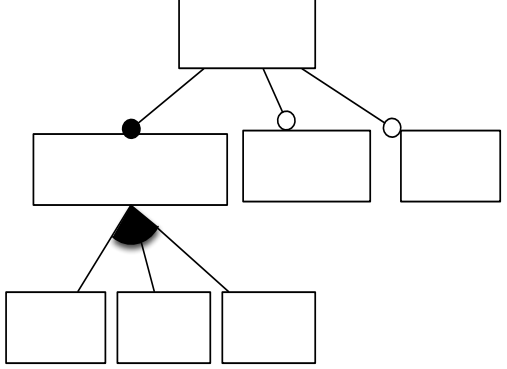
Configuration

Software Generator
(derivation engine)


Unused flexibility

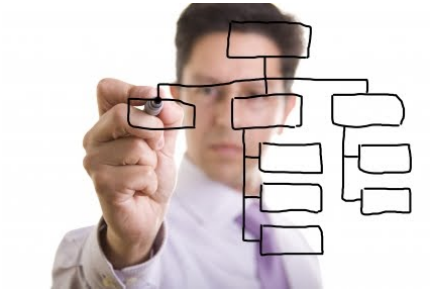

Illegal variant

Feature Model

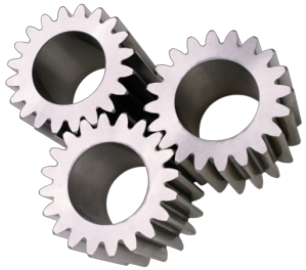


not, and, or, implies

Communicative



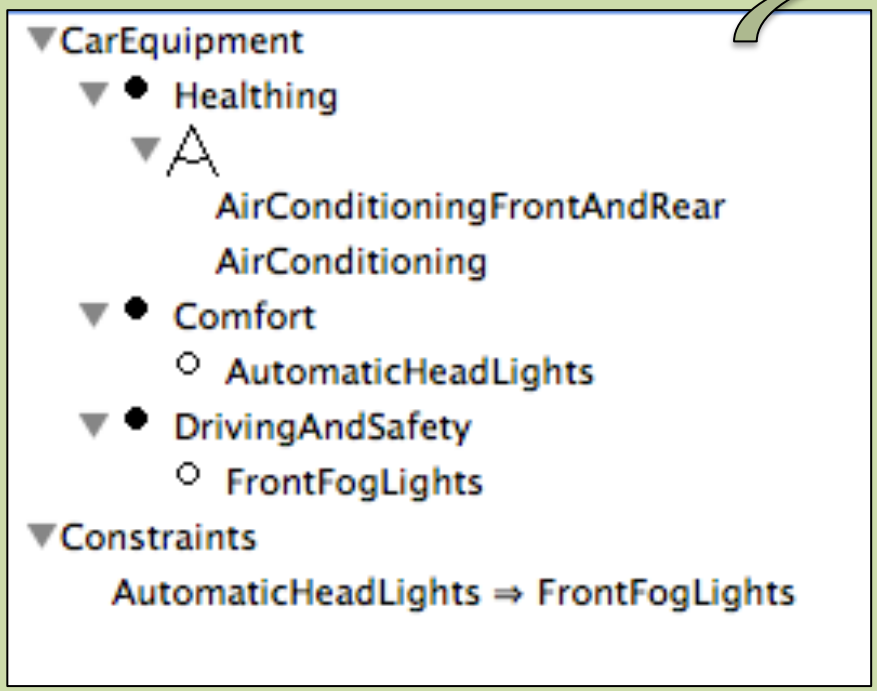
Analytic

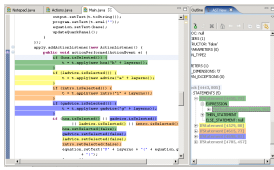


Generative

Feature Models

(defacto standard for modeling variability)

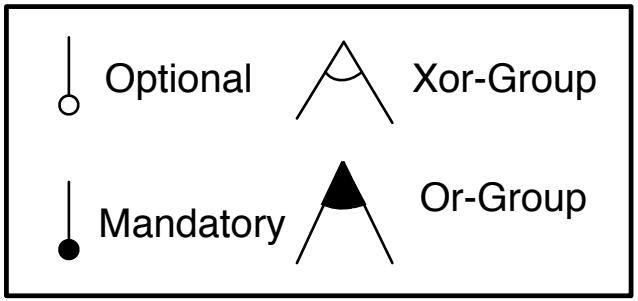




Hierarchy: rooted tree

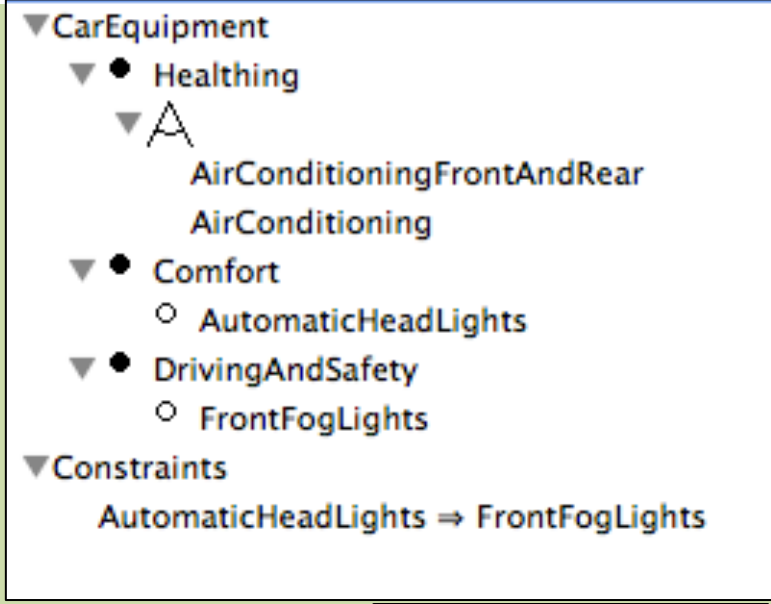
Variability:

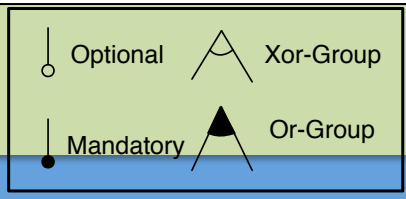
- mandatory,
- optional,
- Groups: exclusive or inclusive features
- Cross-tree constraints



SPLC12Scripts Model | SPLC12Scripts.config

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





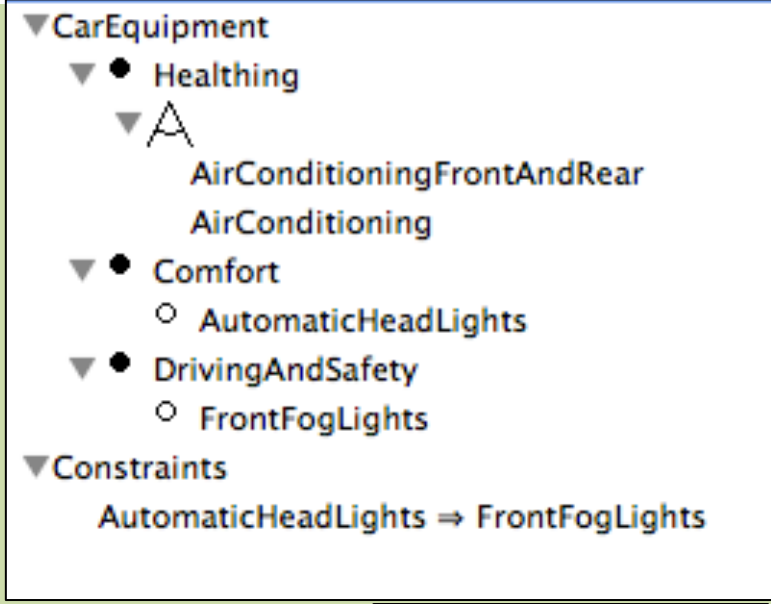
Hierarchy + Variability
=
set of valid configurations

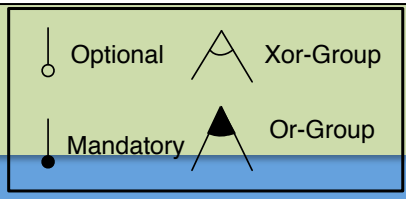
configuration = set of features selected

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

SPLC12Scripts Model | SPLC12Scripts.config

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

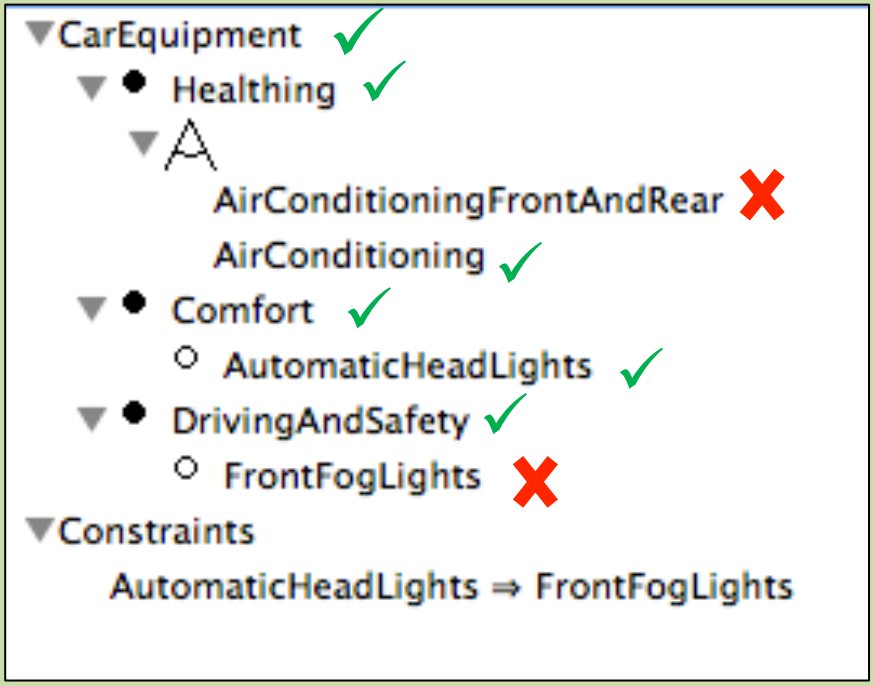


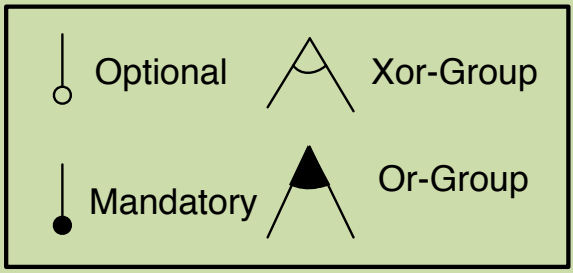


Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning}

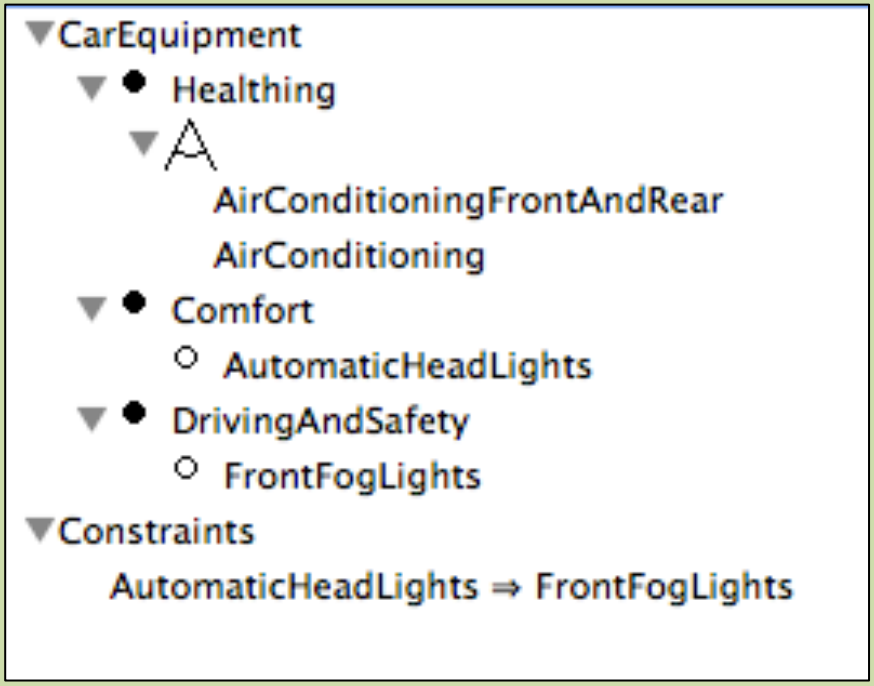


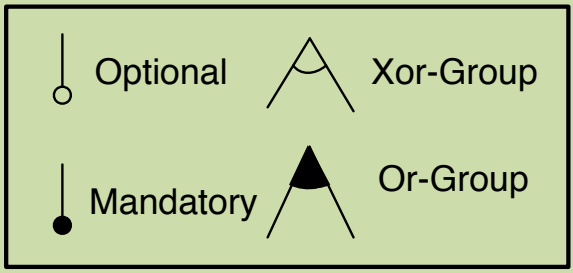


Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

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



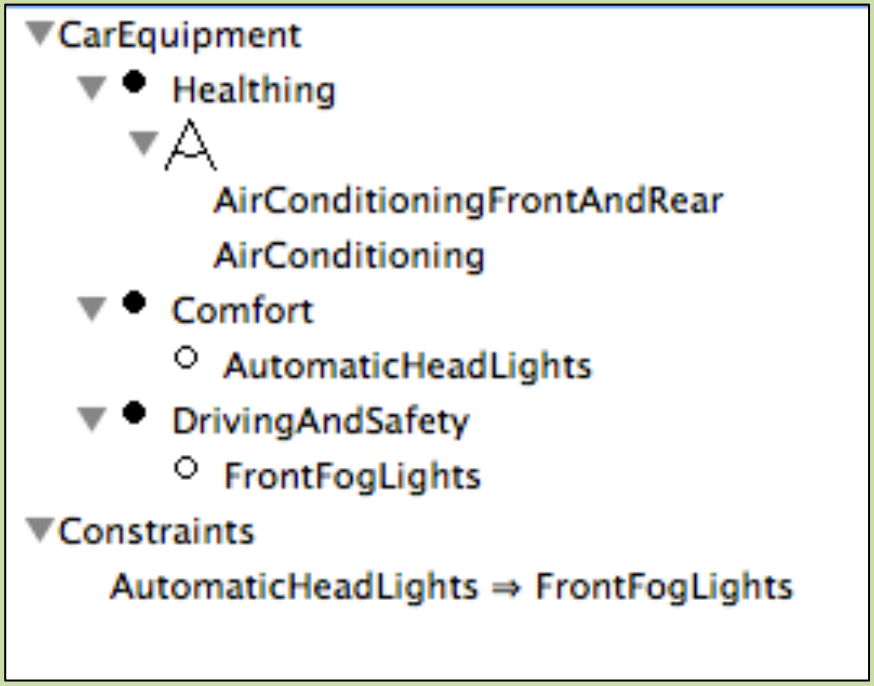


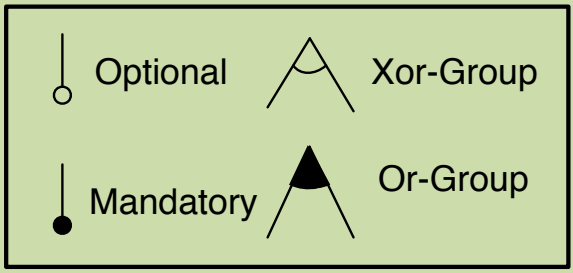
Hierarchy + Variability
=
set of valid configurations

{CarEquipment, Comfort, DrivingAndSafety, Healthing}

X

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

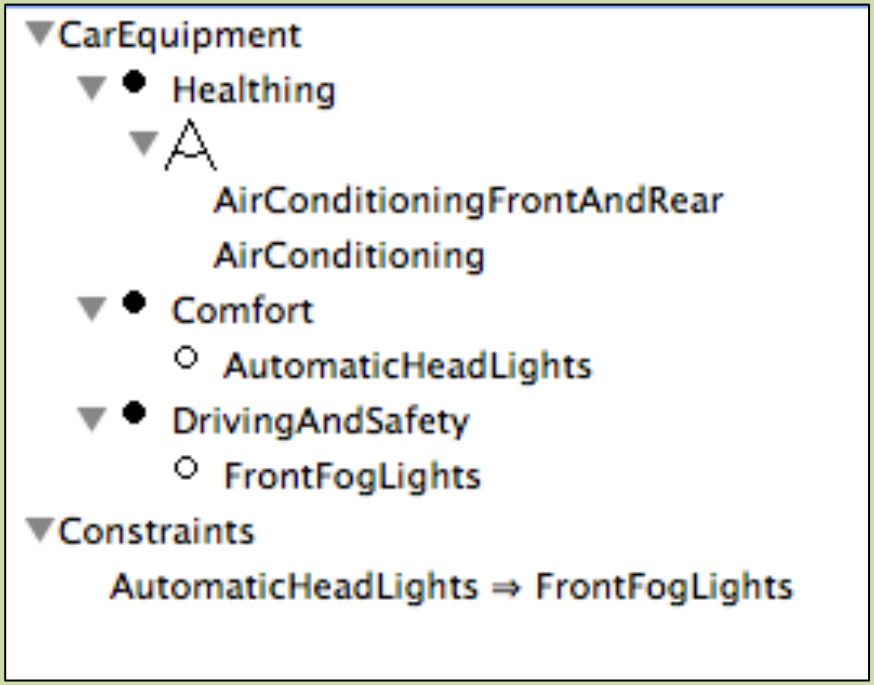


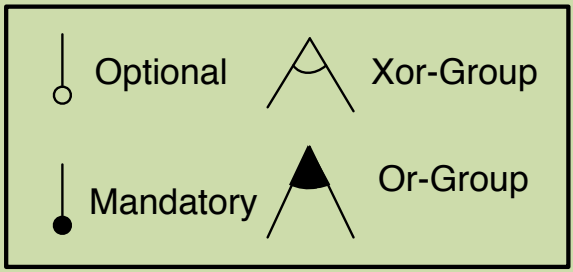


Hierarchy + Variability
=
set of valid configurations

Configuration set (from a basic feature model of car)

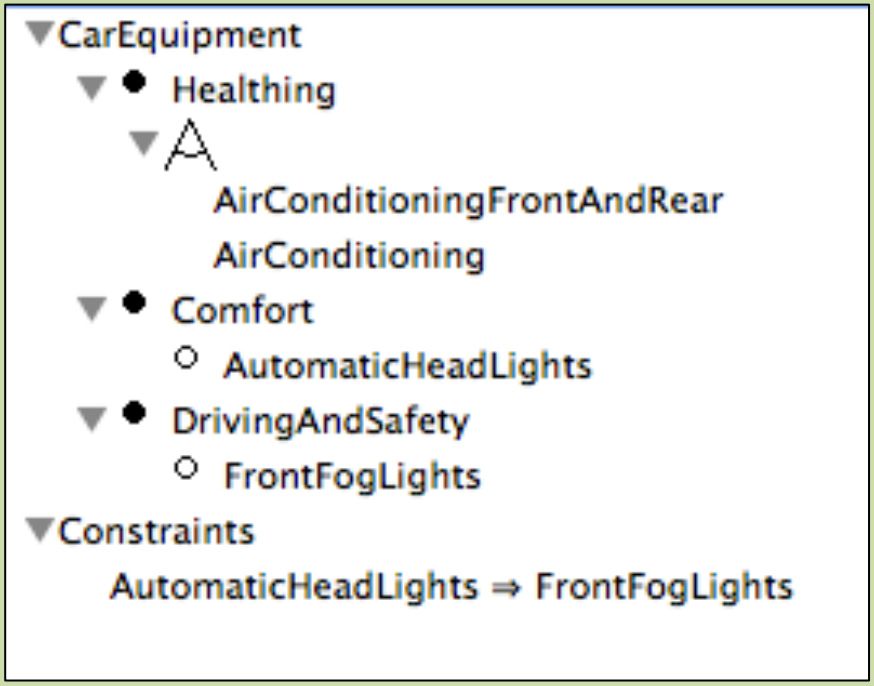
	CarEquipment	Comfort	DrivingAndSafety	Healting	AirConditioning	FrontFogLights	AutomaticHeadLights	AirConditioningFrontAndRear
Car2	yes	yes	yes	yes	yes	yes	yes	no
Car6	yes	yes	yes	yes	no	yes	no	yes
Car1	yes	yes	yes	yes	yes	yes	no	no
Car4	yes	yes	yes	yes	no	no	no	yes
Car5	yes	yes	yes	yes	yes	no	no	no
Car3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

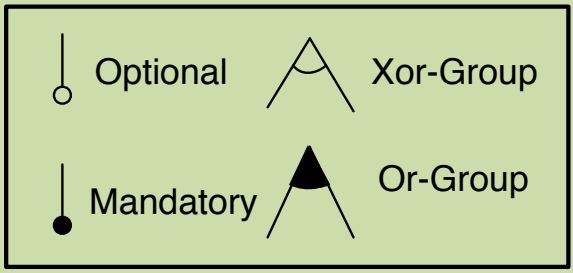




Hierarchy + Variability
=
set of valid configurations

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



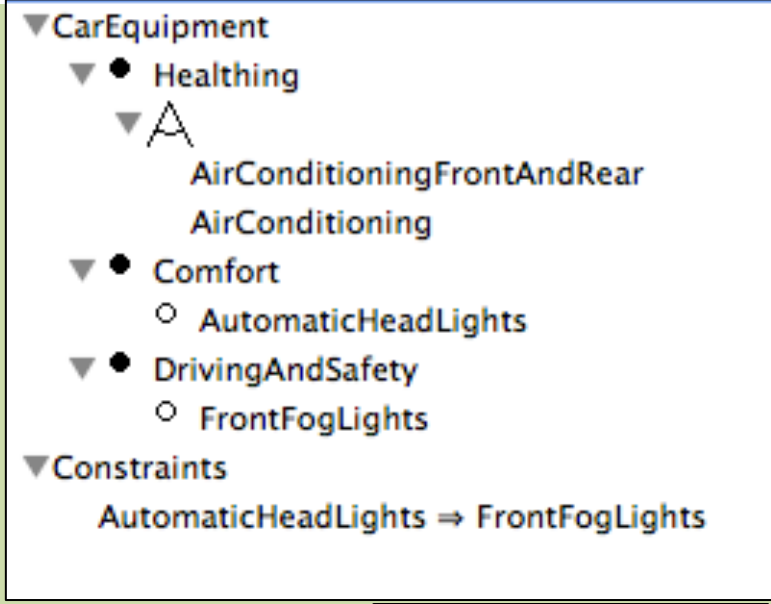


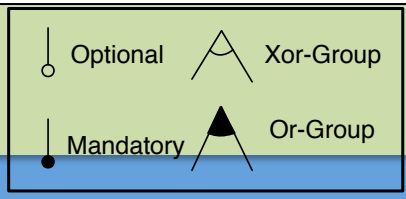
Hierarchy + Variability
=
set of valid configurations

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



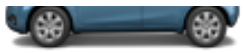


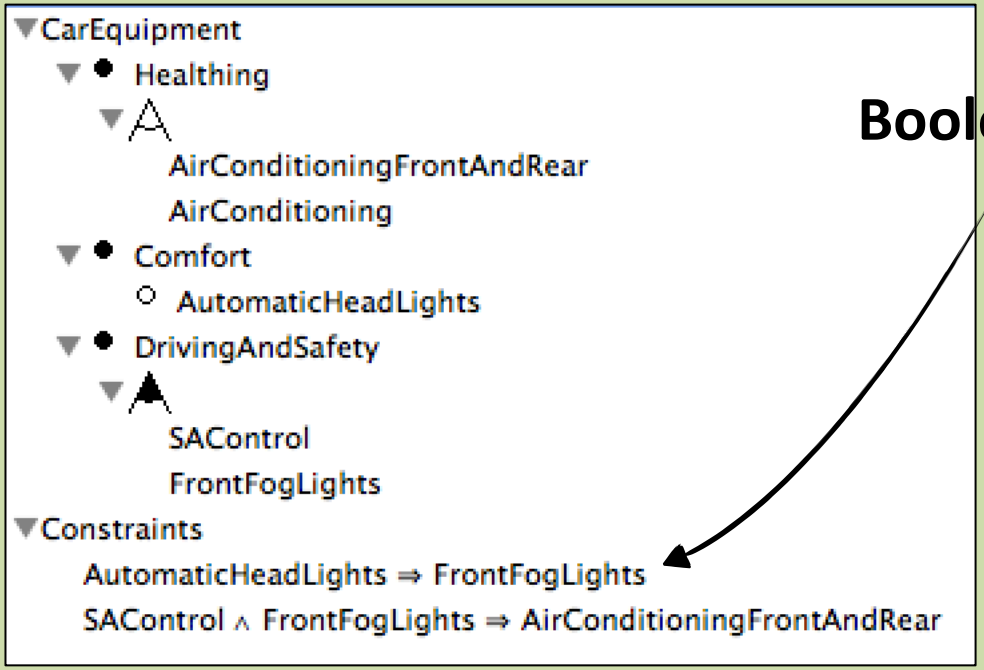
Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

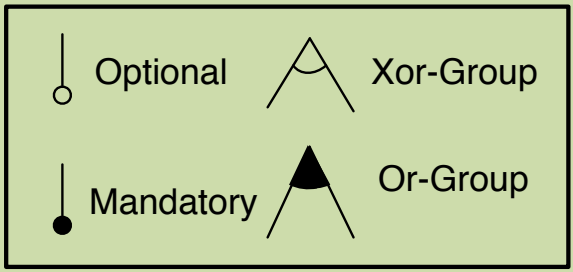
{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning}

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





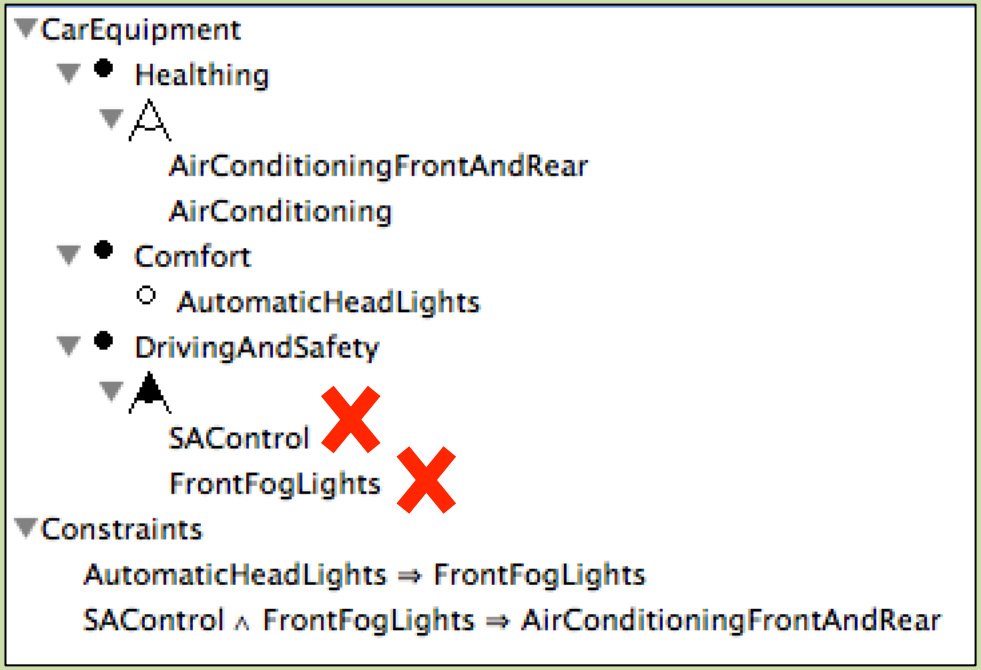
Boolean logic: ^, v, not, implies

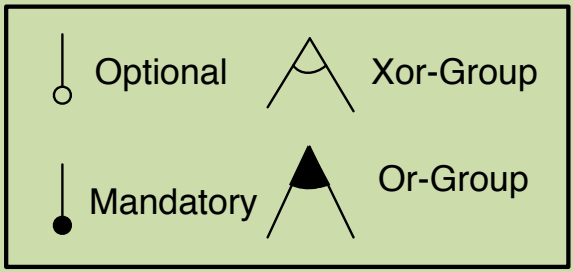


Hierarchy + Variability

=

set of valid configurations



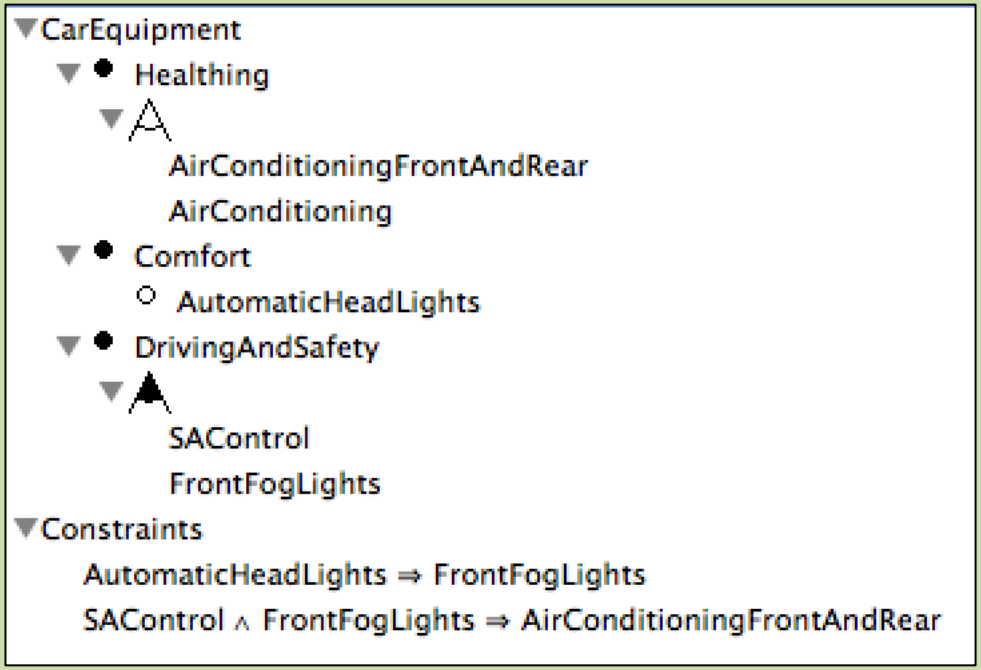


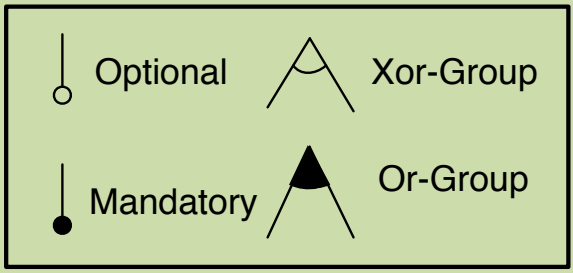
Hierarchy + Variability

=

set of valid configurations

Or-group: at least one!



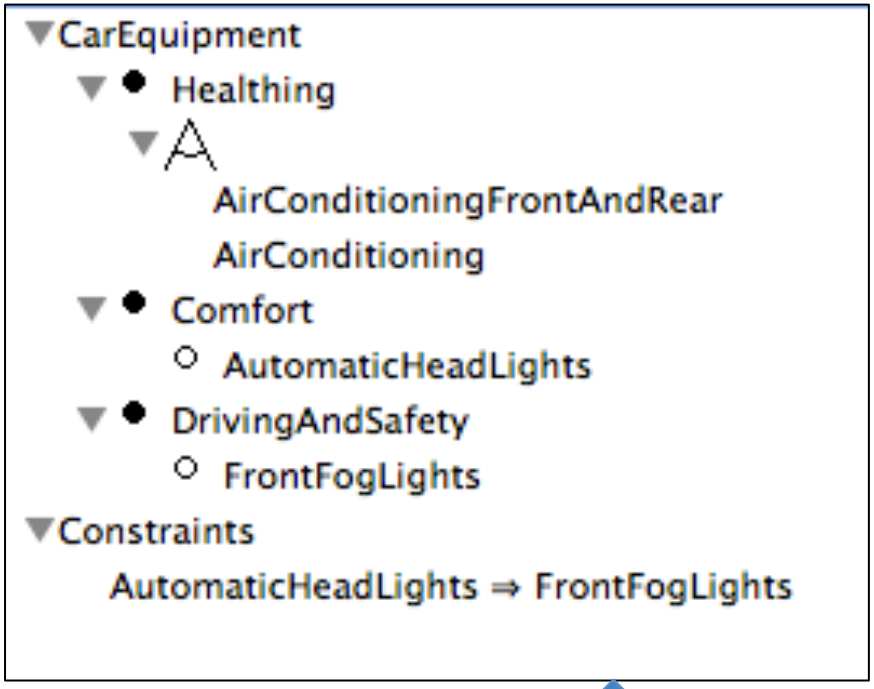


Hierarchy + Variability
=
set of valid configurations

{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

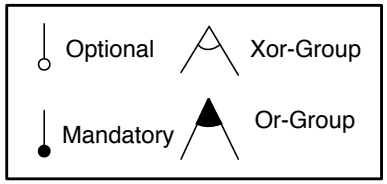
(Boolean) Formula φ

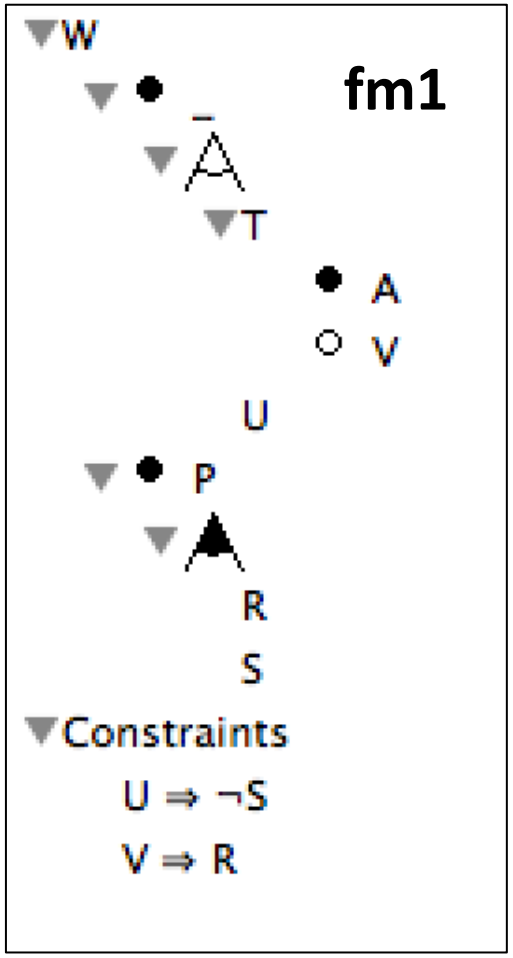
Product	CarEquipment	Comfort	DrivingAndSafety	Healting	AirConditioning	FrontFogLights	AutomaticHeadLights	AirConditioningFrontAndRear
Find	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No	Yes / No
Car1	yes	yes	yes	yes	yes	yes	no	no
Car2	yes	yes	yes	yes	yes	yes	yes	no
Car3	yes	yes	yes	yes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

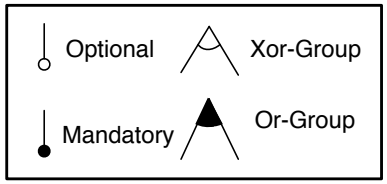


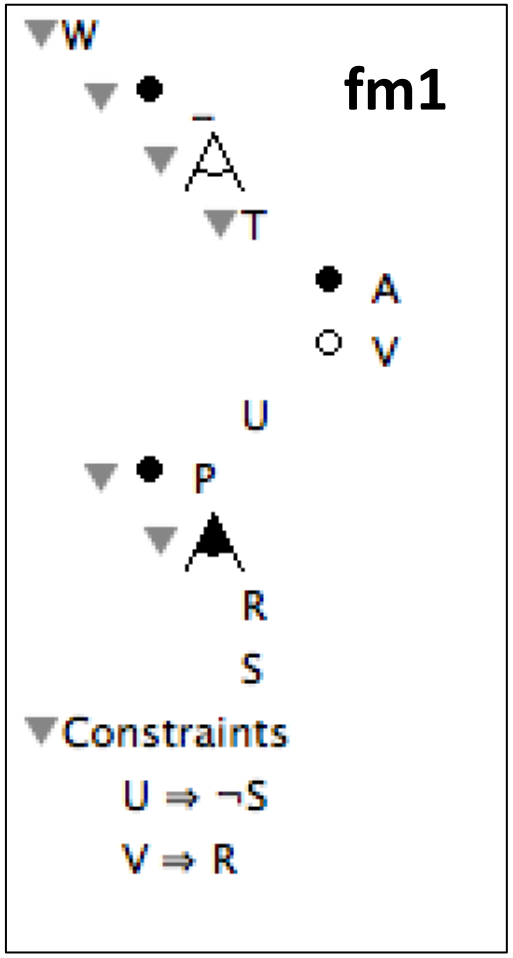


$$[\![fm1]\!] = \{$$
$$\{W, P, R, S, T, A, V\},$$
$$\{W, P, S, T, A\},$$
$$\{W, P, R, T, A\},$$
$$\{W, P, R, U\},$$
$$\{W, P, R, T, V, A\},$$
$$\{W, P, R, S, T, A\},$$
$$\}$$

(Boolean) Feature Models

~ Boolean formula



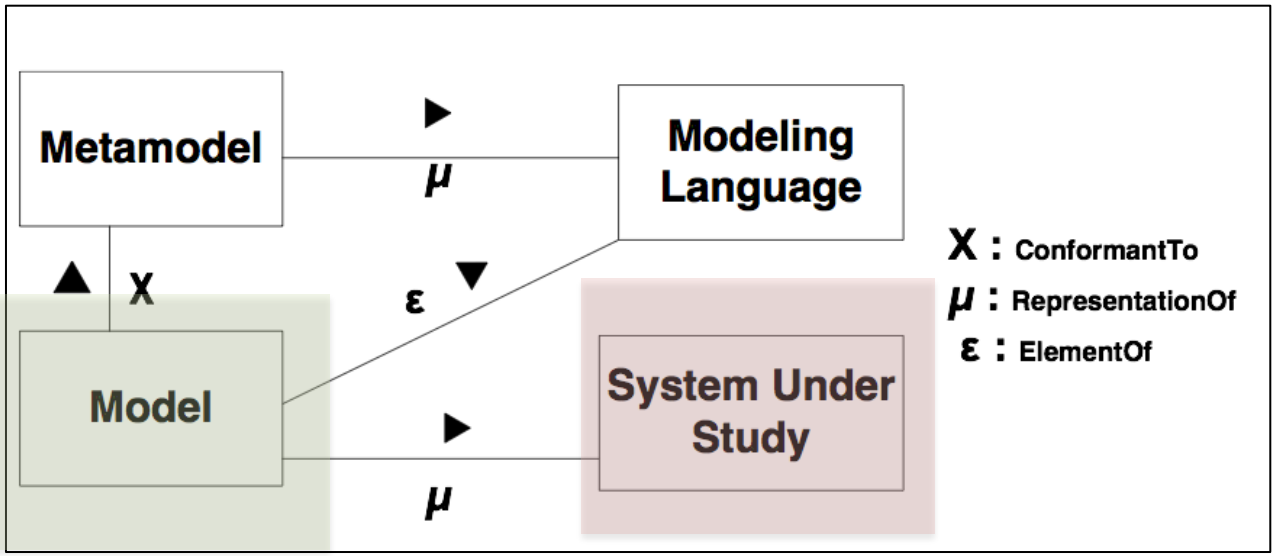


$$
\begin{aligned}
&\phi_{fm_1} = W \;//\; \text{root} \\
&\wedge\, W \Leftrightarrow P \;//\; \text{mandatory} \\
&//\; \text{Or-group} \\
&\wedge\, P \Rightarrow R \vee S \\
&\wedge\, R \Rightarrow P \wedge\, S \Rightarrow P \\
&\wedge\, V \Rightarrow T \;//\; \text{optional} \\
&\wedge\, A \Leftrightarrow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

Languages:

How to specify feature models?

Feature Models



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

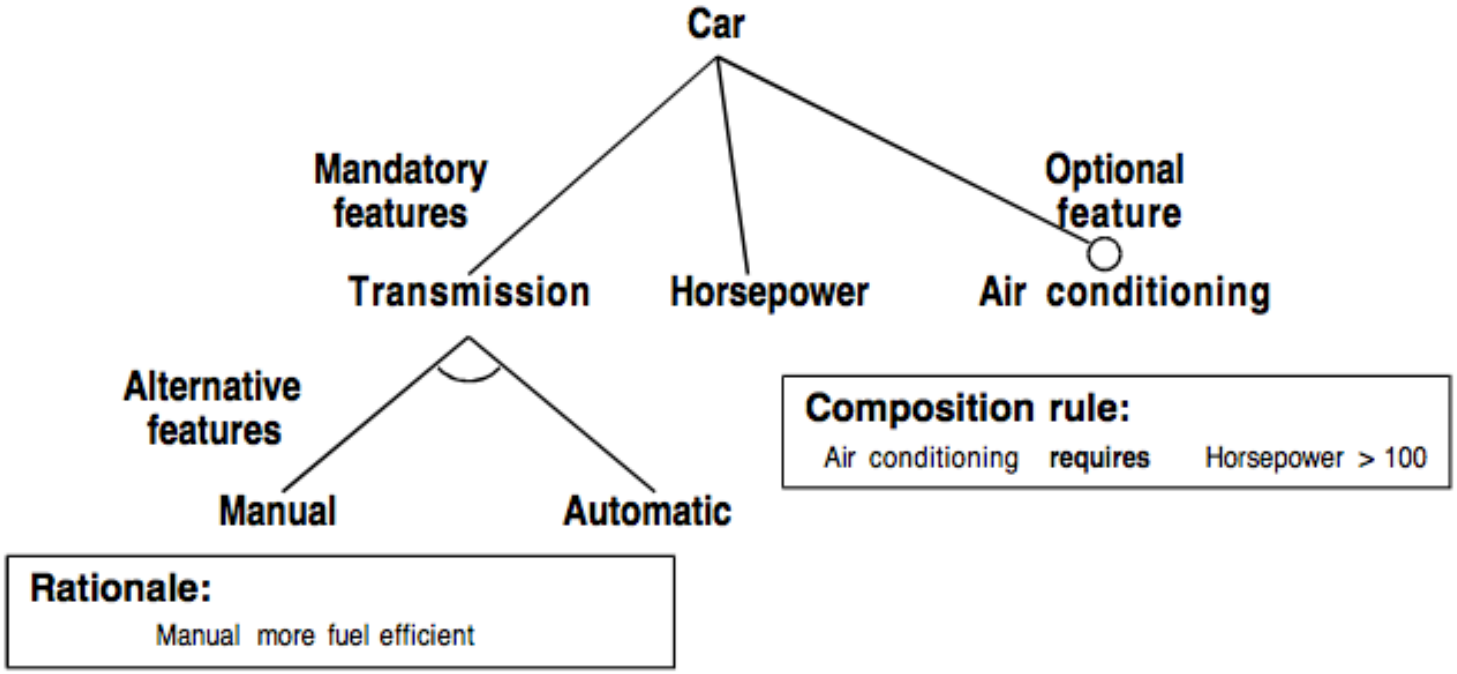
History of Feature Models

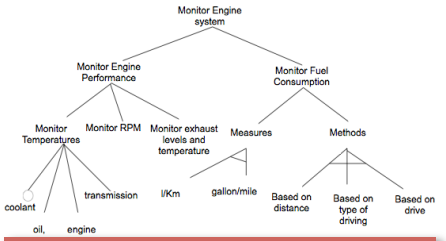
~ often called Feature Diagrams
(suggest a visual representation)

graphical languages

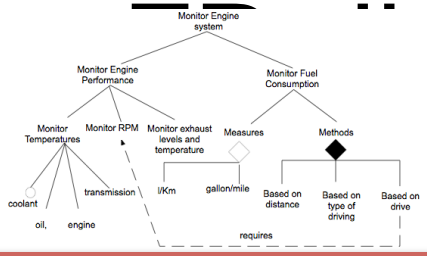
Feature Models

Kang et al. (1990)

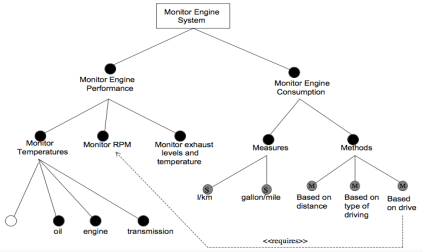




FODA (OFT)
[Kang et al., 1990]

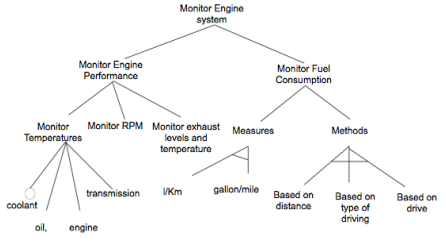


FeatuRSEB (RFD)
[Griss et al., 1998]

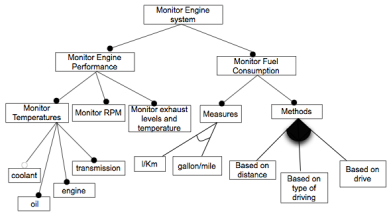


PLUSS (PFT)
[Eriksson et al., 2005]

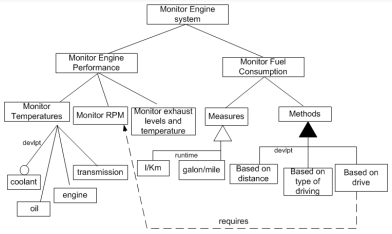
FORM (OFD)
[Kang et al., 1998]



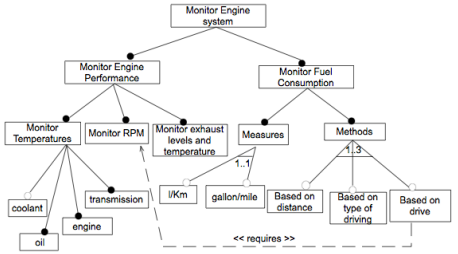
Gen. Prog. (GPFT)
[Czarnecki et al., 2000]

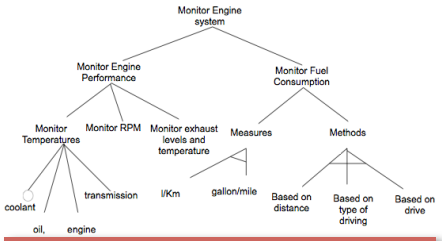


VBFD
[van Gurp et al., 2001]

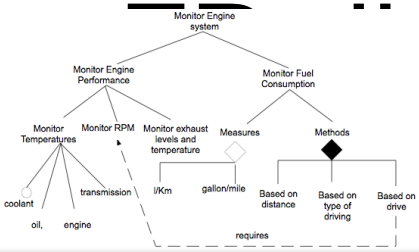


EFD
[Riebisch et al., 2002]

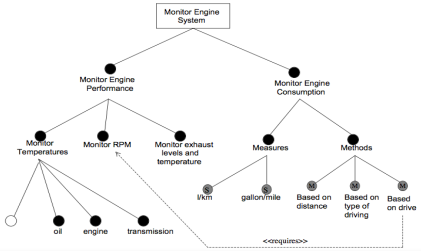




FODA (OFT)
[Kang et al., 1990]



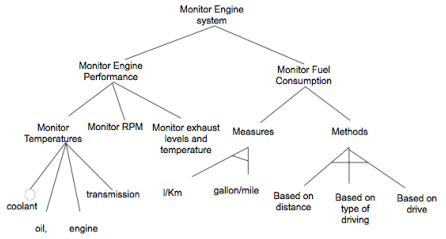
FeatuRSEB (RFD)
[Griss et al., 1998]



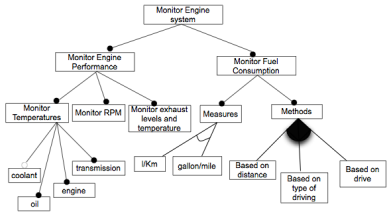
PLUSS (PFT)
[Eriksson et al., 2005]

Aesthetic differences

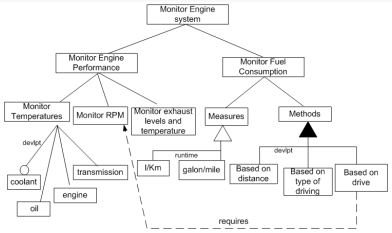
FORM (OFD)
[Kang et al., 1998]



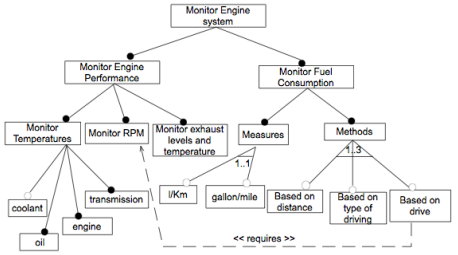
Gen. Prog. (GPFT)
[Czarnecki et al., 2000]



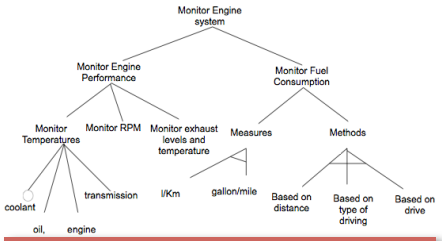
VBFD
[van Gurp et al., 2001]



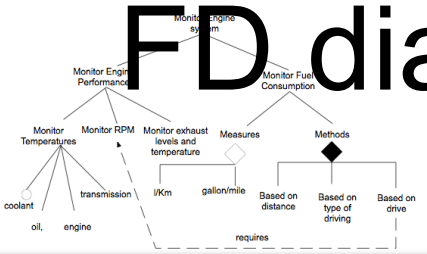
EFD
[Riebisch et al., 2002]



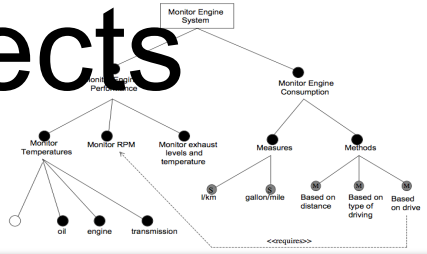
FD dialects



FODA (OFT)
[Kang et al., 1990]



FeatuRSEB (RFD)
[Griss et al., 1998]

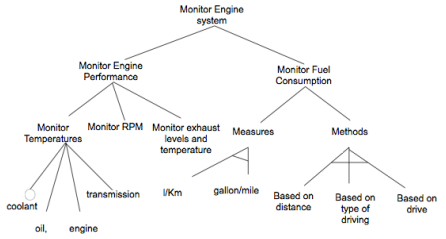


PLUSS (PFT)
[Eriksson et al., 2005]

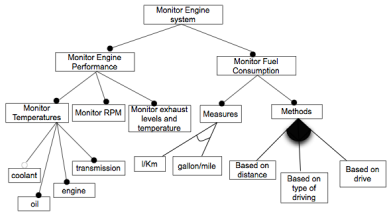
Aesthetic differences

Stronger claims

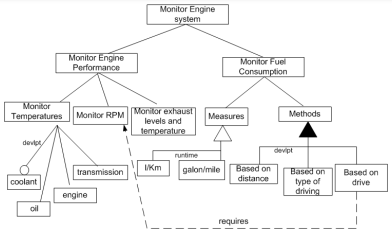
FORM (OFD)
[Kang et al., 1998]



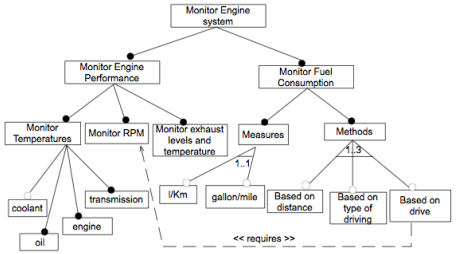
Gen. Prog. (GPFT)
[Czarnecki et al., 2000]



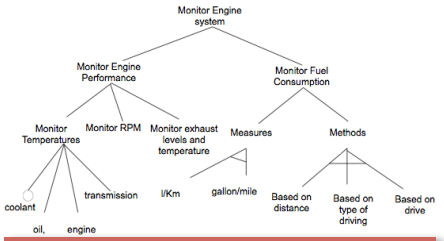
VBFD
[van Gurp et al., 2001]



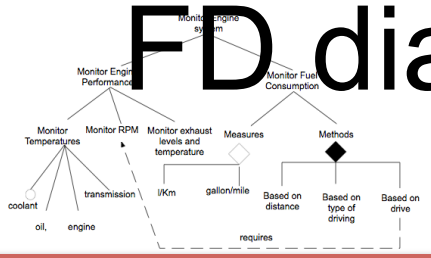
EFD
[Riebisch et al., 2002]



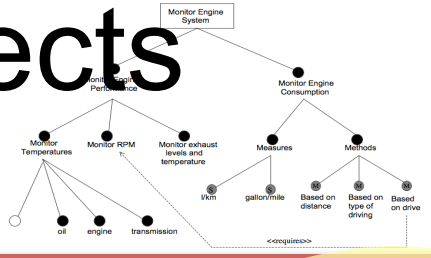
FD dialects



FODA (OFT)
[Kang et al., 1990]



FeatuRSEB (RFD)
[Griss et al., 1998]



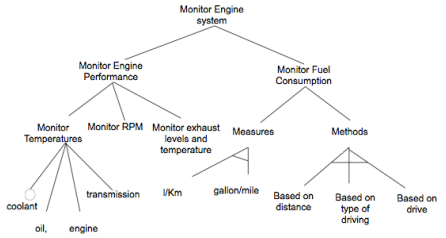
PLUSS (PFT)
[Eriksson et al., 2004]

Aesthetic differences

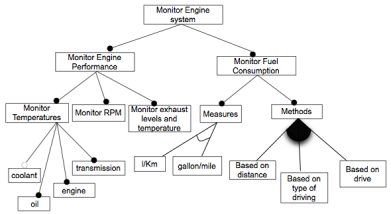
Stronger claims

Vague use of terms
syntax, semantics, expressiveness, ambiguity…

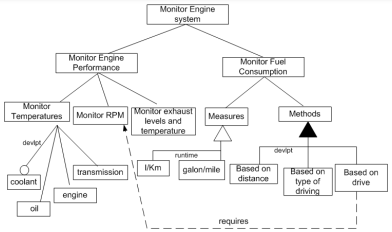
FORM (OFD)
[Kang et al., 1998]



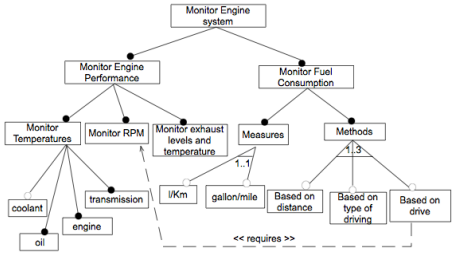
Gen. Prog. (GPFT)
[Czarnecki et al., 2000]



VBFD
[van Gurp et al., 2001]



EFD
[Riebisch et al., 2002]



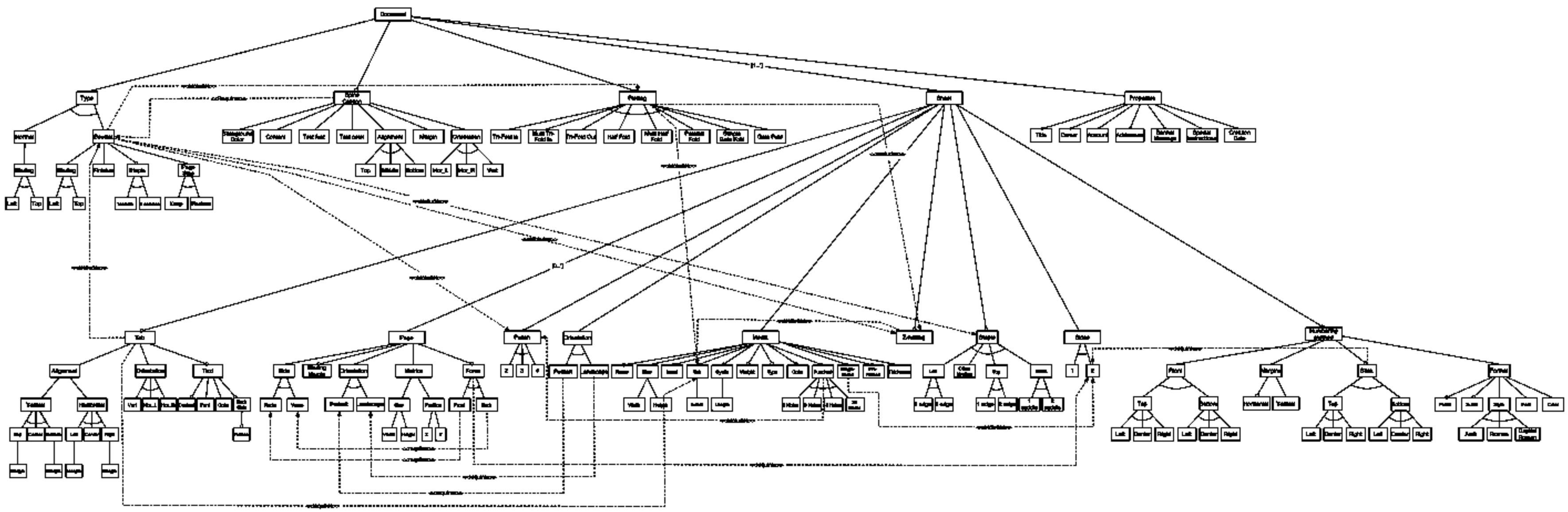
Concrete (graphical) syntax matters…

But semantics even more!

Feature Models

~ Feature Diagrams
(suggest a visual representation)

but that's not mandatory
that does not necessary scale
or respond to users' concerns



Type
Left
Top
Half Fold

Textual variability languages

- Andreas Classen, Quentin Boucher, Patrick Heymans: A text-based approach to feature modelling: Syntax and semantics of TVL. Sci. Comput. Program. 76(12): 1130-1143 (2011)
- Czarnecki et al. “Clafer: Unifying Class and Feature Modeling” SoSyM’15
- 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
- Mauricio Alférez, José A. Galindo, Mathieu Acher, and Benoit Baudry. Modeling Variability in the Video Domain: Language and Experience Report (2014). Research Report RR-8576

FAMiliAR

(FeAture Model scrIpt Language for manIpulation and Automatic Reasoning)

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

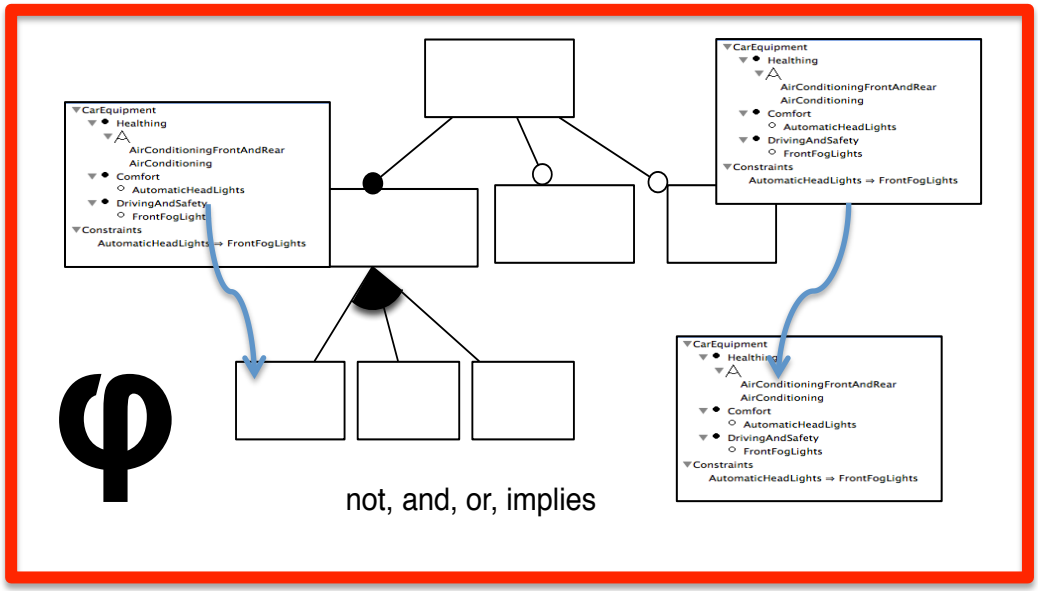
COMMON VARIABILITY LANGUAGE

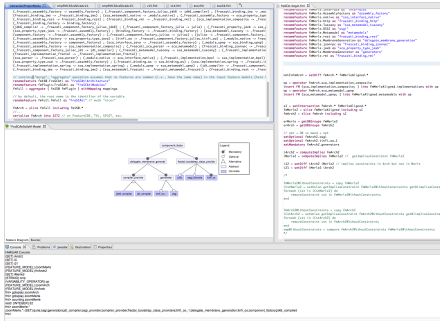
S.P.L.O.T. Software Product Lines Online Tools

IDE Feature

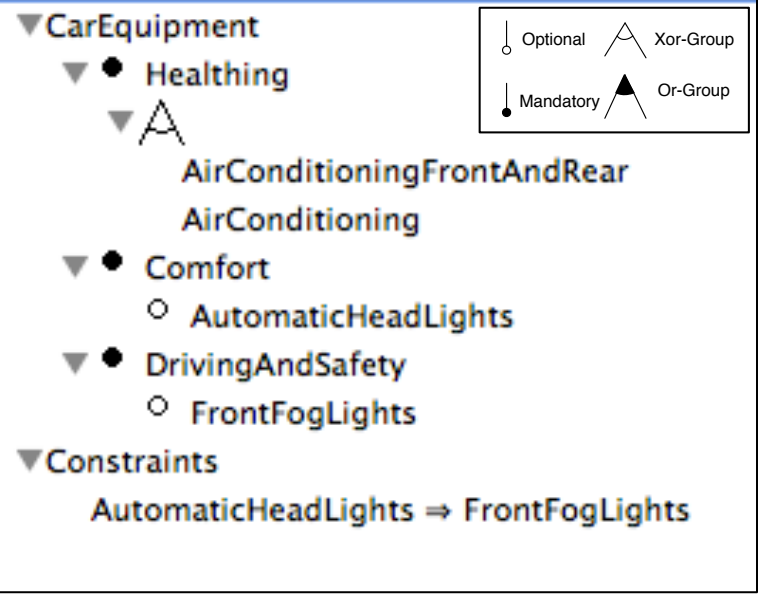
TVL

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



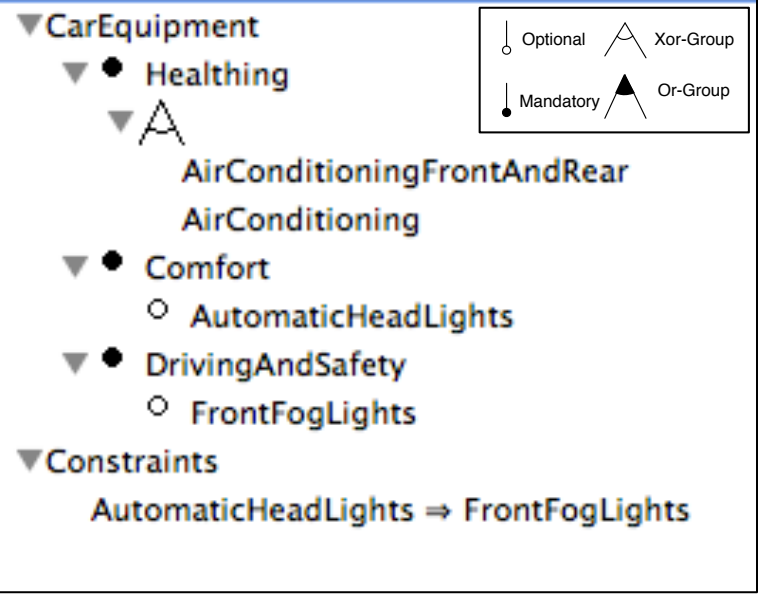

```
fml> convert fmCarEquipment into SPLOT
res1: (STRING) <feature_model name="fmCarEquipment">
<meta>
<data name="description"/>
<data name="creator"/>
<data name="address"/>
<data name="email"/>
<data name="phone"/>
<data name="website"/>
<data name="organization"/>
<data name="department"/>
<data 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>
```

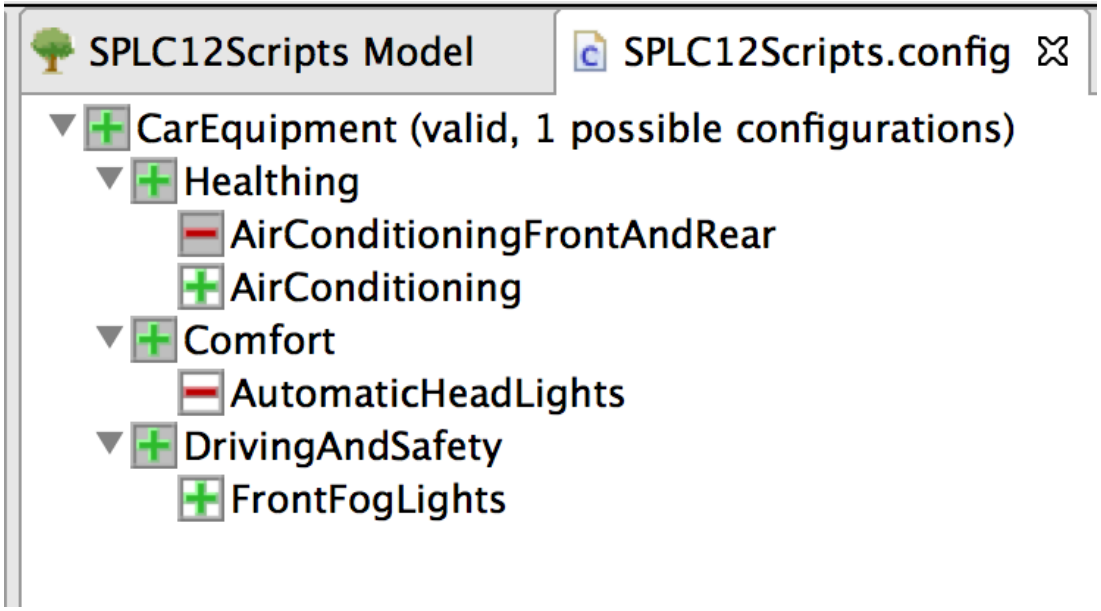
S.P.L.O.T.
Software Product Lines Online Tools

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 ; )
```

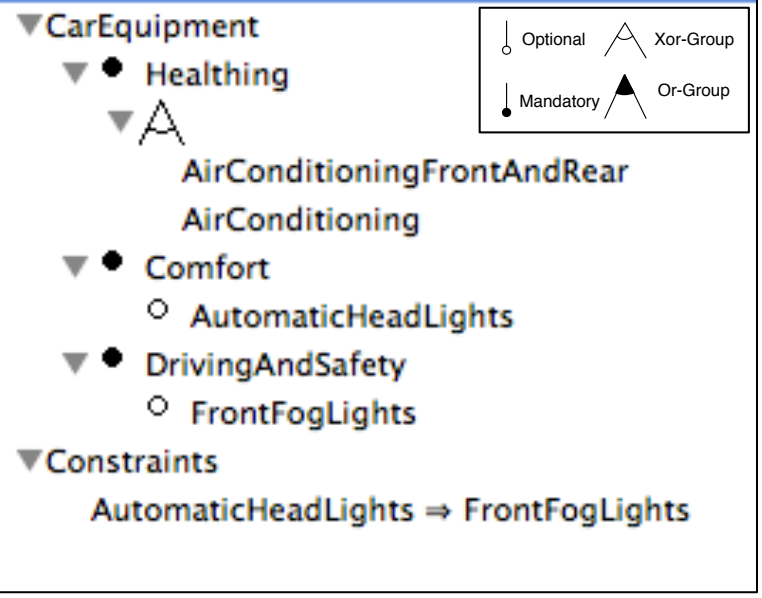
```
MacBook-Pro-de-Mathieu-3:Documents macher1$ java -Xmx1024M -jar FML-basic-1.1.jar FMLTestRepository/carEquipTuto.fml
FAMILIAR (for FeAture Model scrIpt Language for manIpulation and Automatic Reasoning)  version 1.1 (beta)
http://familiar-project.github.com/
fml> ls
(FEATURE_MODEL) fmCarEquipment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



{CarEquipment, Comfort,
DrivingAndSafety,
Healthing}

X

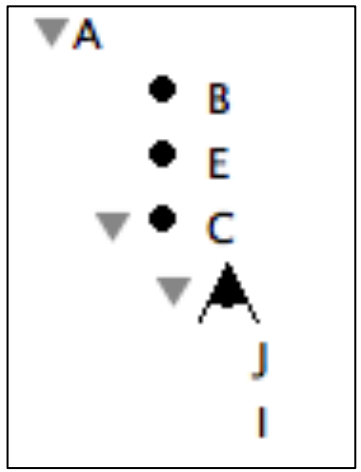
{AirConditioning, FrontFogLights}
{AutomaticHeadLights, AirConditioning,
FrontFogLights}
{AutomaticHeadLights, FrontFogLights,
AirConditioningFrontAndRear}
{AirConditioningFrontAndRear}
{AirConditioning}
{AirConditioningFrontAndRear, FrontFogLights}

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

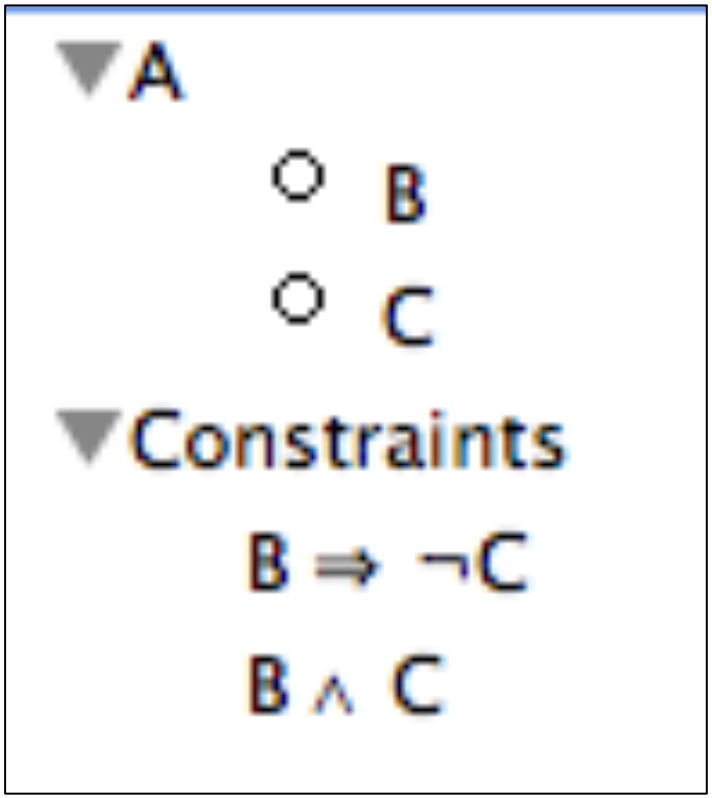
```
fml> s1 = configs fmCarEquipment
s1: (SET) {{Comfort;CarEquipment;FrontFogLights;DrivingAndSafety;AirConditioning;Healthing};{AirConditioningFrontAndRear;CarEquipment;Comfort;Driving
AndSafety;Healthing};{FrontFogLights;DrivingAndSafety;AutomaticHeadLights;CarEquipment;Healthing;AirConditioningFrontAndRear;Comfort};{Healthing;CarE
quipment;AirConditioning;DrivingAndSafety;Comfort};{Healthing;CarEquipment;Comfort;FrontFogLights;DrivingAndSafety;AirConditioningFrontAndRear};{Auto
maticHeadLights;DrivingAndSafety;CarEquipment;AirConditioning;Healthing;Comfort;FrontFogLights}}
fml> size s1
res0: (INTEGER) 6
fml> counting fmCarEquipment
res1: (DOUBLE) 6.0
```

```
fml> co = cores fmCarEquipment
co: (SET) {CarEquipment;Healthing;DrivingAndSafety;Comfort}
fml> fmCarEquipment.*
res6: (SET) {DrivingAndSafety;AirConditioningFrontAndRear;Comfort;Healthing;FrontFogLights;AirConditioning;AutomaticHeadLights;CarEquipment}
fml> setDiff fmCarEquipment.* co
res7: (SET) {AutomaticHeadLights;FrontFogLights;AirConditioning;AirConditioningFrontAndRear}
fml>
```

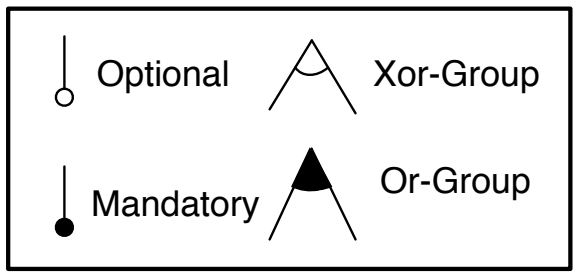
```
fm1 = FM (A : B [C] ; B : E F ; C : (I|J) ; )

r1 = root fm1
s = children r1
s1 = children fm1.A
assert (s eq s1) // equality of the two sets
ft1 = parent fm1.F
str1 = name ft1
ft2 = parent F // parent fm1.F

// another FM
fm2 = FM (A : B C E ; C : (I|J)+ ; )
ft3 = fm2.B
ft4 = name B // ambiguity
```

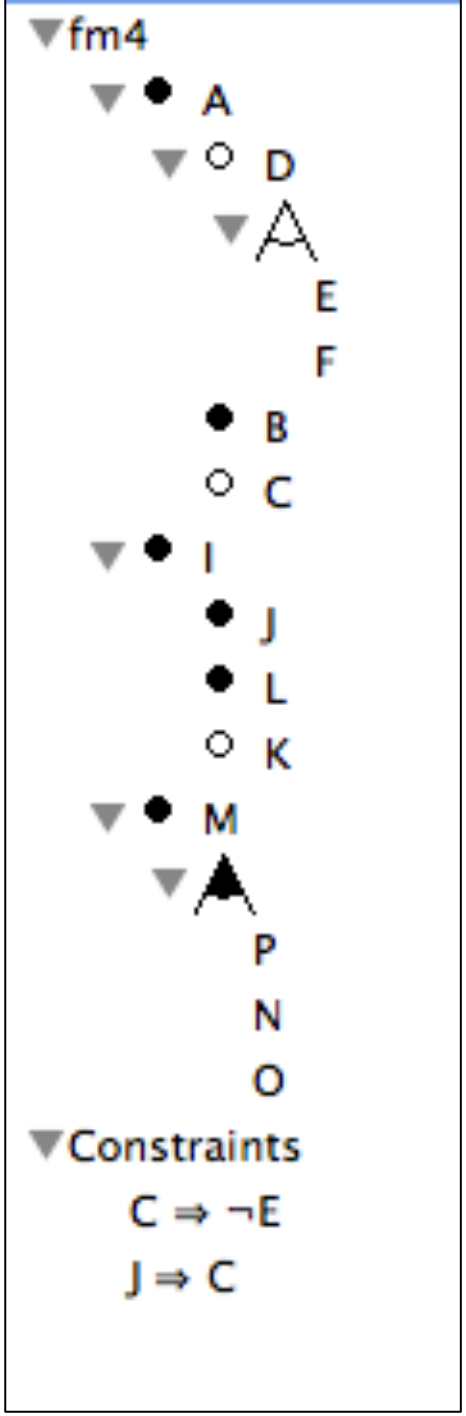


I want to analyze and play with my specification!

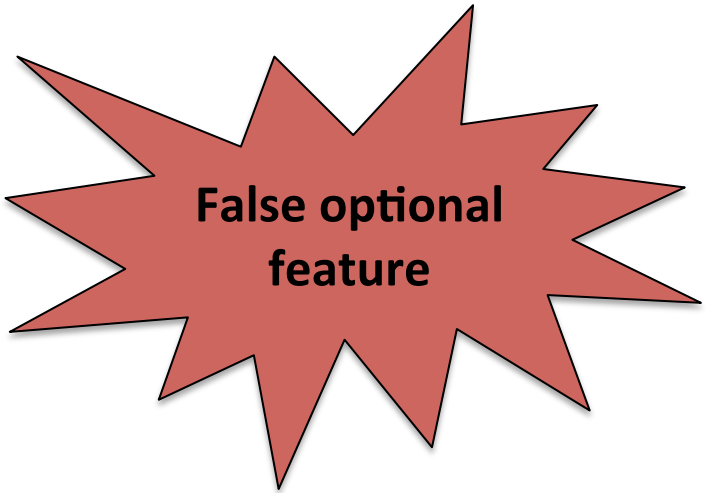


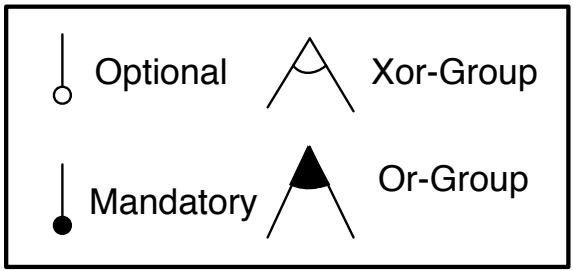
Empty set of configurations

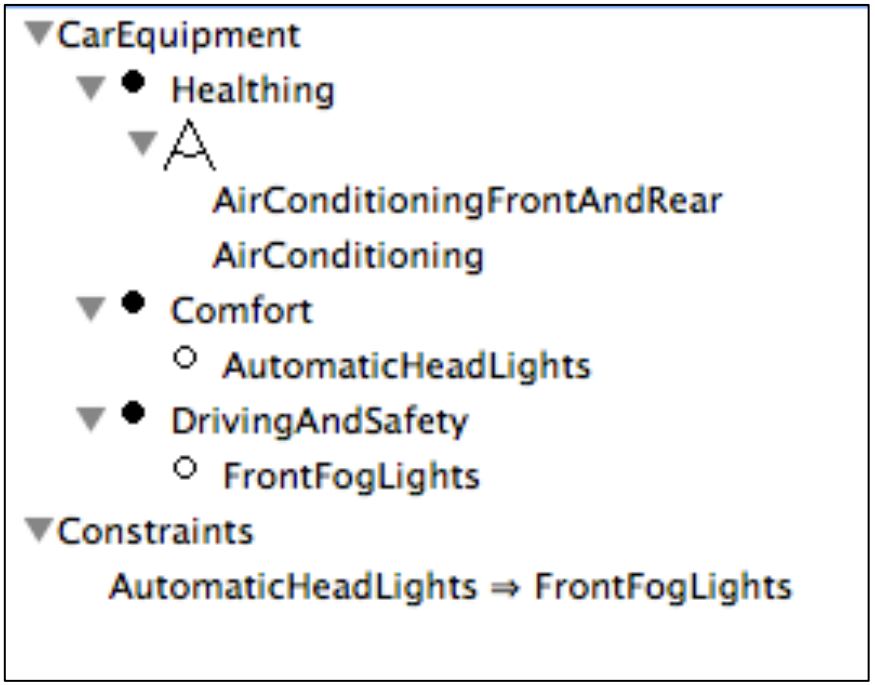


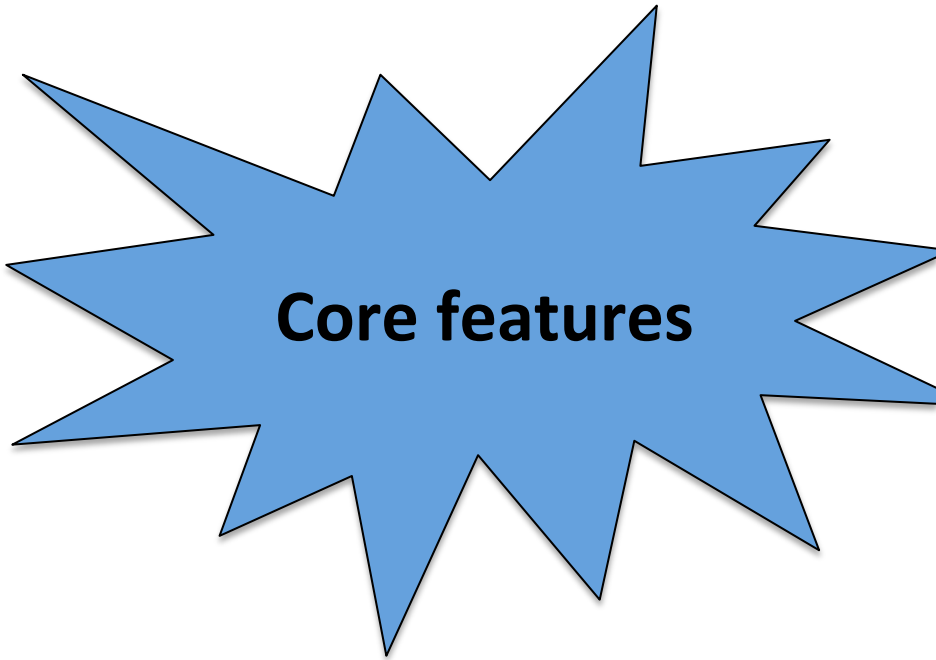

fm4
A
D
E
F
B
C
I
J
L
K
M
P
N
O
Constraints
C ⇒ ¬E
J ⇒ C


Dead feature

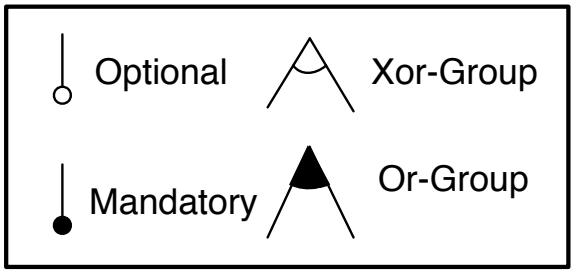

False optional feature

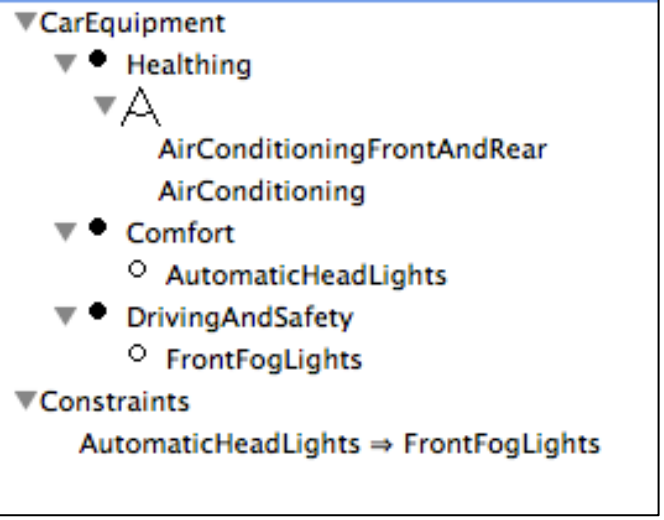

Optional
Xor-Group
Mandatory
Or-Group

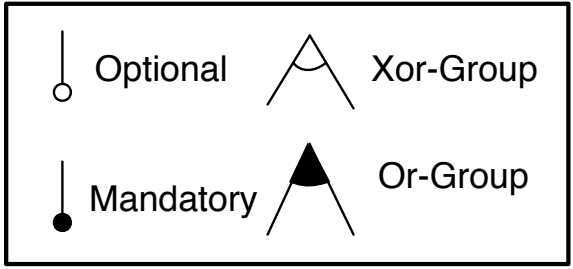


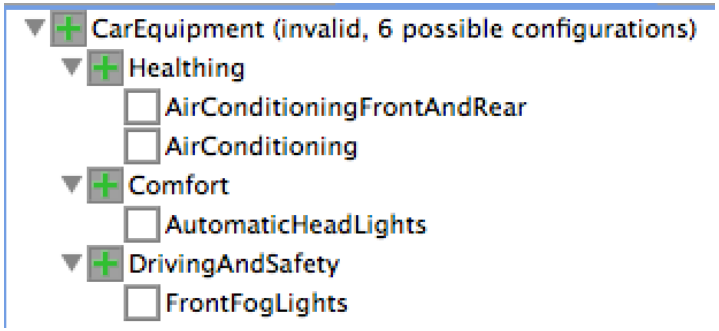


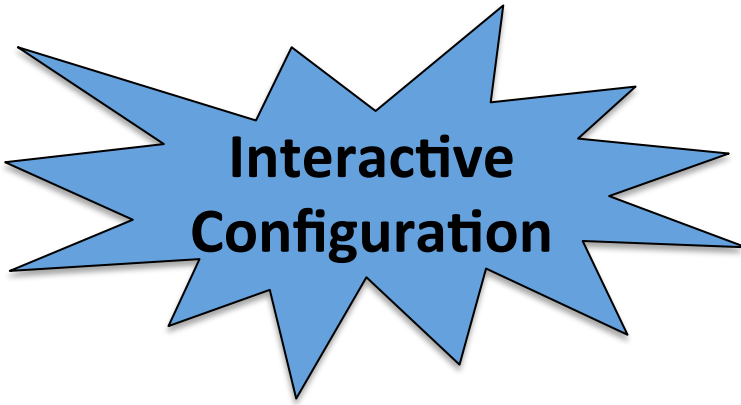
{CarEquipment, Comfort, DrivingAndSafety, Healthing}

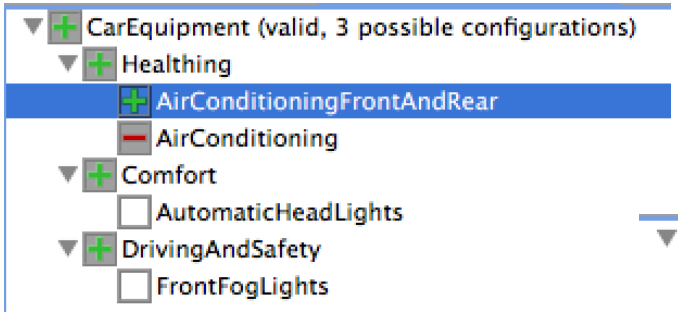


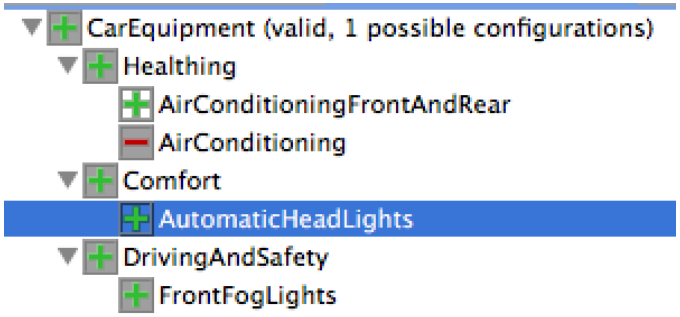

CarEquipment
Healthing
AirConditioningFrontAndRear
AirConditioning
Comfort
AutomaticHeadLights
DrivingAndSafety
FrontFogLights
Constraints
AutomaticHeadLights ⇒ FrontFogLights


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

	PL													CP						DL			Multi			Others																No support		
	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]
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	C	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

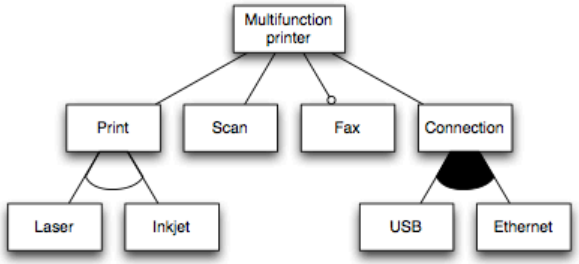
Decision problems and complexity

- 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

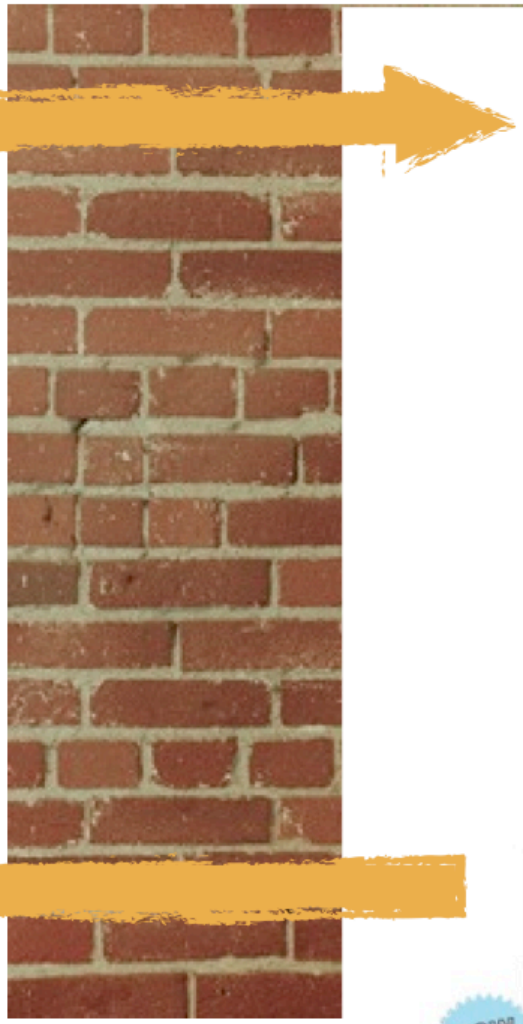
How to automate analysis of your feature models?

Binary Decision Diagram (BDD)
SAT solver

Typical implementations



result



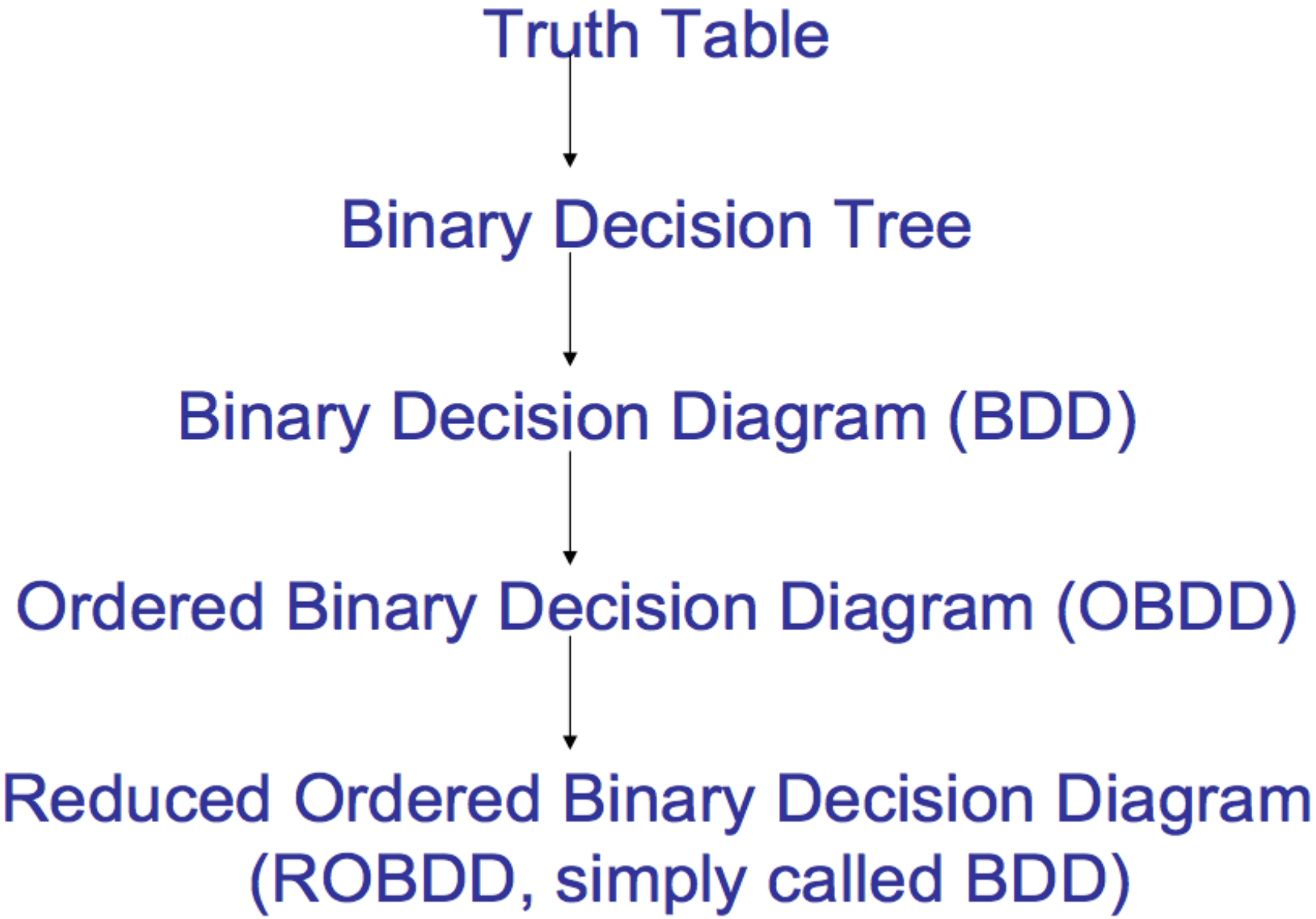
logics

solvers

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

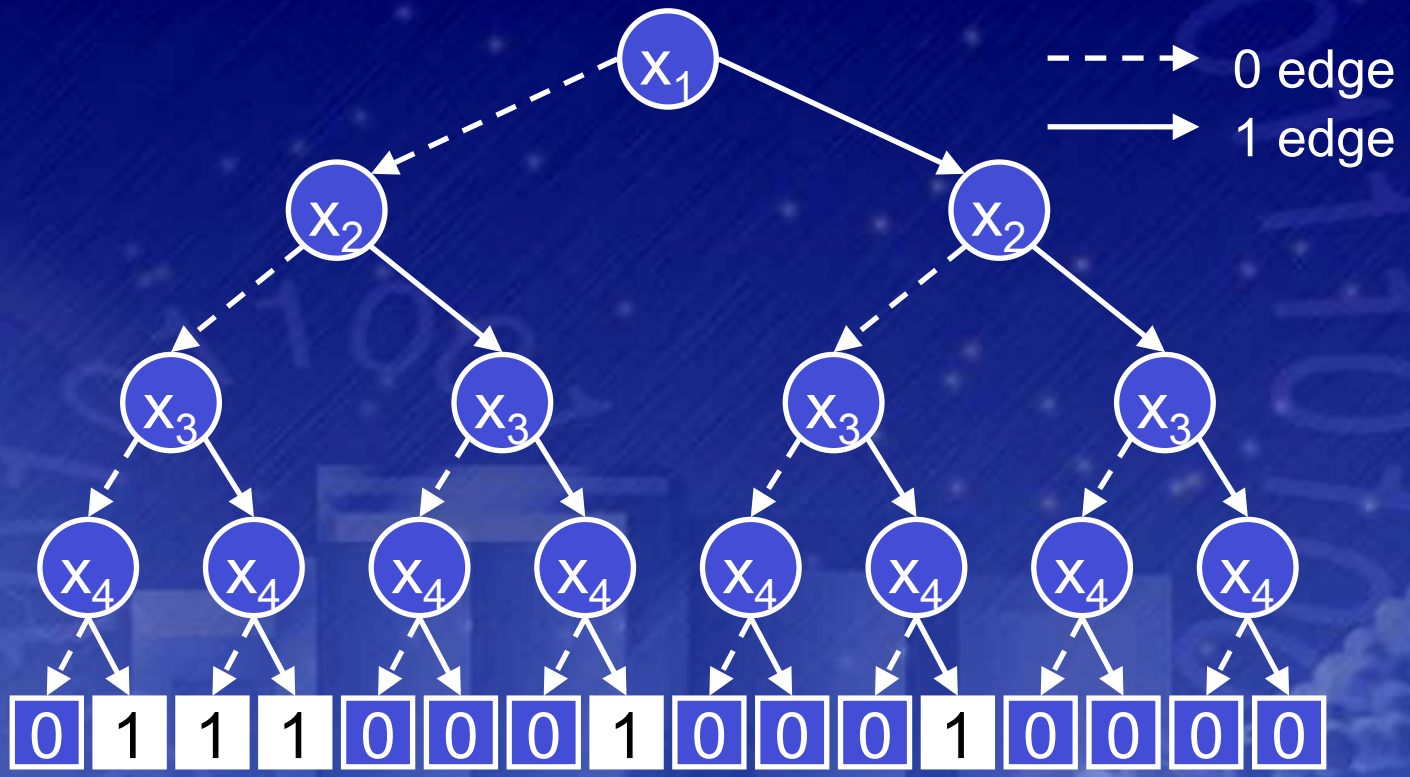
BDDs from Truth Tables



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

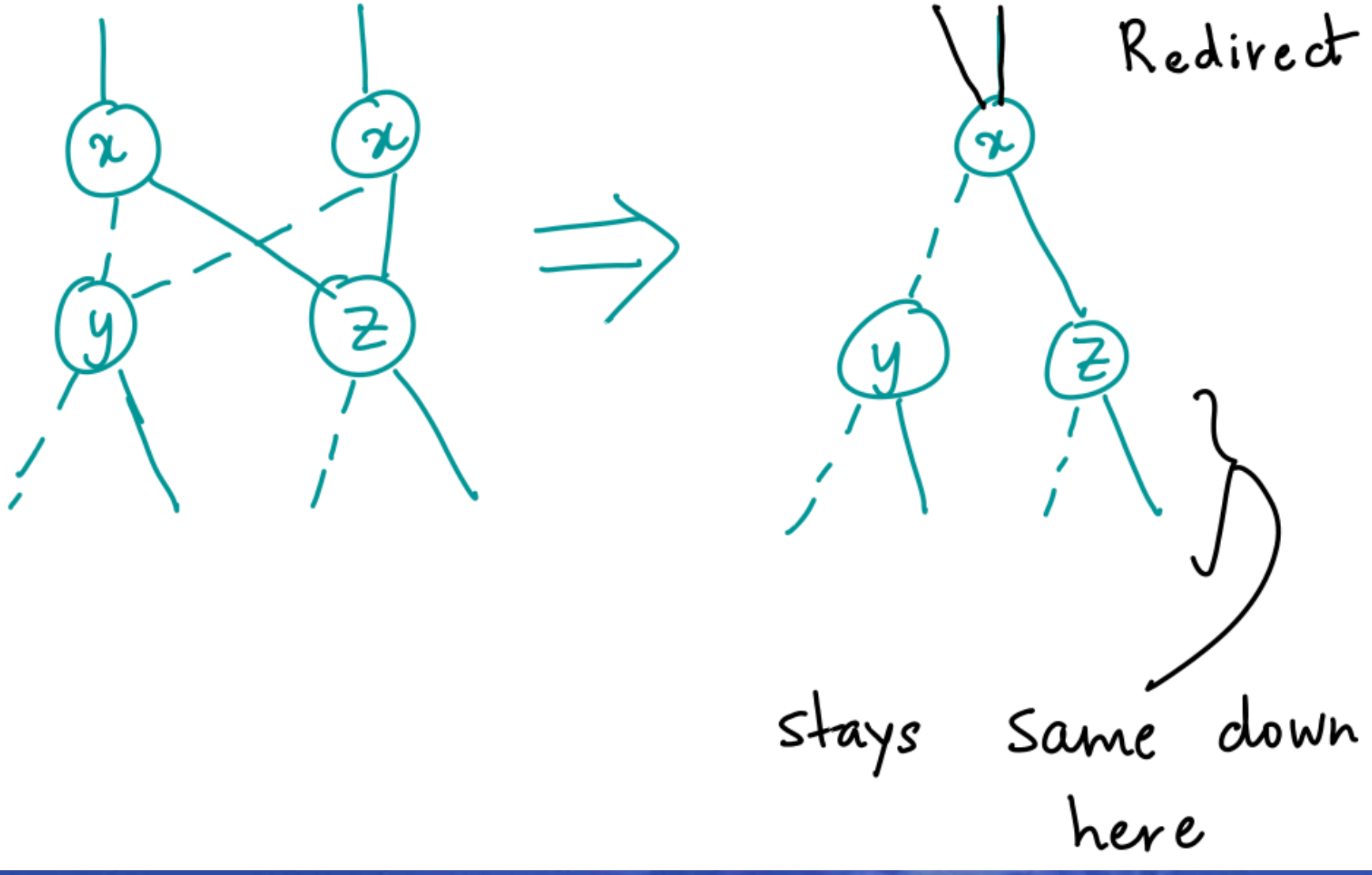
- Identify Redundancies

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

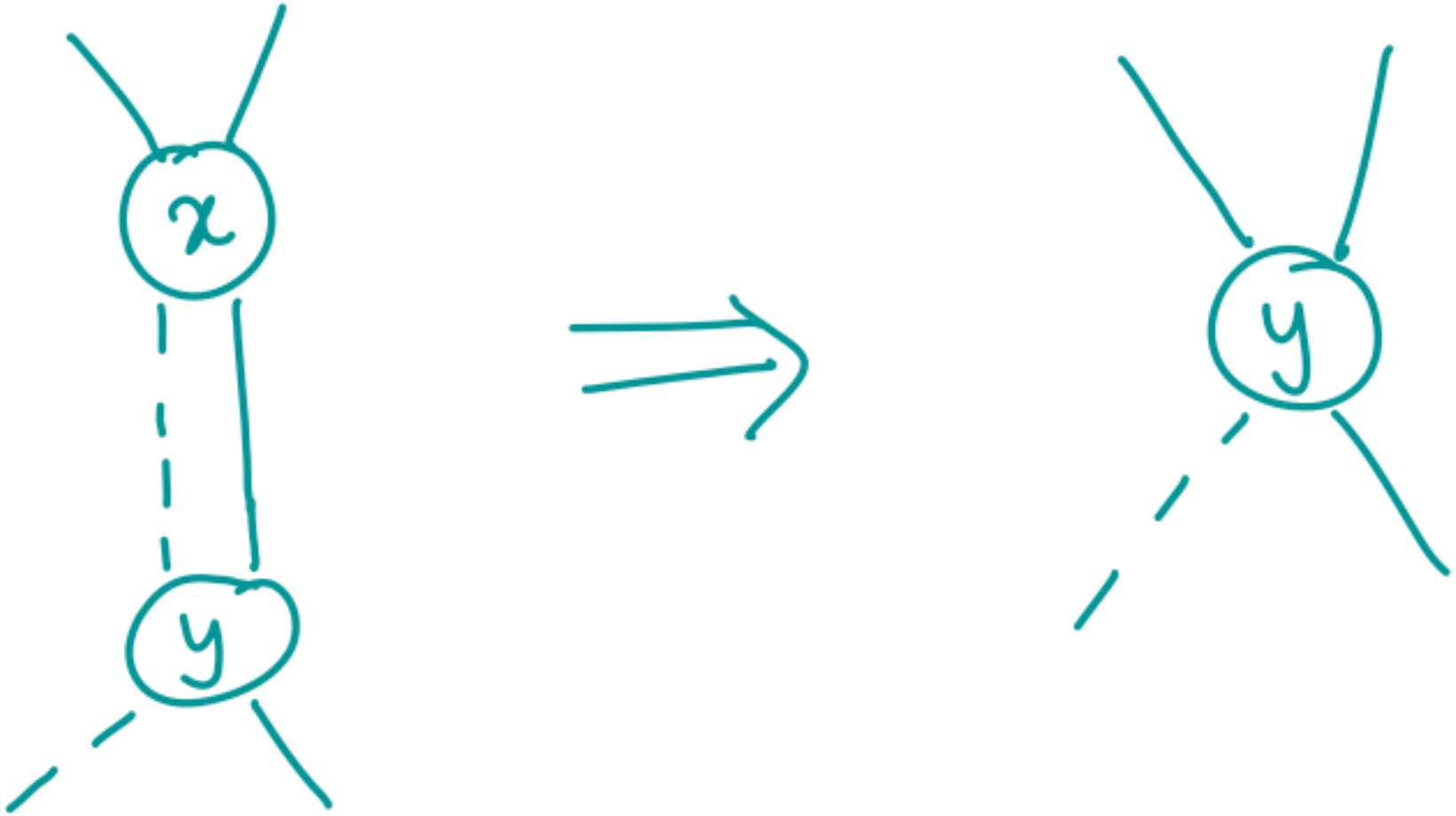
Merge equivalent leaves

"a" is either 0 or 1

Merge isomorphic nodes

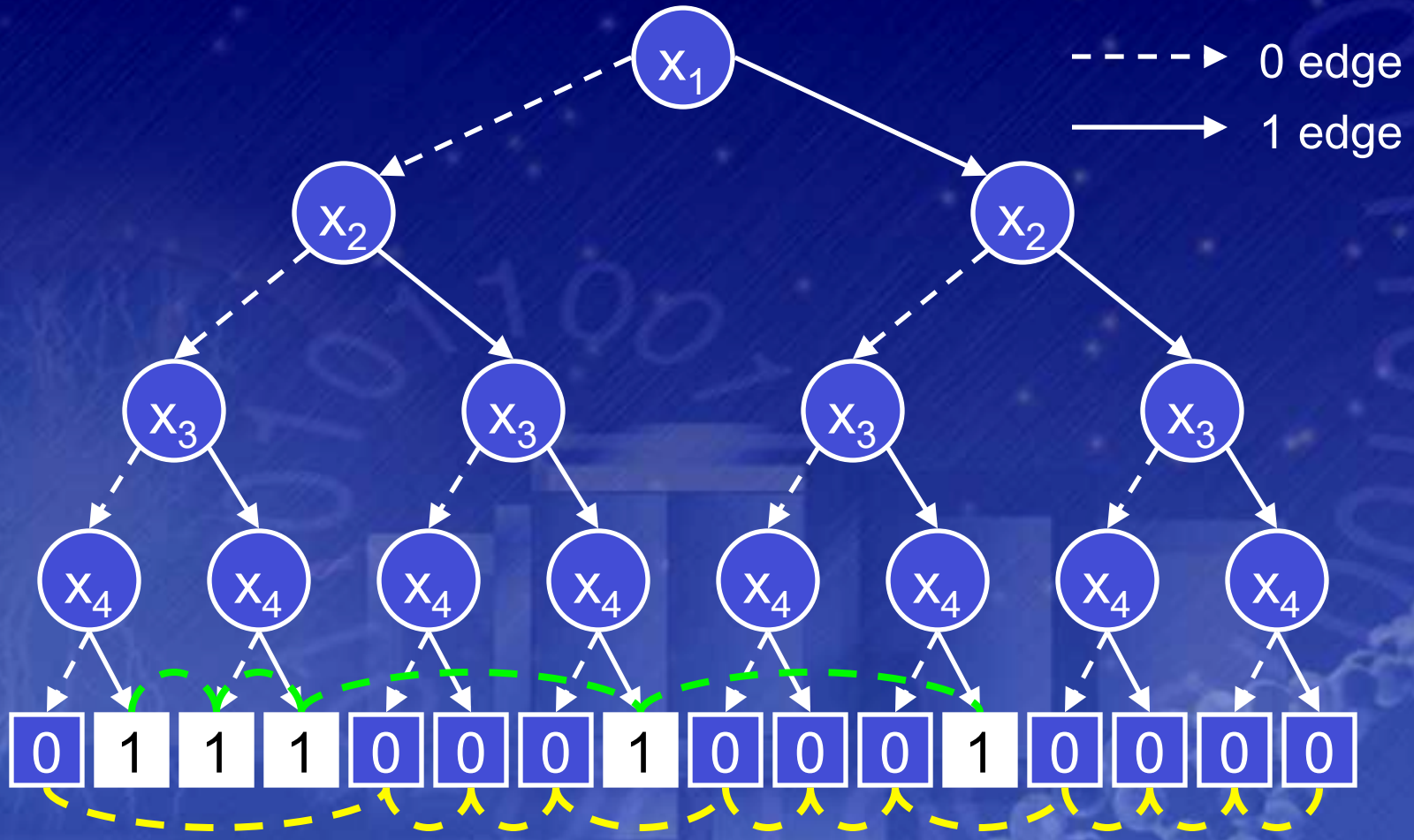


Eliminate redundant tests



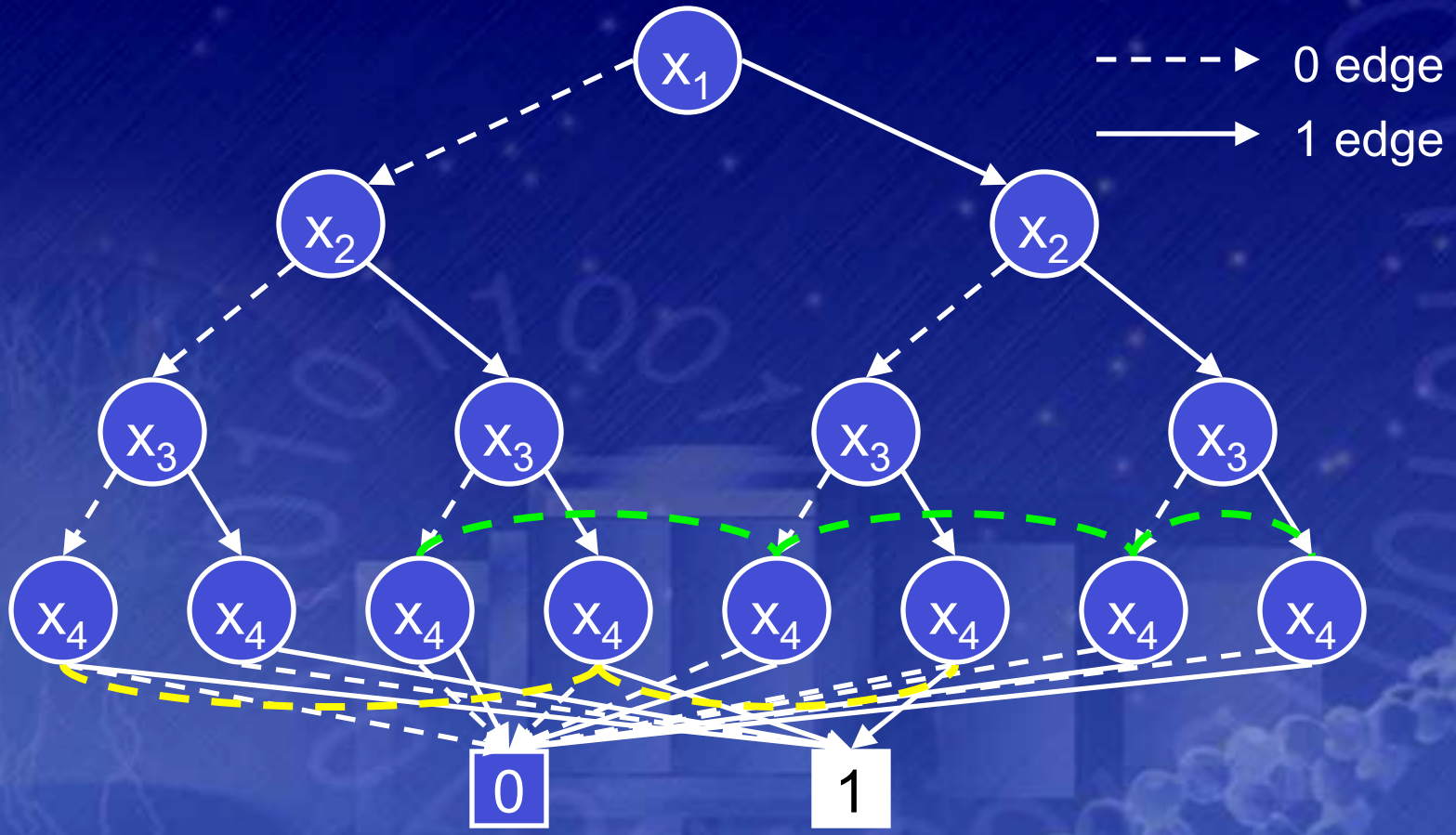
Binary Decision Diagrams

- Collapse redundant nodes.



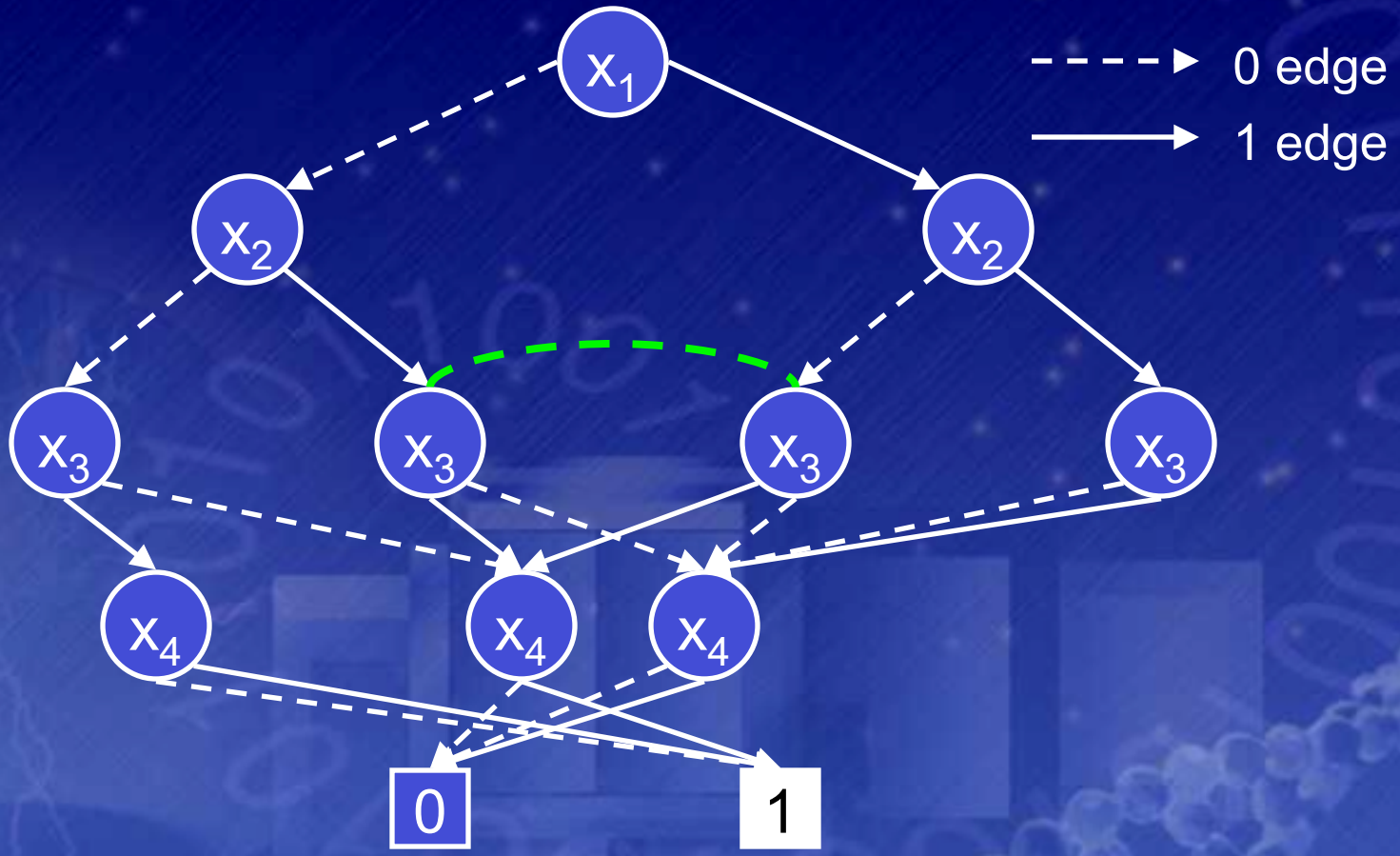
Binary Decision Diagrams

- Collapse redundant nodes.



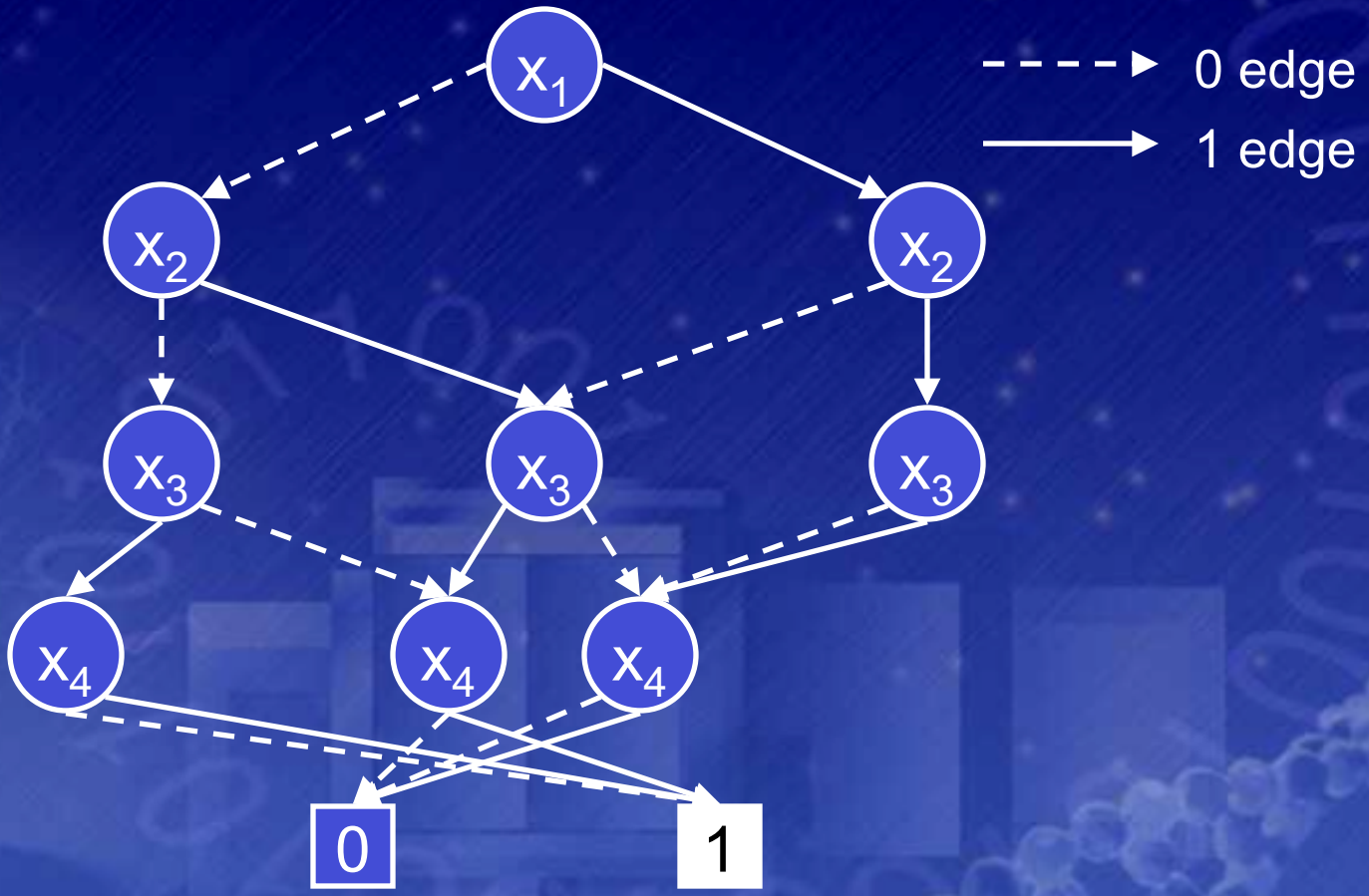
Binary Decision Diagrams

- Collapse redundant nodes.



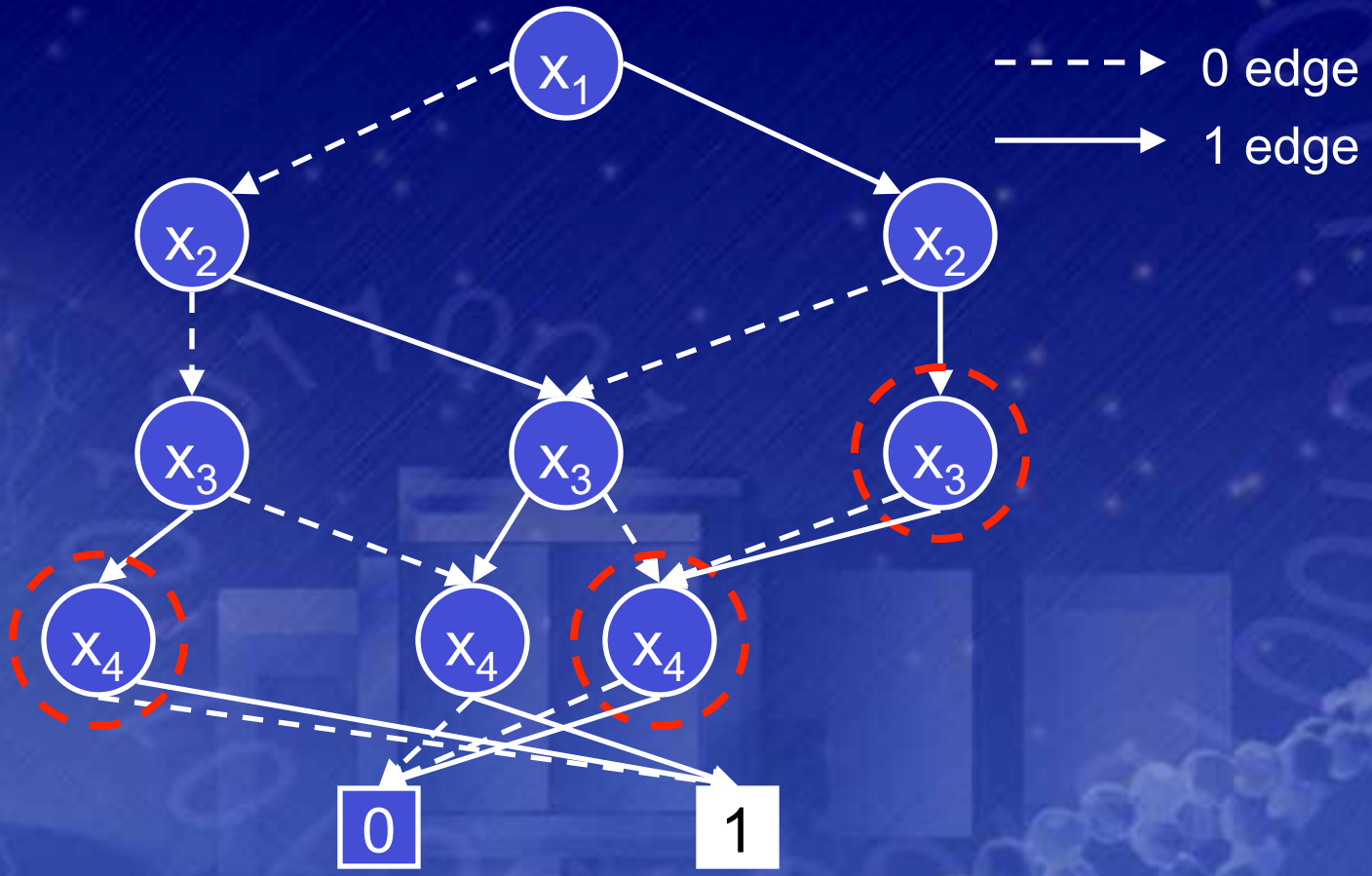
Binary Decision Diagrams

- Collapse redundant nodes.



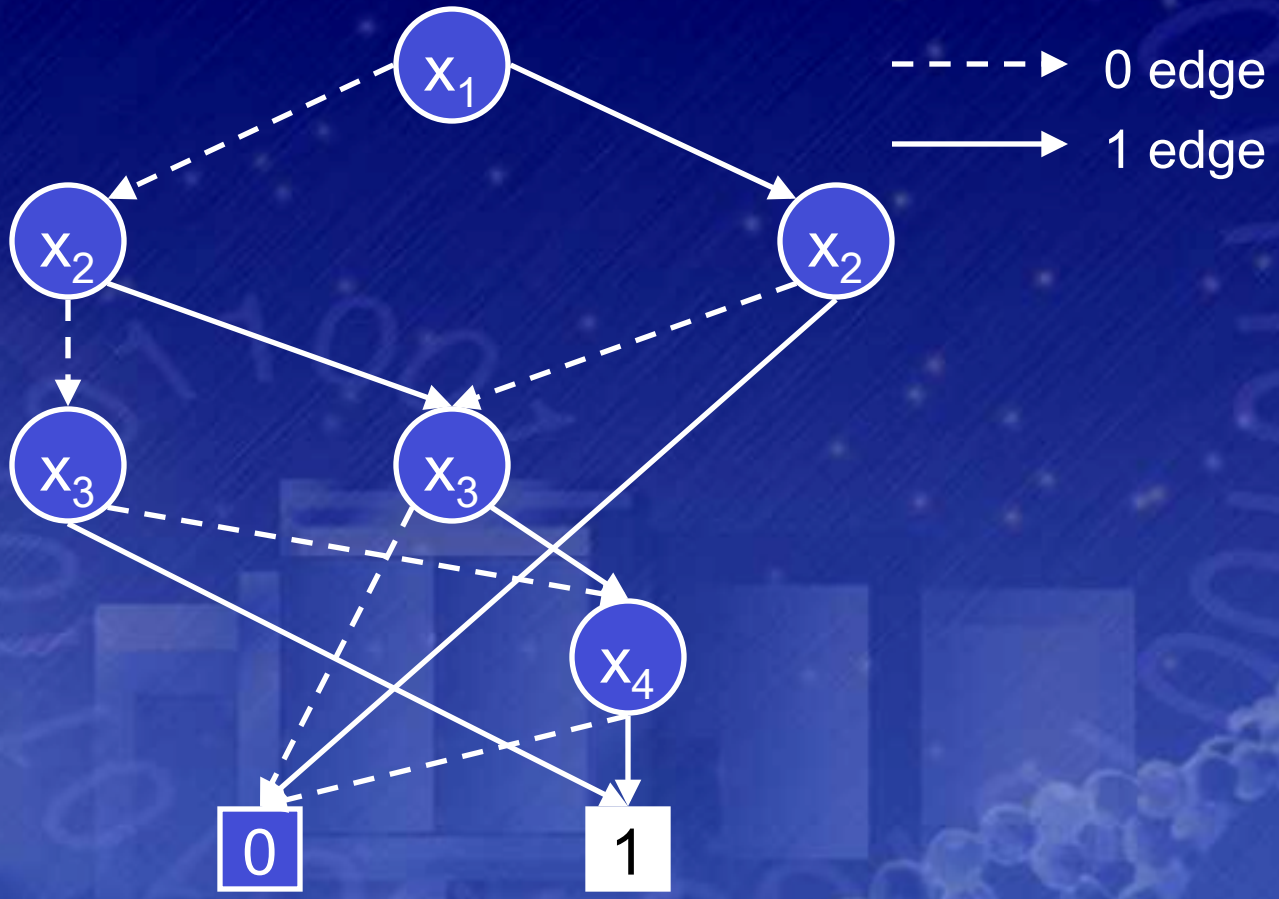
Binary Decision Diagrams

- Eliminate unnecessary nodes.



Binary Decision Diagrams

- Eliminate unnecessary nodes.

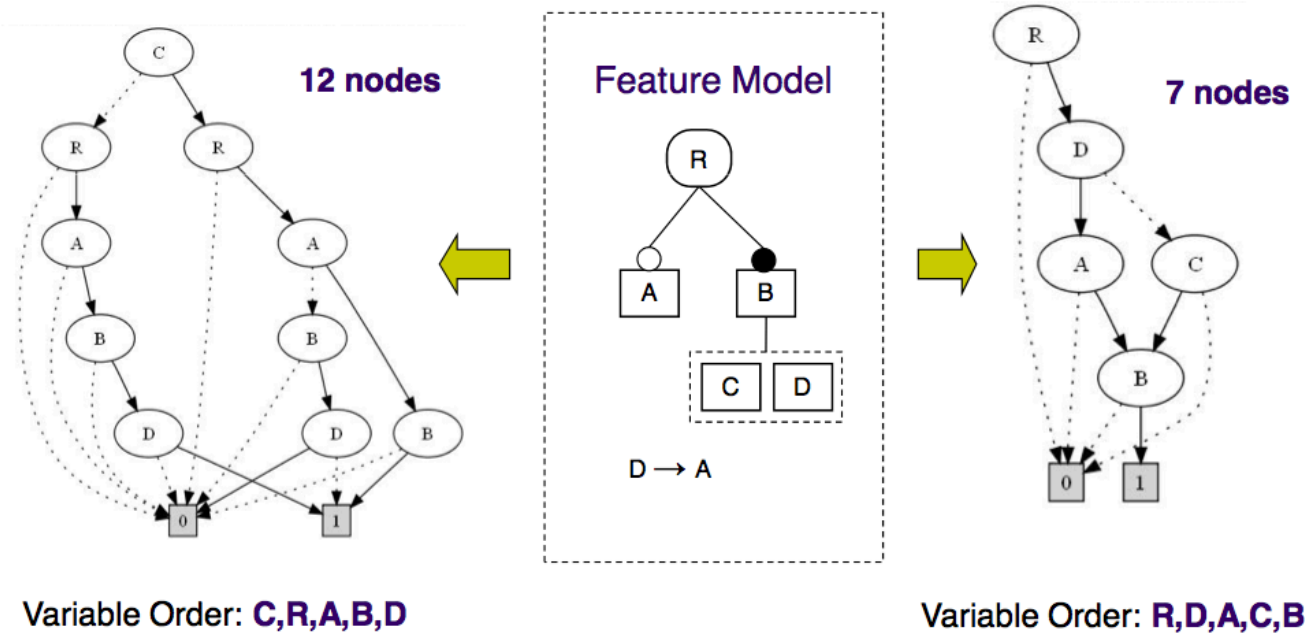


Binary Decision Diagrams (BDDs)

- 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

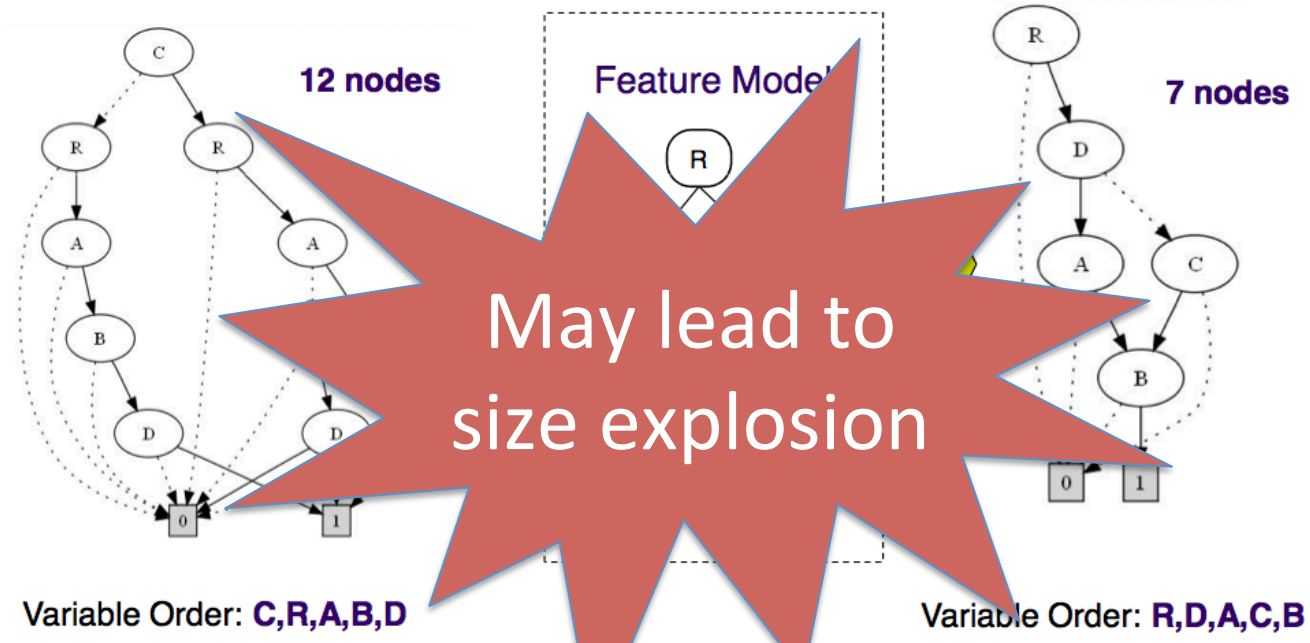
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**



[Mendonca, slide]

Binary Decision Diagrams (BDDs): Theoretical Problem

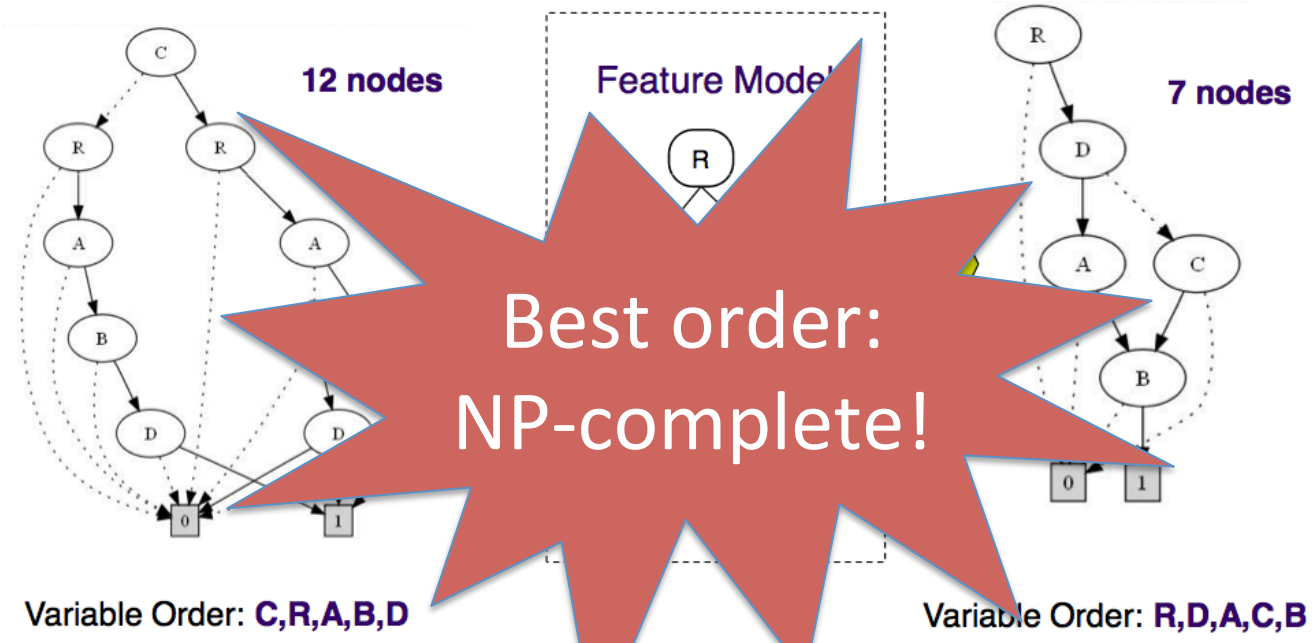
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**



[Mendonca, slide]

Binary Decision Diagrams (BDDs): Theoretical Problem

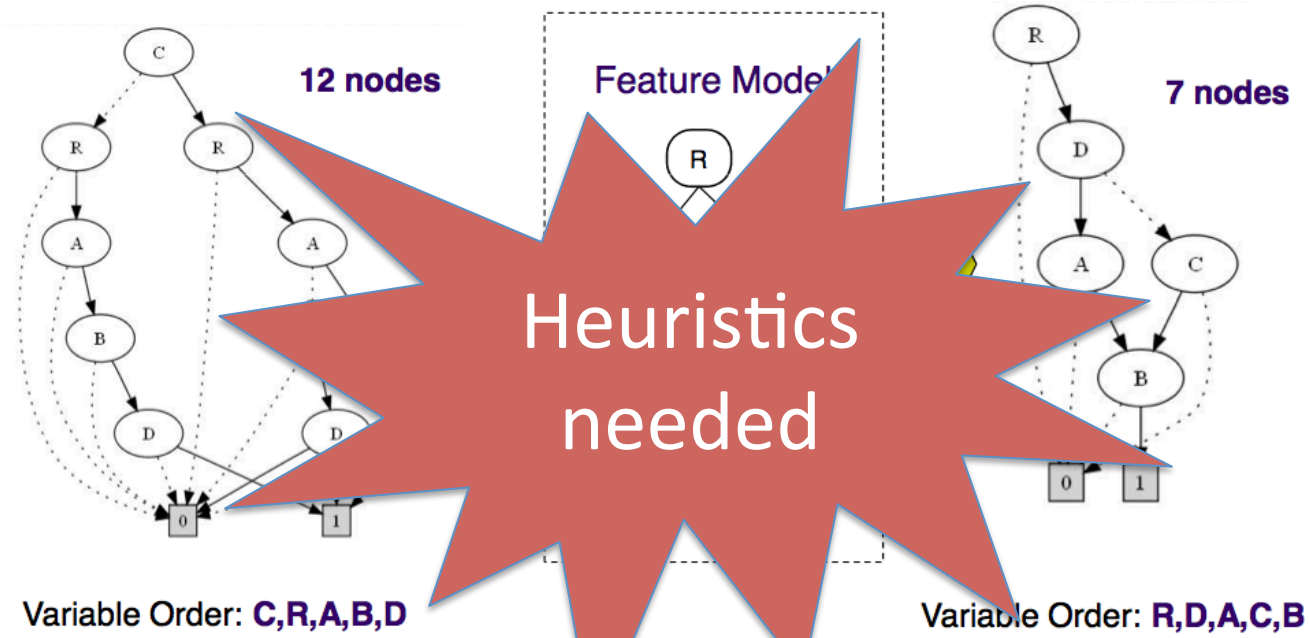
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**



[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**



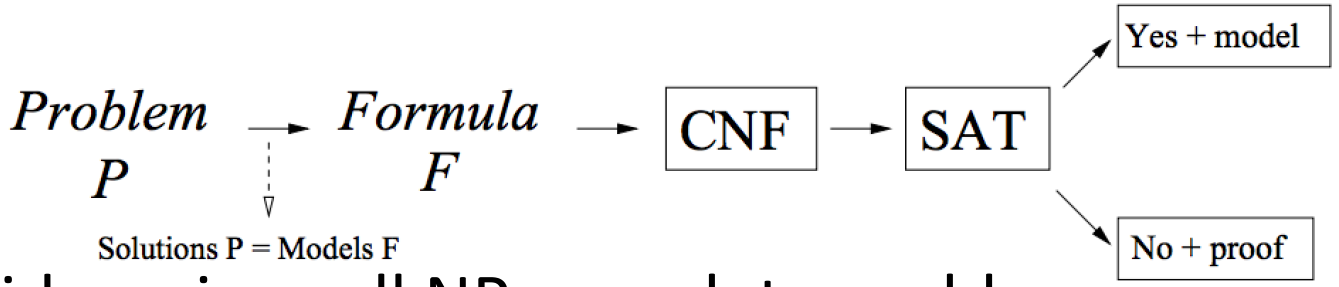
[Mendonca, slide]

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;

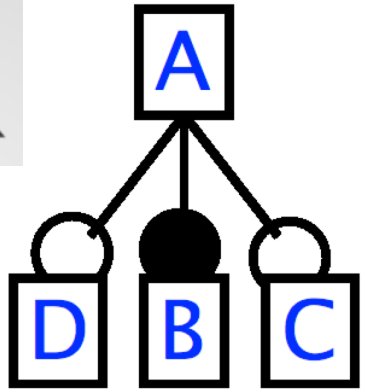
satisfy(ϕ) {
  if every clause of ϕ has a true literal, return T;
  if BCP() == F, return F;
  assign appropriate values to all pure literals;
  choose an $x \in V$ that is unassigned in A,
    and choose $v \in \{T, F\}$.
  $A(x) = v$;
  if satisfy(ϕ) return T;
  $A(x) = \neg v$;
  if satisfy(ϕ) return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

How to automate analysis of your feature models?

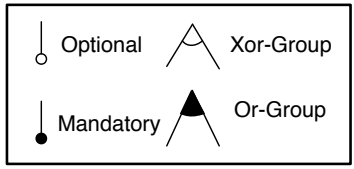
Let us use BDDs and SAT solvers

A ^
A ⇔ B ^
C => A ^
D => A

FAMiliAR



FM



φ

```
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")
```

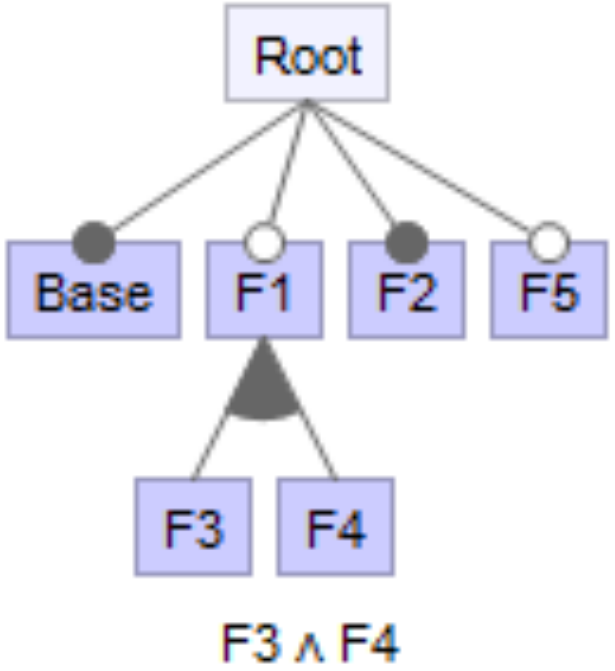
```
fml> c1 = cores fm1
c1: (SET) {B;A}
fml> c1bis = cores fm1bis
```

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

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

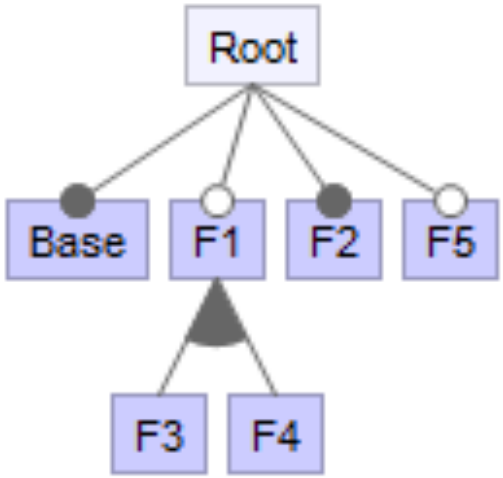
Consistency

- SAT-Solver
  - SAT(FM)



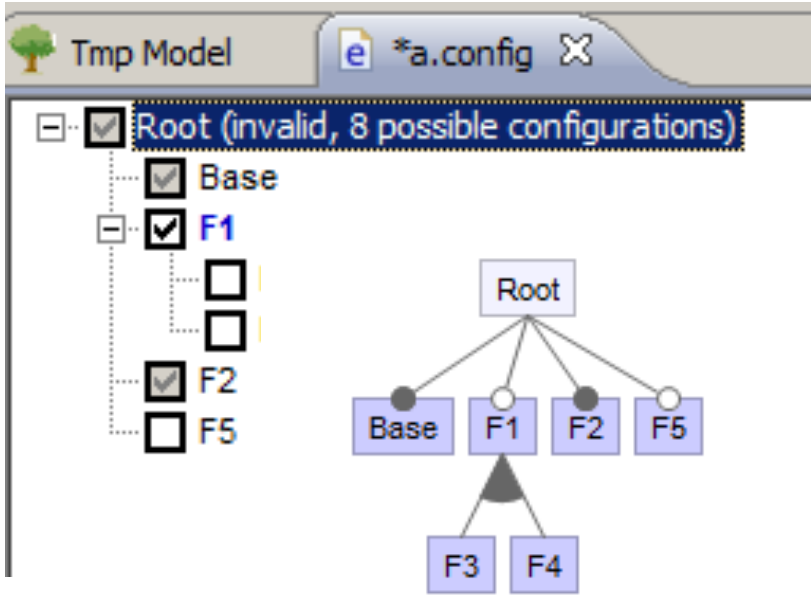
Core and dead features

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

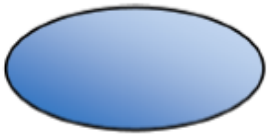
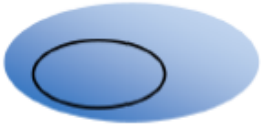
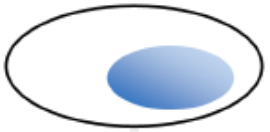
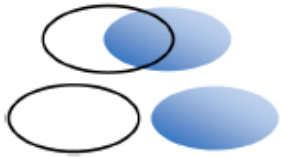


Partial configuration

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



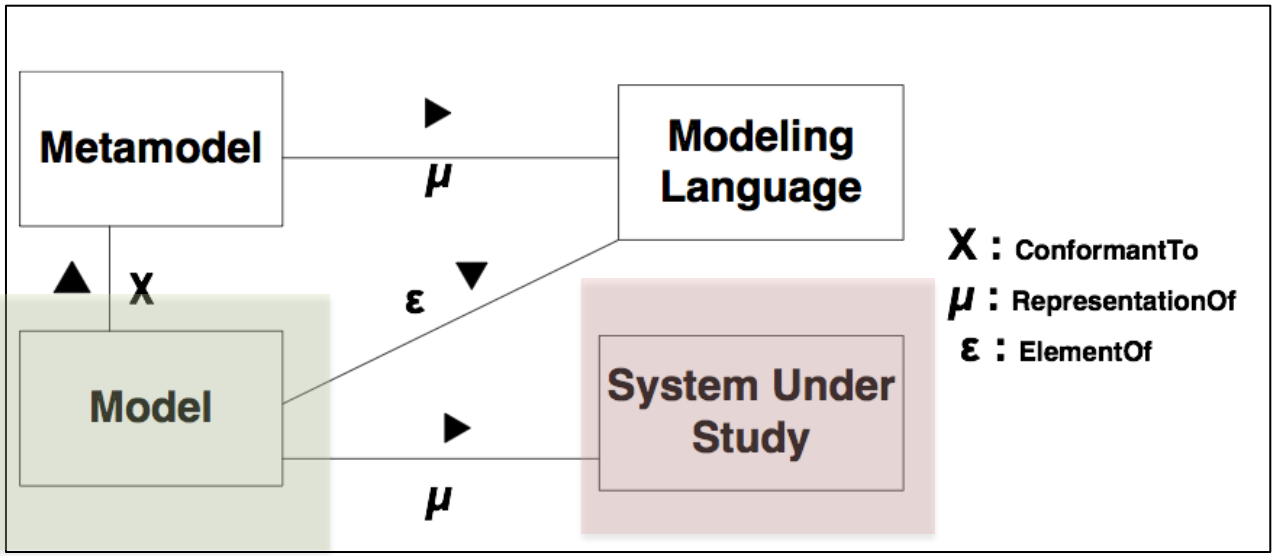
Relationship between feature models

	No Products Added	Products Added
No Products Deleted	 Refactoring	 Generalization
Products Deleted	 Specialization	 Arbitrary Edit

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

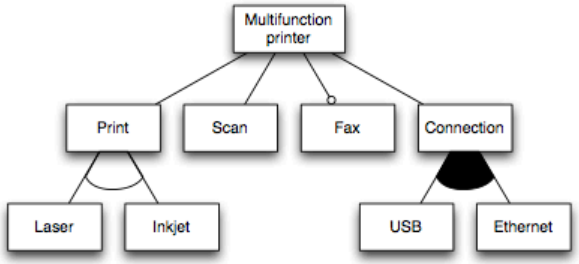
Recap

Feature Models

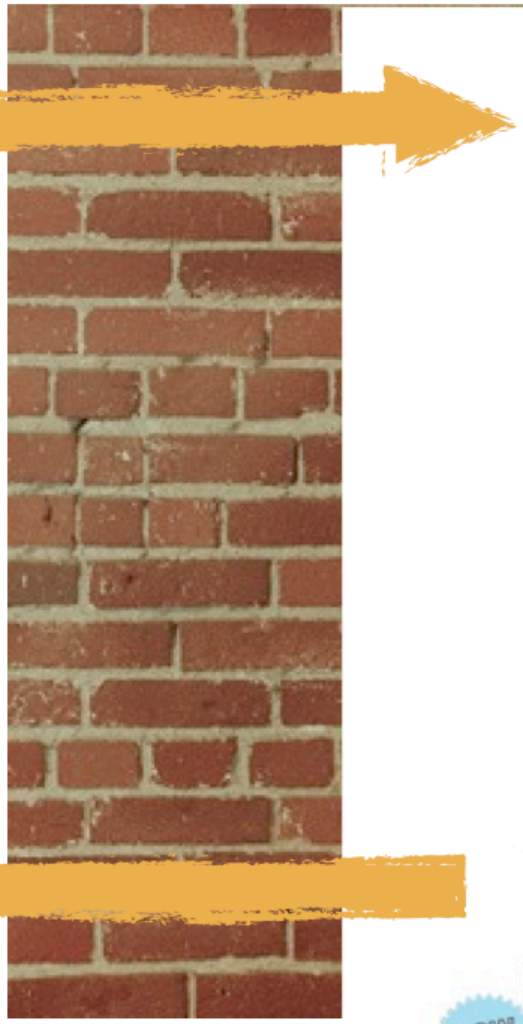


- CarEquipment
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights
- Constraints
  - AutomaticHeadLights ⇒ FrontFogLights

Typical implementations



result



logics

solvers

FAMiliAR

(FeAture Model scrIpt Language for manIpulation and Automatic Reasoning)

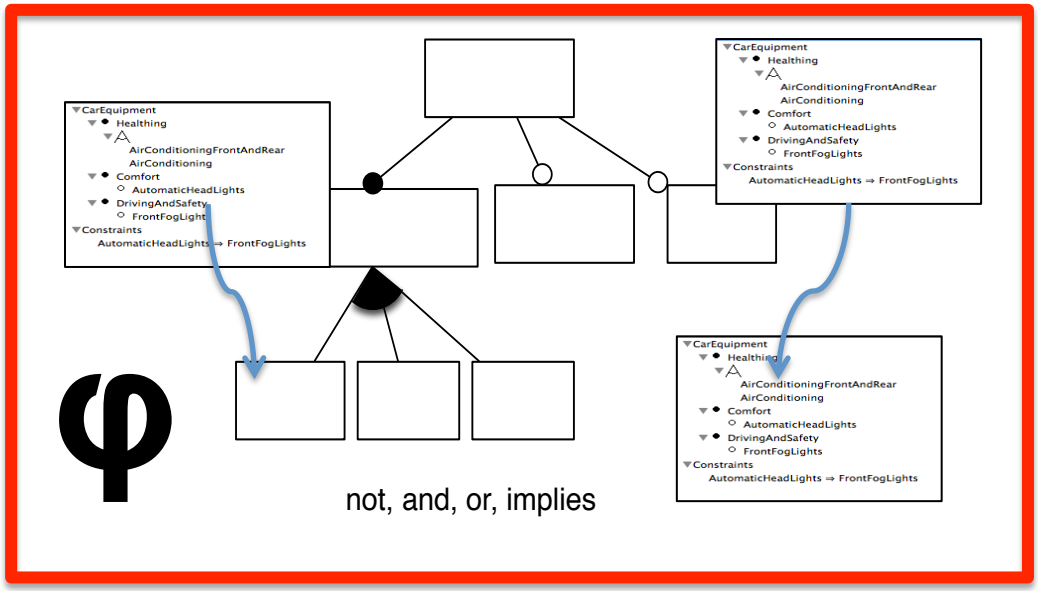
http://familiar-project.github.com/

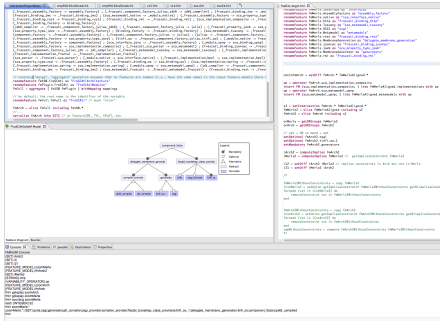
COMMON VARIABILITY LANGUAGE

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

IDE Feature

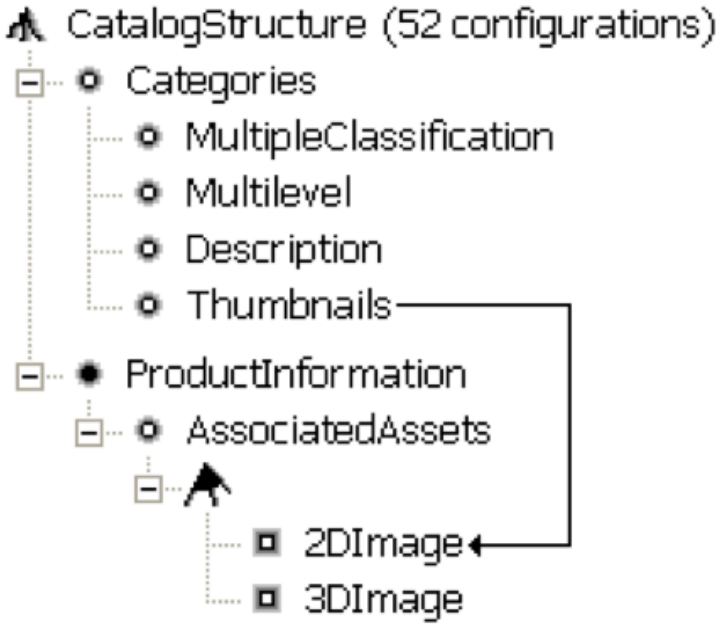
TVL
DIMACS

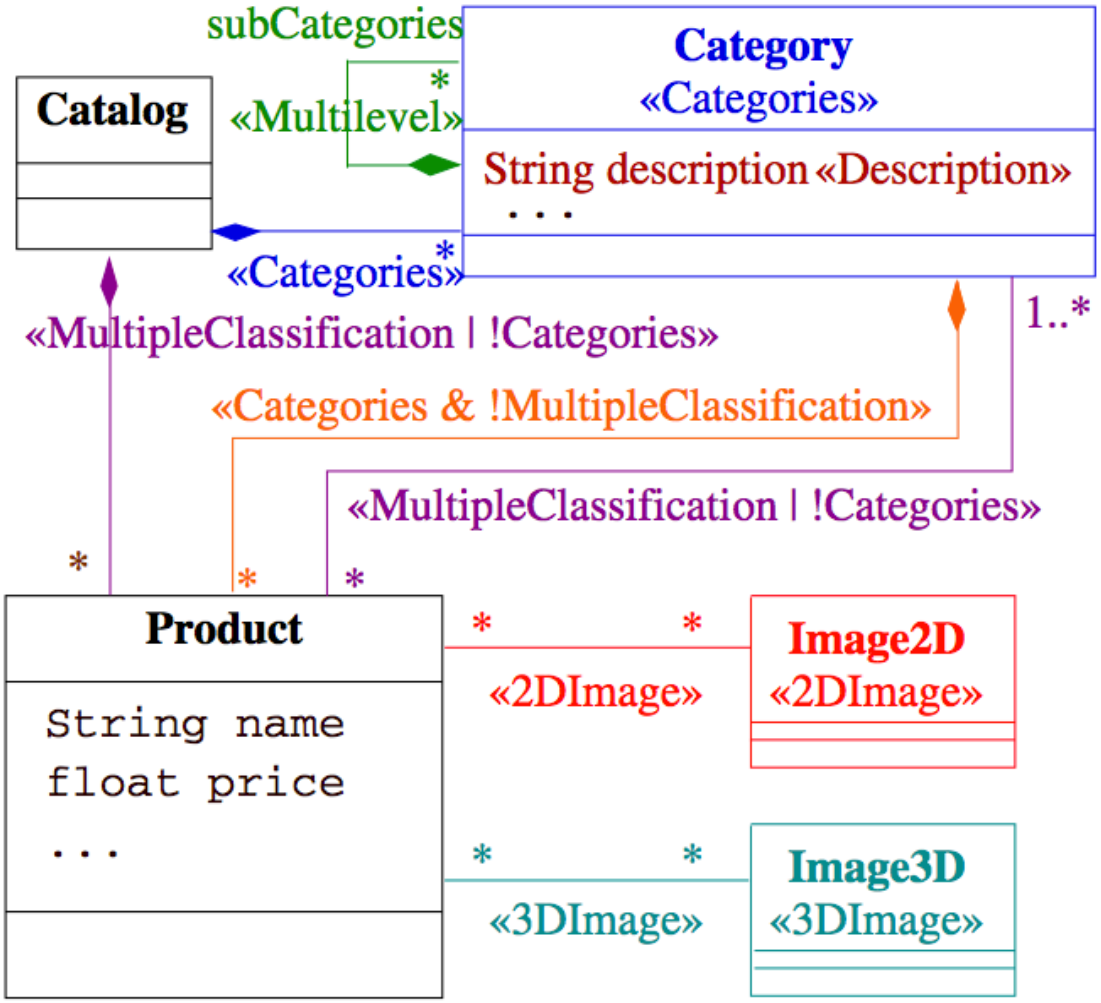


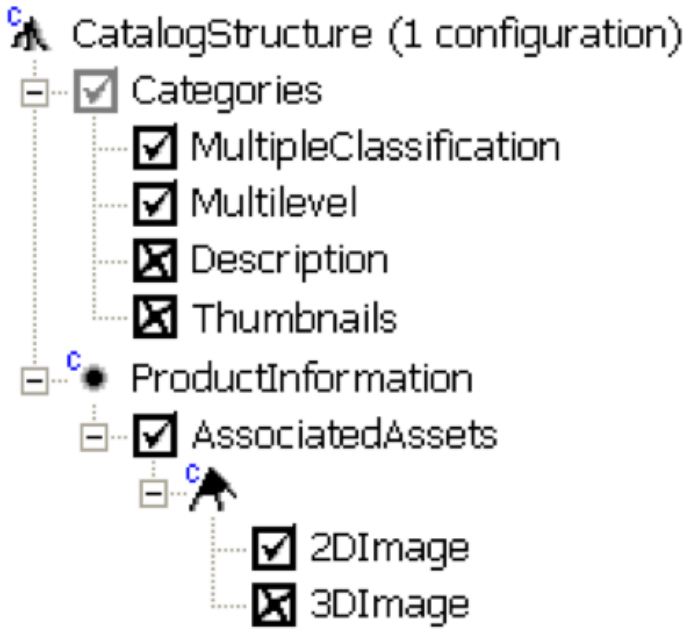


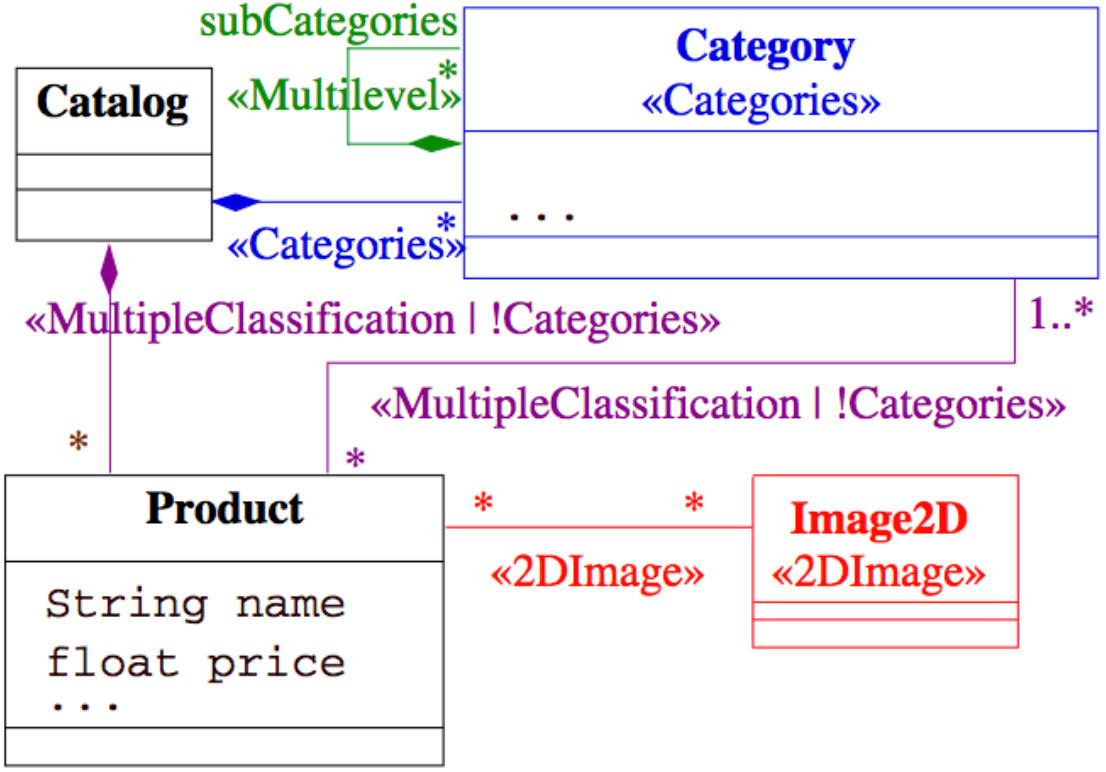
importing, exporting, composing, decomposing, editing, configuring, reverse engineering, computing "diffs", refactoring, testing, and reasoning about (multiple) variability models

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

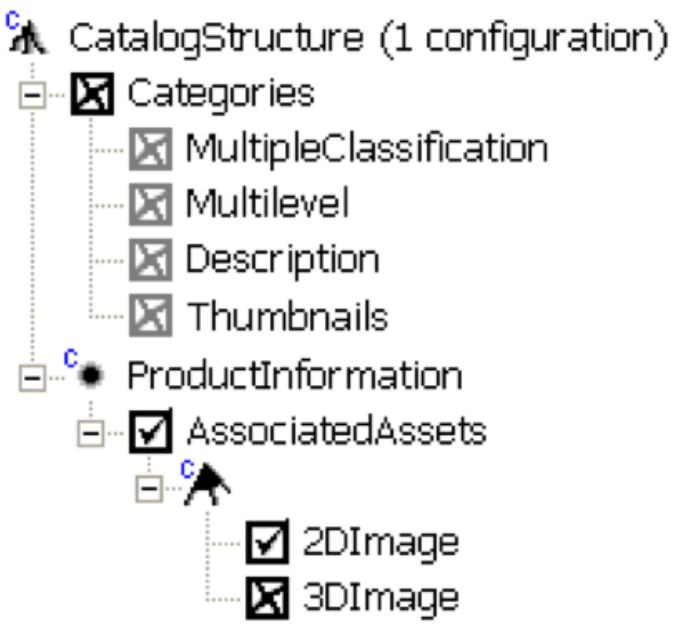

CatalogStructure (52 configurations)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage

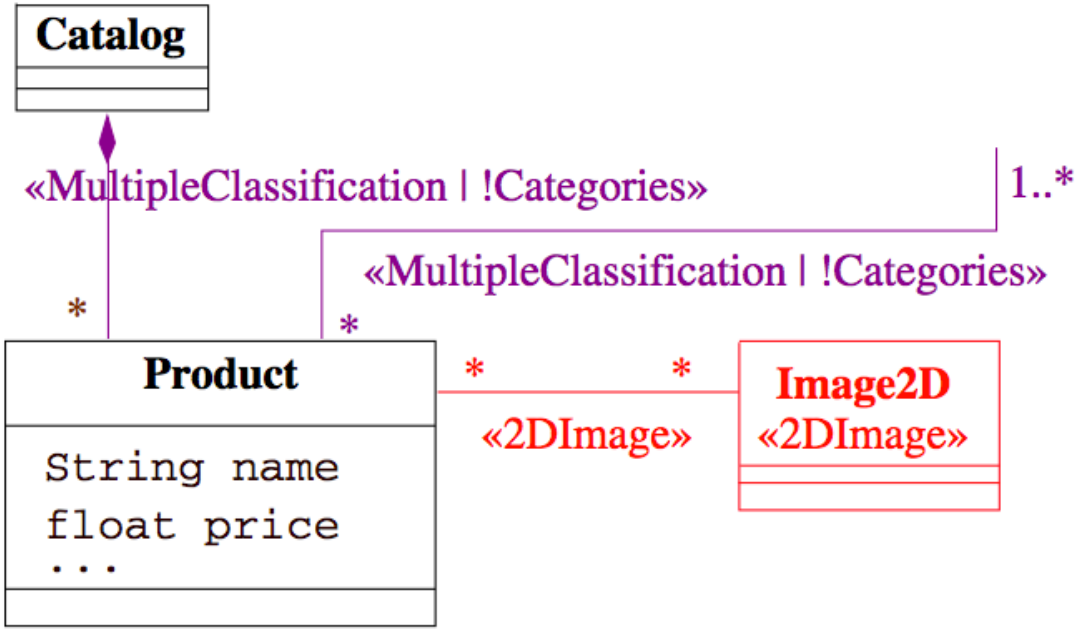

subCategories
Catalog
«Multilevel»
Category
«Categories»
String description «Description»
. . .
«Categories»
«MultipleClassification | !Categories»
«Categories & !MultipleClassification»
«MultipleClassification | !Categories»
1..*
Product
String name
float price
. . .
«2DImage»
Image2D
«2DImage»
«3DImage»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


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

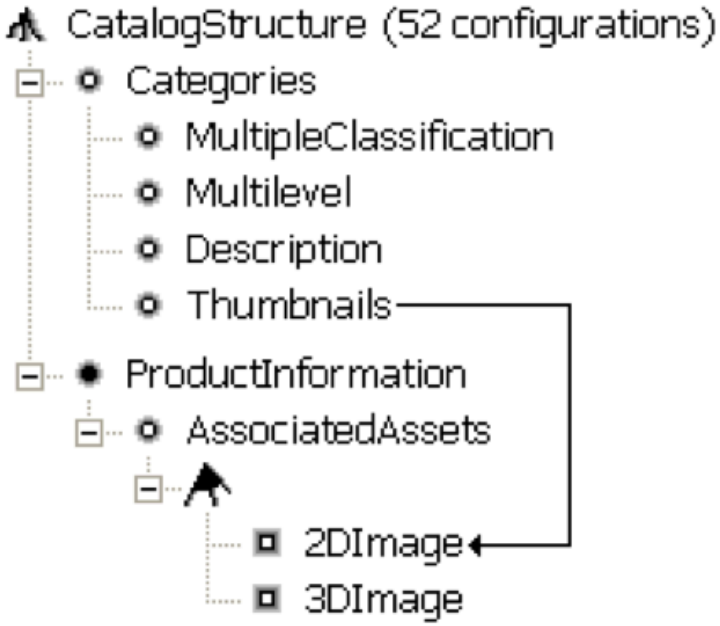

subCategories
Catalog
«Multilevel»
Category
«Categories»
...
«Categories»
«MultipleClassification | !Categories»
1..*
«MultipleClassification | !Categories»
Product
String name
float price
...
«2DImage»
Image2D
«2DImage»

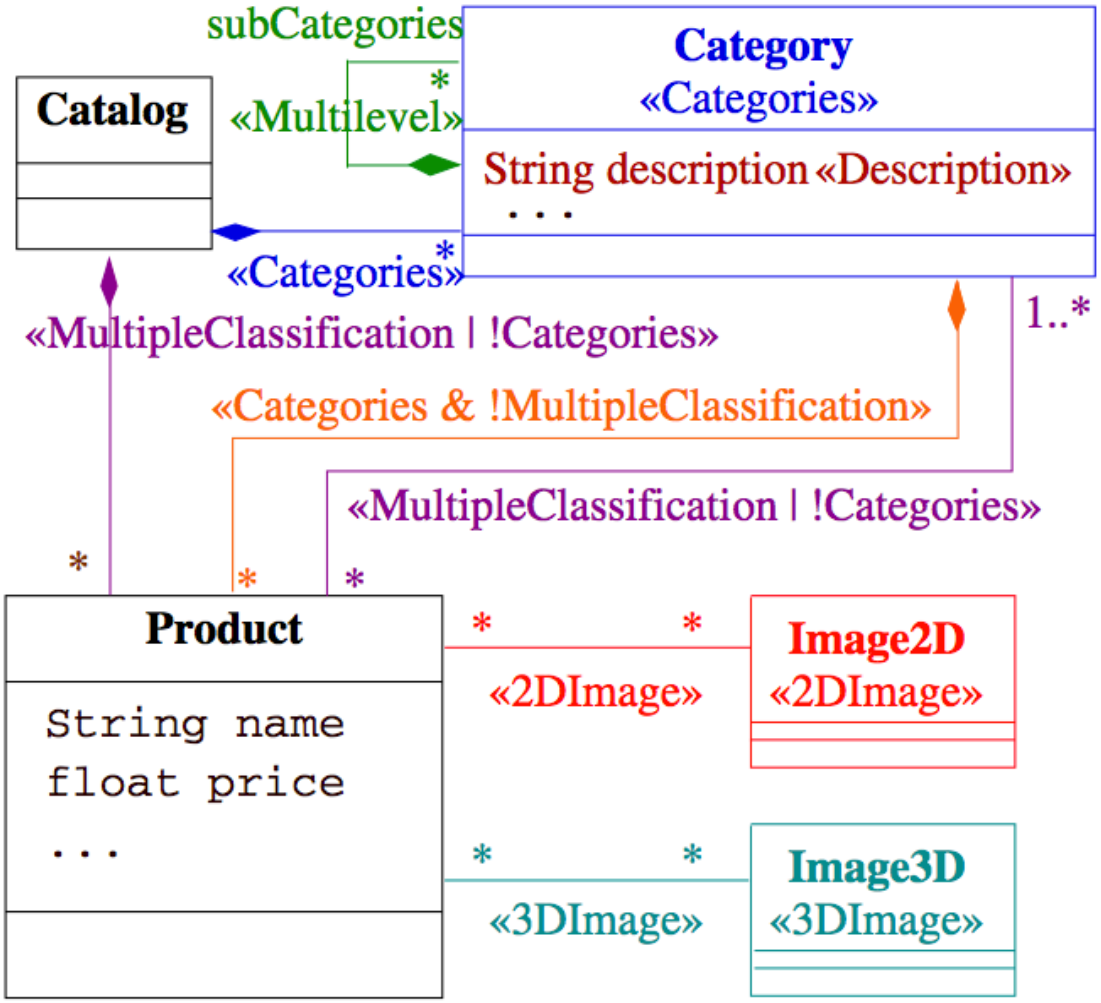
Ooops


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»

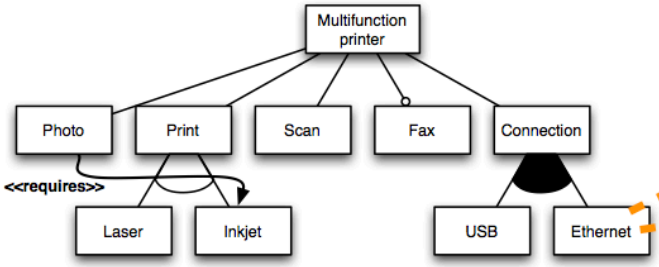
Safe composition? No!



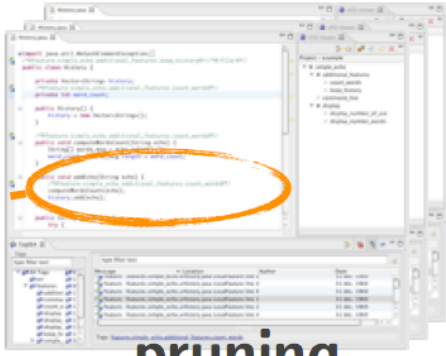


Product Derivation

feature model



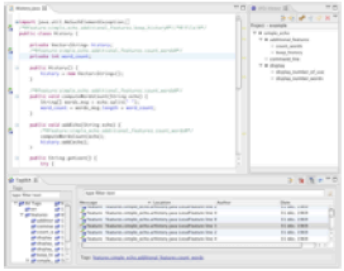
variable model and code assets



pruning,
composition,
weaving,
transformation
...

configuration

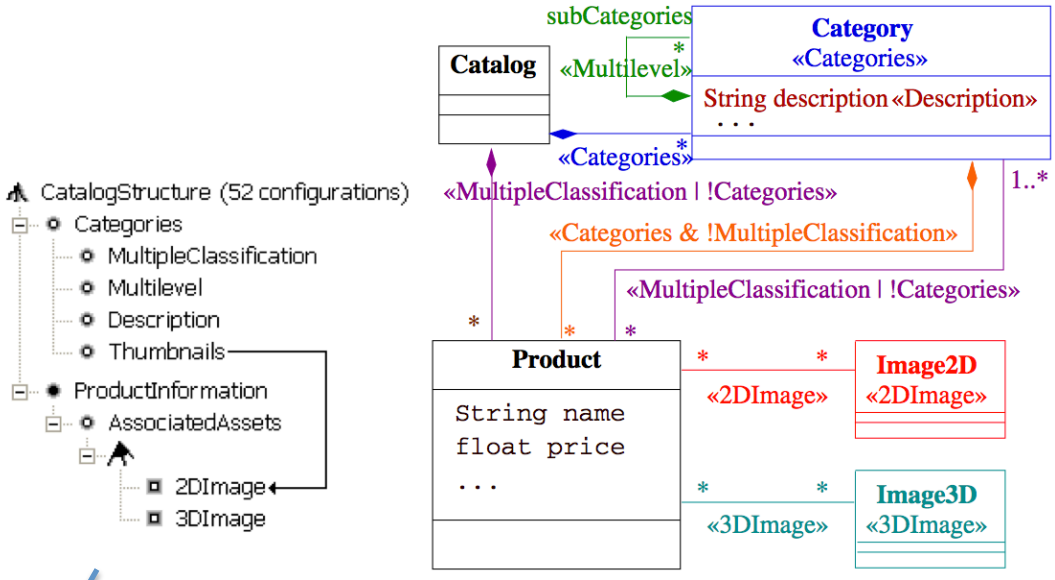
{ MP, Photo, Print, Inkjet, Scan,
Fax, Connection, USB, Ethernet }



product spec

product

Safe composition: how does it work?



$$
\begin{array}{ll}
q_{FM} = & \\
\textit{root:} & cs \wedge \\
\textit{child-parent:} & (\texttt{ct} \Rightarrow \texttt{cs}) \wedge (\texttt{mc} \Rightarrow \texttt{ct}) \wedge (\texttt{ml} \Rightarrow \texttt{ct}) \wedge \\
 & (\texttt{ds} \Rightarrow \texttt{ct}) \wedge (\texttt{tn} \Rightarrow \texttt{ct}) \wedge (\texttt{pi} \Rightarrow \texttt{cs}) \wedge \\
 & (\texttt{aa} \Rightarrow \texttt{pi}) \wedge (\texttt{i2} \Rightarrow \texttt{aa}) \wedge (\texttt{i3} \Rightarrow \texttt{aa}) \wedge \\
\textit{group:} & (\texttt{aa} \Rightarrow \textit{choice}_{1,2}(\texttt{i2}, \texttt{i3})) \wedge \\
\textit{mandatory:} & (\texttt{cs} \Rightarrow \texttt{pi}) \wedge \\
\textit{additional:} & (\texttt{tn} \Rightarrow \texttt{i2})
\end{array}
$$

Another propositional formula