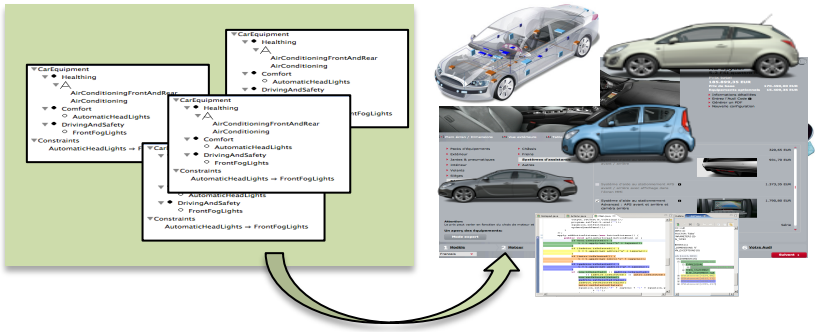


Model-based Software Product Lines

Variability Models (Part 2)

Mathieu Acher

Maître de Conférences

mathieu.acher@irisa.fr

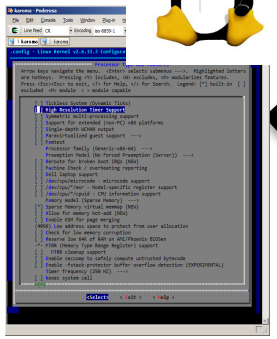
UNIVERSITÉ DE RENNES 1

istic Informatique Électronique

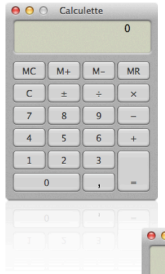
Material

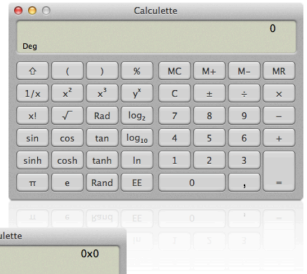
http://mathieuacher.com/teaching/M2R/

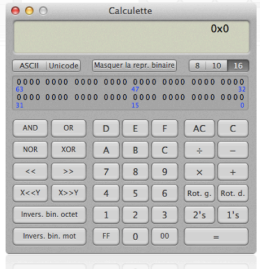
Previously



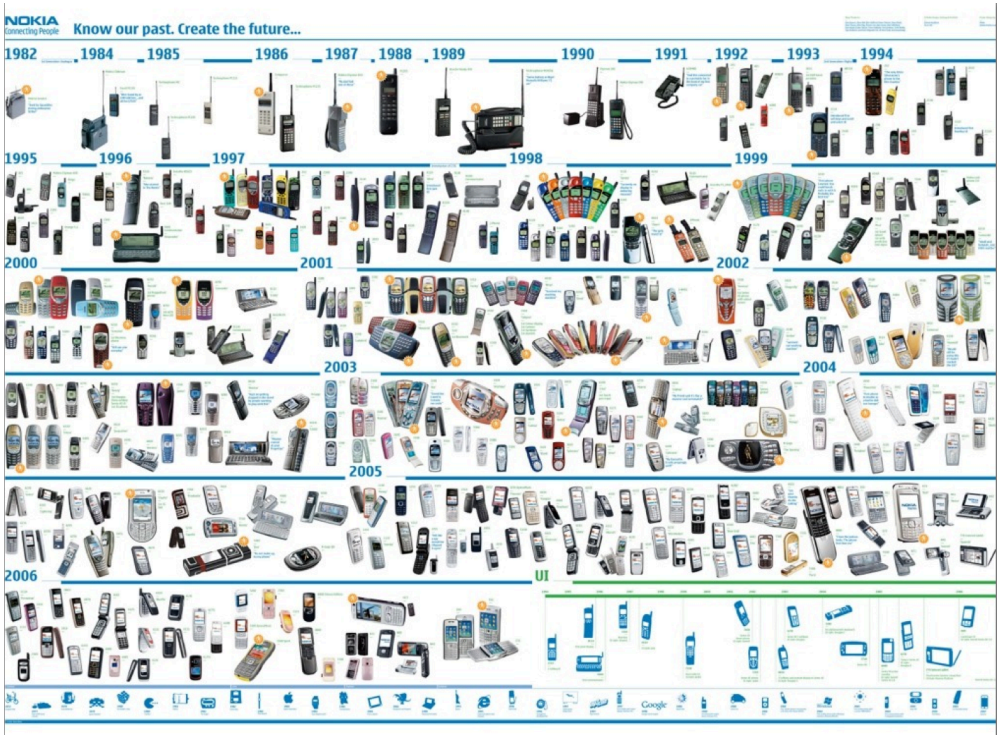
Linux Kernel







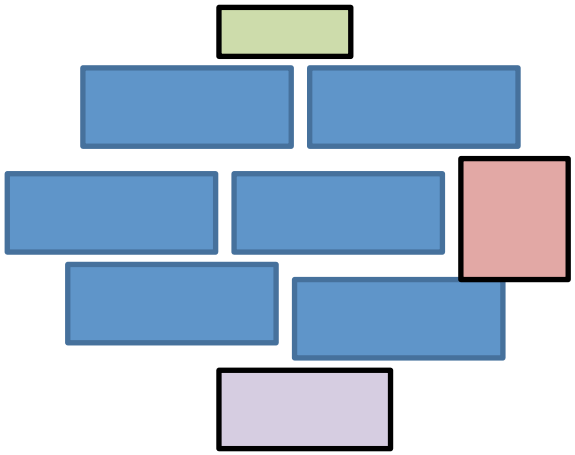
2000 features



Software Product Line Engineering

Factoring out **commonalities**

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



Managing **variabilities**

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

Variability Management

Common features

print – connect with computer…

Variable features

fax – scan – USB port…

Product-specific features

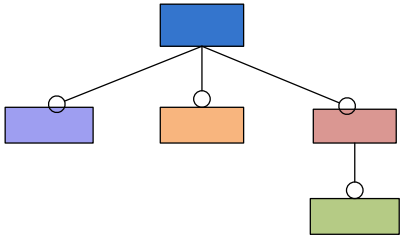
serial port

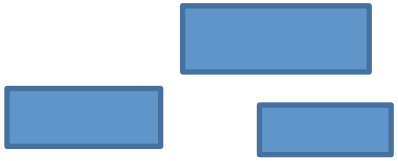
Variability

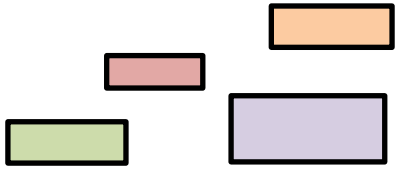
"the ability of a system to be efficiently extended, changed, customized or configured for use in a particular context"

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

Domain engineering (development for reuse)





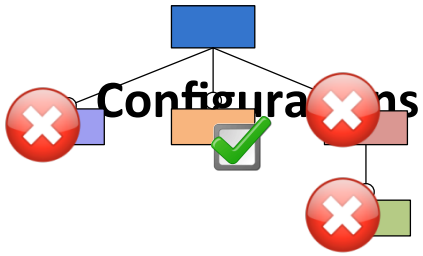


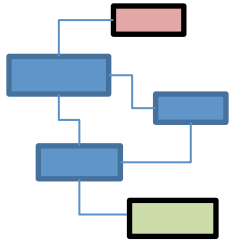
Common assets

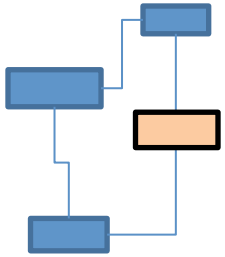
Variants

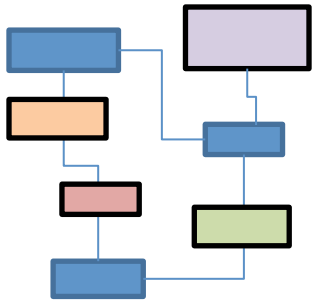
Variability Model

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









product$_1$

product$_2$

product$_n$

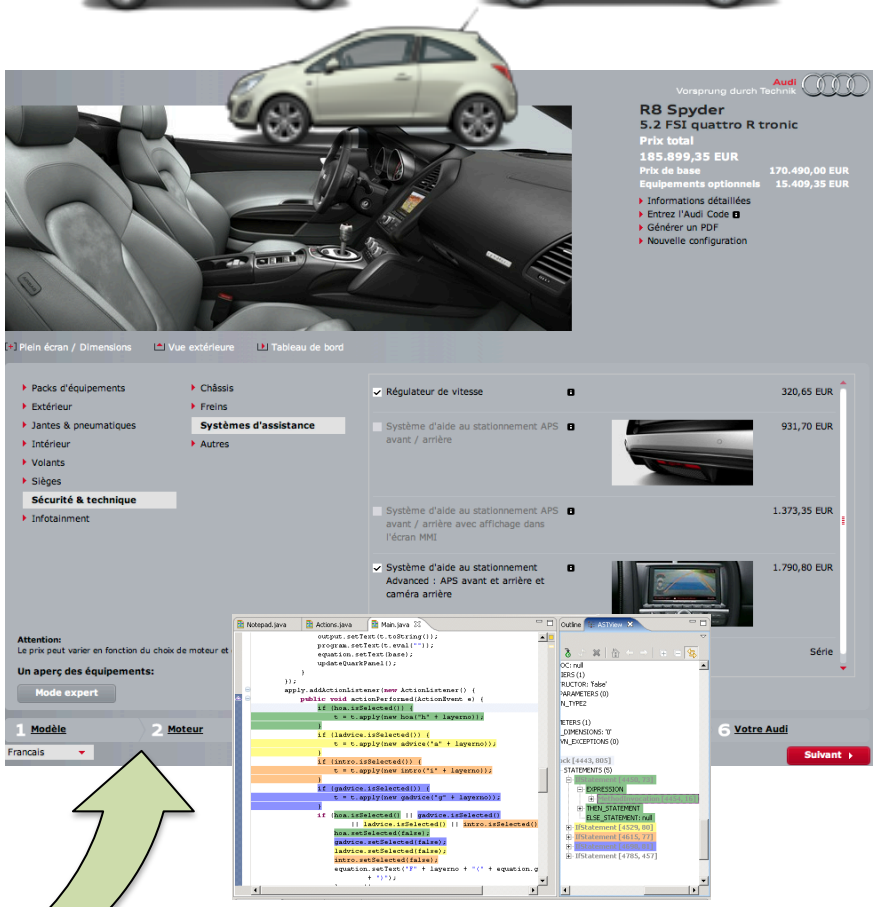
Application engineering (development with reuse)

Modeling Variability:

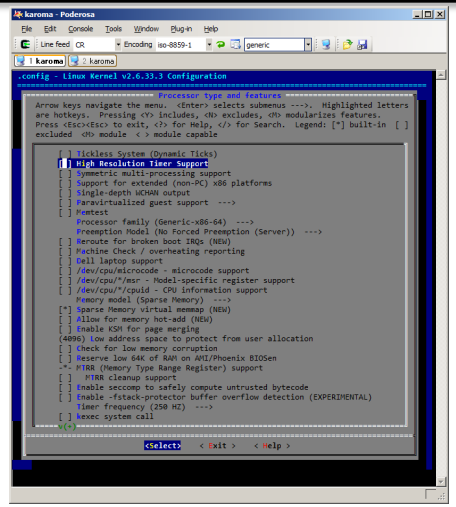
Why Should I Care?

Variability Models

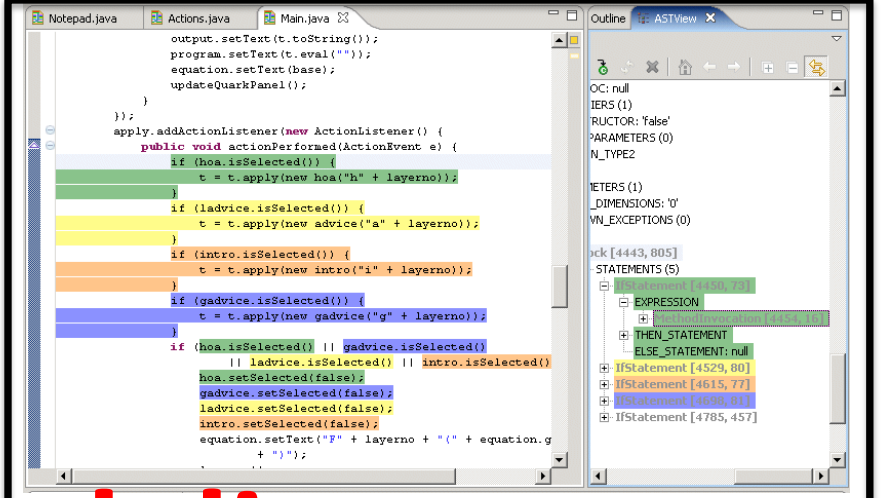
- CarEquipment
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights
- Constraints
  - AutomaticHeadLights ⇒ FrontFogLights



#1 precise specification is needed

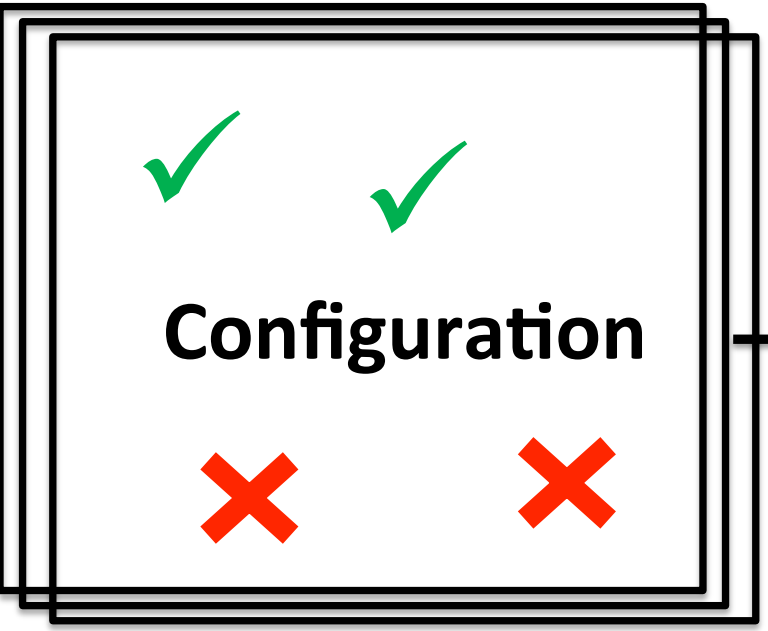


Variability Model

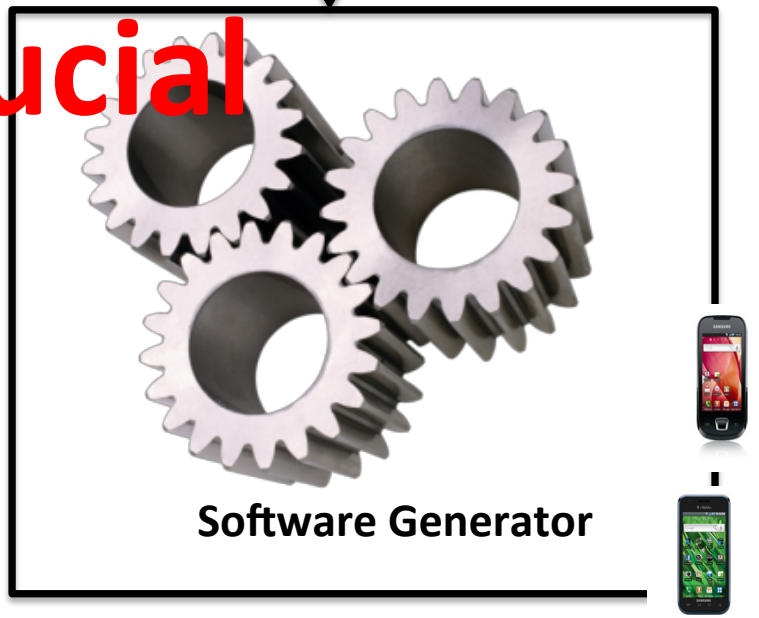


Domain Artefacts (e.g., source code)

Modeling variability is crucial

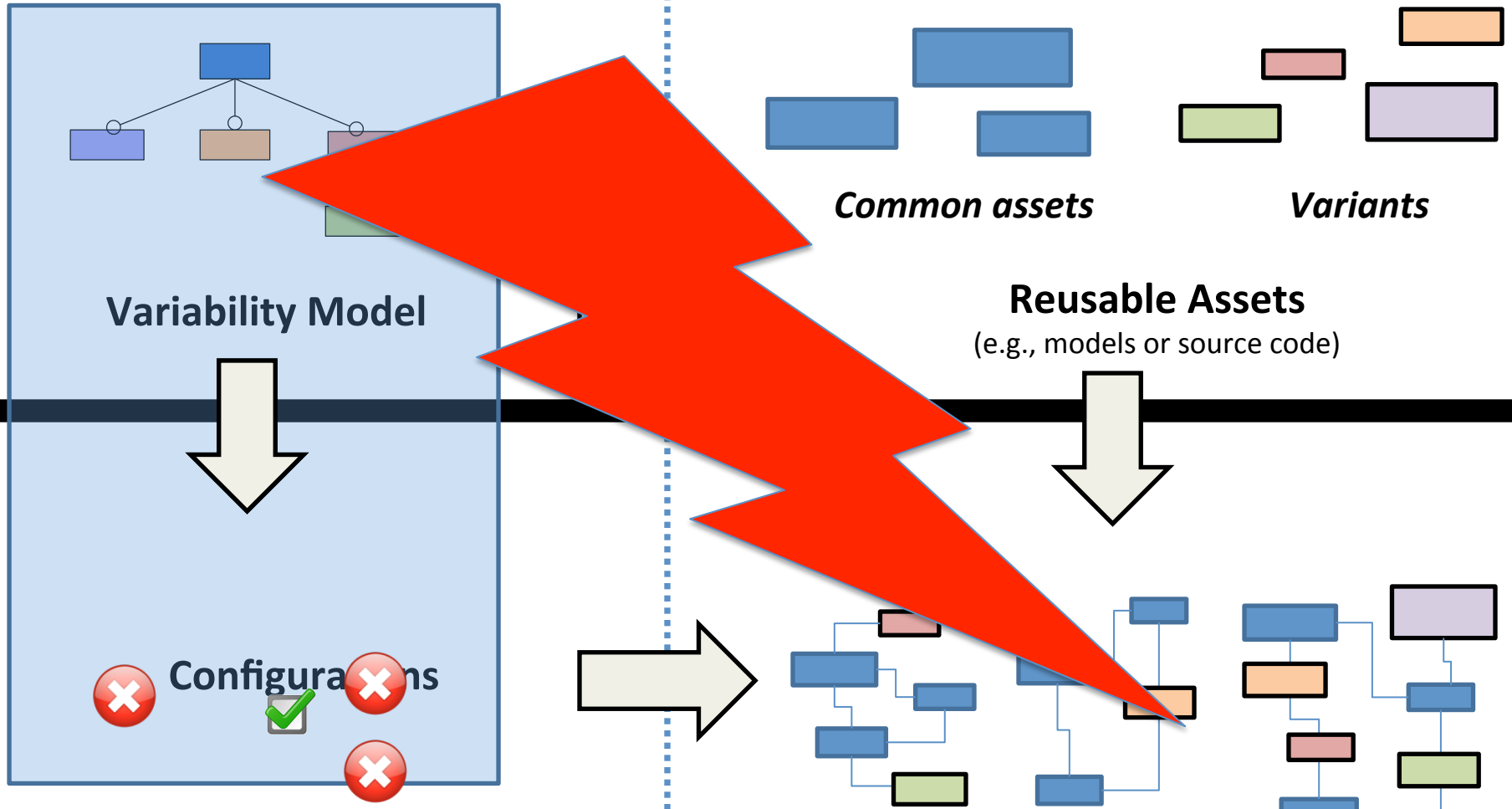


Configuration



Software Generator

Domain engineering (development for reuse)

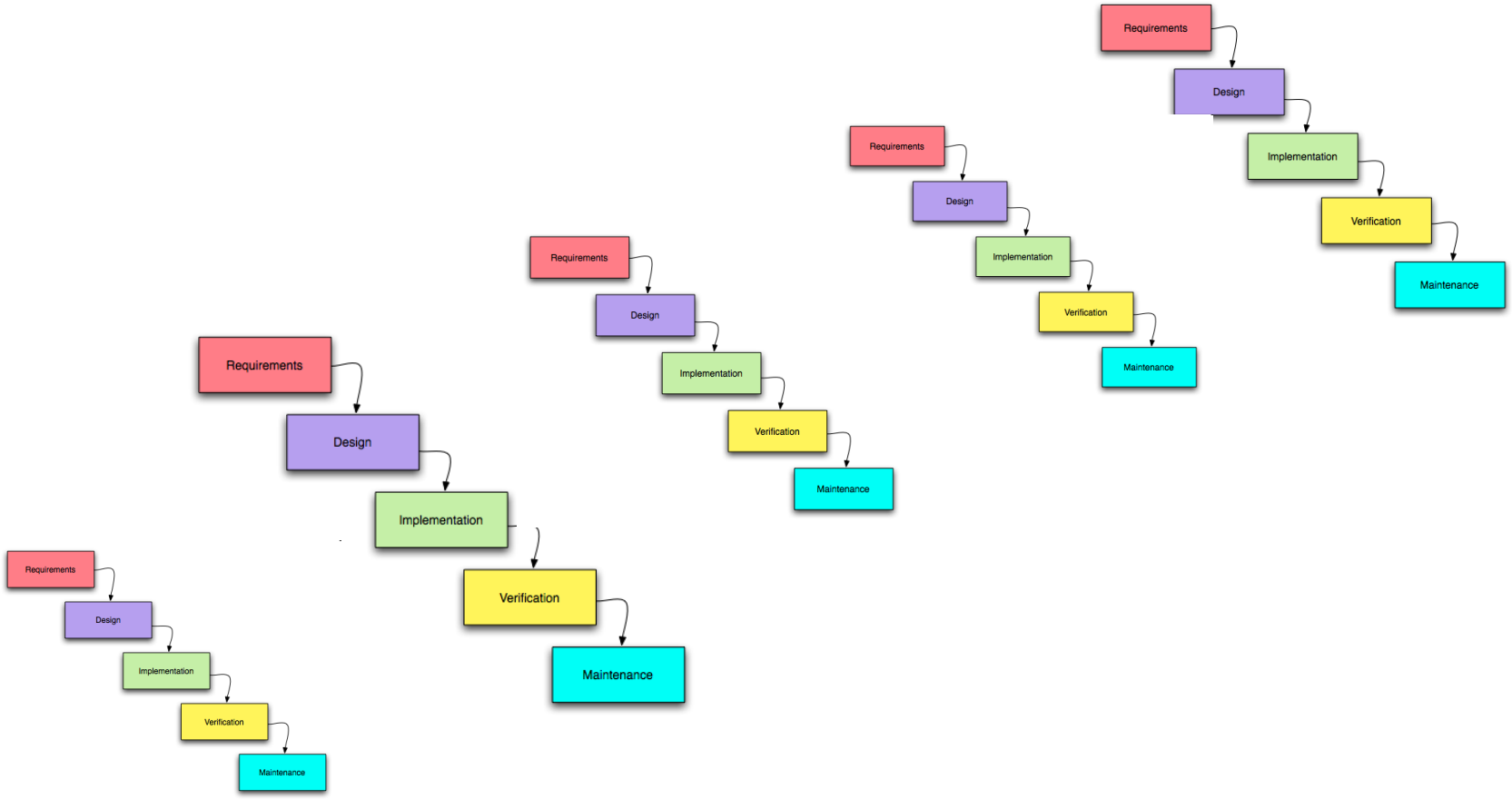


Application engineering (development with reuse)


Illegal variant


Unused flexibility

#2 variability cross cuts all artefacts (from requirements to code)

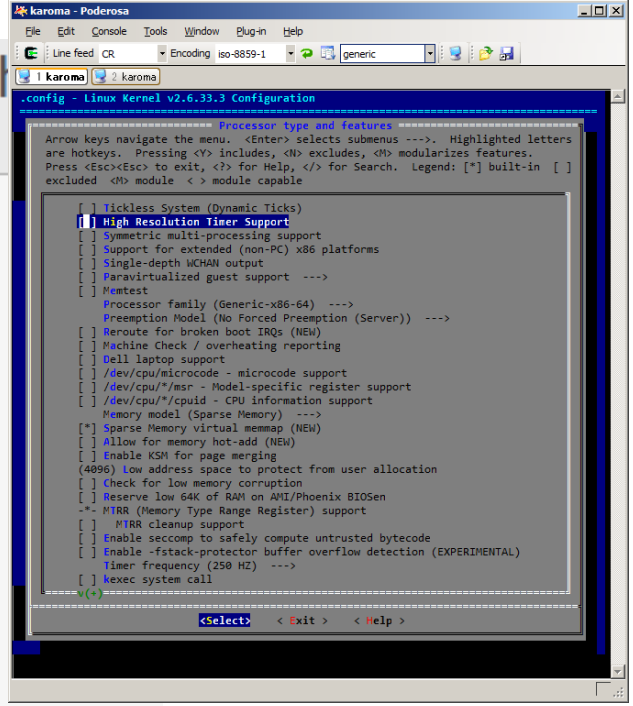

Requirements
Design
Implementation
Verification
Maintenance
Requirements
Design
Implementation
Verification
Maintenance
Requirements
Design
Implementation
Verification
Maintenance
Requirements
Design
Implementation
Verification
Maintenance
Requirements
Design
Implementation
Verification
Maintenance

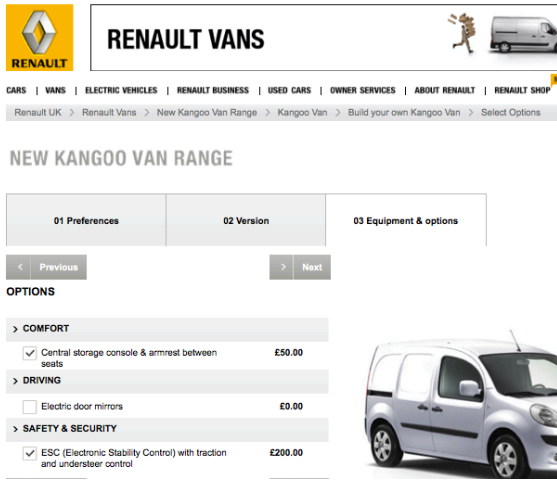
httpd.conf -- win32 Apach

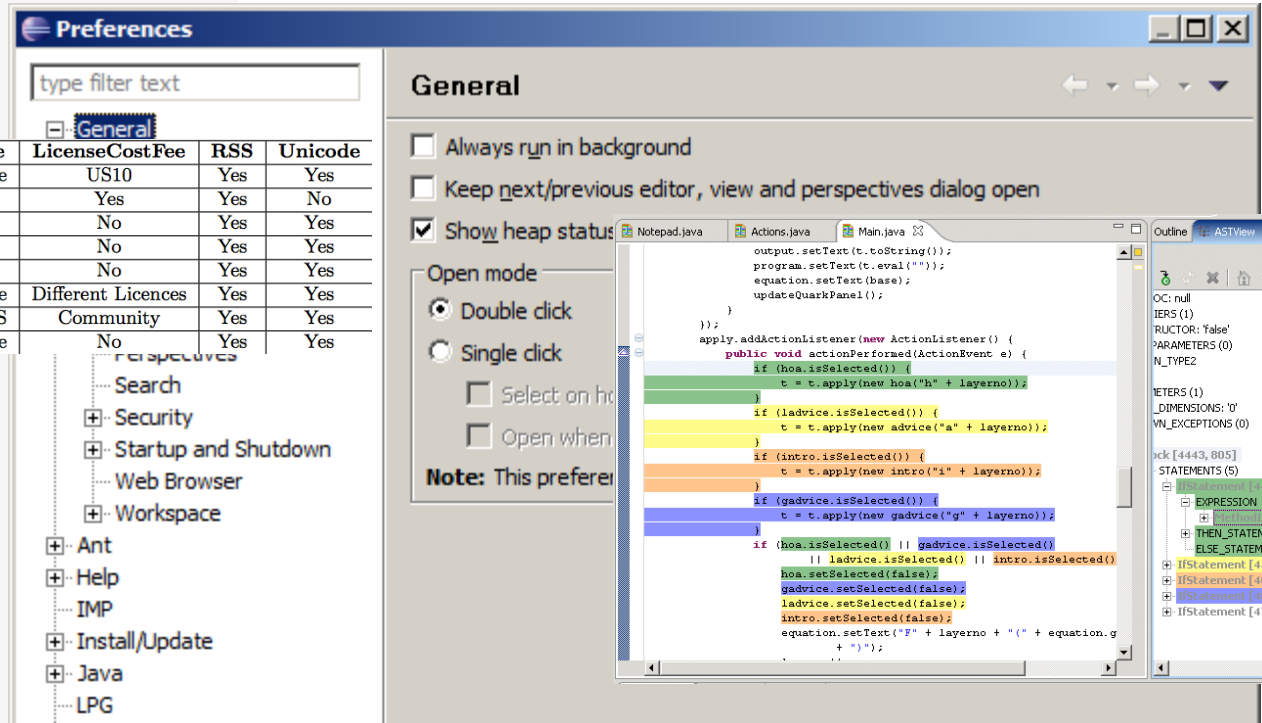
Building a Web Server, for Windows

```
Listen 80
ServerRoot "/www/Apache2"
DocumentRoot "/www/webroot"

ServerName localhost:80
ServerAdmin admin@localhost

ServerSignature On
ServerTokens Full

DefaultType text/plain
AddDefaultCharset ISO-8859-1

UseCanonicalName Off

HostnameLookups Off

Error
LogL

PidF

Time

KeepAlive On
MaxKeepAliveRequests 100
KeepAliveTimeout 15


<IfModule mpm_winnt.c>
  ThreadsPerChild 250
  MaxRequestsPerChild 0
</IfModule>
```

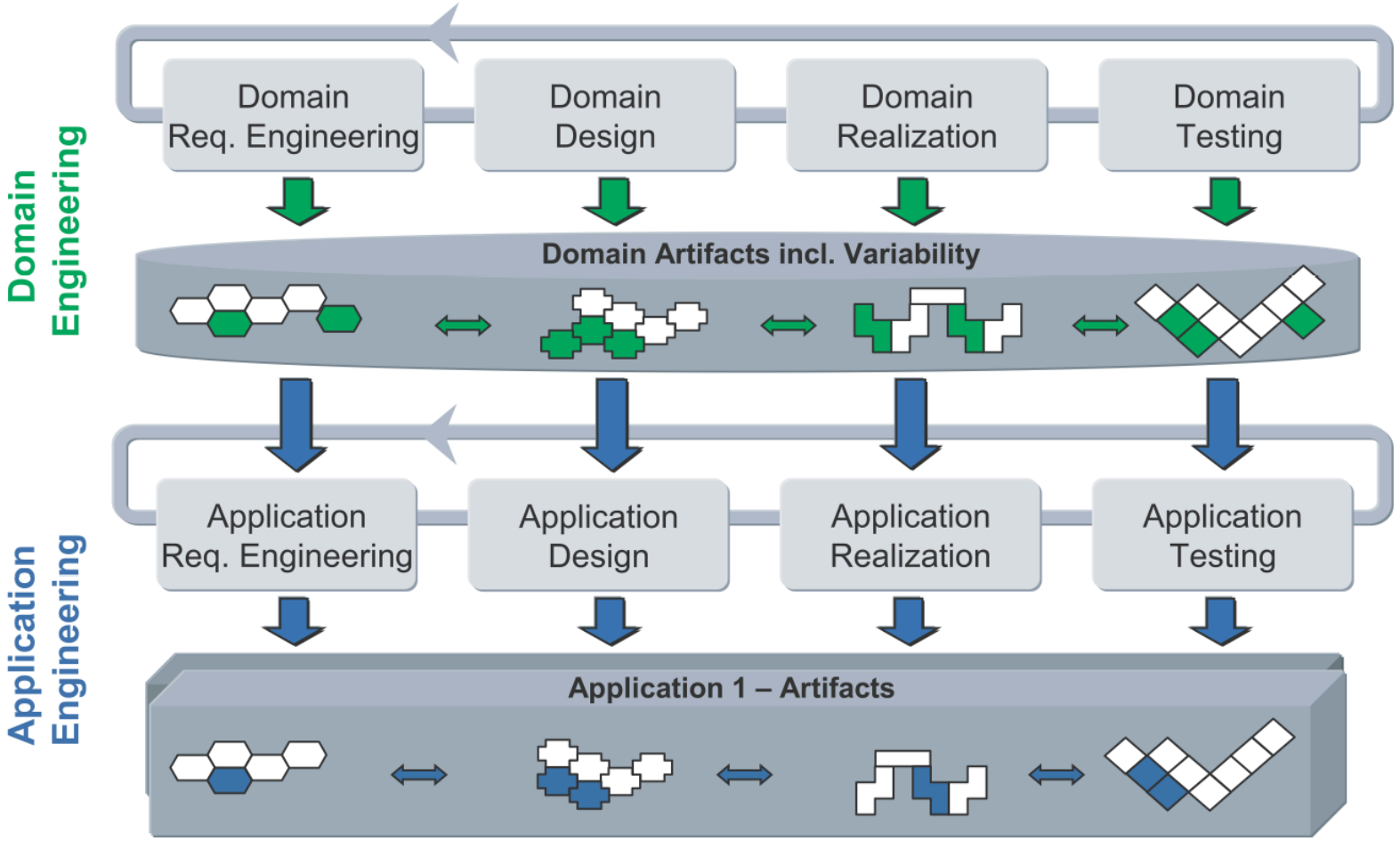
Identifier	License	Language	Storage	LicenseCostFee	RSS	Unicode
Confluence	Commercial	Java	Database	US10	Yes	Yes
PBwiki	Nolimit	No	No	Yes	Yes	No
MoinMoin	GPL	Python	Files	No	Yes	Yes
DokuWiki	GPL2	PHP	Files	No	Yes	Yes
PmWiki	GPL2	PHP	Files	No	Yes	Yes
DrupalWiki	GPL2	PHP	Database	Different Licences	Yes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







Software Product-Line Engineering



#3 automation
(exponential number of products)

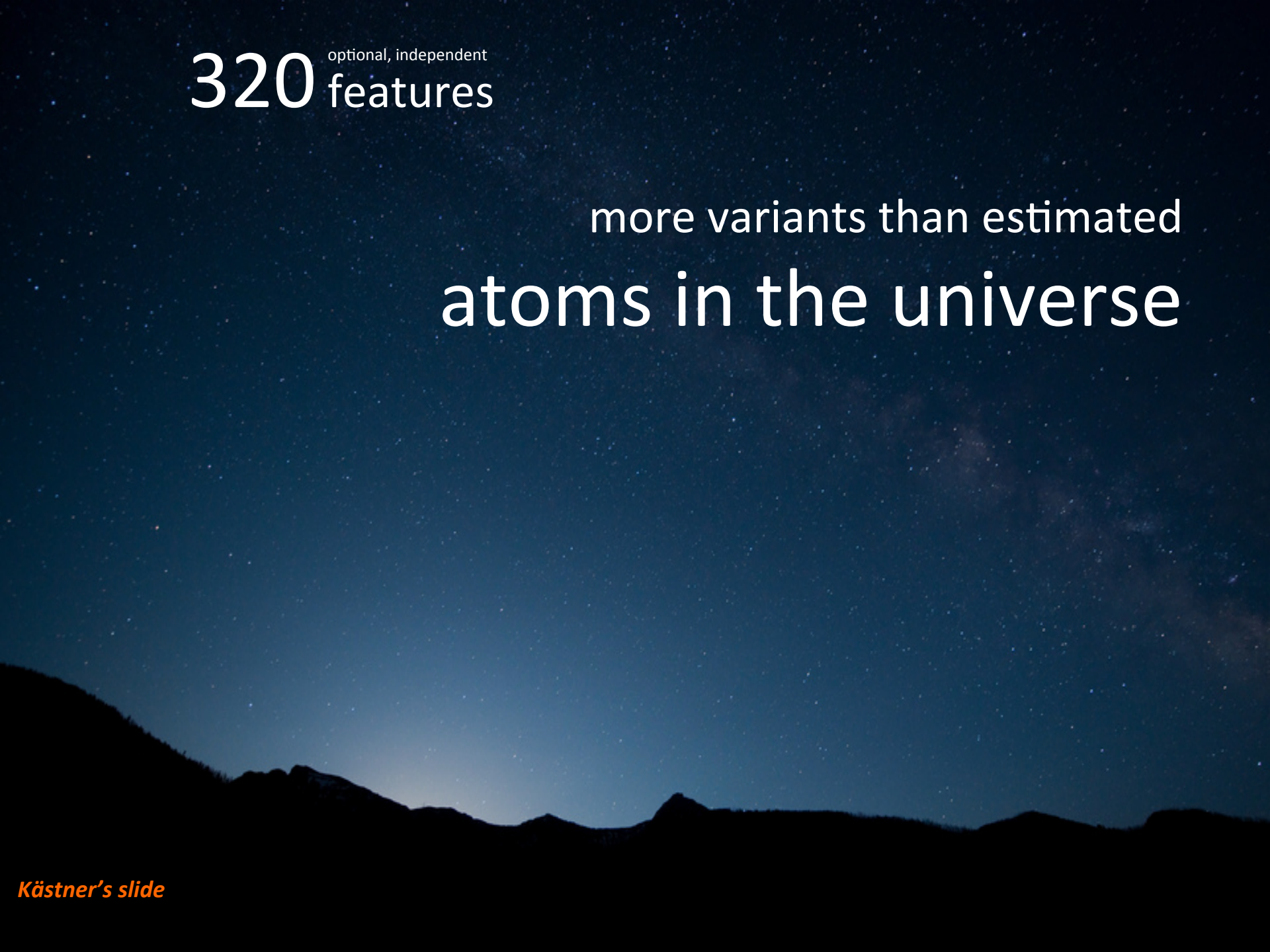
Kästner's slide

33 optional, independent features

a unique variant for every

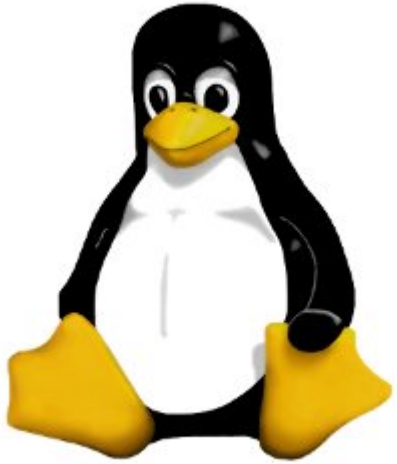
person on this planet

Kästner's slide


320 optional, independent features
more variants than estimated
atoms in the universe
Kästner's slide

2000 features

10000 features



Variability Model

- **Abstraction** of some aspect of a system under study (SUS)
  - Some details are hidden or removed to **simplify** and focus attention
  - **general** concepts can be formulated by abstracting common properties of instances
- In a form that can be **mechanically analyzed**
  - properties of a SUS
  - model transformation
    - Generation of another abstraction (including code)

Variability Models: Why?
(3 arguments, summary)

precise specification

variability is orthogonal / cross cuts different artefacts

automation

Variability models are needed. Now we are convinced…

#1 Formalism? Theory?
modeling principles, semantics, logics

#2 Language?
to elaborate/build variability models

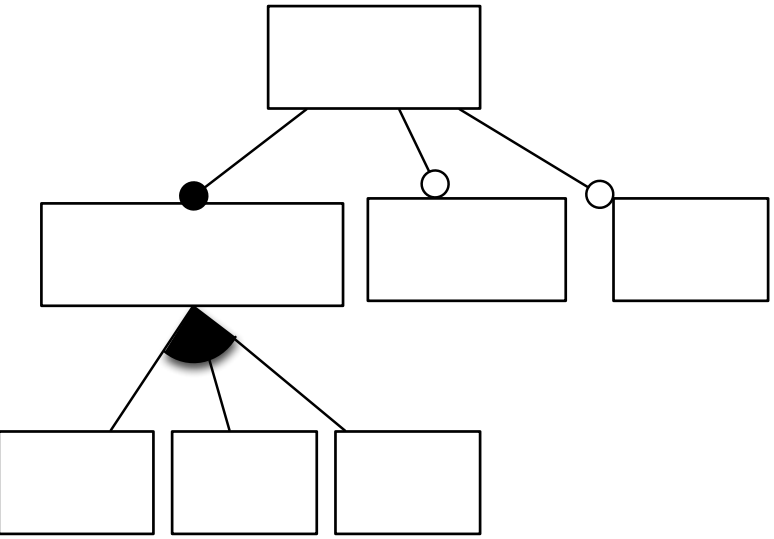
#3 Reasoning techniques?
to analyze properties of an SPL (scalable and automated way)

#4 Applications of variability models?
variability model and so what?

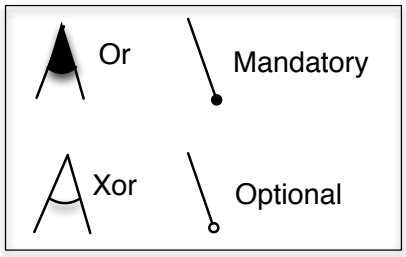
Feature models,

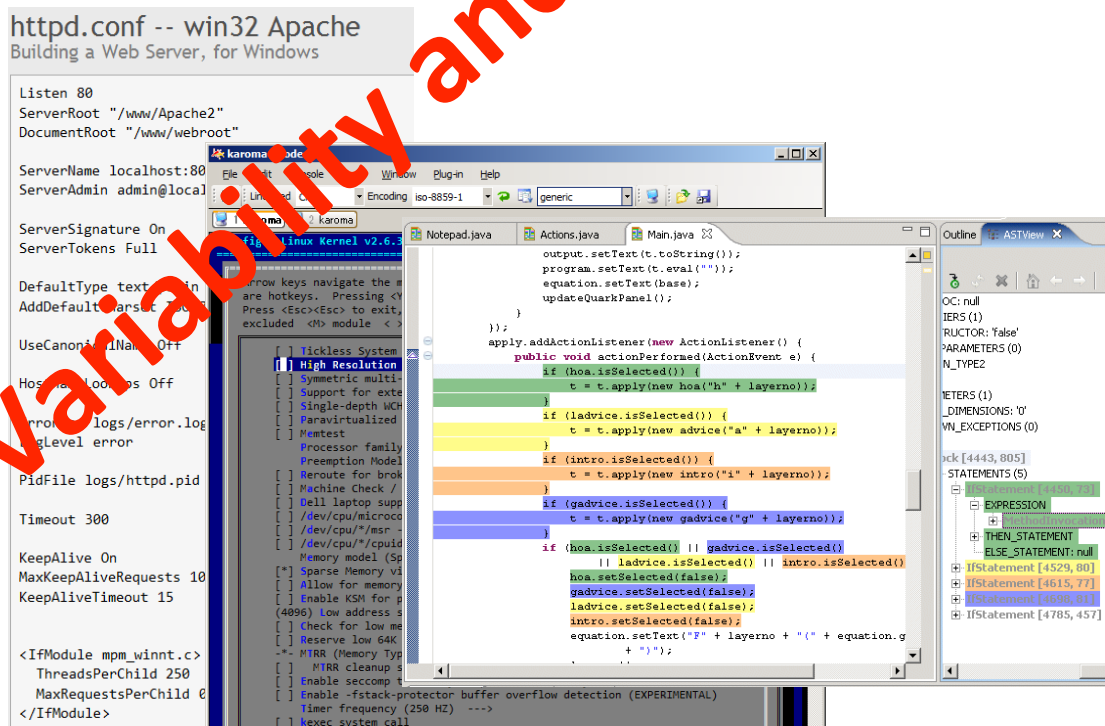
the defacto standard
for modeling variability

Feature Models



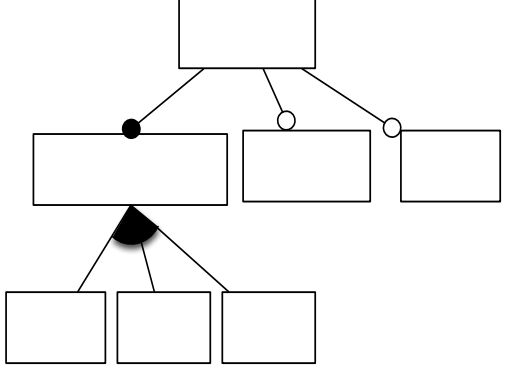
not, and, or, implies





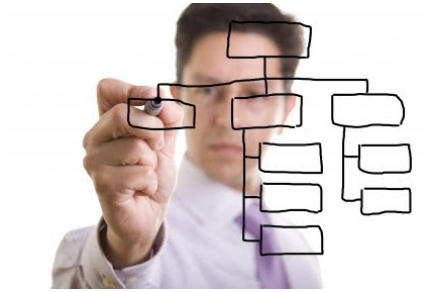
Variability and Configuration

Feature Model

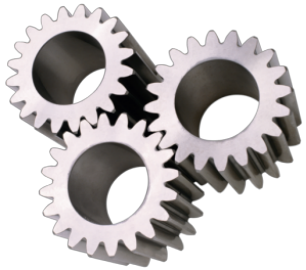


not, and, or, implies

Communicative



Analytic



Generative

Feature Model
de facto standard

- Research
  - 2500+ citations of [Kang et al., 1990] on Google Scholar
  - Central to many generative approaches
    - at requirements or code level
  - Tools & Languages (GUIDSL/FeatureIDE, SPLOT, FaMa, FAMILIAR)

- Industry
  - Tools (Gears, pure::variants),
- Common Variability Language (CVL)

Feature Model: Applications

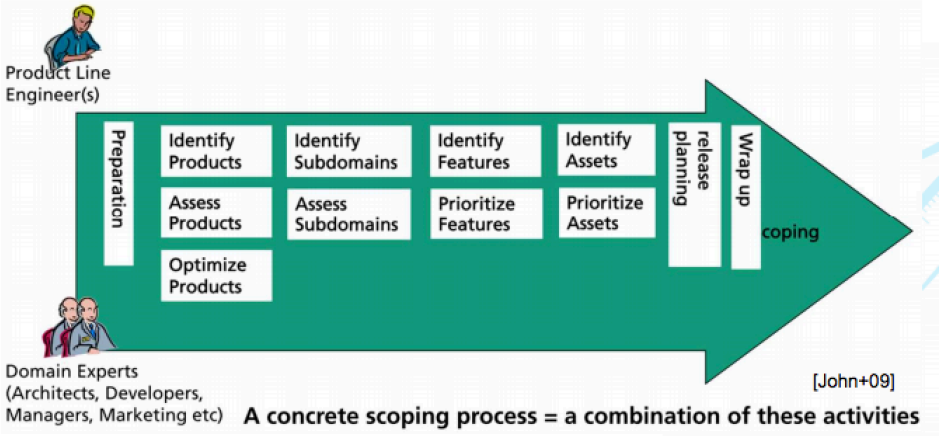
- Domain Analysis
- Configurators
- Automated Derivation of “Product”
- Testing
- Model Checking

Feature Models:

Principles

Feature modeling is…

- … about identifying
  - common features of concepts
  - variable features of concepts
  - and their dependencies
  - and documenting them in a coherent model, called a feature model
- Core activity of domain-engineering methods



Feature and Psychology

- The human mind processes lots of information
- Important function of human information processing: reducing the amount of information
- Describing concepts by features
  - Clustering
  - Abstraction (Classification/Generalization)
  - Vocabulary construction
- Classifying a perceptional unit as an instance of a concept
- Analysis of similarity
  - Features common to entities, activities, events, relationships, structures, etc.
- Analysis of variations
  - Features that are unique to entities, activities, events, relationships, structures, etc.

Modeling variability (and commonality)

- **Features** are a convenient way to think about commonalities and differences between family members
  - e.g. "MP3 player" – some phones have it, some don't
- Features are « usually » **coarse-grained** abstractions
  - Abstract from detailed requirements
    - e.g. "when in shuffle mode, the player selects the next track using a random function"
  - Abstract from design/implementation details
    - e.g. "MP3_Player is a concrete subclass of Media_Player »
- Depends on what you are modeling and for which purpose
  - Elicitating requirements with customers (informal document)
  - VS builiding a representation of your code (based on CPP directives)

What's a **feature**? (survey excerpt)

- *[Kang et al.]* "a prominent or distinctive **user-visible aspect, quality** or **characteristic** of a software system or systems"
- *[Kang et al.]* "distinctively identifiable **functional abstractions** that must be implemented, tested, delivered, and maintained"
- *[Eisenecker and Czarnecki]*. "**anything users** or client programs might want to **control about a concept**"
- *[Bosch et al.]* "A **logical unit of behaviour** specified by a set of **functional** and **non-functional requirements**."
- *[Chen et al.]* "a **product characteristic** from user or customer views, which essentially consists of a cohesive **set of** individual **requirements**"
- *[Batory]* "an elaboration or augmentation of an entity(s) that introduces a **new service**, **capability** or relationship"

Requirements to Code

Feature Models:

Syntax

Audi
Vorsprung durch Technik

R8 Spyder

5.2 FSI quattro R tronic

Prix total

171.216,00 EUR

Prix de base	**170.490,00 EUR**
Equipements optionnels	**726,00 EUR**

- Informations détaillées
- Entrez l'Audi Code
- Générer un PDF
- Nouvelle configuration

[+] Plein écran / Dimensions | Fermer la capote | Habitacle | Tableau de bord

Packs

Aucun pack n'est proposé pour ce modèle.

Couleurs

Blanc Ibis
Noir
Prix: 0,00 EUR

Couleurs métallisées à partir de 0,00 EUR

Couleurs à effet perlé à partir de 0,00 EUR

Couleurs personnalisées Audi exclusive

Audi exclusive

Couleur capote

Noir

Jantes

4 Jantes alu 5 BRANCHES ROTOR finition titane 8,5 x 19 à l'avant, 11 x 19 à l'arrière. Pneus 235/35 R19 à l'avant et 305 /30 R19 à l'arrière
Prix: 726,00 EUR

19" à partir de 0,00 EUR

1 Modèle | 2 Moteur | 3 Extérieur | 4 Intérieur | 5 Option | 6 Votre Audi

Audi
Vorsprung durch Technik

R8 Spyder
5.2 FSI quattro R tronic

Prix total

185.899,35 EUR

Prix de base	**170.490,00 EUR**
Equipements optionnels	**15.409,35 EUR**

- Informations détaillées
- Entrez l'Audi Code
- Générer un PDF
- Nouvelle configuration

[+] Plein écran / Dimensions Vue extérieure Tableau de bord

- Packs d'équipements
- Extérieur
- Jantes & pneumatiques
- Intérieur
- Volants
- Sièges
- **Sécurité & technique**
- Infotainment

- Châssis
- Freins
- **Systèmes d'assistance**
- Autres

excludes

☑ Régulateur de vitesse		320,65 EUR
☐ Système d'aide au stationnement APS avant / arrière		931,70 EUR
☐ Système d'aide au stationnement APS avant / arrière avec affichage dans l'écran MMI		1.373,35 EUR
☑ Système d'aide au stationnement Advanced : APS avant et arrière et caméra arrière		1.790,80 EUR
● Audi hill assist : assistance au démarrage en côte		Série

Attention:
Le prix peut varier en fonction du choix de moteur et des équipements.

Un aperç des équipements:

Mode expert

Réinitialiser la sélection

1 Modèle 2 Moteur 3 Extérieur 4 Intérieur 5 Option 6 Votre Audi

Audi
Vorsprung durch Technik

A5 Sportback
3.0 TDI quattro S tronic

Prix total
54.460,15 EUR

Prix de base	50.570,00 EUR
Equipements optionnels	3.890,15 EUR

- Informations détaillées
- Entrez l'Audi Code
- Nouvelle configuration

[+] Plein écran / Dimensions

Vérification de votre sélection

Cet équipement nécessite un équipement complémentaire:

☑ GPS Plus avec disque dur		2.934,25 EUR

Voici les équipements complémentaires possibles:

○ Ordinateur de bord en couleur avec programme efficiency **Remarque:** uniquement sur les modèles avec système Start-Stop et uniquement disponible en combinaison avec l'autoradio Concert, l'autoradio Symphony ou un système de navigation		181,50 EUR
○ Pack Intenso Plus		3.100,00 EUR

- Extérieur
- Jantes & pneumatiques
- Intérieur
- Volants
- Sièges
- Sécurité & technique
- **Infotainment**

2.934,25 EUR

● Sans appareil de navigation — Série

Attention:
Le prix peut varier en fonction du choix de moteur et des équipements.

Un aperç des équipements:

Mode expert

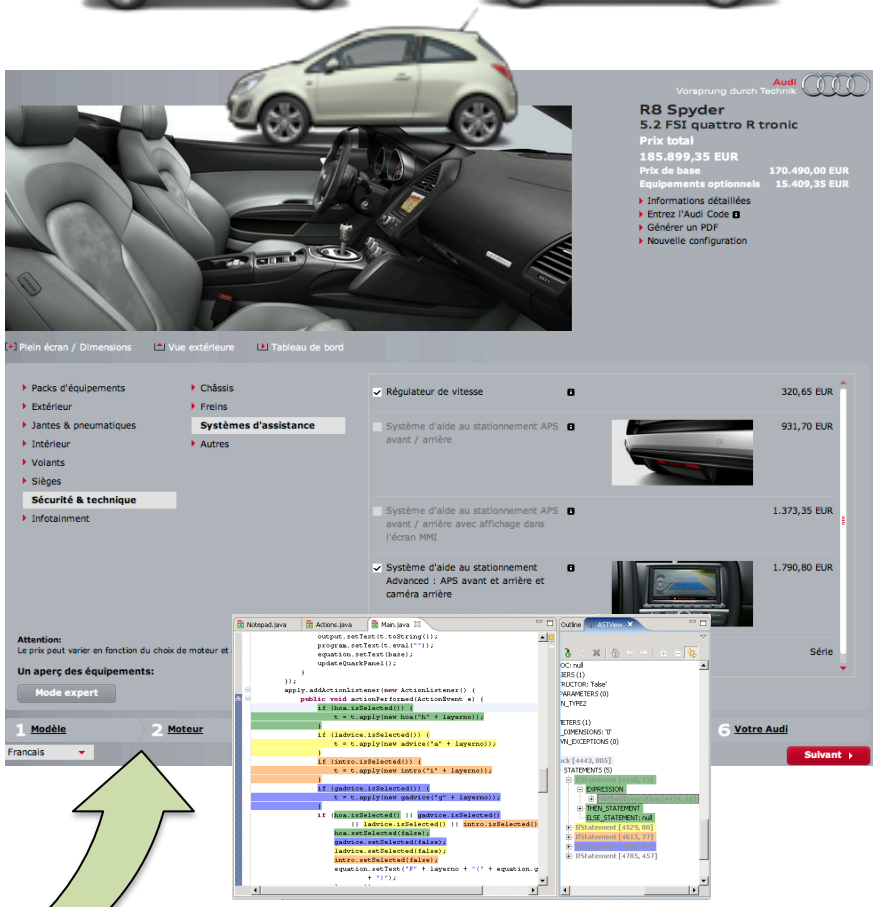
Réinitialiser la sélection

1 Modèle | 2 Moteur | 3 Extérieur | 4 Intérieur | 5 Option | 6 Votre Audi

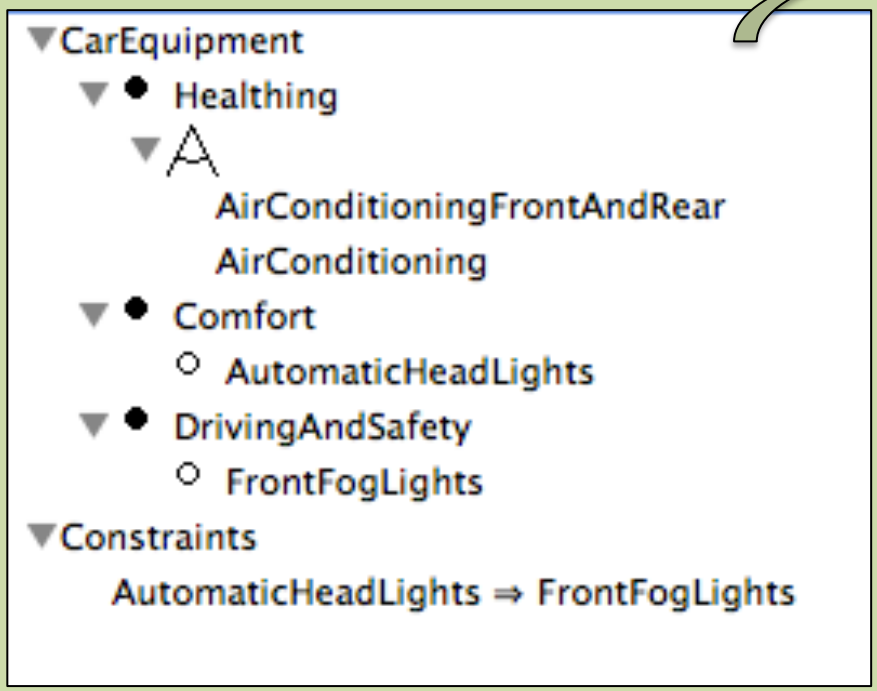
Francais

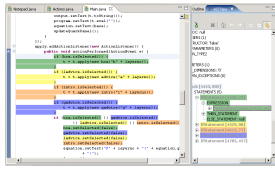
Suivant

Feature Models



Feature Models (Background)

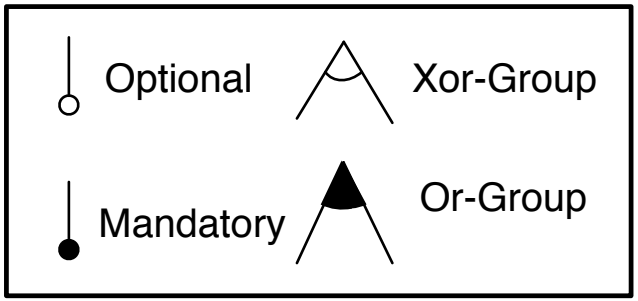


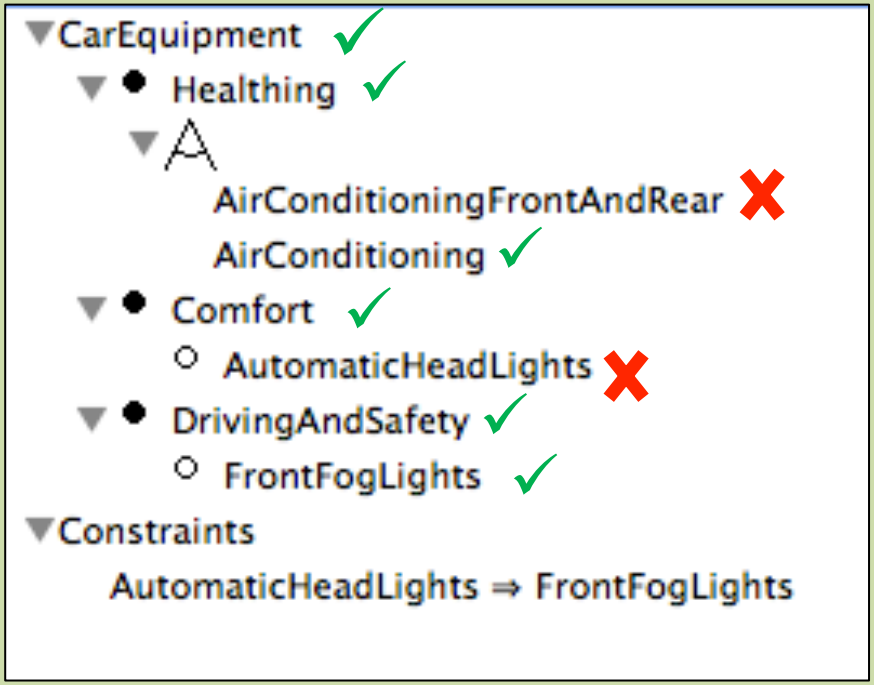


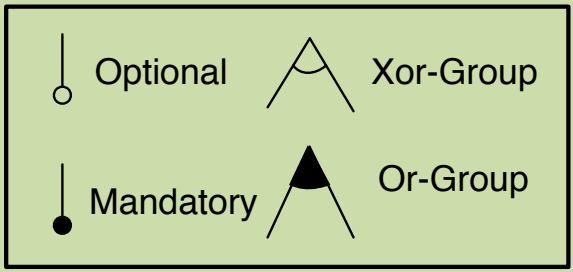
Hierarchy: rooted tree

Variability:

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



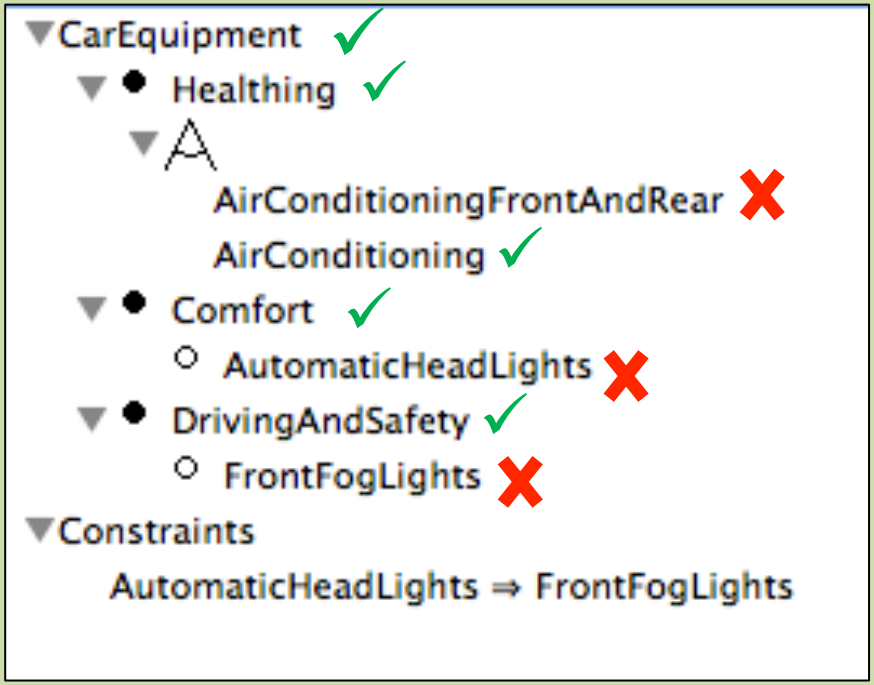


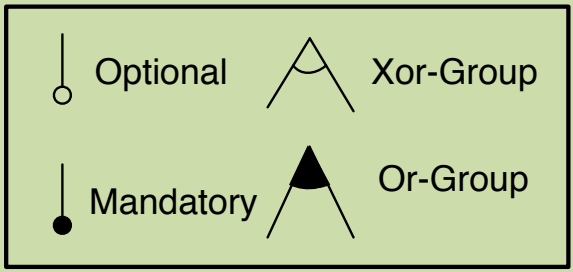


Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

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

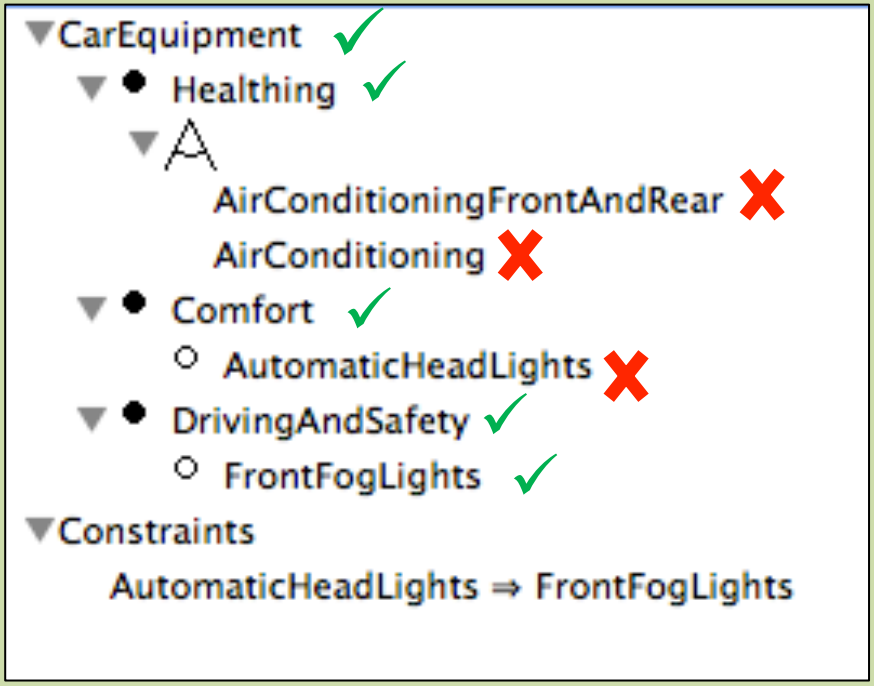


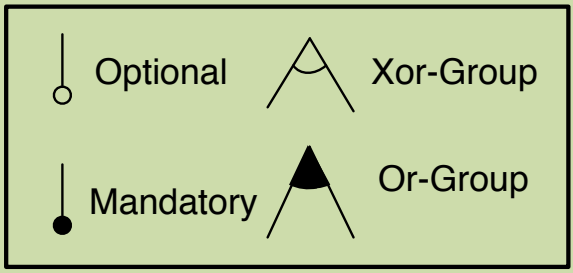


Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning}

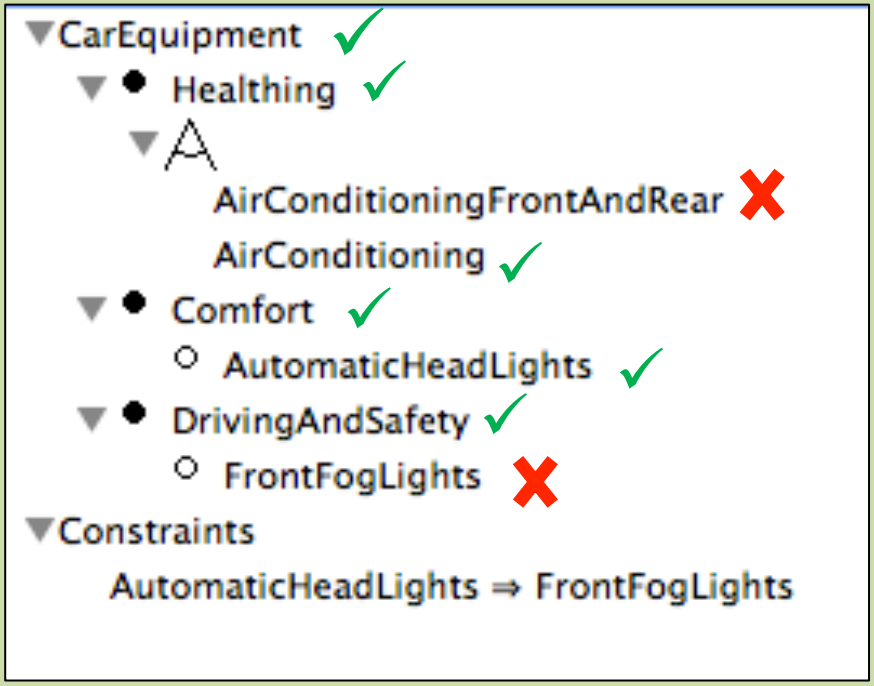


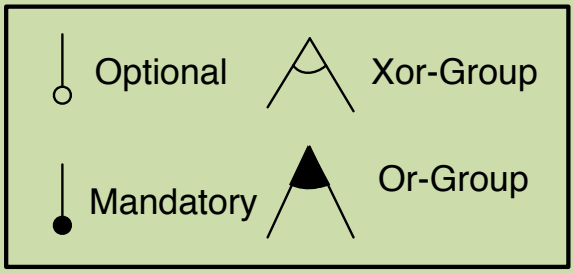


Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

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

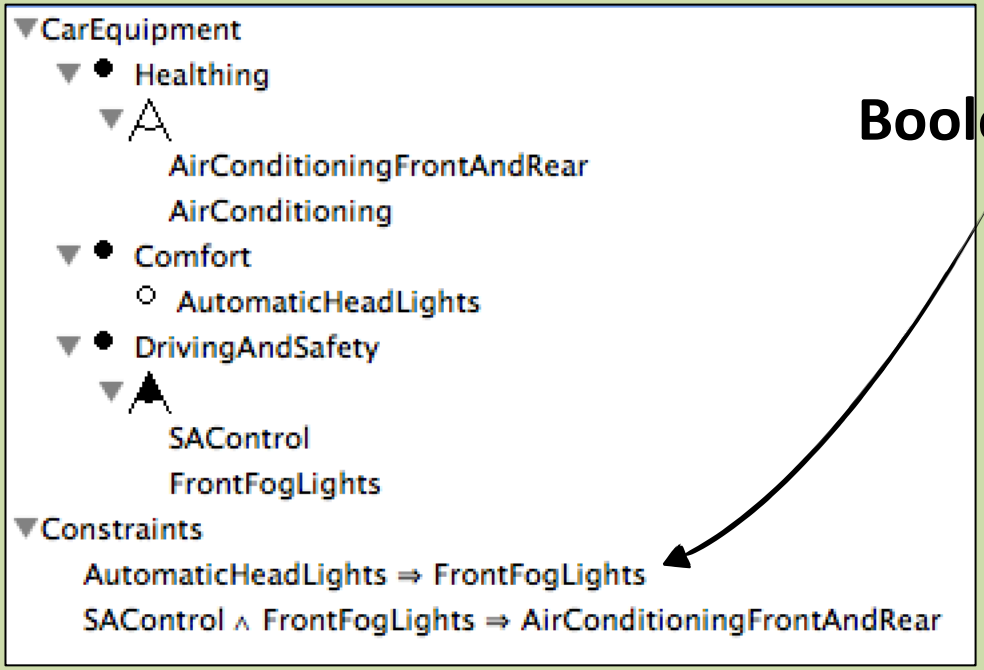




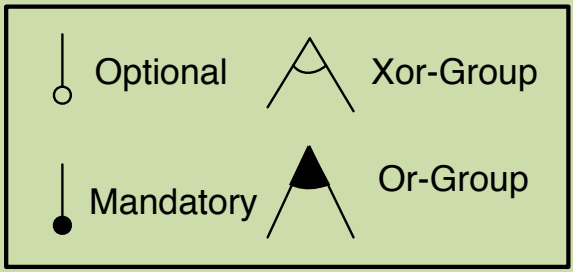
Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

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



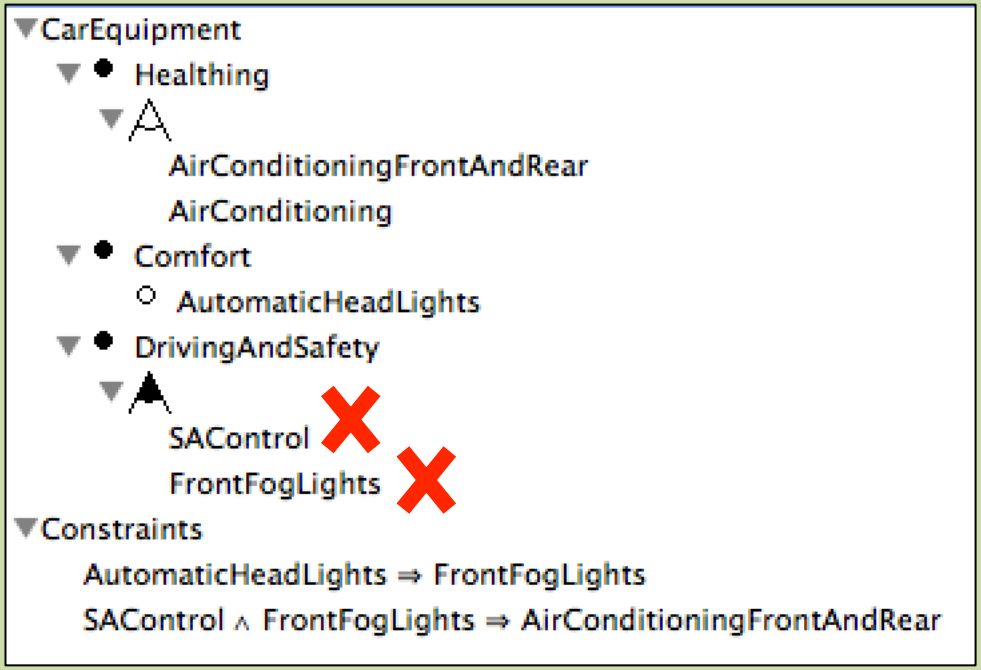
Boolean logic: ^, v, not, implies

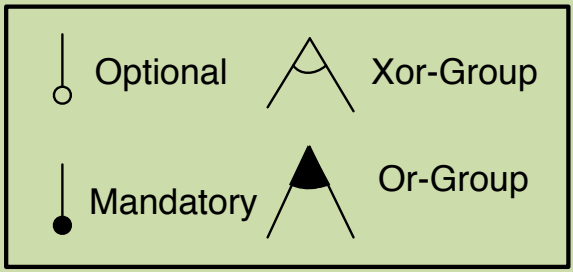


Hierarchy + Variability

=

set of valid configurations



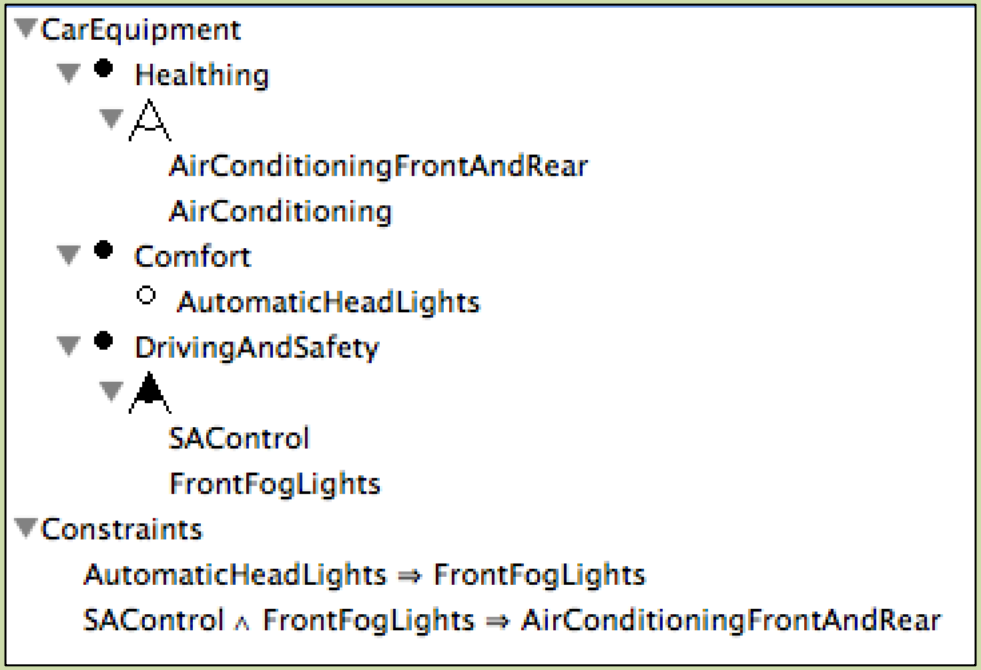


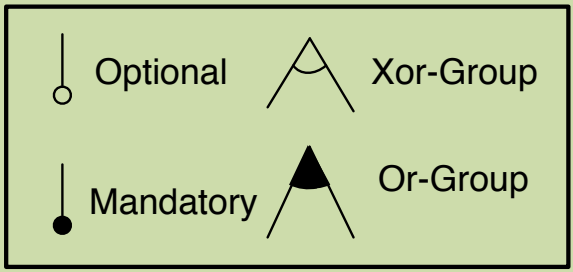
Hierarchy + Variability

=

set of valid configurations

Or-group: at least one!





Hierarchy + Variability
=
set of valid configurations

{CarEquipment, Comfort, DrivingAndSafety, Healthing}

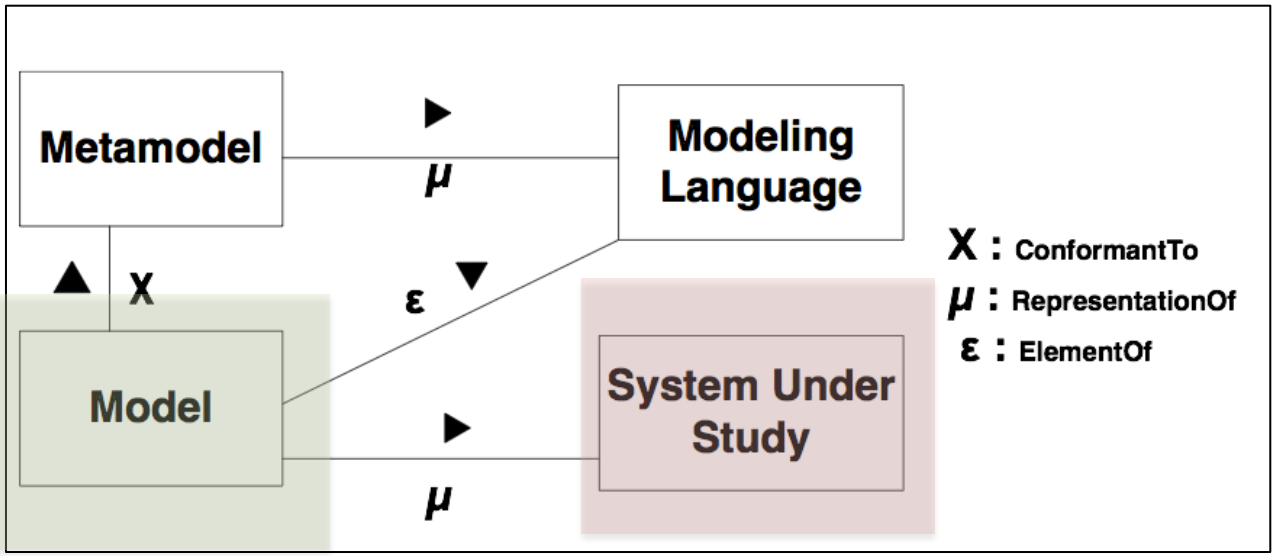
X

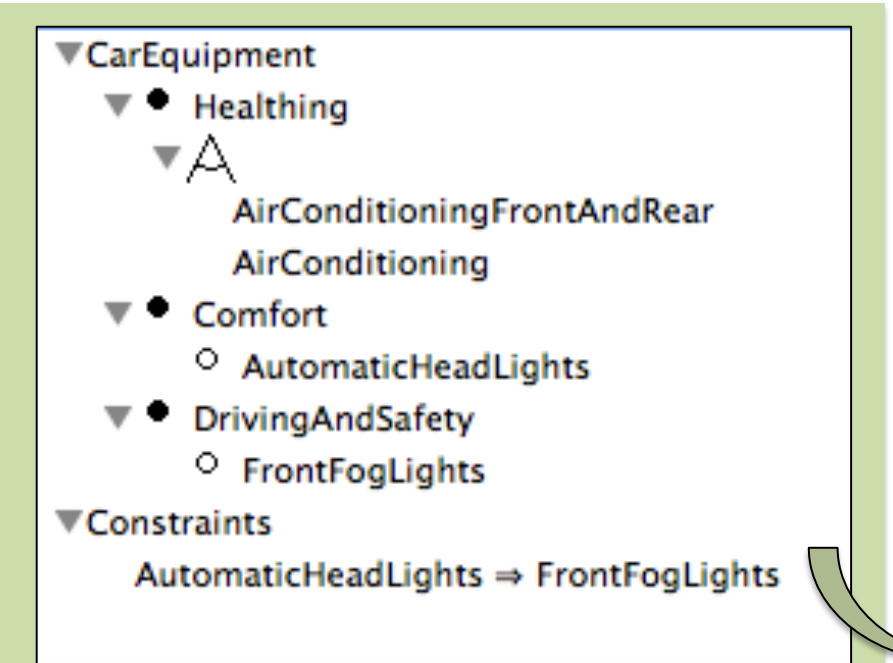
{AirConditioningFrontAndRear, FrontFogLights, SAControl}
{AirConditioningFrontAndRear, SAControl}
{AutomaticHeadLights, AirConditioning, FrontFogLights}
{AirConditioningFrontAndRear, SAControl, AutomaticHeadLights, FrontFogLights}
{FrontFogLights, AirConditioning}
{AutomaticHeadLights, AirConditioningFrontAndRear,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

Languages:

How to specify feature models?

Feature Models





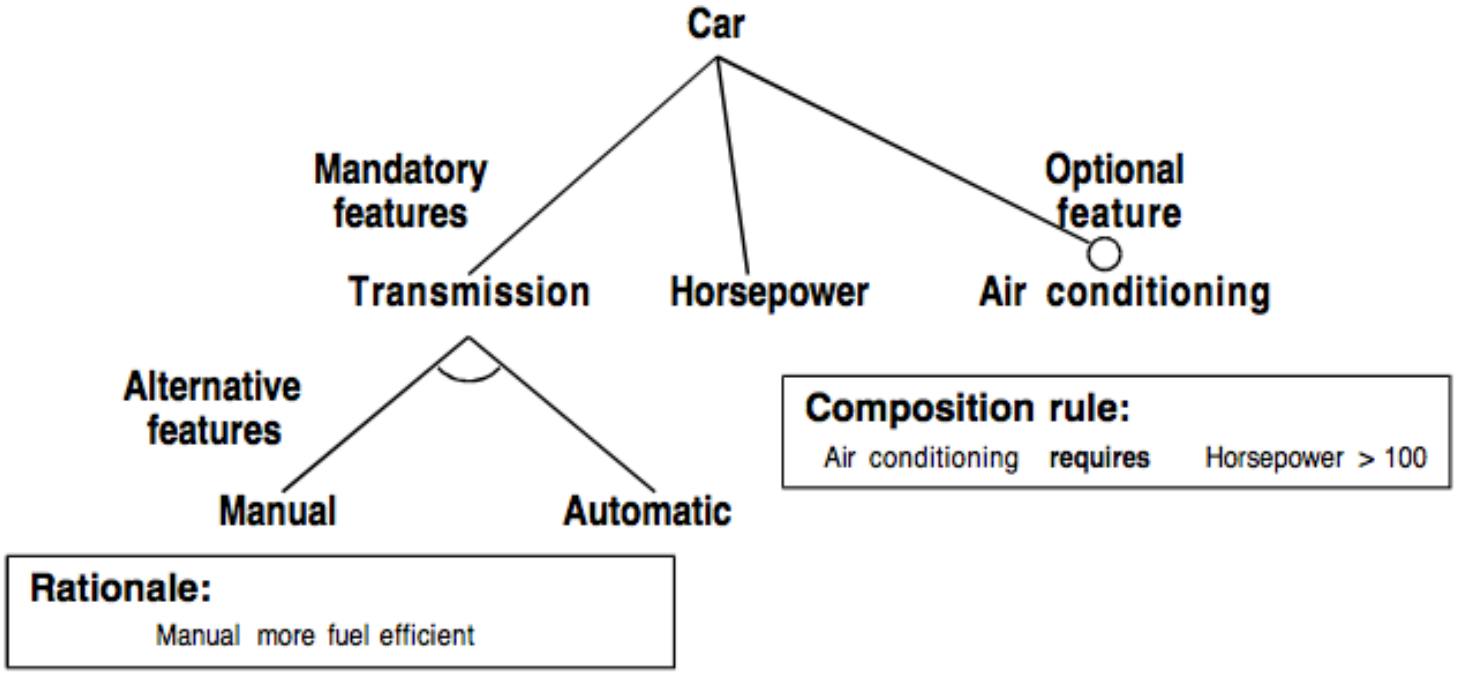
History of Feature Models

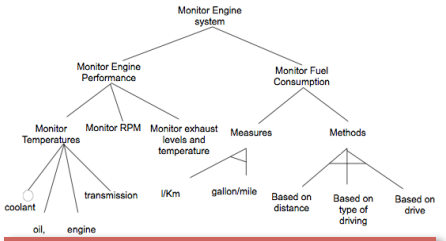
~ often called Feature Diagrams
(suggest a visual representation)

graphical languages

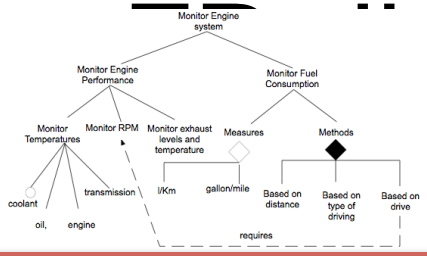
Feature Models

Kang et al. (1990)

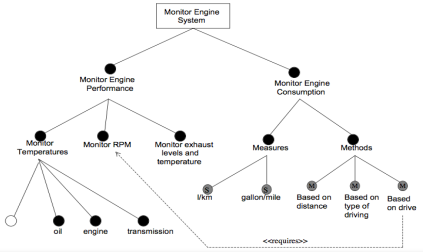




FODA (OFT)
[Kang et al., 1990]

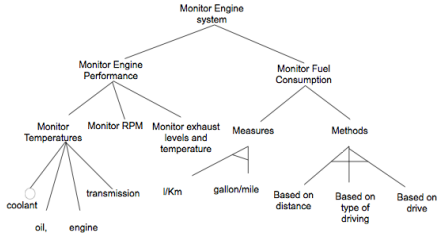


FeatuRSEB (RFD)
[Griss et al., 1998]

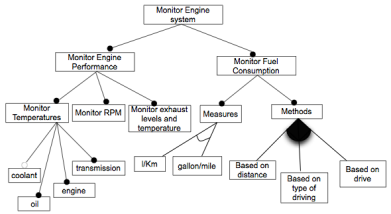


PLUSS (PFT)
[Eriksson et al., 2005]

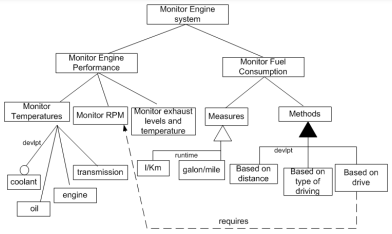
FORM (OFD)
[Kang et al., 1998]



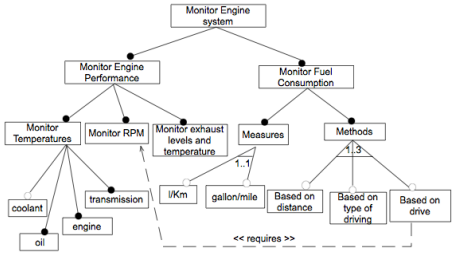
Gen. Prog. (GPFT)
[Czarnecki et al., 2000]

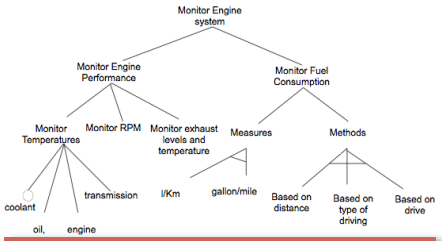


VBFD
[van Gurp et al., 2001]

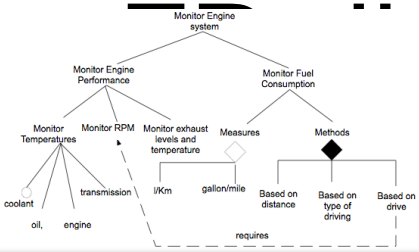


EFD
[Riebisch et al., 2002]

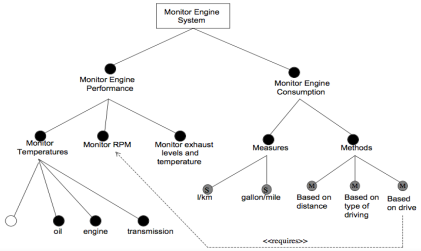




FODA (OFT)
[Kang et al., 1990]



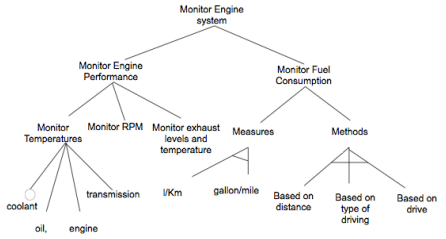
FeatuRSEB (RFD)
[Griss et al., 1998]



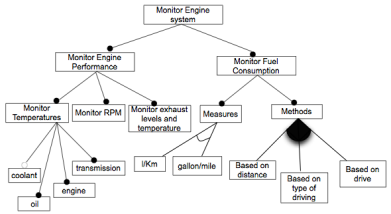
PLUSS (PFT)
[Eriksson et al., 2005]

Aesthetic differences

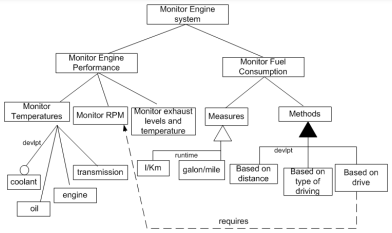
FORM (OFD)
[Kang et al., 1998]



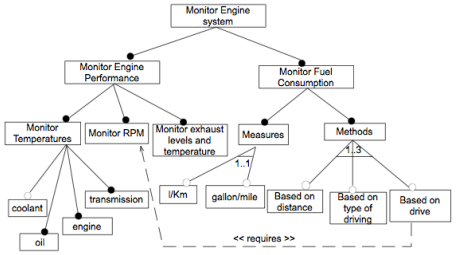
Gen. Prog. (GPFT)
[Czarnecki et al., 2000]



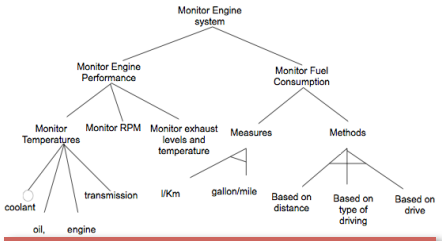
VBFD
[van Gurp et al., 2001]



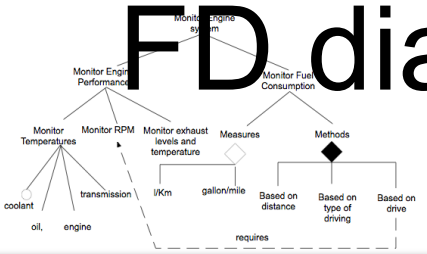
EFD
[Riebisch et al., 2002]



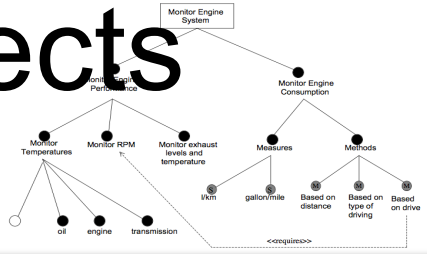
FD dialects



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



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

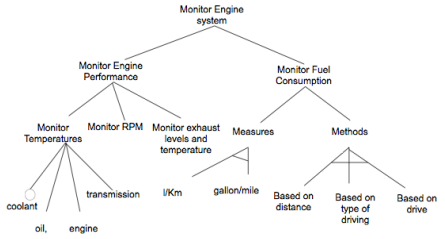


PLUSS (PFT)
[Eriksson et al., 2005]

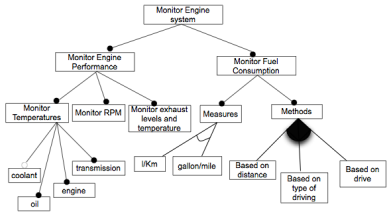
Aesthetic differences

Stronger claims

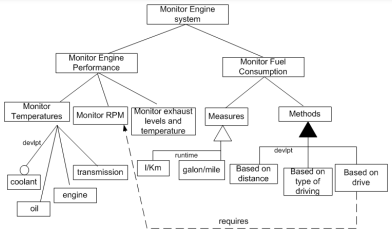
FORM (OFD)
[Kang et al., 1998]



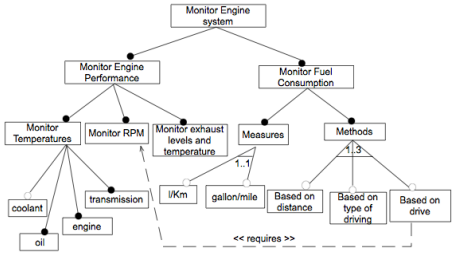
Gen. Prog. (GPFT)
[Czarnecki et al., 2000]



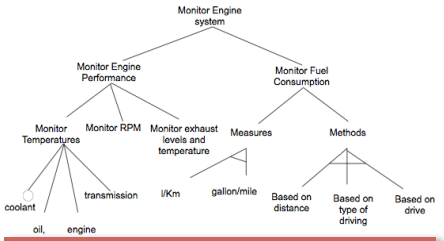
VBFD
[van Gurp et al., 2001]



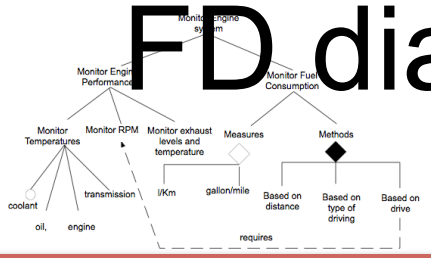
EFD
[Riebisch et al., 2002]



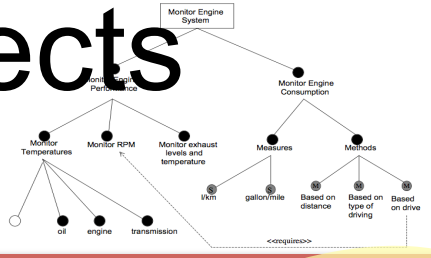
FD dialects



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



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



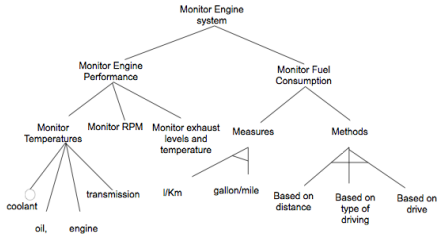
PLUSS (PFT)
[Eriksson et al., 2004]

Aesthetic differences

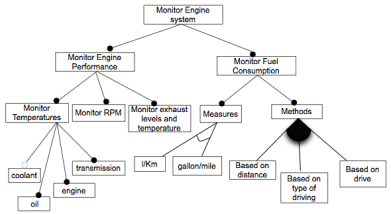
Stronger claims

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

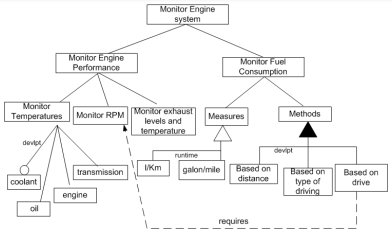
FORM (OFD)
[Kang et al., 1998]



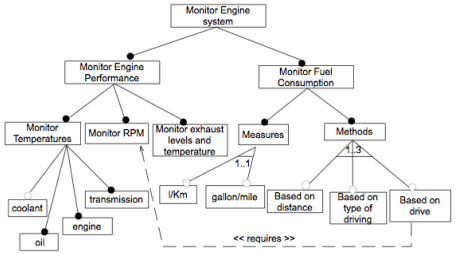
Gen. Prog. (GPFT)
[Czarnecki et al., 2000]



VBFD
[van Gurp et al., 2001]



EFD
[Riebisch et al., 2002]



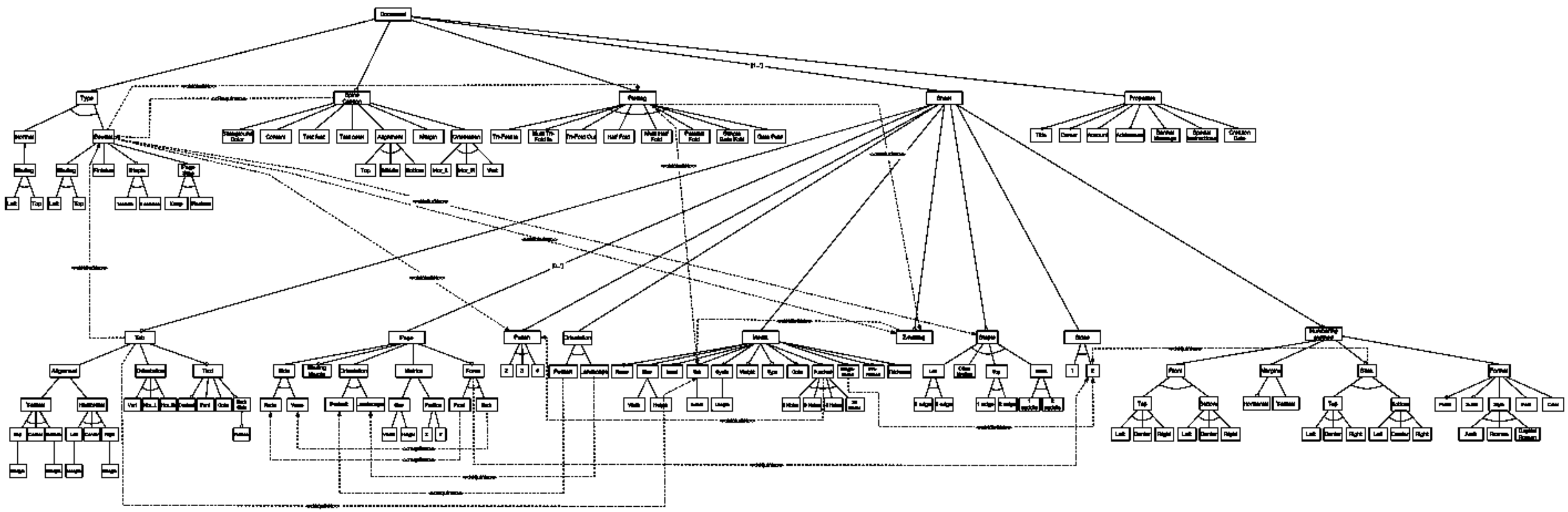
Concrete (graphical) syntax matters…

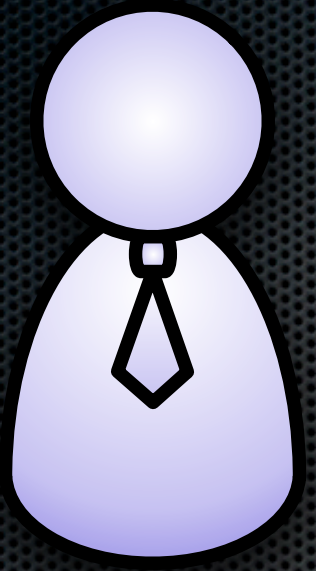
But semantics even more!

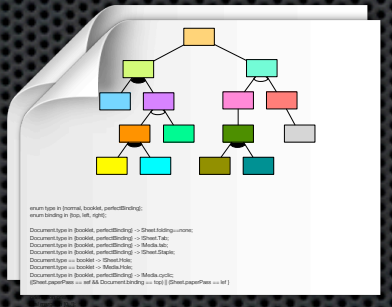
Feature Models

~ Feature Diagrams
(suggest a visual representation)

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

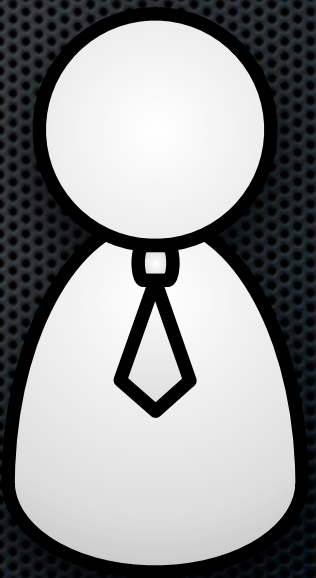

Type
Top
Left
Half Fold
Title

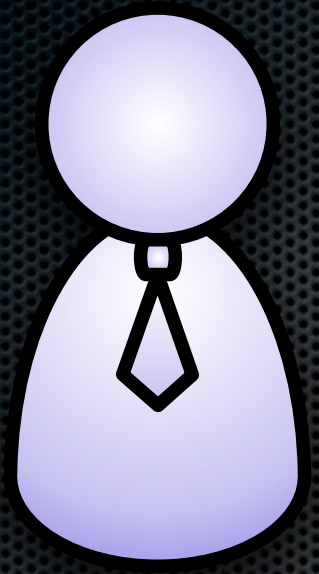




enum type in {normal, booklet, perfectBinding};
enum binding in {top, left, right};

Document.type in {booklet, perfectBinding} -> Sheet.folding == none;
Document.type in {booklet, perfectBinding} -> Sheet.Tab;
Document.type in {booklet, perfectBinding} -> !Media.tab;
Document.type in {booklet, perfectBinding} -> Sheet.Staple;
Document.type == booklet -> Sheet.Hole;
Document.type == booklet -> !Media.Hole;
Document.type in {booklet, perfectBinding} -> !Media.cyclic;
{|Sheet.paperPass == ref && Document.binding == top|} {|Sheet.paperPass == ref |}





Visualisation will help, right ?

- Tools create their own problems
- Lock-in, dependance
- Some information is inevitably textual
  - OCL constraints in UML
  - Minispecs in StateCharts
  - Attributes and constraints in FD

Why not use text altogether ?

- There are powerful tools for that
- Helps with
  - Editing
  - Transformation
  - Versioning
  - Information exchange
  - …
- Graphical views can be generated anyway

Not a new idea

Language	User Friendly	Attributes	Cardinalities	Cross tree constraints	Modula-risation
Van Deursen et al.	✔				
GUIDSL (AHEAD)	✔				
SXFM	✔				
XML-based (FAMA etc.)		✔	✔	✔	

Furthermore

- Intuitive C-like syntax
- Formal semantics
- Statically typed
- Editing facilities
  - Textual editor with auto-completion

TVL : allOf – and decomposition

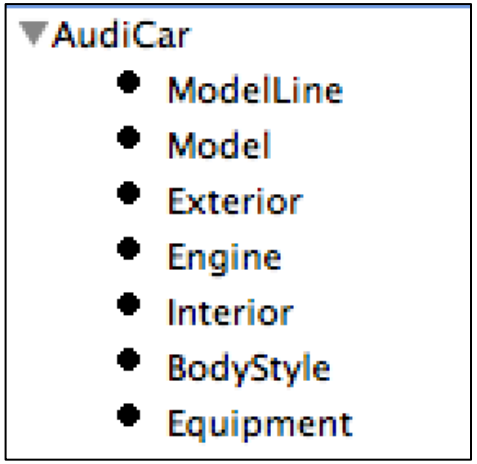
root feature

and decomposition

all subfeatures must be present in the product

sub features

```
root AudiCar {
        group allOf {
        ModelLine,
        BodyStyle,
        Model,
        Engine,
        Exterior,
        Interior,
        Equipment
        }

}
```



TVL : optional features

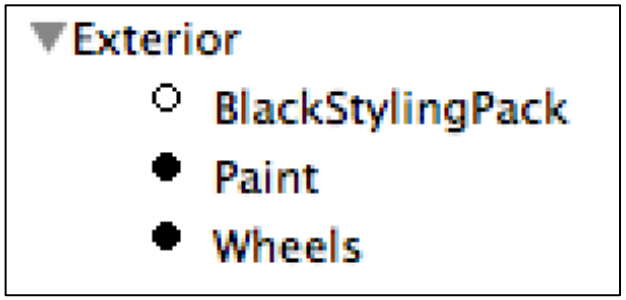
```
Exterior{
    group allof {
            Wheels,
            Paint,
            opt BlackStylingPack
    }
}
```

all subfeatures

must be present in the product

except this one

it is optional



TVL : one of – xor decomposition

feature definition

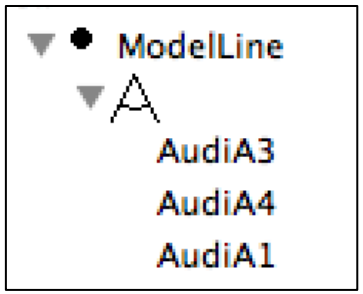
xor decomposition

one and only one subfeature must be present in the product

```
ModelLine {
        group oneof {
                AudiA1 {
                        AudiA1 -> (A1 || A1Sportback);
                        AudiA1 -> (Door3 || Sportback);
                },
                AudiA3
                {
                        AudiA3 -> (A3 || A3Sportback || A3Cabriolet
                                || S3 || S3Sportback || RS3Sportback);
                        AudiA3 -> (Door3 || Sportback || Cabriolet);
                },
                AudiA4
                {
                        AudiA4 -> (A4Avant || A4AllroadQuattro || S4Avant);
                        AudiA4 -> (Avant || AllroadQuattro);
                },
        }

}
```

constraints definition

Boolean expressions in terms of features that must remain true



TVL : someOf – or decomposition

feature definition

or decomposition

one or more subfeatures must be present in the product

```
Equipment{
            group someof {
                        Package,
                        EquipExterior,
                        Tyres,
                        EquipInterior,
                        SteeringWheels,
                        Seats,
                        SafetyAndTechnology,
                        AudiAndCommunication,
                        Warranty
            }
}
```

TVL : Modularity

```
root Car {
          group allOf {
          BodyStyle,
          Model,
          Engine,
          Equipment }

}

include(BodyStyle.tvl);
include(Model.tvl);
include(Engine.tvl);
include(Equipment.tvl);
```

Split model in reusable parts using external files

TVL : Modularity

BodyStyle.tvl

```
BodyStyle {
    group oneof {
           Avant,
           Door3,
           Sportback,
           Coupe,
           Cabriolet,
           Roadster,
           Spyder,
           SUV
     }

}
```

Model.tvl

```
Model {
    group oneof {
           A1,
           A3,
           S3,
           A4,
           S4,
           A5,
           S5,
           RS5,
           A6,
           A7,
           A8,
           A8L,
           R8
           Q3,
           Q5,
           Q7,
           TT,
           TTS,
           TT RS
     }
}
```

Engine.tvl

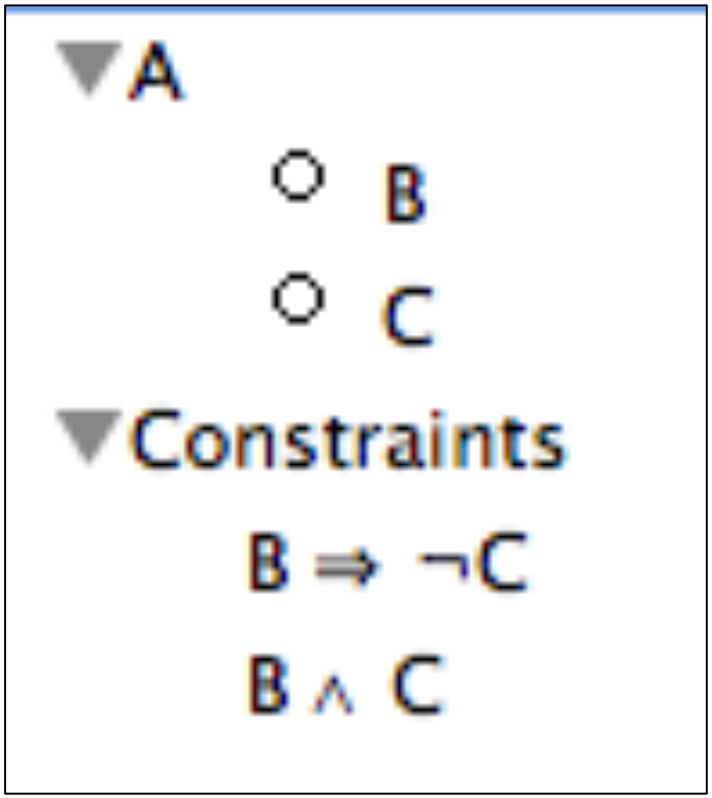
```
Engine {
   group oneof {
      SE12TFSI,
      CE14TFSI,
      CE14TFSI119,
      Sport12TFSI,
      Sport14TFSI,
      Sport14TFSI119,
      BlackEdition14TFSI,
      SLine12TFSI,
      SLine14TFSI,
      SE16TDI,
      CE16TDI,
      CE20TDI,
      ...
     }

}
```

TVL : Tools

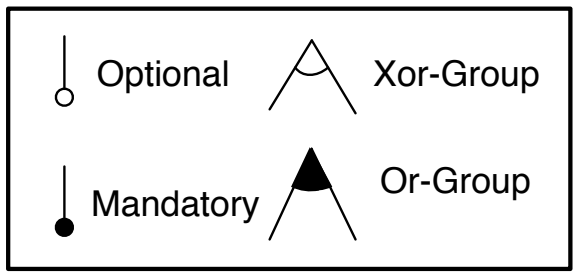
- Key references
  - Classen, A.; Boucher, Q. and Heymans, P. A Text-based Approach to Feature Modelling: Syntax and Semantics of TVL. In Science of Computer Programming (SCP), Special Issue on Software Evolution
  - **Hubaux, A.; Boucher, Q.; Hartman, H.; Michel, R. and Heymans, P. Evaluating a Text-based Feature Modelling Language: Four Industrial Case (SLE 2010)**
  - http://info.fundp.ac.be/tvl/
- Support
  - Parser / Syntactic and semantic checker
  - SAT-based checker for Boolean models

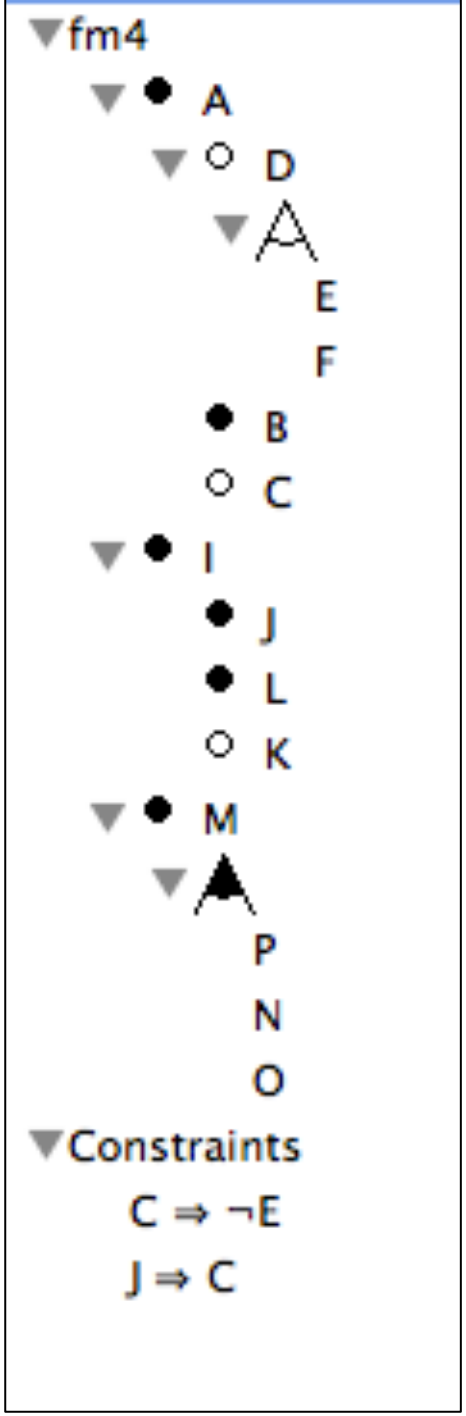
- Lessons learned
  - Textual vs graphical languages
  - Semantics of your language matter

I want to analyze and play with my specification!

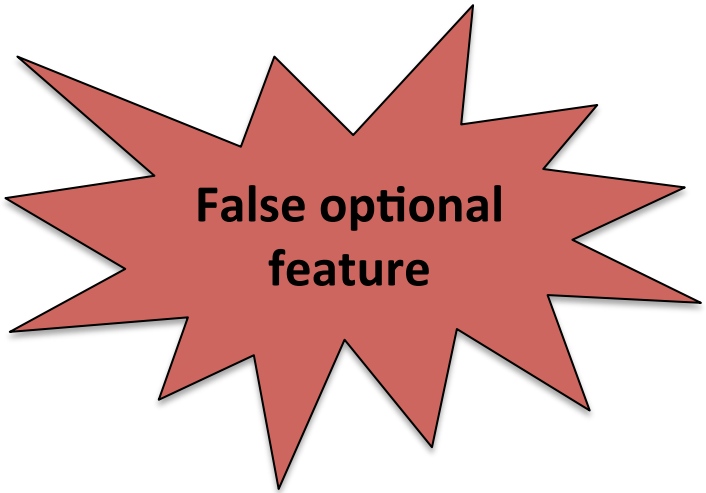


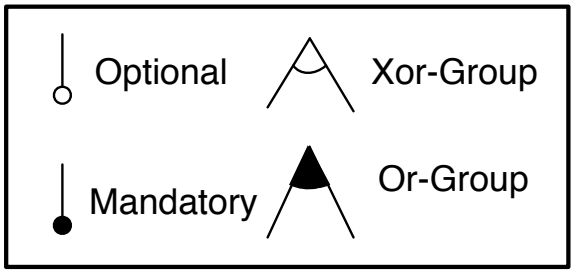
Empty set of configurations

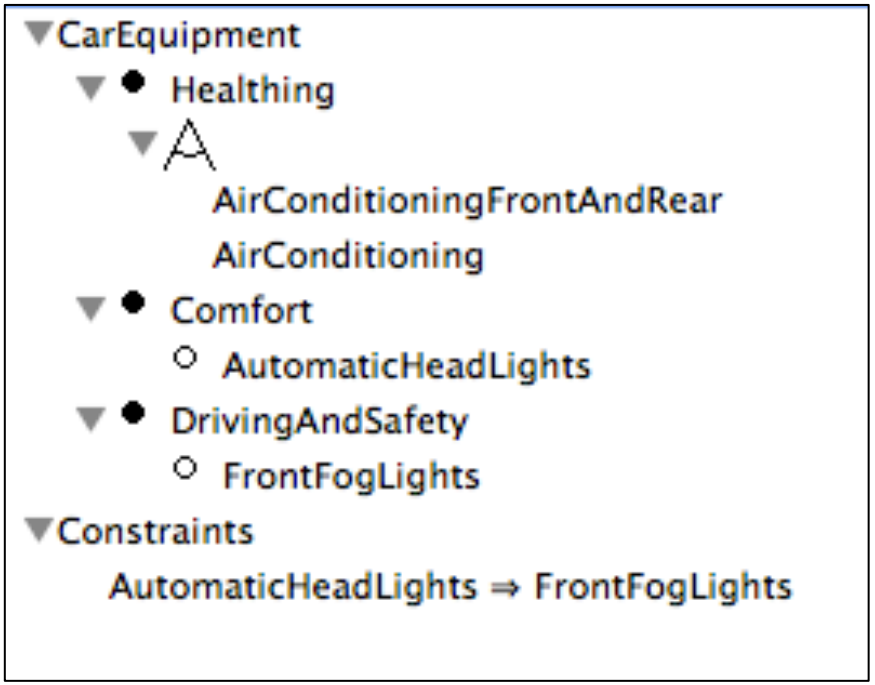


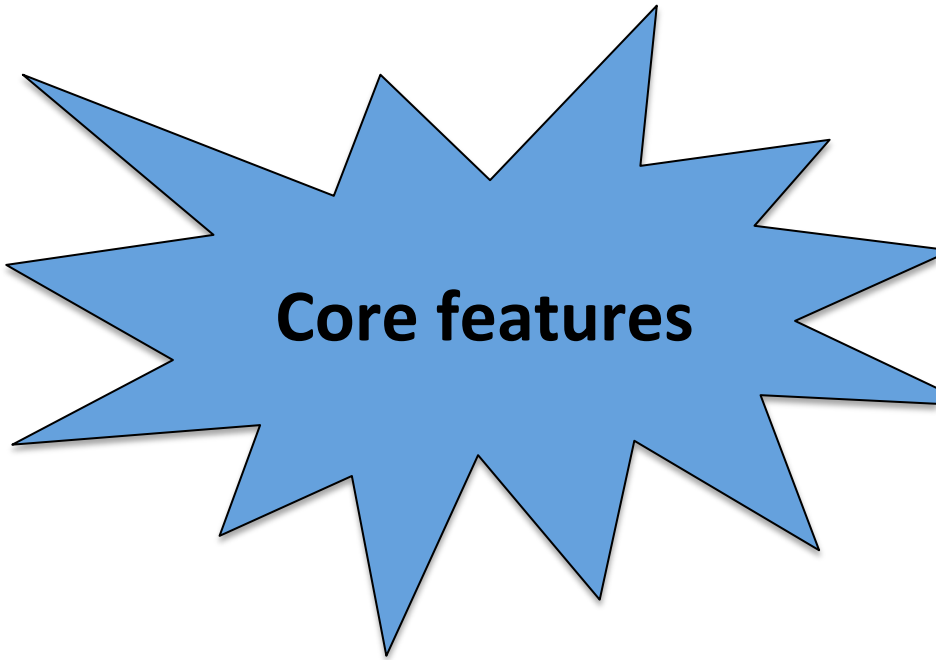

fm4
A
D
E
F
B
C
I
J
L
K
M
P
N
O
Constraints
C ⇒ ¬E
J ⇒ C


Dead feature

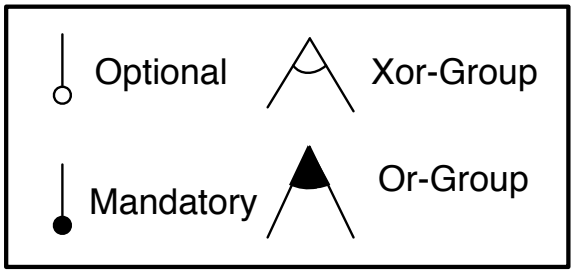

False optional feature

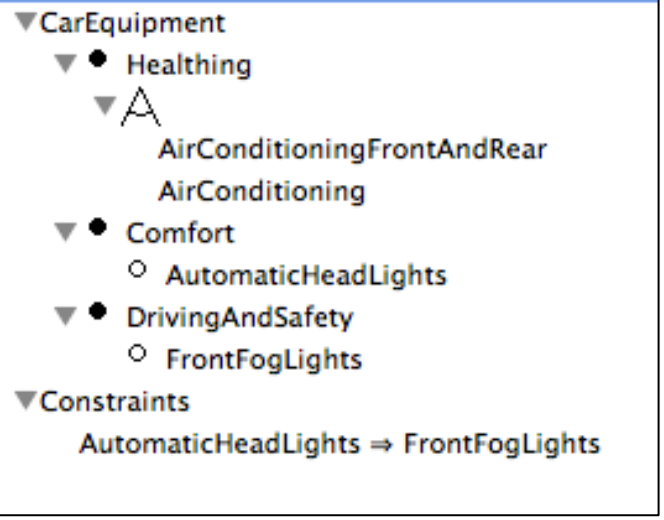

Optional
Xor-Group
Mandatory
Or-Group

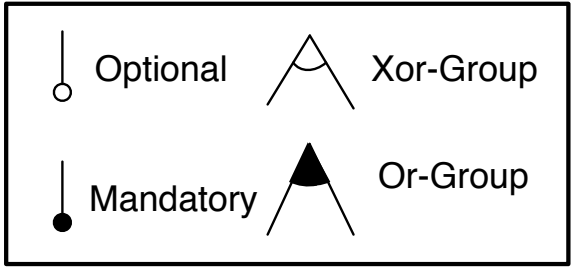


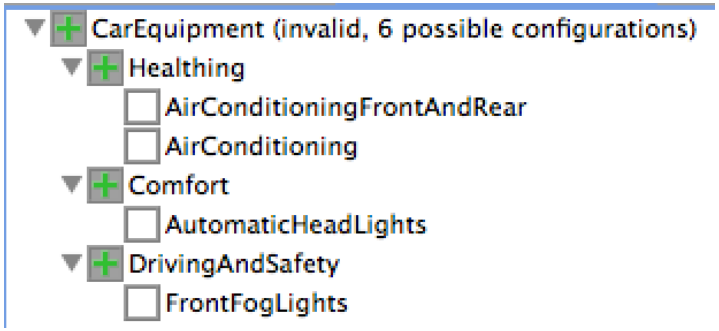


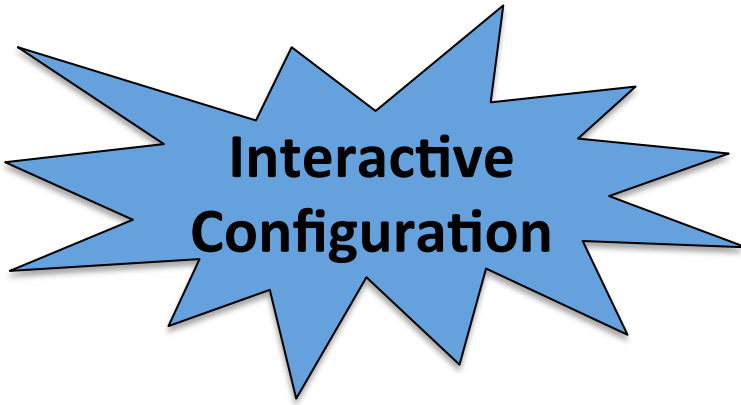
{CarEquipment, Comfort, DrivingAndSafety, Healthing}

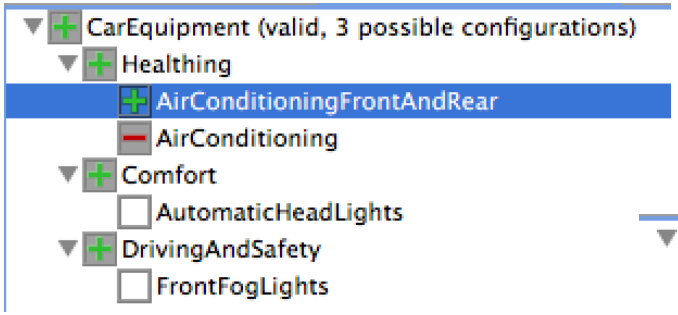


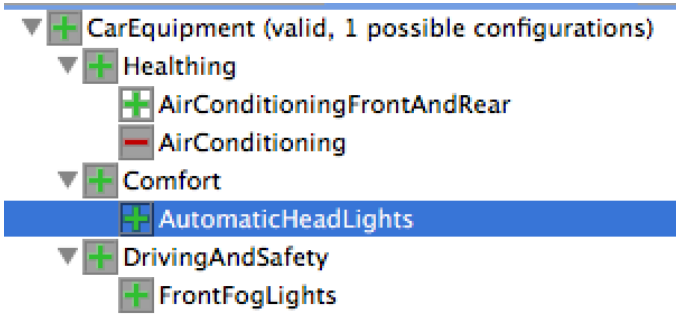

CarEquipment
Healthing
AirConditioningFrontAndRear
AirConditioning
Comfort
AutomaticHeadLights
DrivingAndSafety
FrontFogLights
Constraints
AutomaticHeadLights ⇒ FrontFogLights


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

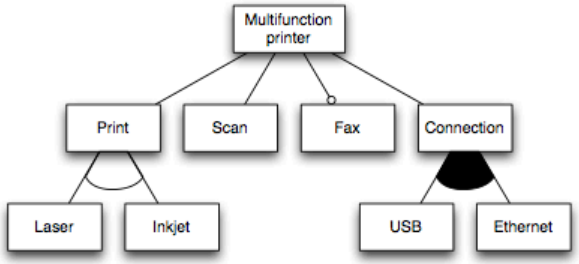
Decision problems and complexity

- 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

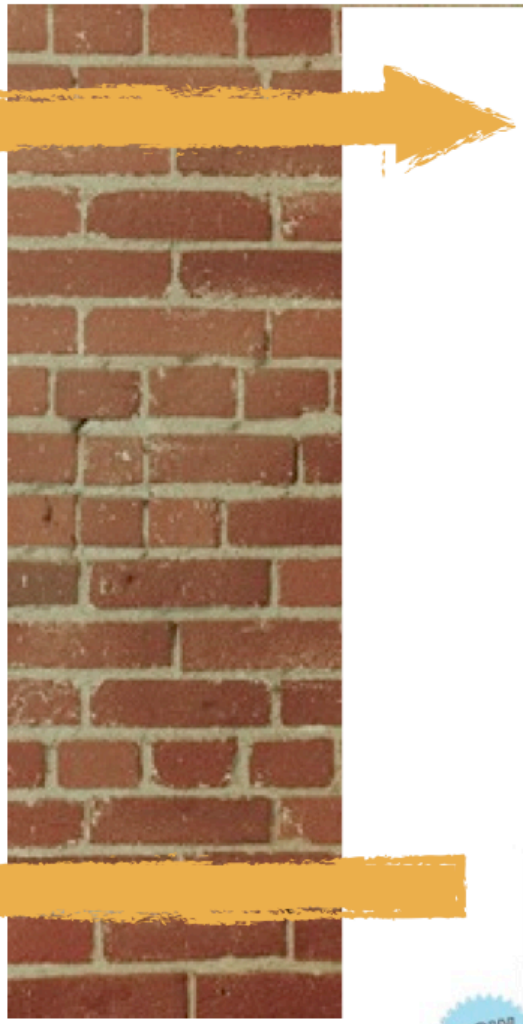
How to automate analysis of your feature models?

Binary Decision Diagram (BDD)
SAT solver

Typical implementations



result

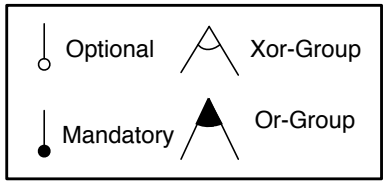


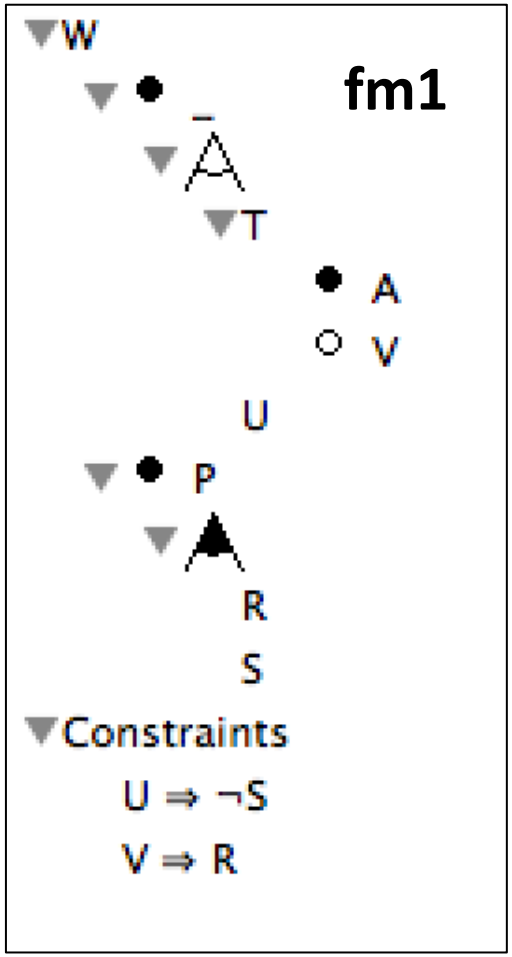
logics

solvers

(Boolean) Feature Models

Hierarchy + Variability = set of valid configurations

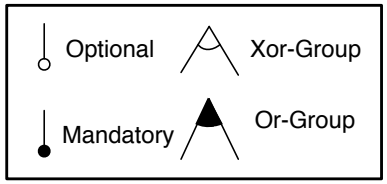


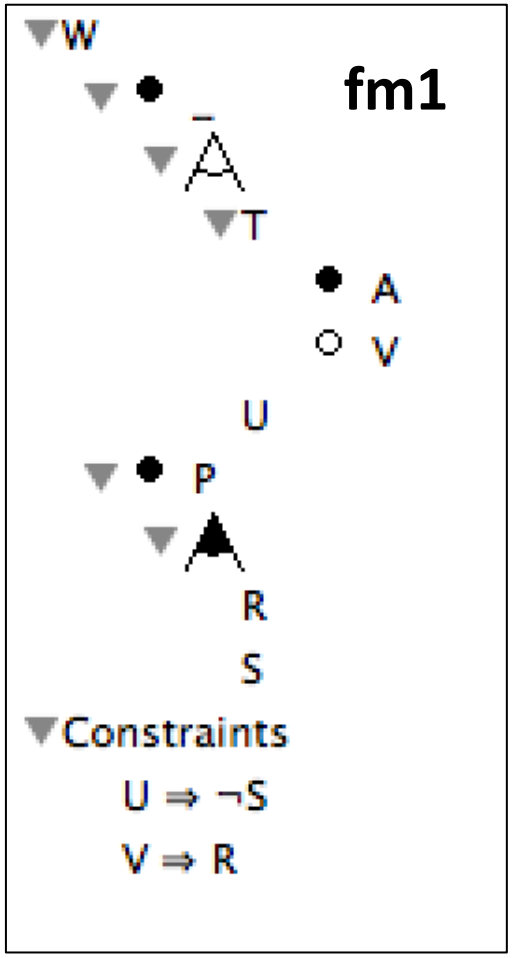


$$[\![fm1]\!] = \{$$
$$\{W, P, R, S, T, A, V\},$$
$$\{W, P, S, T, A\},$$
$$\{W, P, R, T, A\},$$
$$\{W, P, R, U\},$$
$$\{W, P, R, T, V, A\},$$
$$\{W, P, R, S, T, A\},$$
$$\}$$

(Boolean) Feature Models

~ Boolean formula



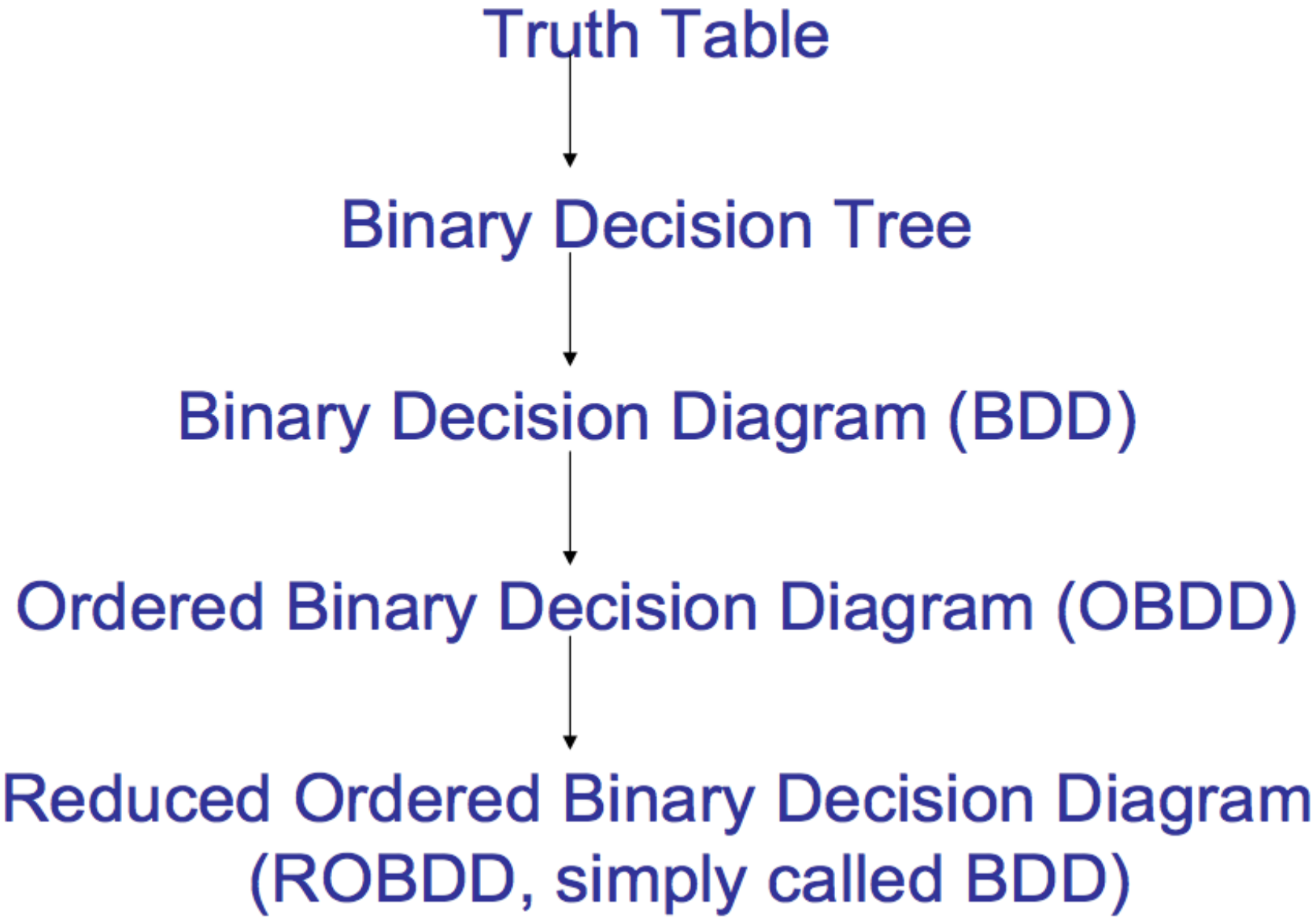


$$
\begin{aligned}
&\phi_{fm_1} = W \;//\; \text{root} \\
&\wedge\, W \Leftrightarrow P \;//\; \text{mandatory} \\
&//\; \text{Or-group} \\
&\wedge\, P \Rightarrow R \vee S \\
&\wedge\, R \Rightarrow P \wedge\, S \Rightarrow P \\
&\wedge\, V \Rightarrow T \;//\; \text{optional} \\
&\wedge\, A \Leftrightarrow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

Truth table, boolean function

from		*to*		
x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

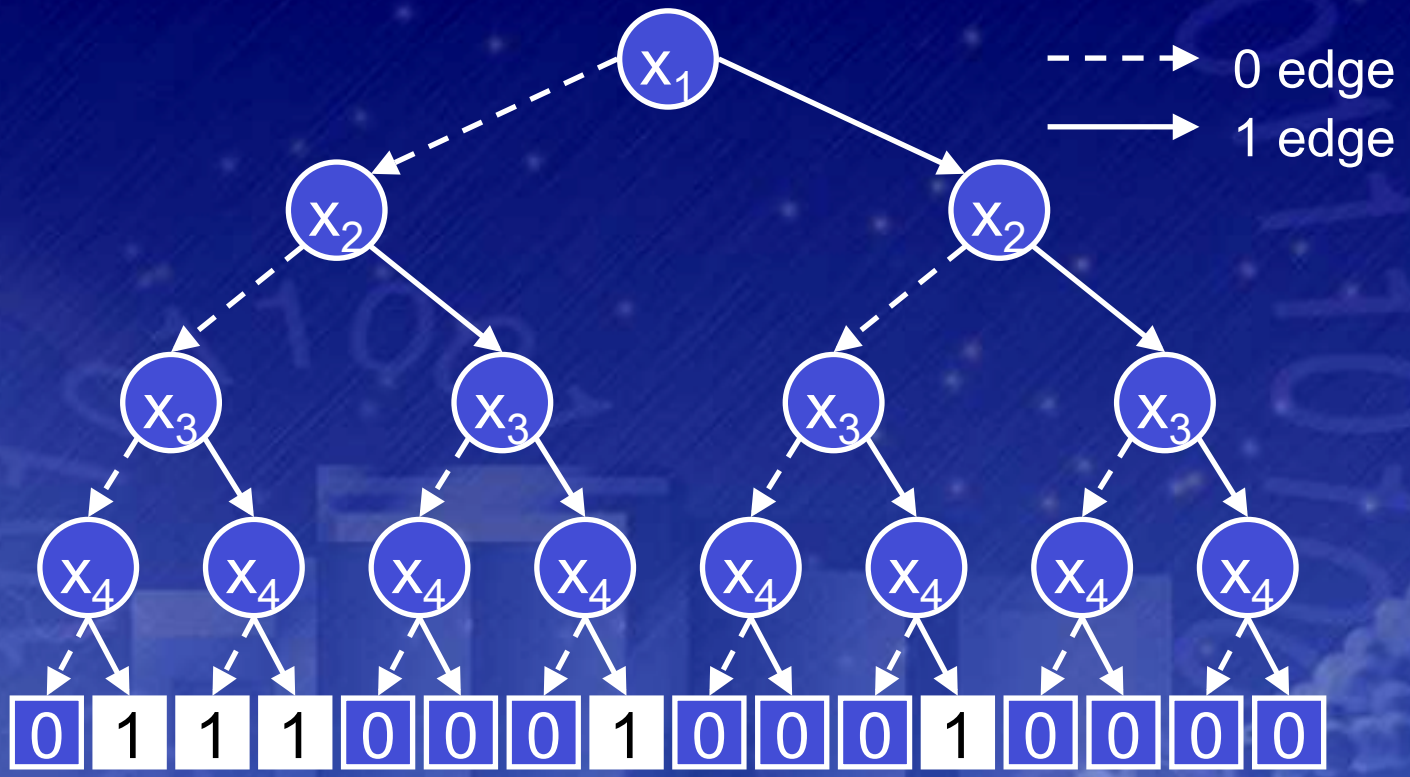
BDDs from Truth Tables



Binary Decision Diagrams (Bryant 1986)

encoding of a truth table.

from		*to*		
x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Reduction

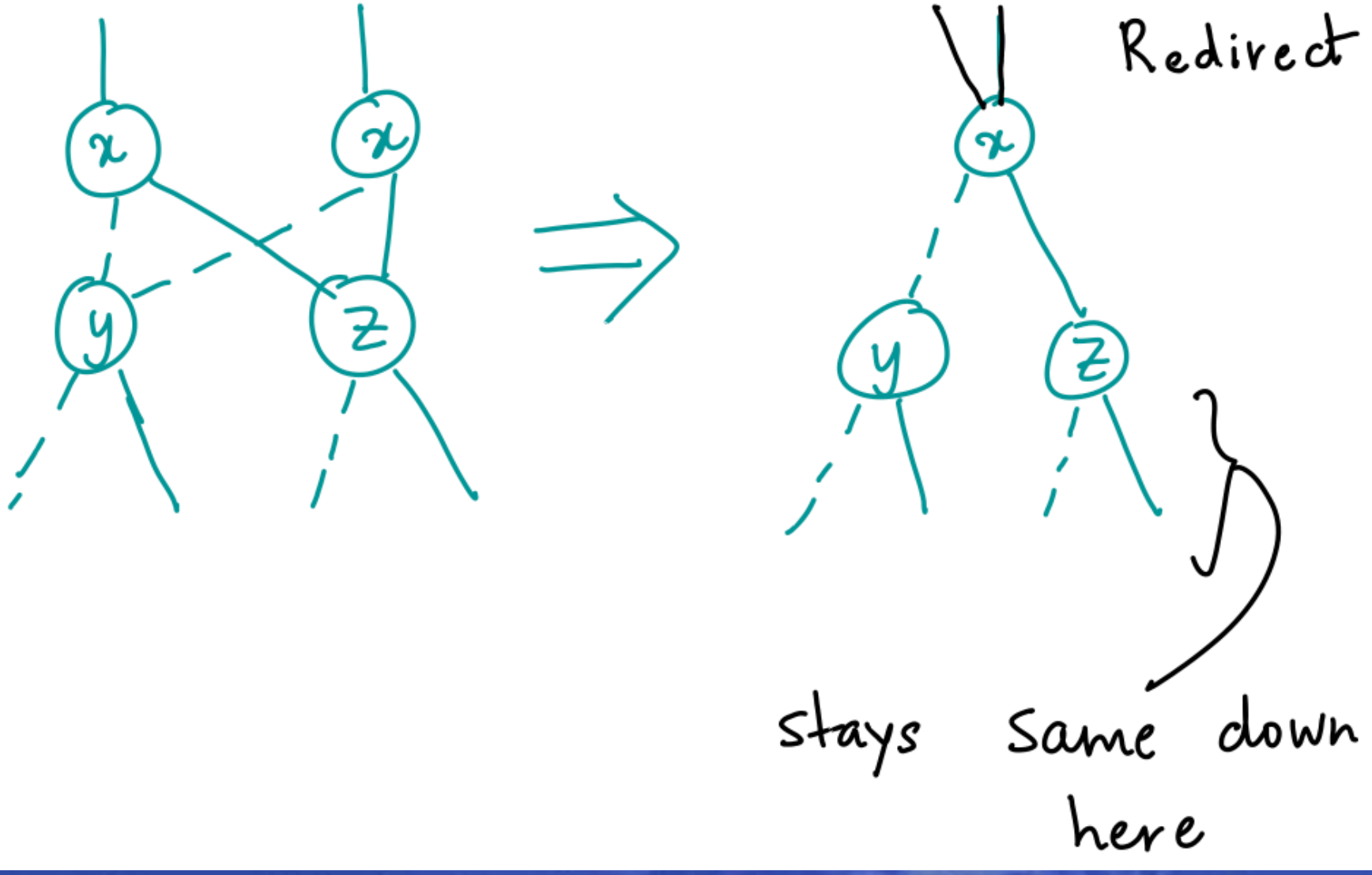
- Identify Redundancies

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

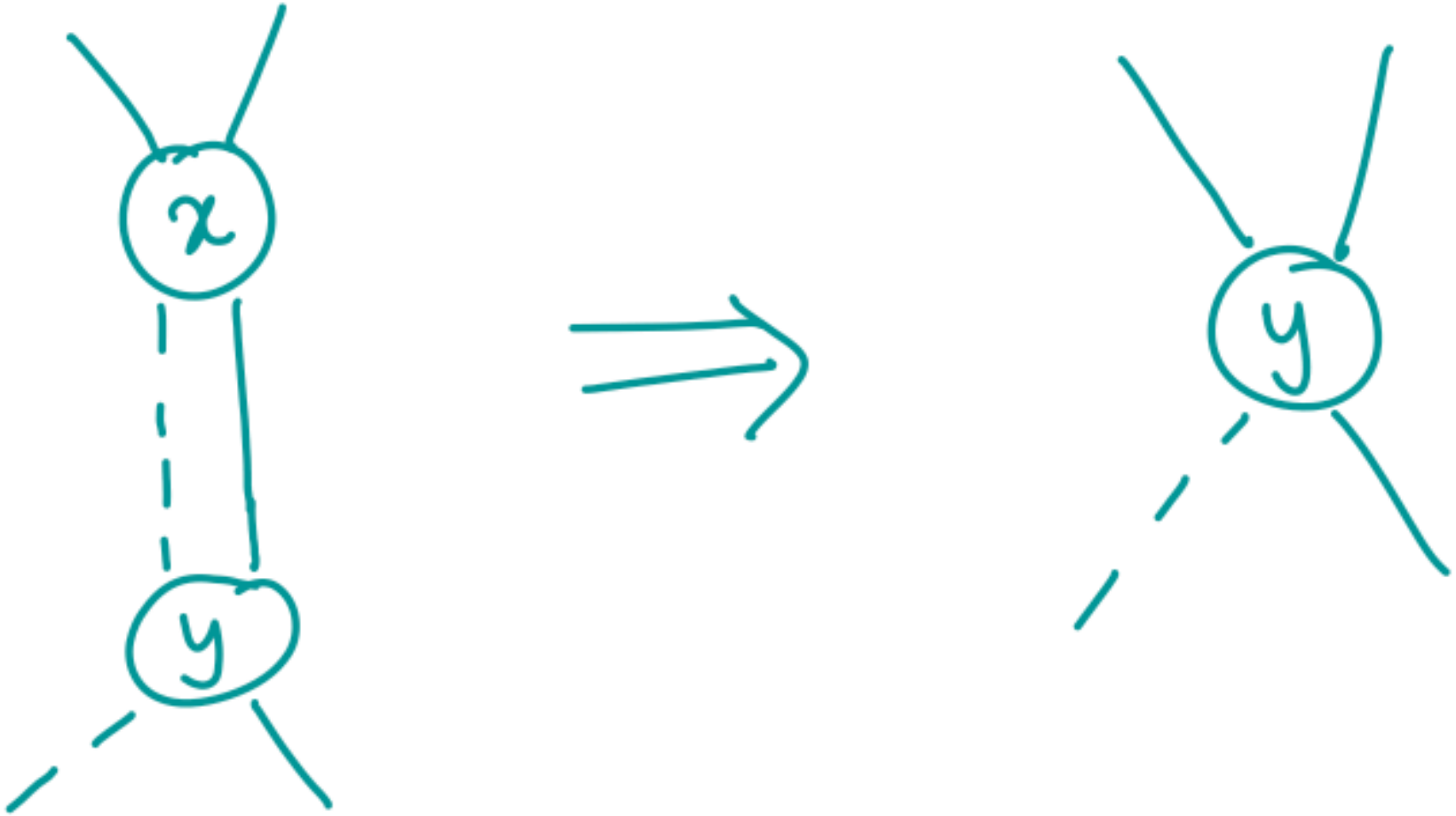
Merge equivalent leaves

"a" is either 0 or 1

Merge isomorphic nodes

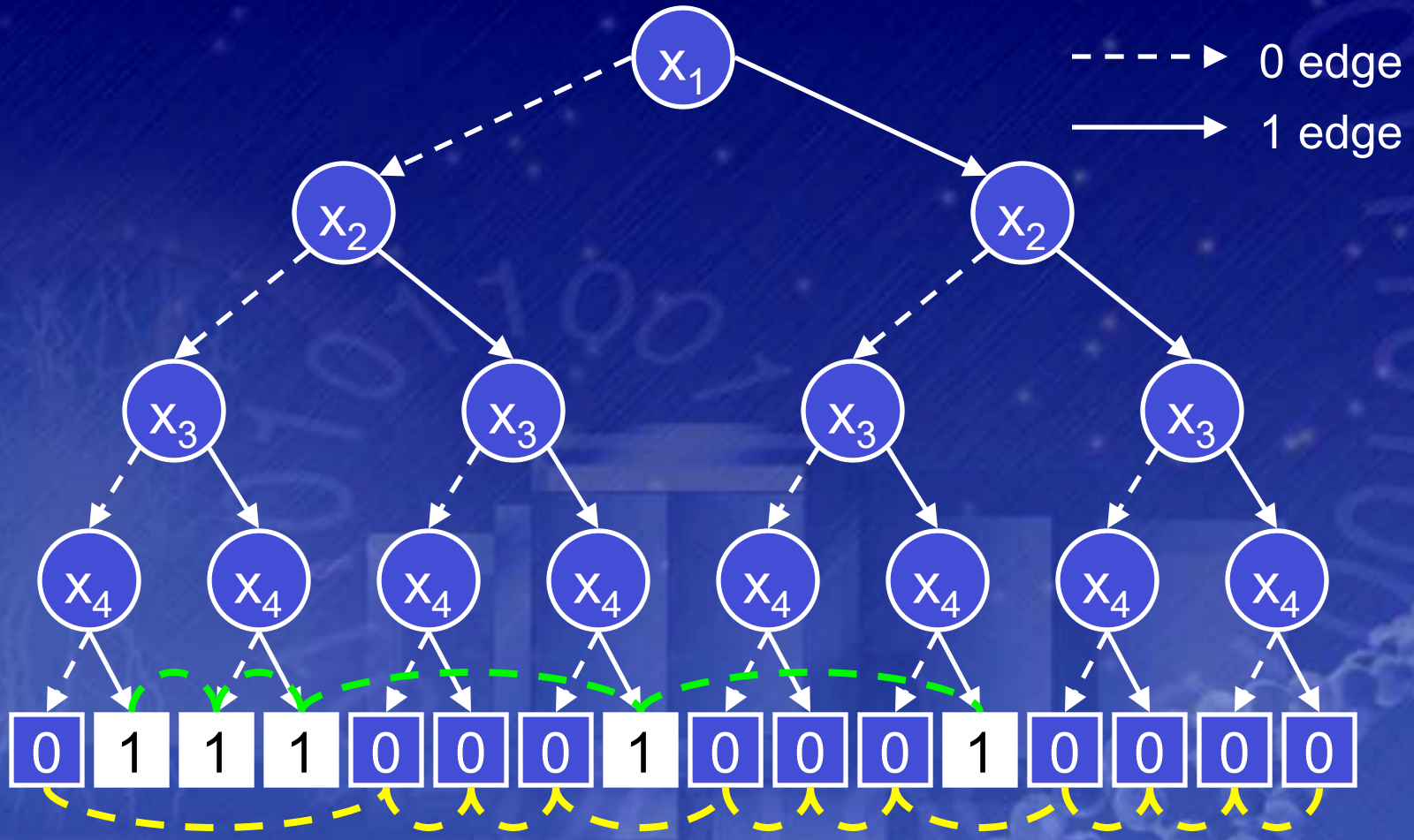


Eliminate redundant tests



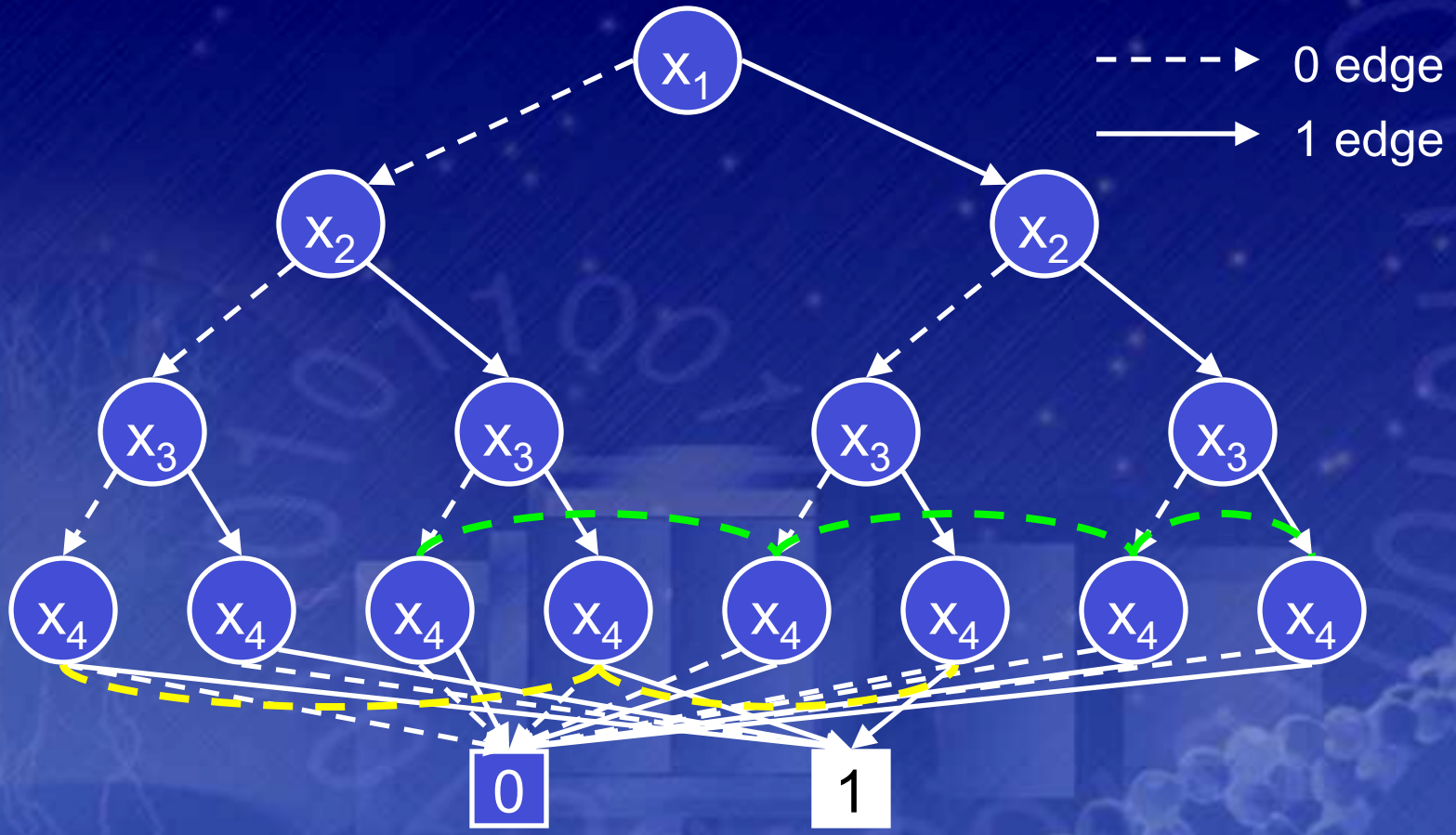
Binary Decision Diagrams

- Collapse redundant nodes.



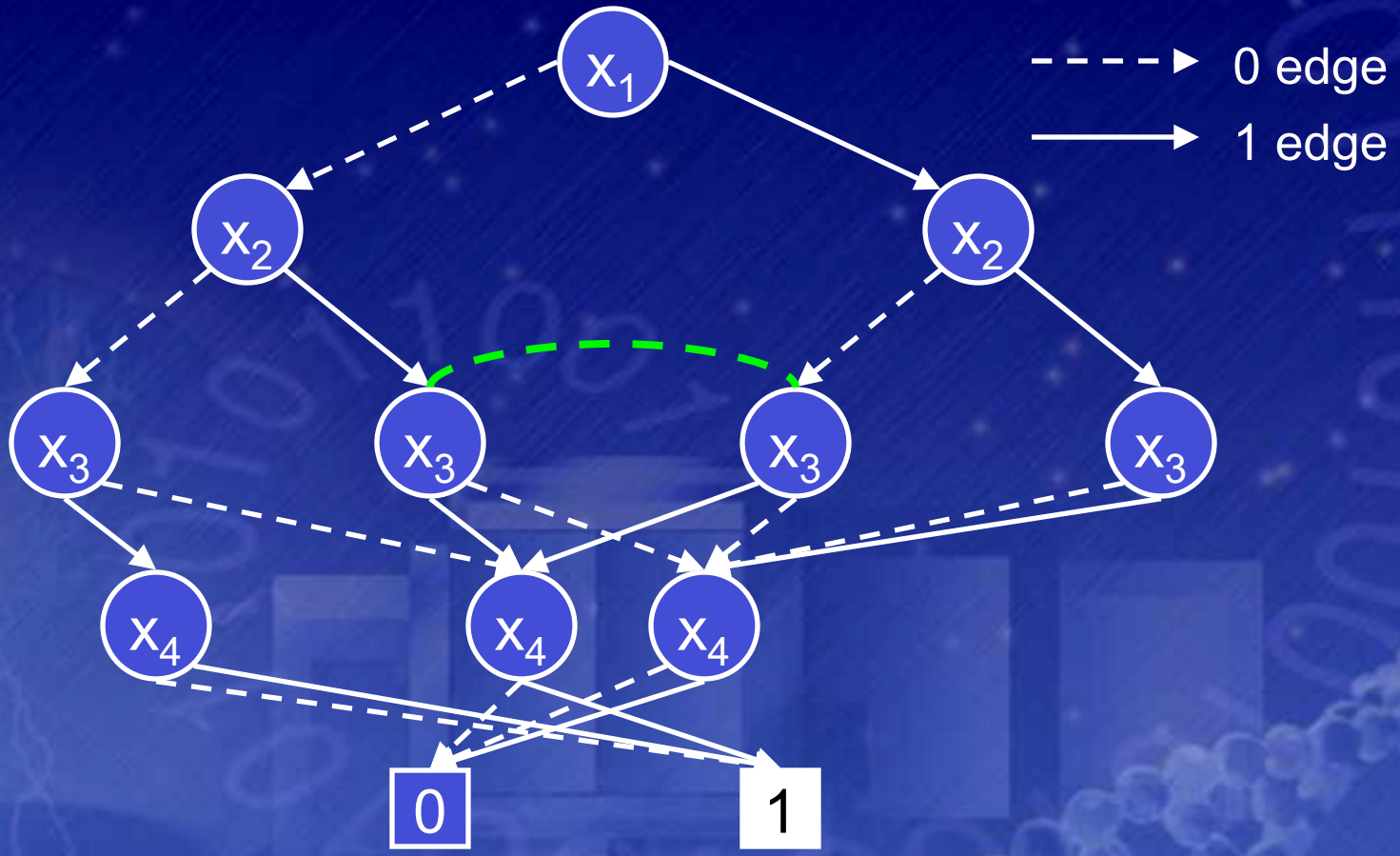
Binary Decision Diagrams

- Collapse redundant nodes.



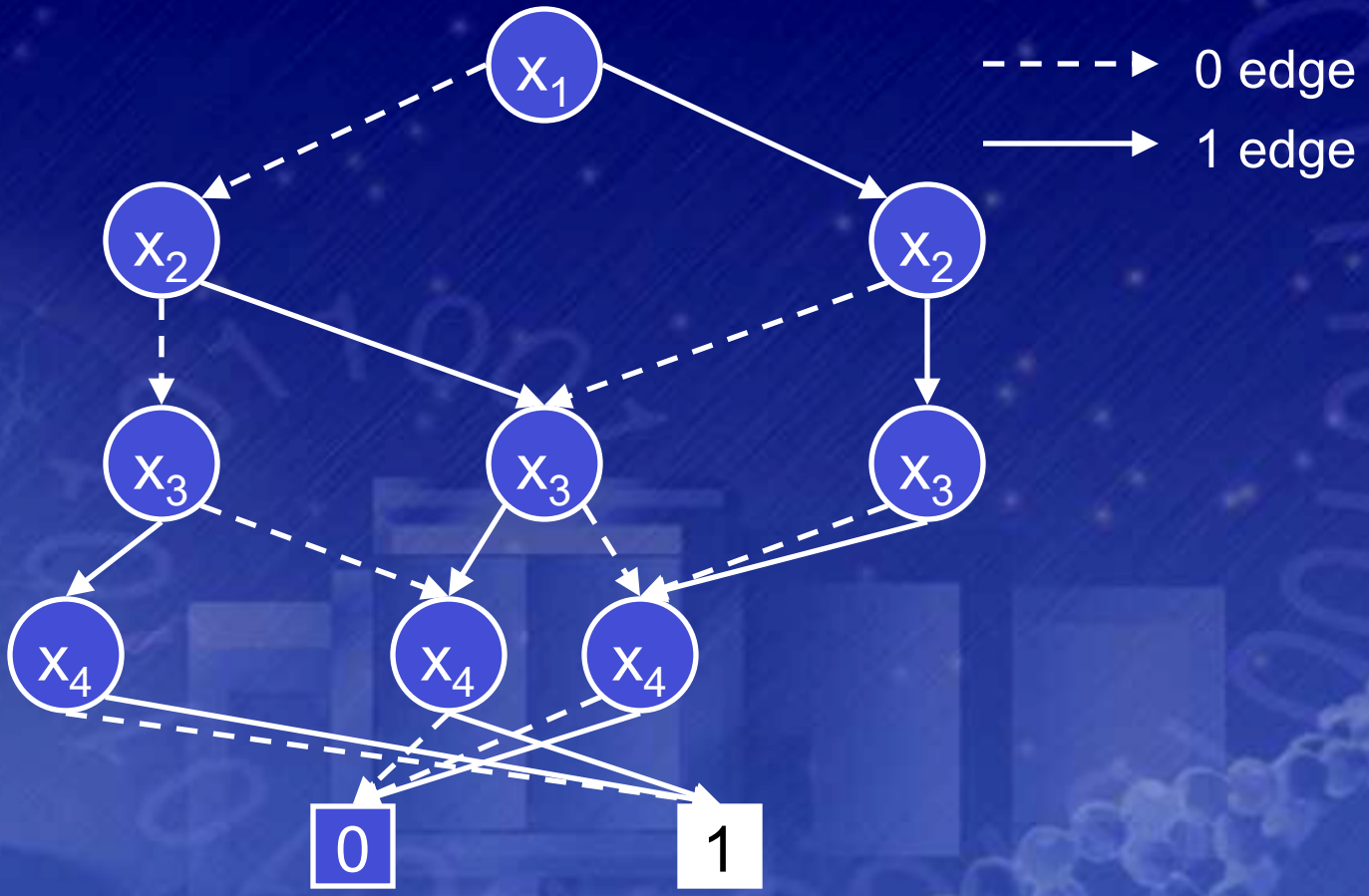
Binary Decision Diagrams

- Collapse redundant nodes.



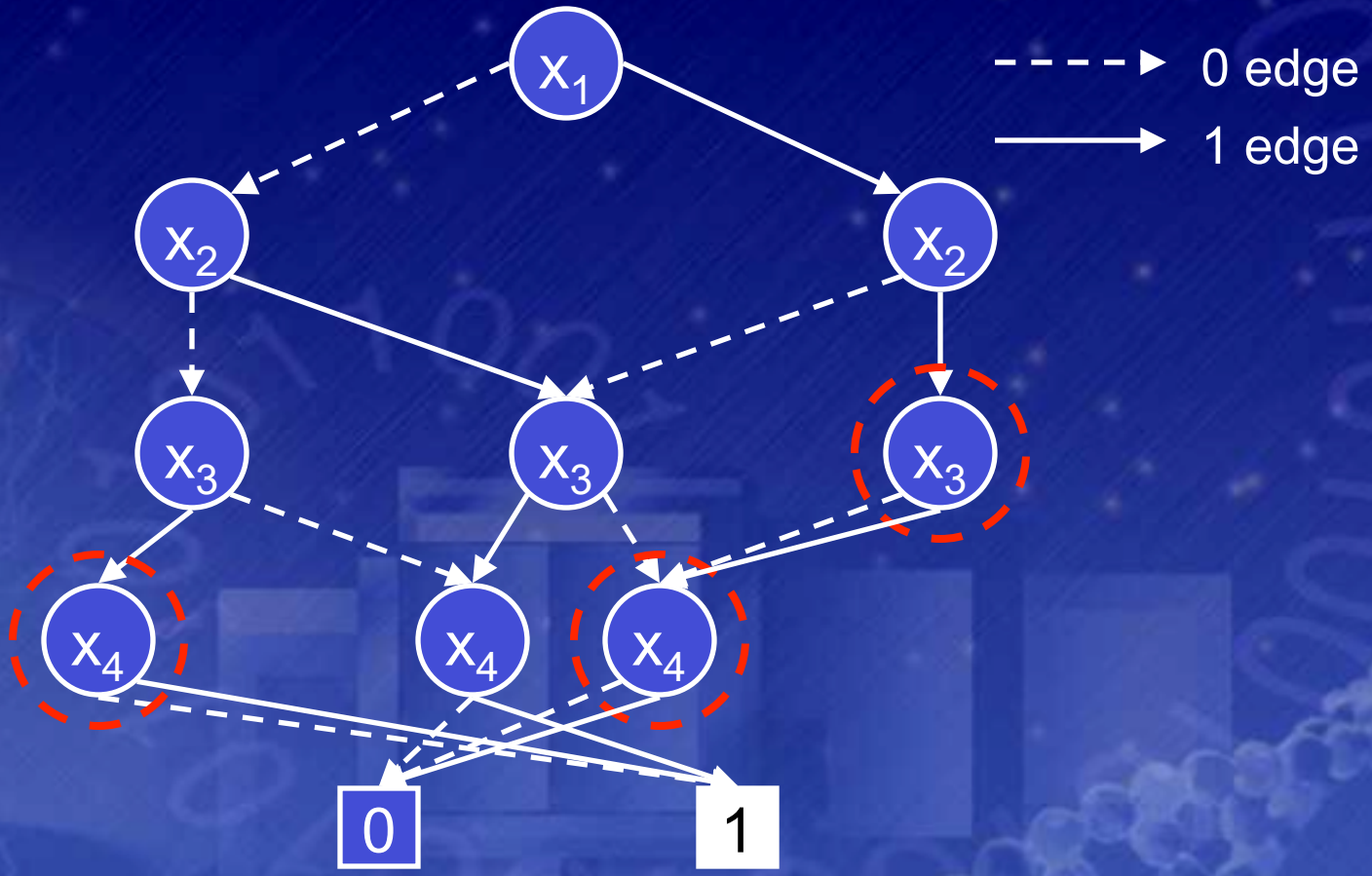
Binary Decision Diagrams

- Collapse redundant nodes.



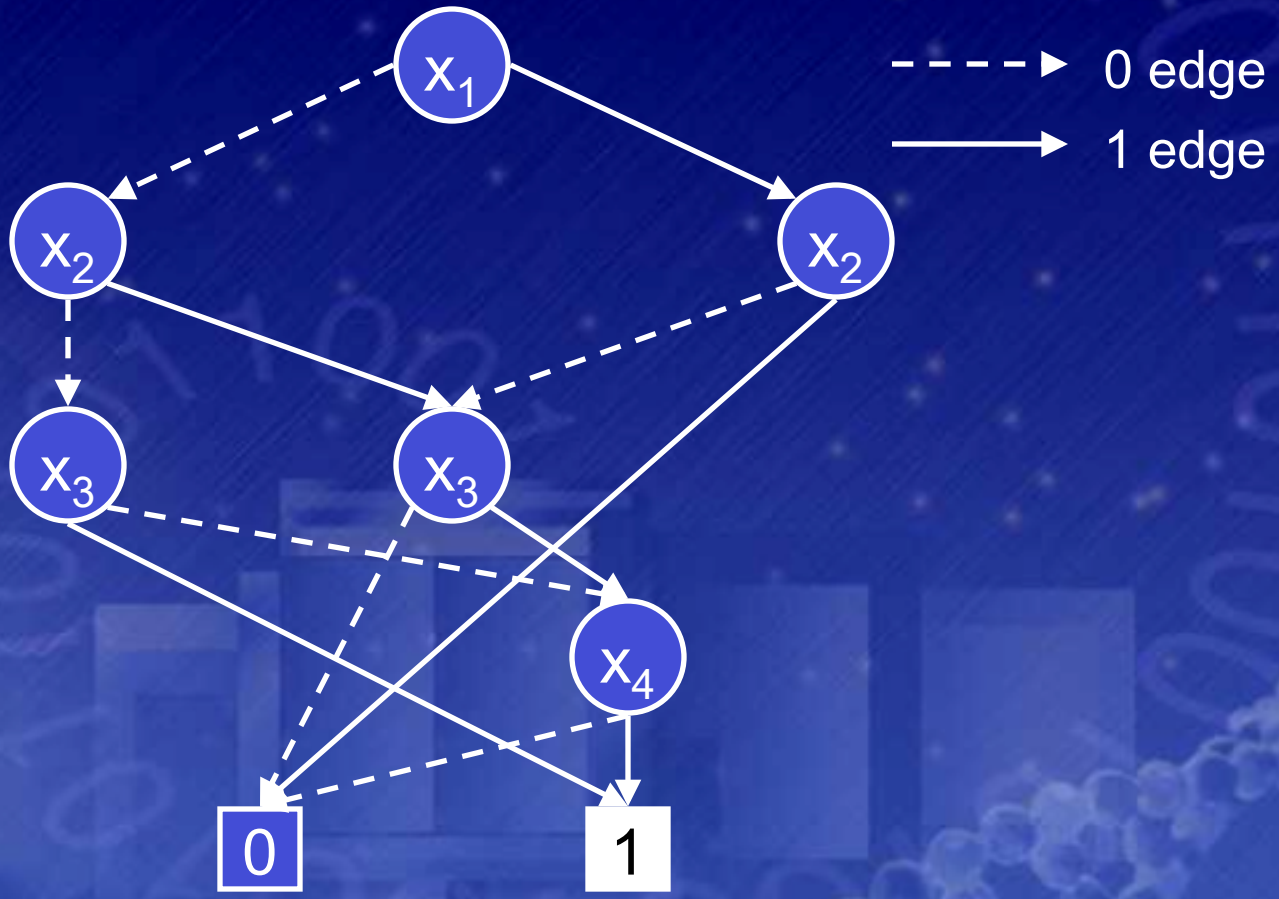
Binary Decision Diagrams

- Eliminate unnecessary nodes.



Binary Decision Diagrams

- Eliminate unnecessary nodes.

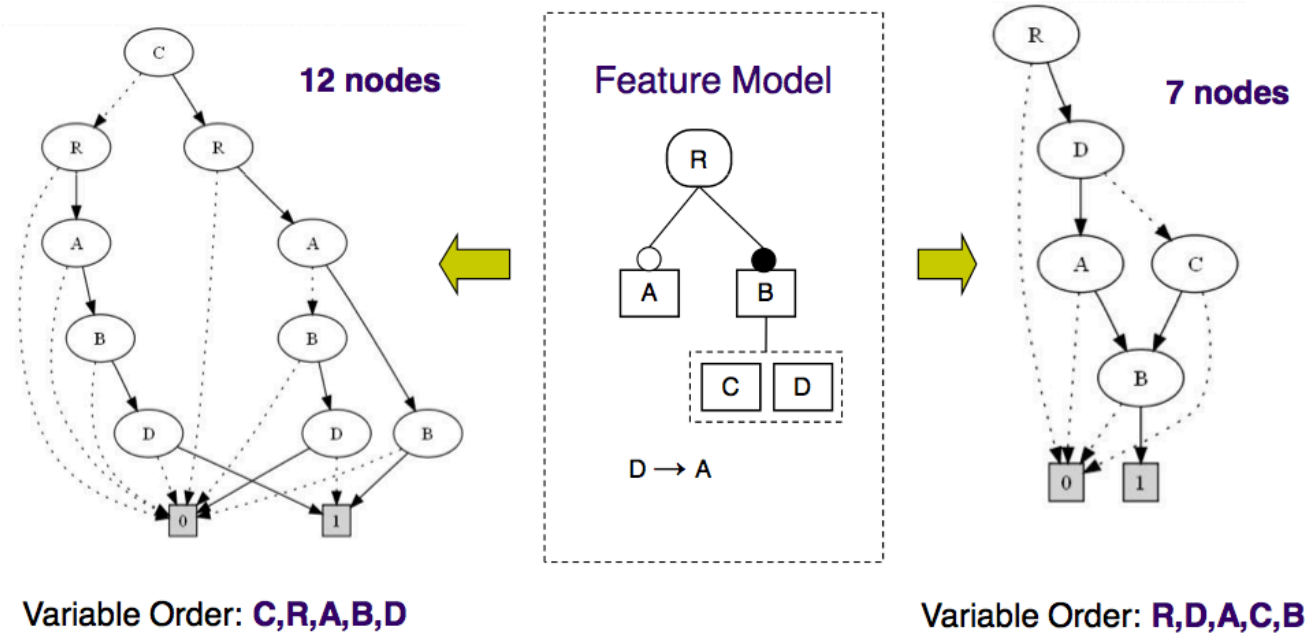


Binary Decision Diagrams (BDDs)

- 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

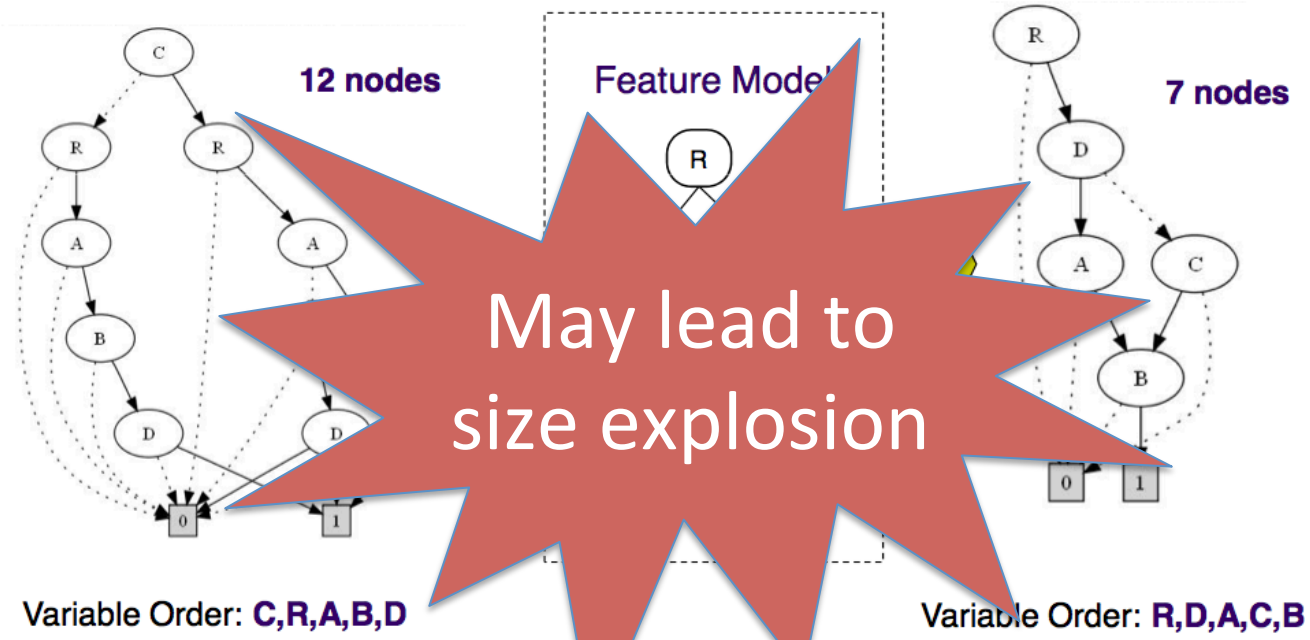
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**



[Mendonca, slide]

Binary Decision Diagrams (BDDs): Theoretical Problem

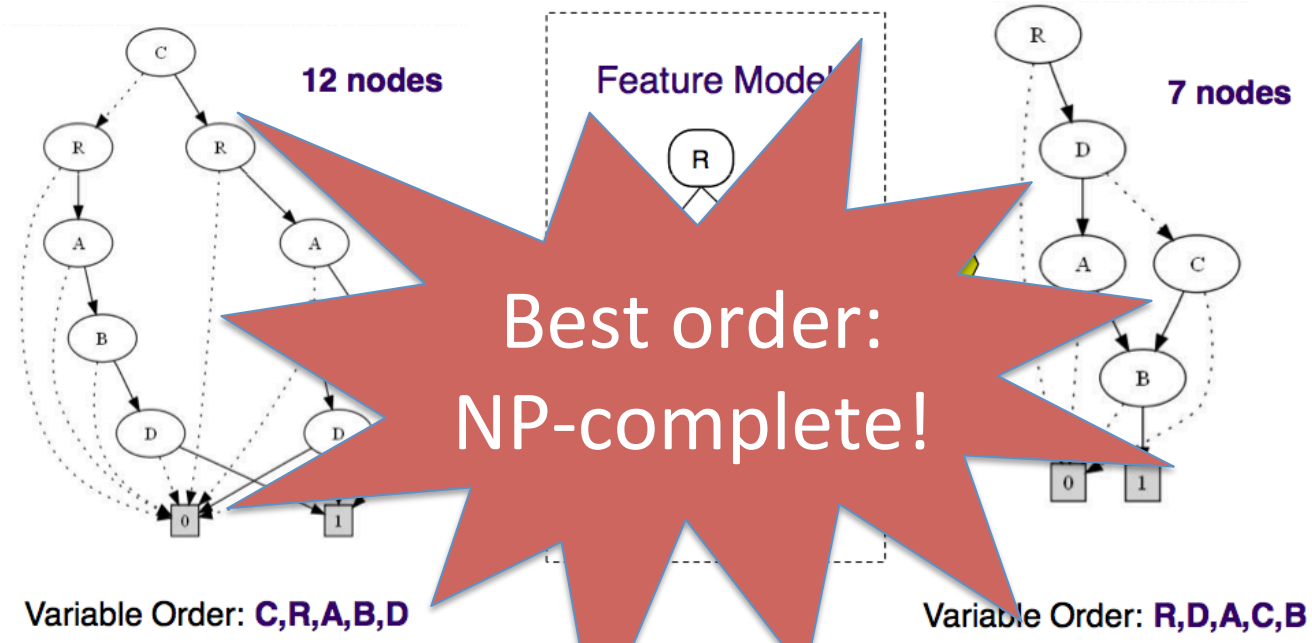
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**



[Mendonca, slide]

Binary Decision Diagrams (BDDs): Theoretical Problem

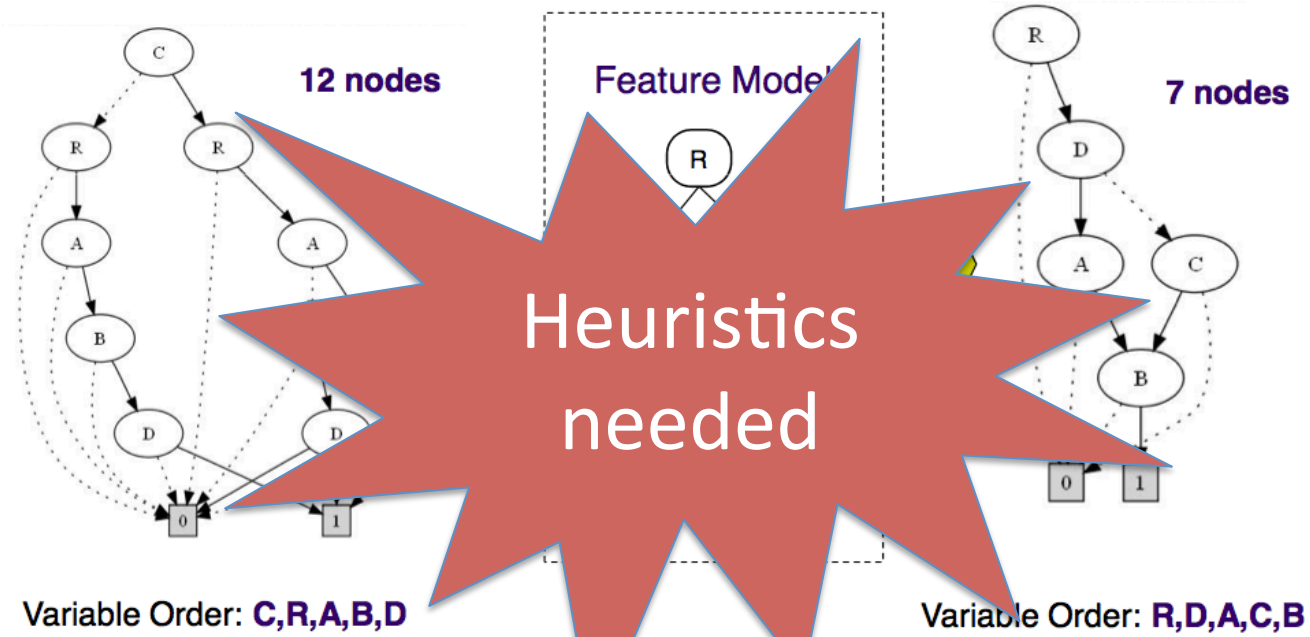
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**



[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**



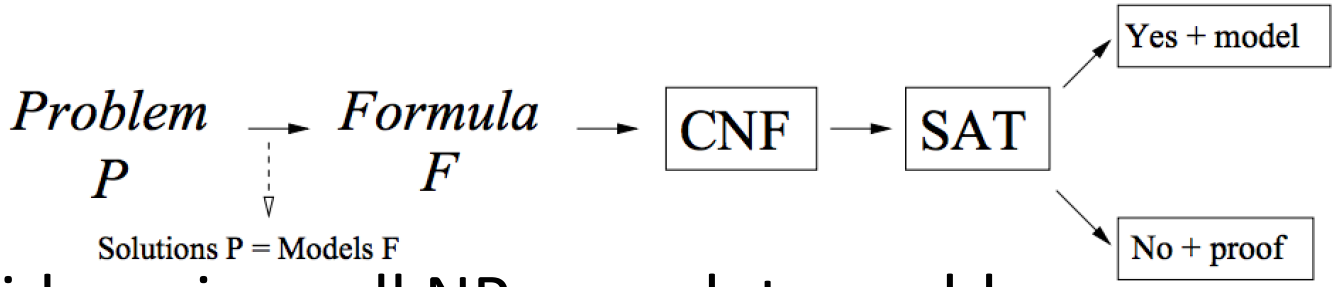
[Mendonca, slide]

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;

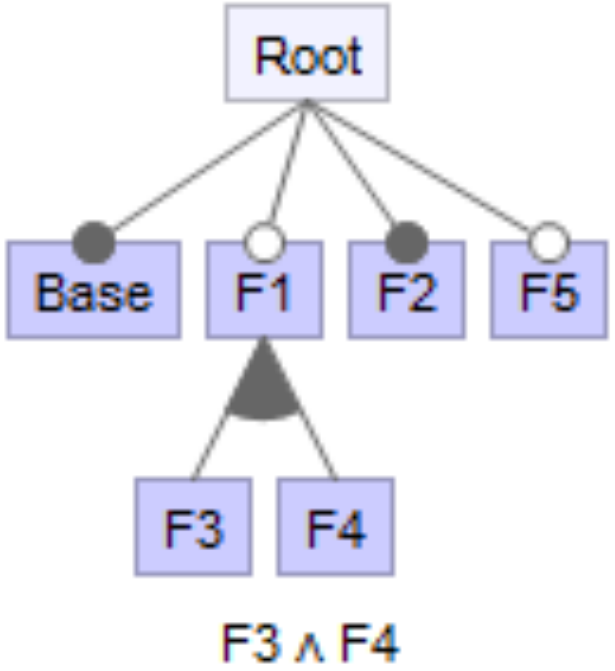
satisfy(ϕ) {
  if every clause of ϕ has a true literal, return T;
  if BCP() == F, return F;
  assign appropriate values to all pure literals;
  choose an $x \in V$ that is unassigned in A,
    and choose $v \in \{T, F\}$.
  $A(x) = v$;
  if satisfy(ϕ) return T;
  $A(x) = \neg v$;
  if satisfy(ϕ) return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

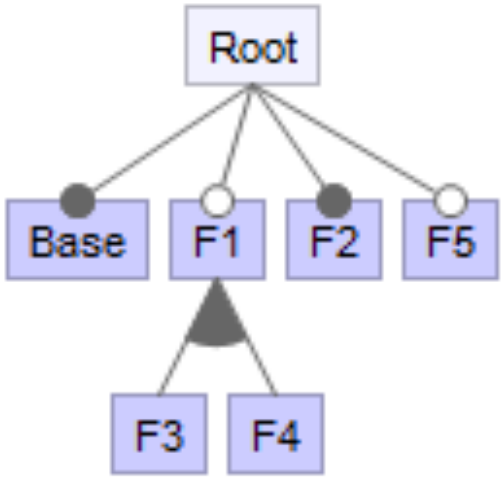
Consistency

- SAT-Solver
  - SAT(FM)



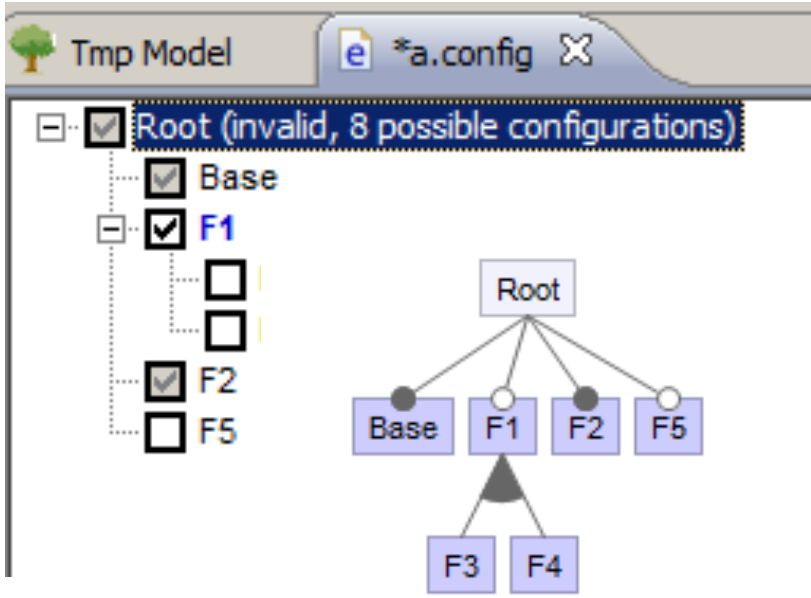
Core and dead features

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

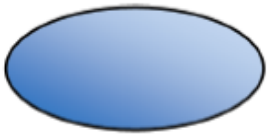
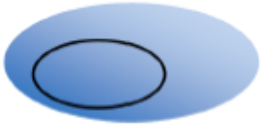
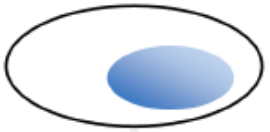
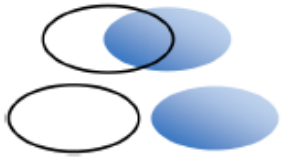


Partial configuration

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



Relationship between feature models

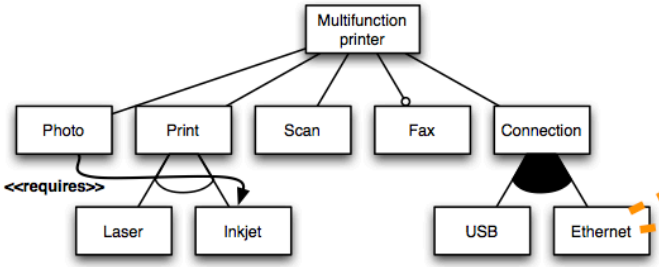
	No Products Added	Products Added
No Products Deleted	 Refactoring	 Generalization
Products Deleted	 Specialization	 Arbitrary Edit

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

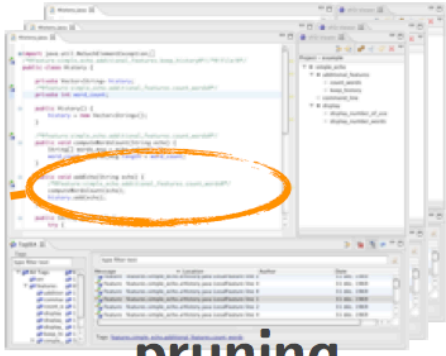
Recap

Product Derivation

feature model



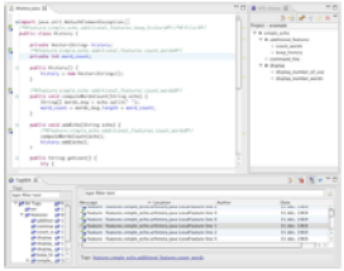
variable model and code assets



pruning,
composition,
weaving,
transformation
...

configuration

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



product spec

product

Feature model in the SPL framework

- Not restricted to document/communicate
- Product/model derivation
  - reminder of previous course
- Safe composition and automated reasoning
- Generation of Configurators

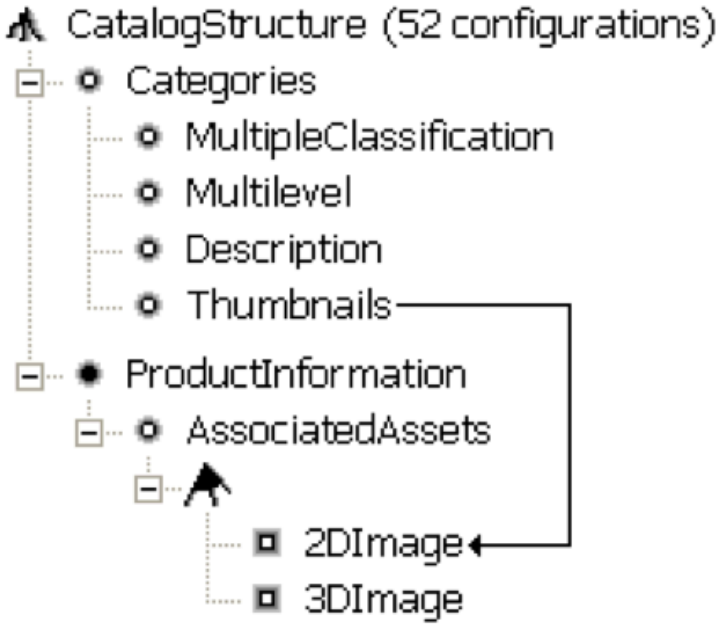
Feature Models

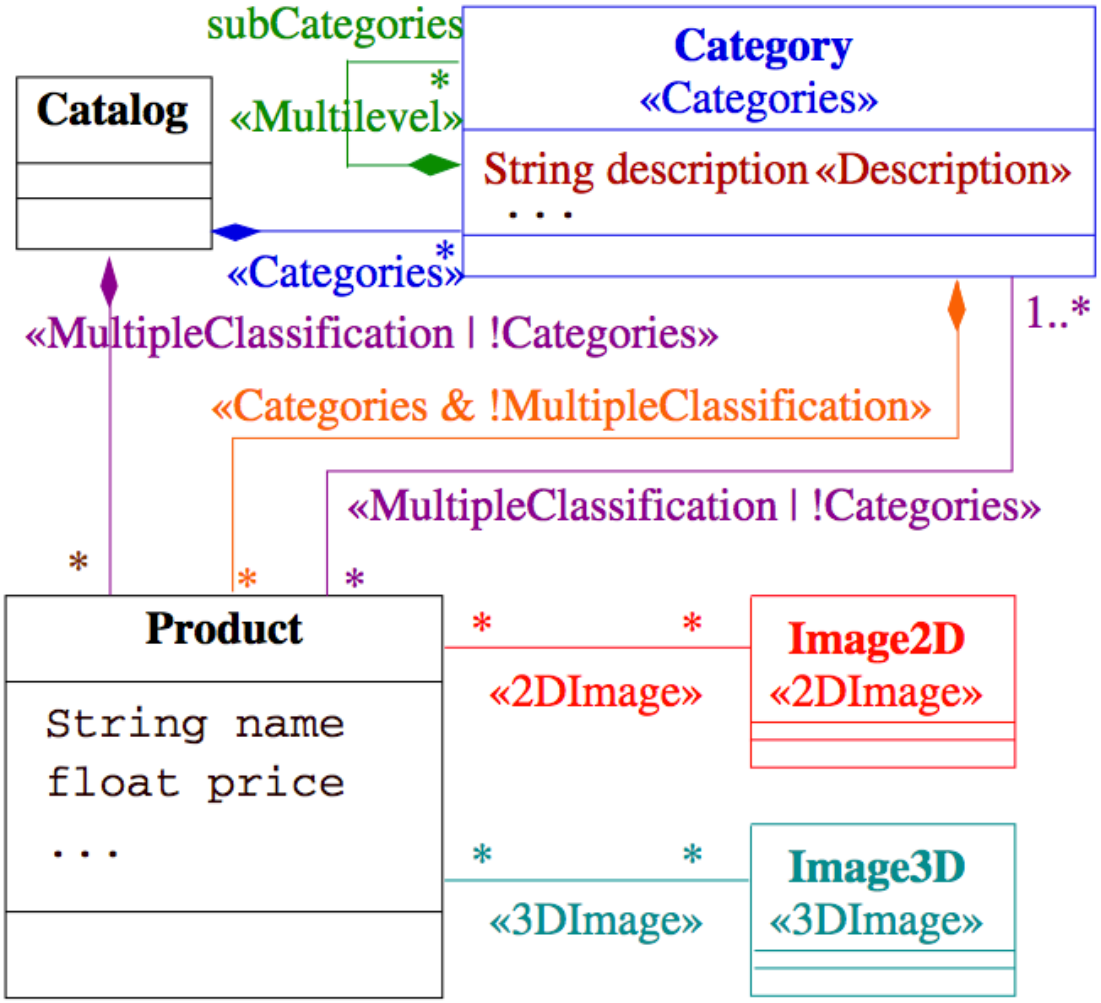
+

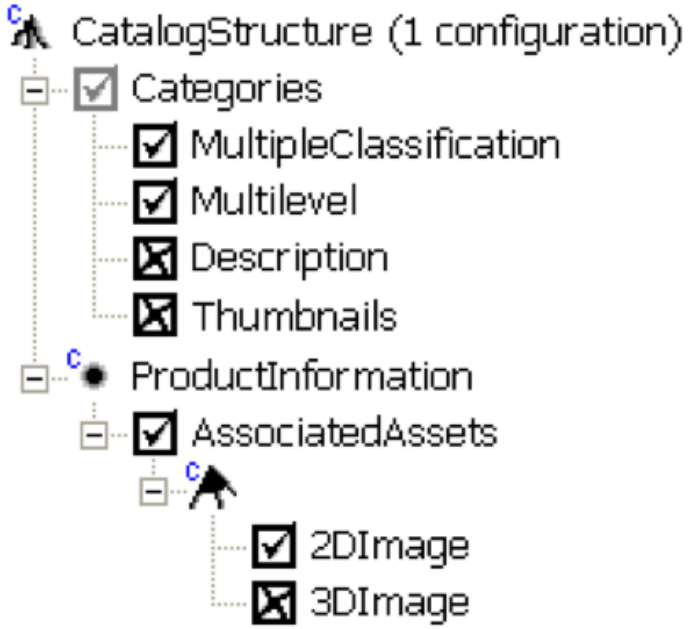
other artefacs

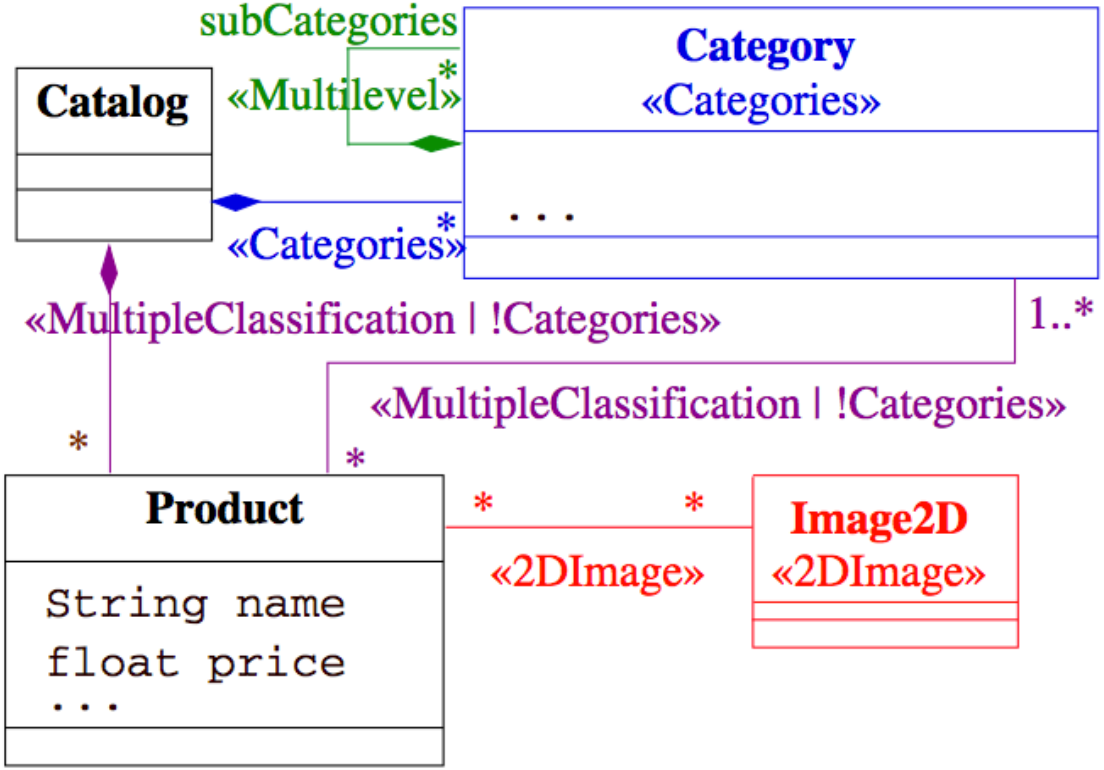
(what we can do with feature models)

“features” in feature models: a label

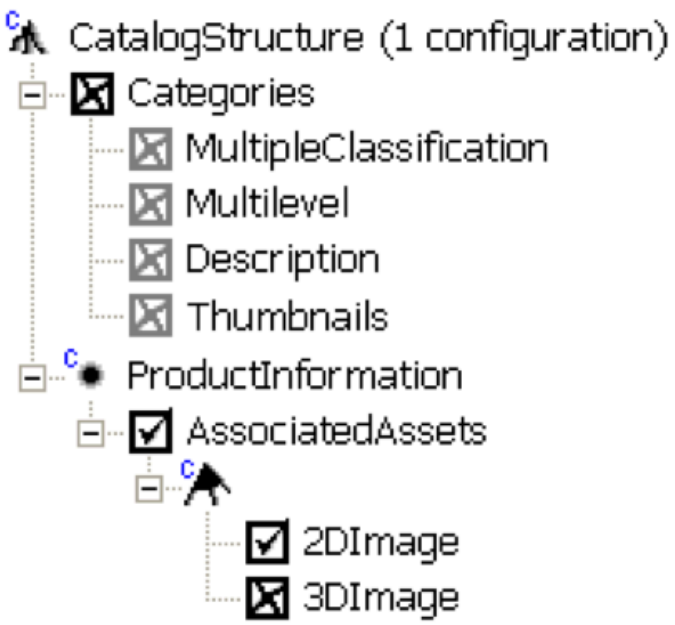

CatalogStructure (52 configurations)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage

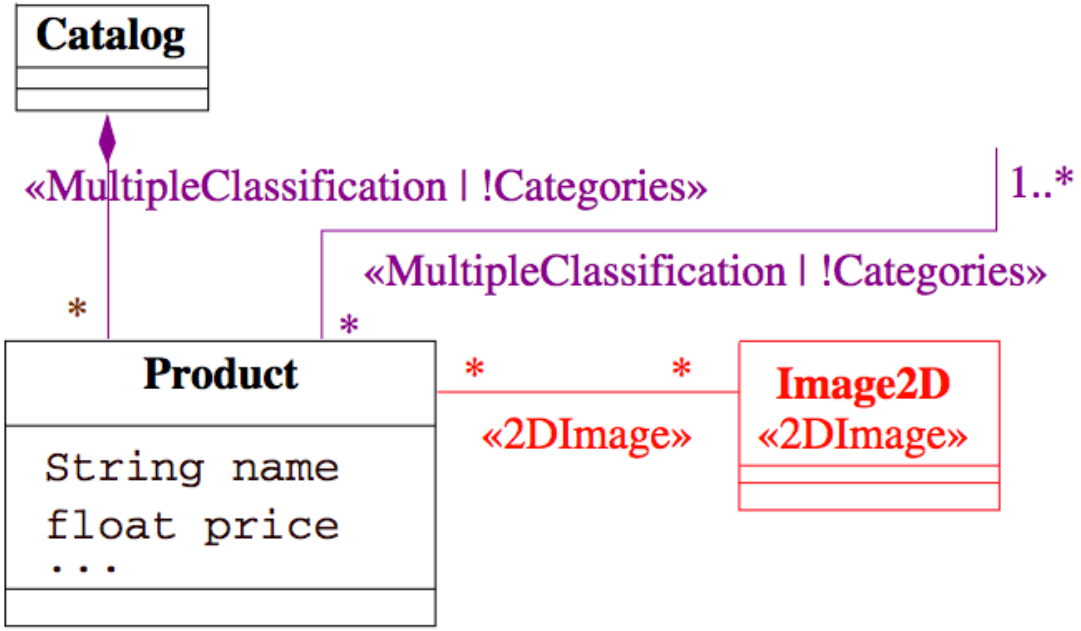

subCategories
Catalog
«Multilevel»
Category
«Categories»
String description «Description»
. . .
«Categories»
«MultipleClassification | !Categories»
1..*
«Categories & !MultipleClassification»
«MultipleClassification | !Categories»
Product
String name
float price
. . .
«2DImage»
Image2D
«2DImage»
«3DImage»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


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

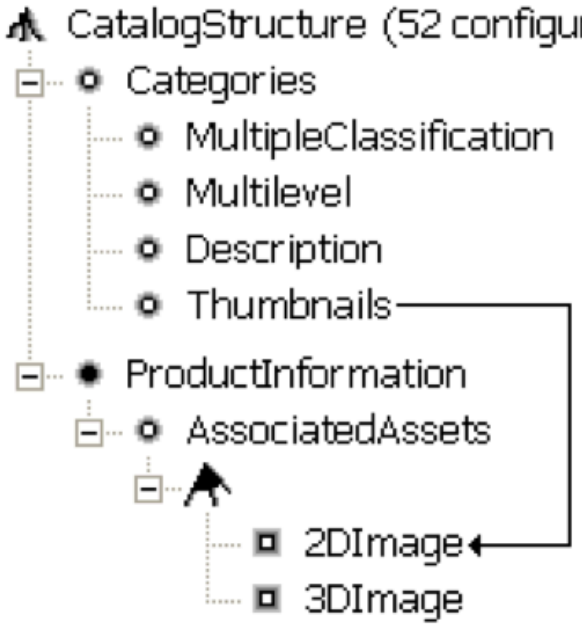

subCategories
Catalog
«Multilevel»
Category
«Categories»
...
«Categories»
«MultipleClassification | !Categories»
1..*
«MultipleClassification | !Categories»
Product
String name
float price
...
«2DImage»
Image2D
«2DImage»

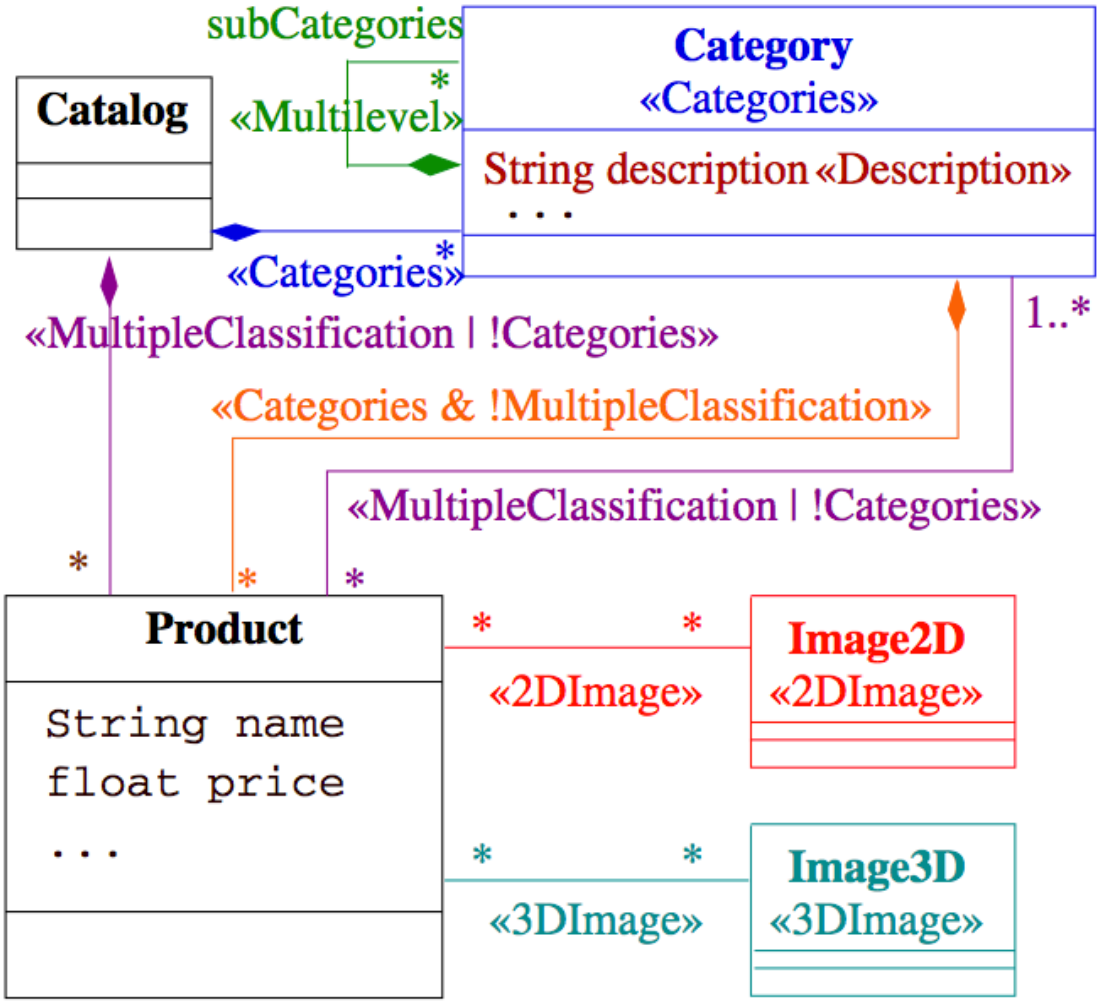
Ooops


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

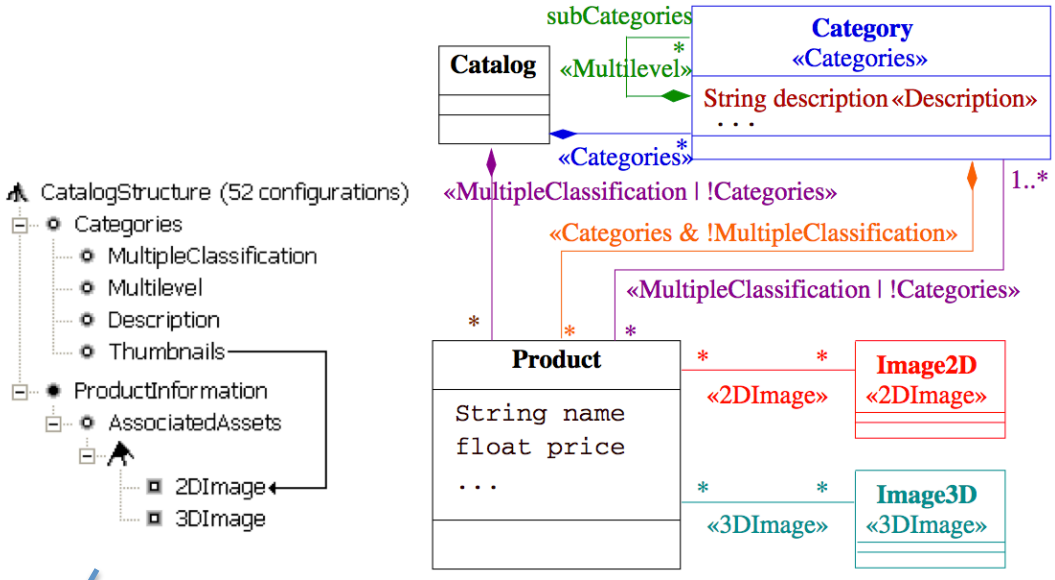

Catalog
«MultipleClassification | !Categories»
1..*
«MultipleClassification | !Categories»
*
*
Product
String name
float price
...
*
*
«2DImage»
Image2D
«2DImage»

Safe composition? No!





Safe composition: how does it work?



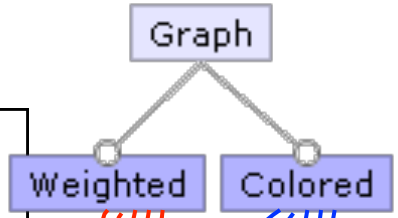
$$
\begin{aligned}
q_{FM} \quad &= \\
\textit{root:} \quad & cs \wedge \\
\textit{child-parent:} \quad & (\texttt{ct} \Rightarrow \texttt{cs}) \wedge (\texttt{mc} \Rightarrow \texttt{ct}) \wedge (\texttt{ml} \Rightarrow \texttt{ct}) \wedge \\
& (\texttt{ds} \Rightarrow \texttt{ct}) \wedge (\texttt{tn}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

Another propositional formula

Mapping: an example

```
class Graph {
  Vector nv = new Vector(); Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++) {
      ((Edge)ev.get(i)).print();
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```




```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;= new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    Color.setDisplayColor(color);
    a.print(); b.print();
    weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Tag and prune

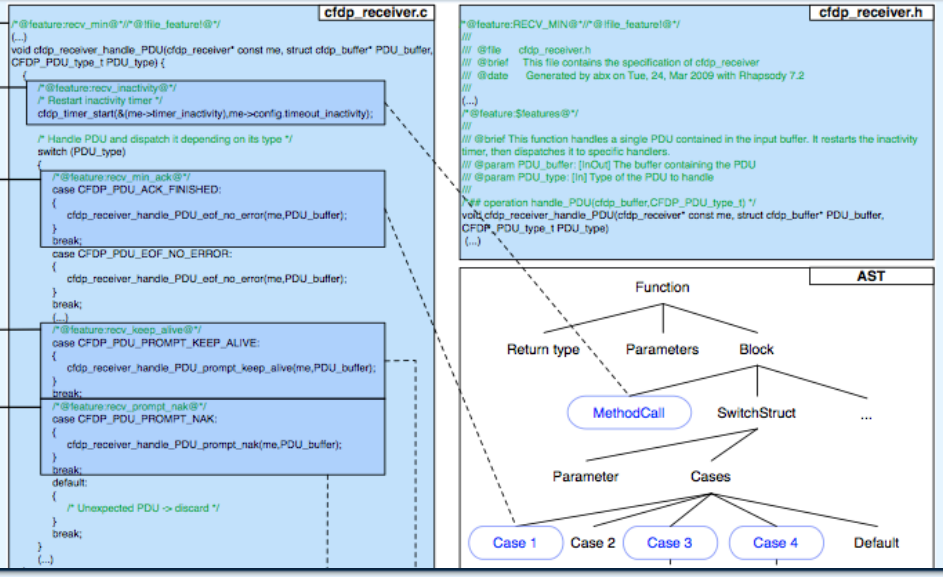
Tag: annotate blocks of code with *features*

Prune: remove blocks tagged with non-selected features

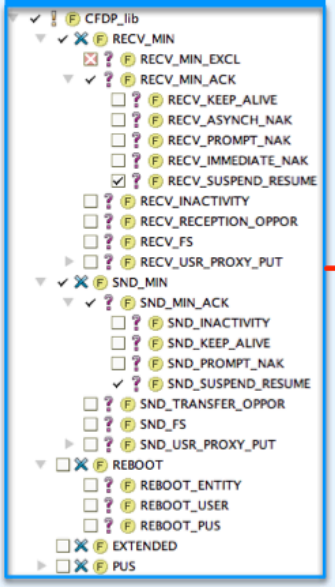
Tagging a whole file with feature *recv_min*

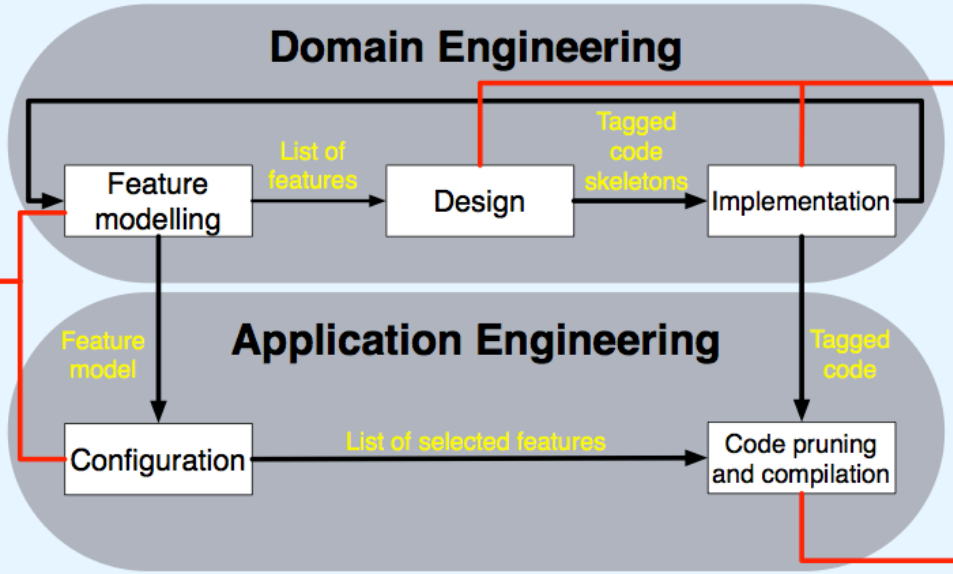
Tagging a function call with feature *recv_inactivity*

Tagging case blocks

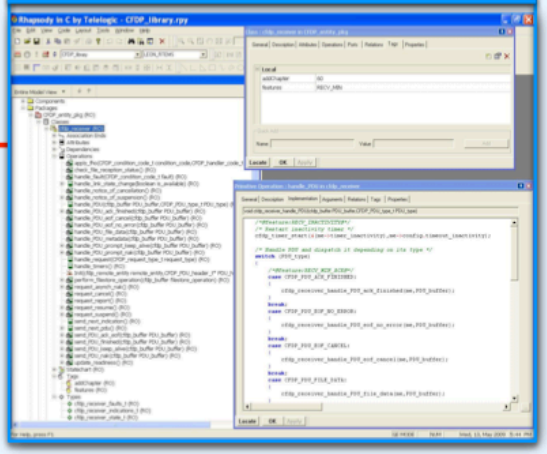


Feature Modelling Tool





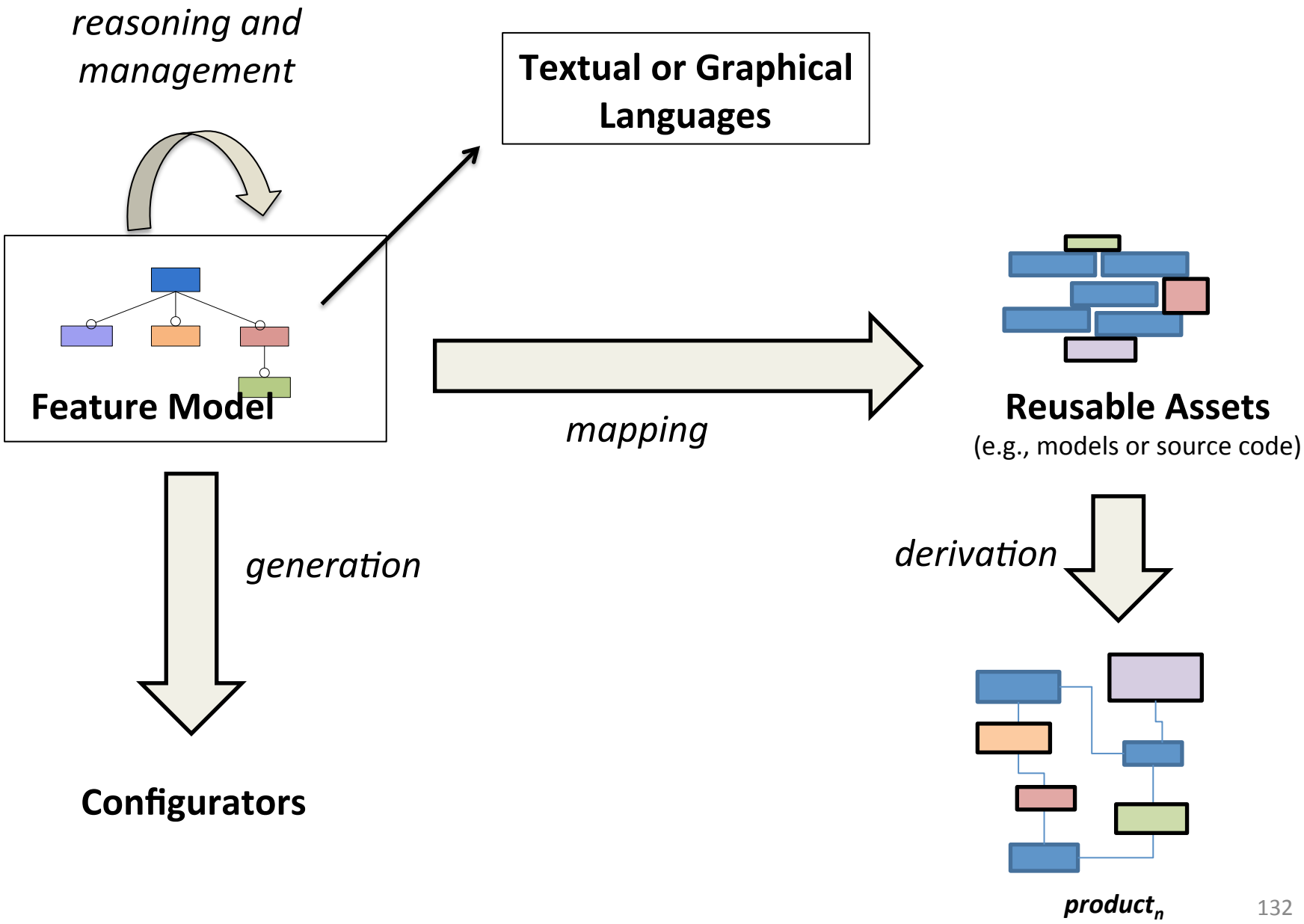
UML Modelling Tool



Source Code Pruner

Summary

- Variability models for mastering complexity: variability is everywhere, precise specification, automated reasoning and management
- Feature Models
  - **Formalism and theory**: modeling principles, semantics, logics
  - **Language** to elaborate/build variability models
  - **Reasoning techniques** to analyze properties of an SPL (scalable and automated way)
  - **Applications**: variability models and so what?


reasoning and management
Textual or Graphical Languages
Feature Model
mapping
Reusable Assets
(e.g., models or source code)
generation
derivation
Configurators
product$_n$

How can I use automated analysis in practice?

Like this?

```
// Feature model path
String featureModelPath = "/Users/mathieuacher/Documents/workspaceScala/SPLAR/resources/models/simple_bike_fm.xml";

// Create feature model object from an XML file (SXFM format - see www.splot-research.org for details)
// If an identifier is not provided for a feature use the feature name as id
FeatureModel featureModel = new XMLFeatureModel(featureModelPath, XMLFeatureModel.USE_VARIABLE_NAME_AS_ID);
// load feature model from
featureModel.loadModel();

// create BDD variable order heuristic
new FTPreOrderSortedECTraversalHeuristic("Pre-CL-MinSpan", featureModel, FTPreOrderSortedECTraversalHeuristic.FORCE_SORT);
VariableOrderingHeuristic heuristic = VariableOrderingHeuristicsManager.createHeuristicsManager().getHeuristic("Pre-CL-MinSpan");

// BDD construction parameters
// - Tuning this parameters can be tricky at times and may require playing a bit
// - For the purpose of this example let's assume "large enough" values
int bddNodeNum = 10000;     // sets the initial size of the BDD table
int bddCacheSize = 10000;  // sets the size of the BDD cache table

// Creates the BDD reasoner
ReasoningWithBDD reasoner = new FMReasoningWithBDD(featureModel, heuristic, 50000, 50000, 60000, "pre-order");

// Initialize the reasoner (BDD is created at this moment)
reasoner.init();

// Use the reasoner
System.out.println("BDD has " + reasoner.getBDD().nodeCount() + " nodes and was built in " + reasoner.getBDDBuildingTime() + " ms");

// Check if feature model is consistent, i.e., has at least one valid configuration
System.out.println("Feature model is " + (reasoner.isConsistent()? "" : " NOT ") + "consistent!");

// Count feature model solutions
System.out.println("Feature model has " + reasoner.countValidConfigurations() + " possible configurations");
```

No. Like this!

```
fm1 = FM ("fm1.tvl") // many other formats
b1 = isValid fm1 // check consistency
n1 = counting fm1 // count the number of valid configurations
```

```
// Feature model path
String featureModelPath = "/Users/mathieuacher/Documents/workspaceScala/SPLAR/resources/models/simple_bike_fm.xml";

// Create feature model object from an XML file (SXFM format - see www.splot-research.org for details)
// If an identifier is not provided for a feature use the feature name as id
FeatureModel featureModel = new XMLFeatureModel(featureModelPath, XMLFeatureModel.USE_VARIABLE_NAME_AS_ID);
// load feature model from
featureModel.loadModel();

// create BDD variable order heuristic
new FTPreOrderSortedECTraversalHeuristic("Pre-CL-MinSpan", featureModel, FTPreOrderSortedECTraversalHeuristic.FORCE_SORT);
VariableOrderingHeuristic heuristic = VariableOrderingHeuristicsManager.createHeuristicsManager().getHeuristic("Pre-CL-MinSpan");

// BDD construction parameters
// - Tuning this parameters can be tricky at times and may require playing a bit
// - For the purpose of this example let's assume "large enough" values
int bddNodeNum = 10000;     // sets the initial size of the BDD table
int bddCacheSize = 10000;  // sets the size of the BDD cache table

// Creates the BDD reasoner
ReasoningWithBDD reasoner = new FMReasoningWithBDD(featureModel, heuristic, 50000, 50000, 60000, "pre-order");

// Initialize the reasoner (BDD is created at this moment)
reasoner.init();

// Use the reasoner
System.out.println("BDD has " + reasoner.getBDD().nodeCount() + " nodes and was built in " + reasoner.getBDDBuildingTime() + " ms");

// Check if feature model is consistent, i.e., has at least one valid configuration
System.out.println("Feature model is " + (reasoner.isConsistent()? "" : " NOT ") + "consistent!");

// Count feature model solutions
System.out.println("Feature model has " + reasoner.countValidConfigurations() + " possible configurations");
```

FAMILIAR: domain-specific language

```
fm1 = FM ("fm1.tvl") // many other formats
b1 = isValid fm1 // check consistency
n1 = counting fm1 // count the number of valid configurations
```

more concise and readable
no need to learn a (set of) API
interoperability facilities

```
// Feature model path
String featureModelPath = "/Users/mathieuacher/Documents/workspaceScala/SPLAR/resources/models/simple_bike_fm.xml";

// Create feature model object from a XML file (SXFM format - see www.splot-research.org for details)
// If an identifier is not provided for a feature use the feature name as id
FeatureModel featureModel = new XMLFeatureModel(featureModelPath, XMLFeatureModel.USE_VARIABLE_NAME_AS_ID);
// load feature model from
featureModel.loadModel();

// create BDD variable order heuristic
new FTPreOrderSortedECTraversalHeuristic("Pre-CL-MinSpan", featureModel, FTPreOrderSortedECTraversalHeuristic.FORCE_SORT);
VariableOrderingHeuristic heuristic = VariableOrderingHeuristicsManager.createHeuristicsManager().getHeuristic("Pre-CL-MinSpan");

// BDD construction parameters
// - Tuning this parameters can be tricky at times and may require playing a bit
// - For the purpose of this example let's assume "large enough" values
int bddNodeNum = 10000;     // sets the initial size of the BDD table
int bddCacheSize = 10000;  // sets the size of the BDD cache table

// Creates the BDD reasoner
ReasoningWithBDD reasoner = new FMReasoningWithBDD(featureModel, heuristic, 50000, 50000, 60000, "pre-order");

// Initialize the reasoner (BDD is created at this moment)
reasoner.init();

// Use the reasoner
System.out.println("BDD has " + reasoner.getBDD().nodeCount() + " nodes and was built in " + reasoner.getBDDBuildingTime() + " ms");

// Check if feature model is consistent, i.e., has at least one valid configuration
System.out.println("Feature model is " + (reasoner.isConsistent()? "" : " NOT ") + "consistent!");

// Count feature model solutions
System.out.println("Feature model has " + reasoner.countValidConfigurations() + " possible configurations");
```

See FAMILIAR