


# Software Product Line Engineering

## modeling and managing variability of software intensive systems


Dr. Mathieu Acher

email: macher@fundp.ac.be

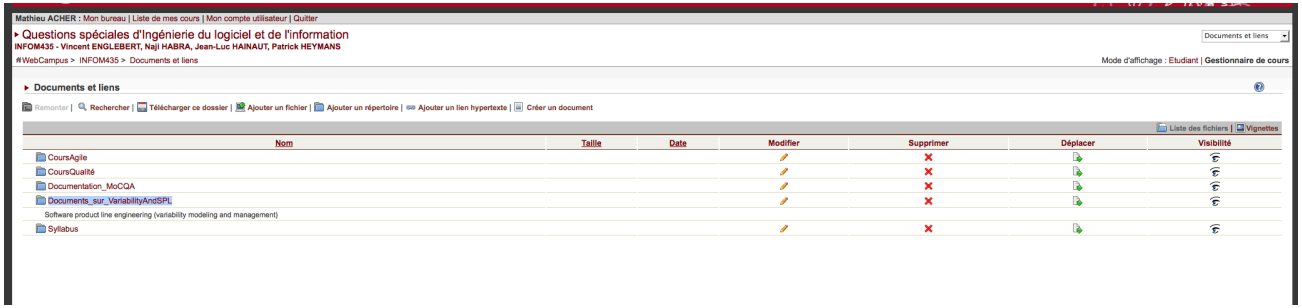
Prof. Patrick Heymans

University of Namur

PReCISE Research Centre

# Material

- http://www.fundp.ac.be/etudes/cours/page_view/INFOM435/
  - Folder: “Documents_sur_VariabilityAndSPL”
  - Slides, exercises, evaluation




- Email: macher@fundp.ac.be

I reuse some material from other colleagues (Czarnecki, Kästner, etc.)

I try to mention references as much as possible

(you can find/read them on the web or simply ask me!)

***[Czarnecki et al. (GPCE'05)]***

# Previously

- **Domain engineering pays off**
  - software product line engineering changes the way software should be developed
  - reuse-in-the-large works best in families of related systems, and thus is domain dependent




- **Application engineering**
- **Generative, model-based approaches**
  - customized products automatically generated from specifications (models)

Today: **Feature Models**

defacto standard for modeling variability

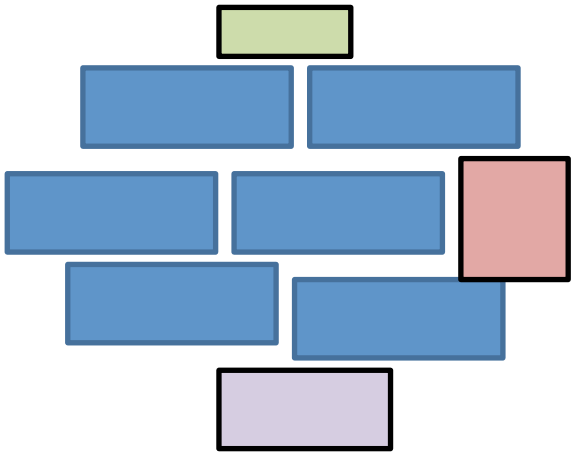
# Today

- **Variability modeling**
- **Feature models**
  - What's in a feature?
  - Rationale
  - Syntax and semantics
  - Overview of automated techniques
- **Feature model in the SPL framework**
  - not restricted to document/communicate
  - safe composition
  - configurators
  - product/model derivation

# Software Product Line Engineering

Factoring out **commonalities**

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

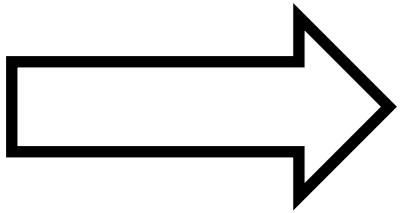


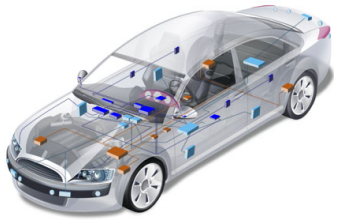
Managing **variabilities**

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

# Software product line engineering

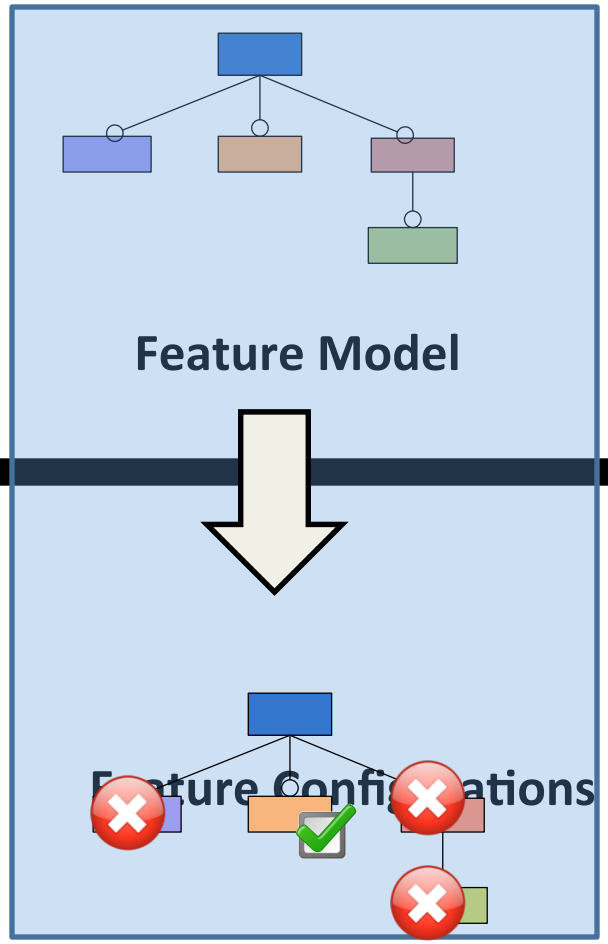
# = modeling and managing variability





# Variability Models: Why?

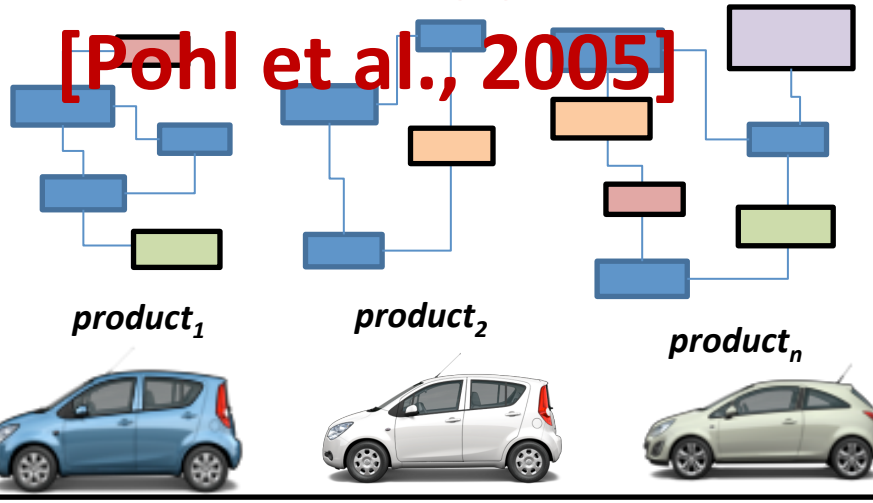
# Domain engineering (development for reuse)





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

"central to the software product line paradigm is the modeling and management of variability, that is, the commonalities and differences in the applications" [Pohl et al., 2005]




# Application engineering (development with reuse)

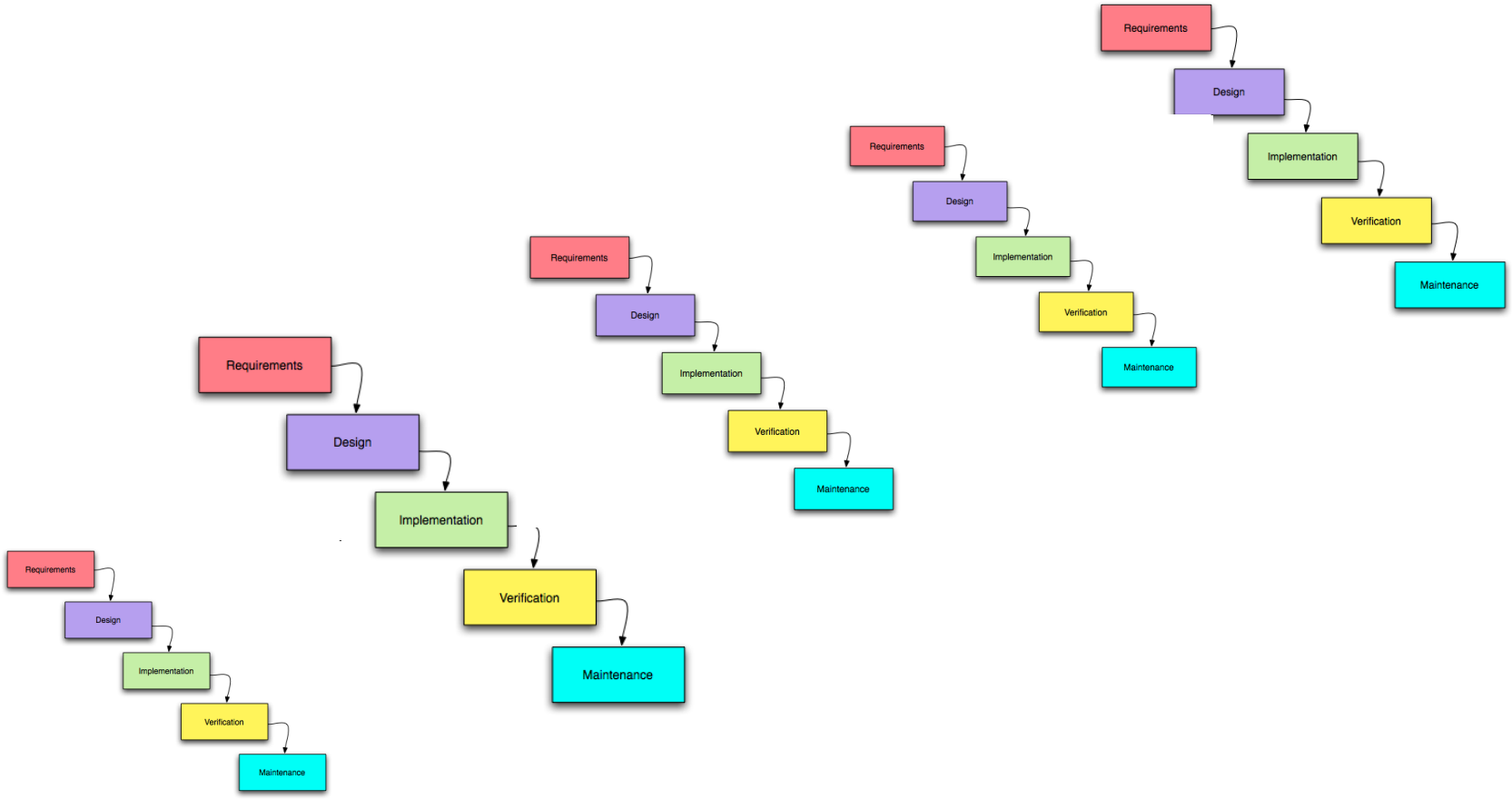
# #1 precise specification is needed


Unused flexibility

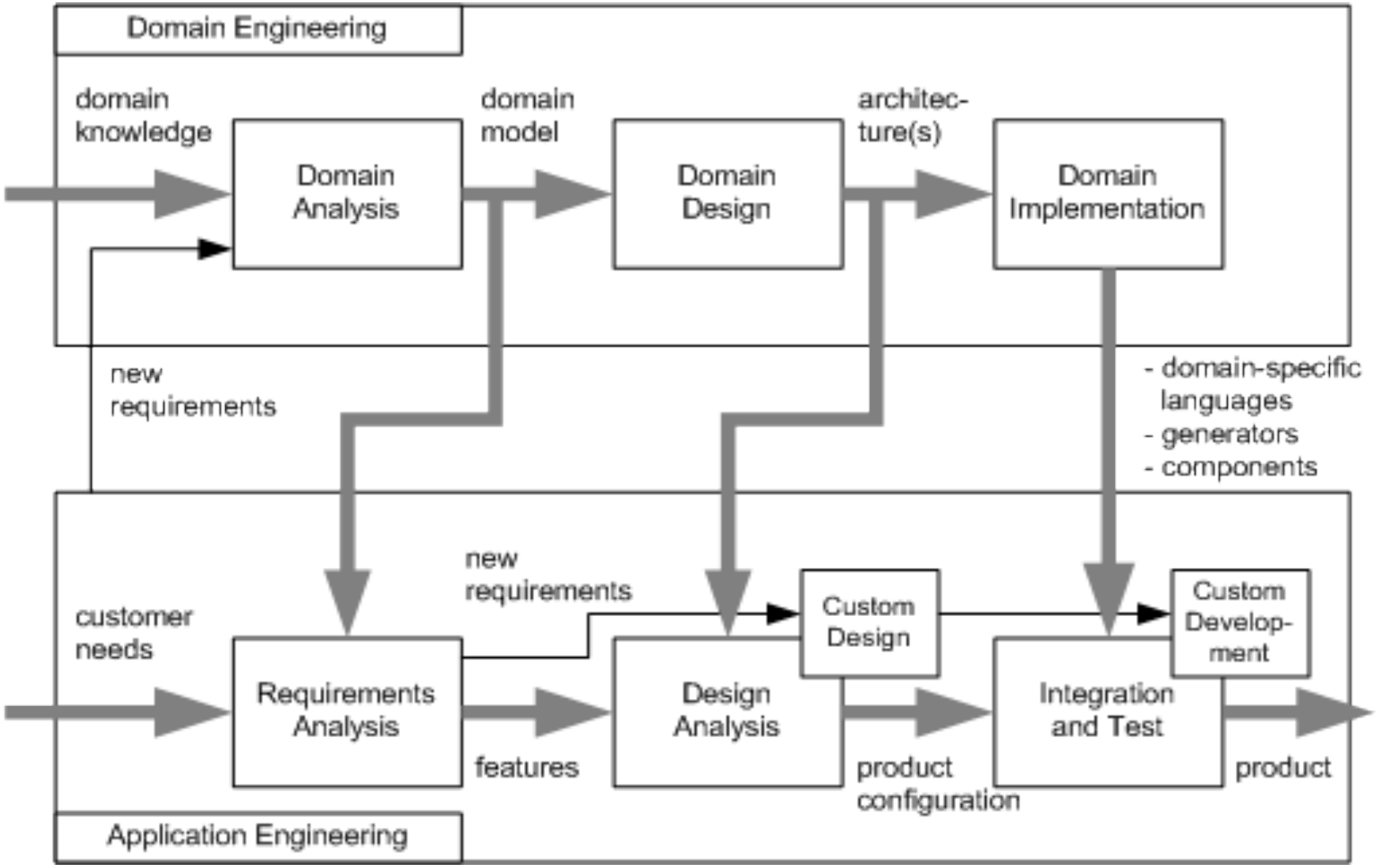

# Illegal Variant

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

# Software Product Line Development?




Time and Effort: not scalable!




*Klaus Pohl (2005)*

# #3 automated reasoning (exponential number of products)

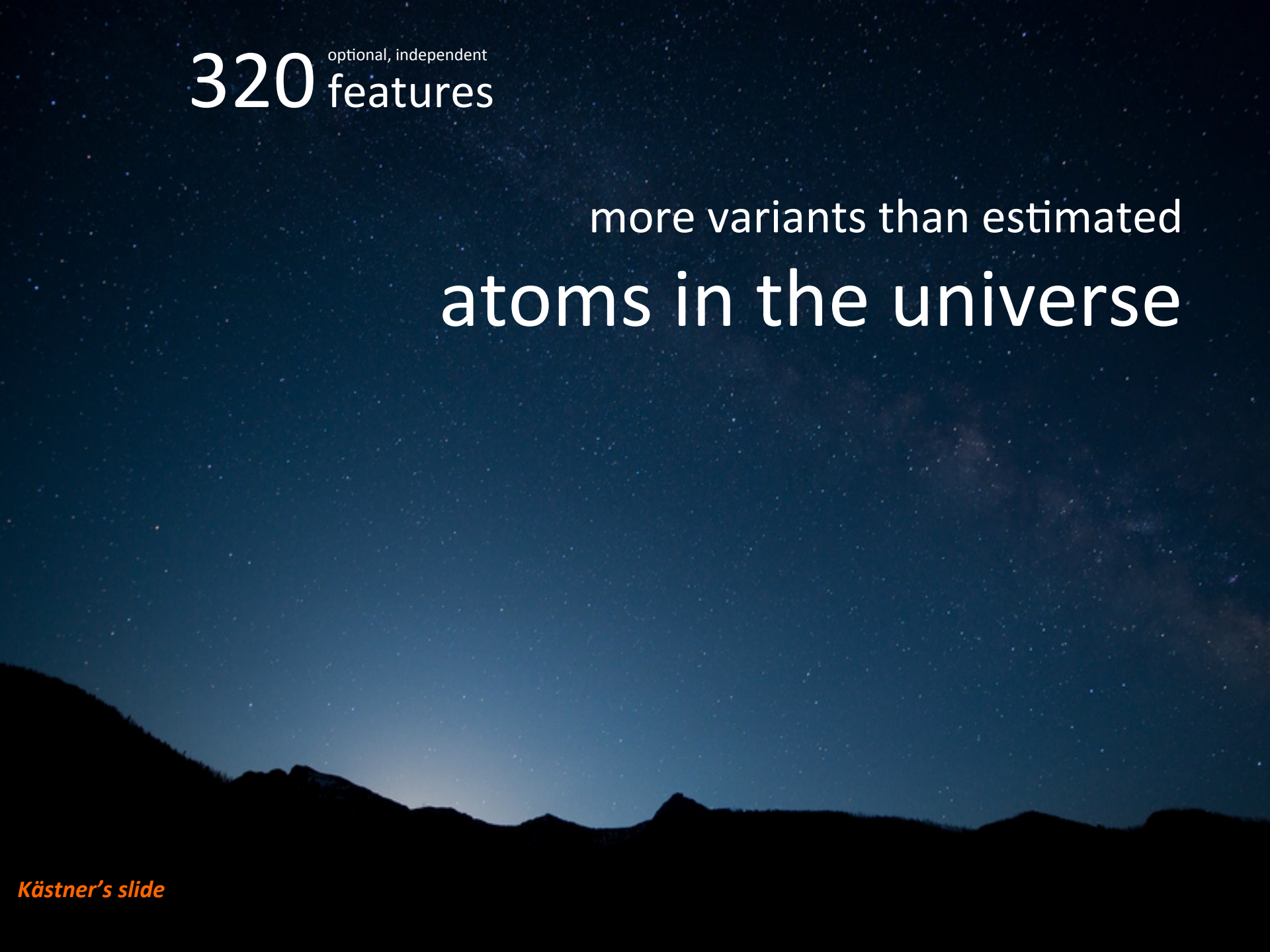

*Kästner's slide*

# 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

# Variability Models: Why?

# 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

# Variability Models: Why?
## (3 arguments, summary)

**precise specification**

**variability is orthogonal / cross cuts different artefacts**

**automated reasoning/analysis**

# 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 Model: de facto standard for modeling variability

- Central to many software product line approaches
  - More than 1000 citations of [Kang et al., 1990] per year
  - Generative programming [Czarnecki et al., 2000]

- Research effort
  - Formal Semantics [Schobbens et al., 2007]
  - Automated Reasoning Techniques [Batory et al., 2005], [Czarnecki et al., 2007], [Mendonca et al., 2009], [Thuem et al., 2009] , [Janota et al., 2010], [Benavides et al., 2010]

$\varphi$

- Tools
  - Commercial: pure::variants, Gears
  - Academic: fmp, FeatureIDE, FaMa, SPLOT, TVL, et

# Feature Models

**#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

**#1 Formalism? Theory?**

**modeling principles, semantics, logics**

# 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
- Feature modeling is a core activity of important 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, e.g. a bird has wings, a bill, and can fly
- • Classifying a perceptional unit as an instance of a concept, e.g. Aunt Martha's bird
- Acquisition and evolution of concepts will not be pursued further in this course

# Modeling variability (and commonality)

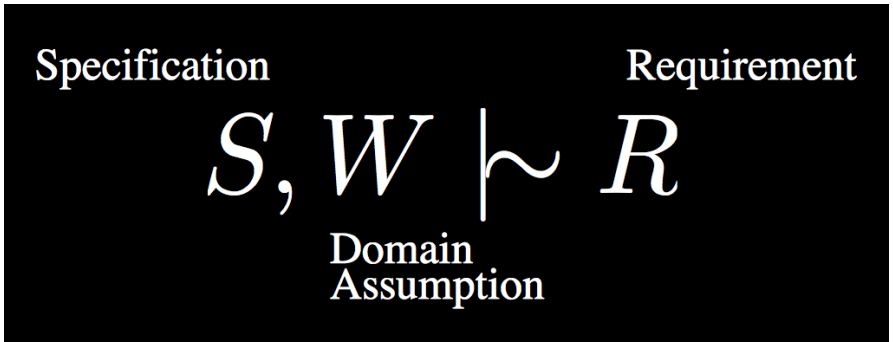
- **Features** are a convenient way to think about commonalities and differences between family members
  - e.g. "MP3 player" – some phones have it, some don't
- Features are usually **coarse-grained** abstractions
  - Abstract from detailed requirements
    - e.g. "when in shuffle mode, the player selects the next track using a random function"
  - Abstract from design/implementation details
    - e.g. "MP3_Player is a concrete subclass of Media_Player"
- An **old** but still **vague** notion

# 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"

# What's in a feature?

- Confusion as to what a feature generally represents (i.e., blackbox view)
- Redefine "Feature" as a 3-tuple of requirements, domain assumptions and specifications
  - Requirements (R)
    - The purpose of the system (optative): '*Notify the police of the presence of burglars*'
  - Domain assumptions (W)
    - The given behaviour of the environment (indicative): *'Burglars cause movement when they break in'* and *'There is a movement sensor monitoring the house, connected to the system'*
  - Specifications (S)
    - What the system should do to satisfy R given W holds (optative): '*Alert the police if the sensor captures movement*'

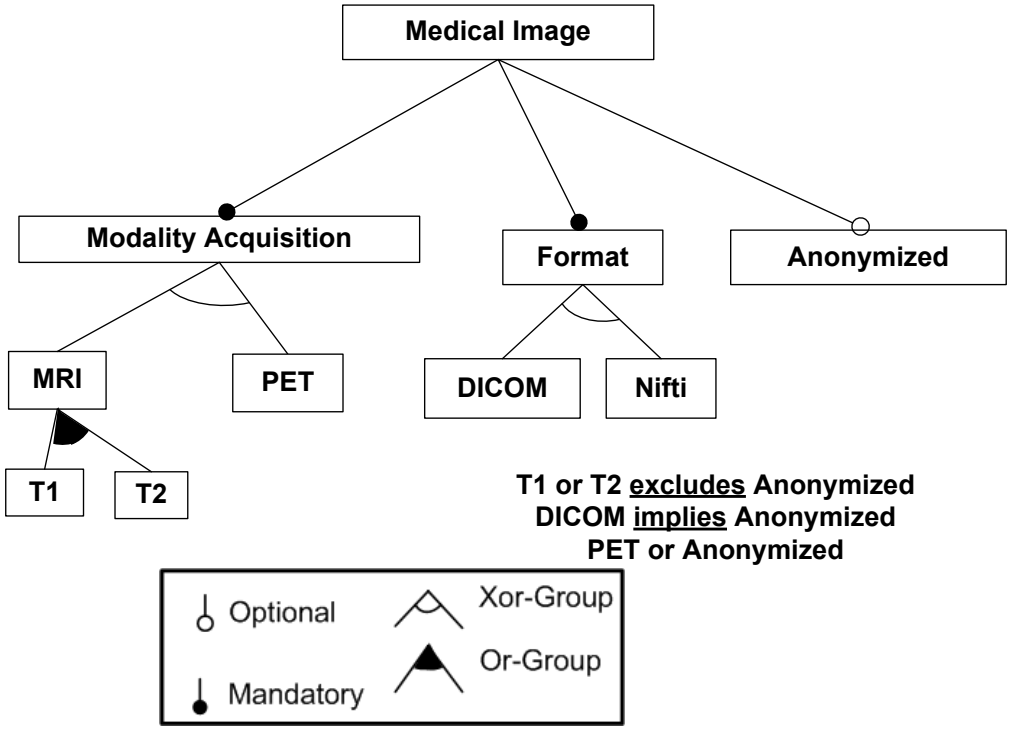


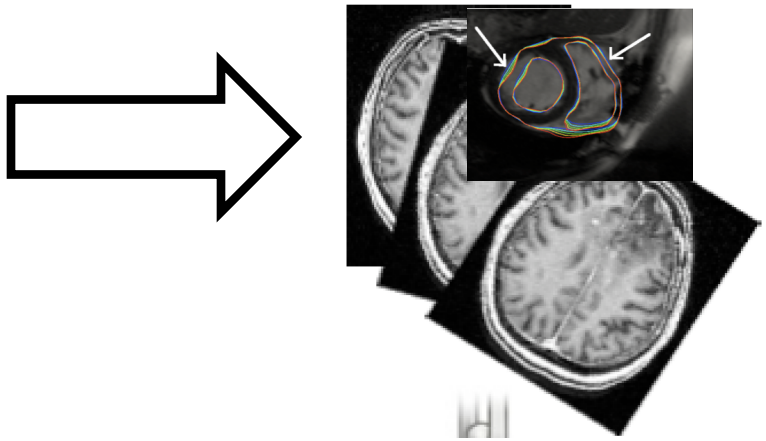
# Feature Models

# Modeling principles, syntax

# Feature Models

describe the common and variable features (characteristics) of a system under study (SUS)



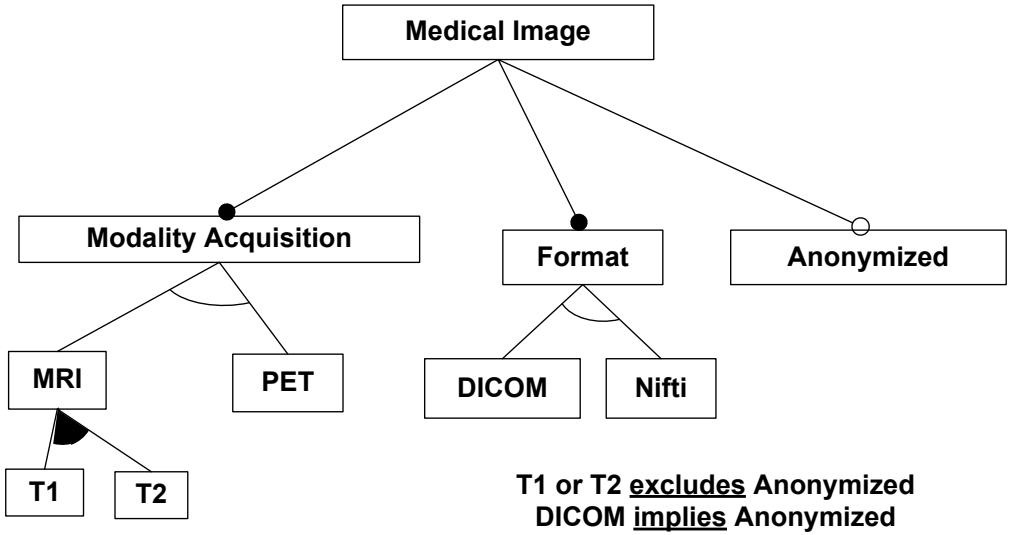



# Feature Models

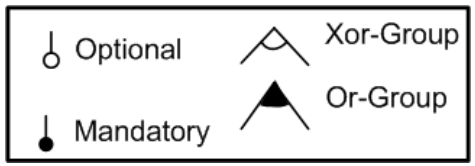
**Hierarchy:** rooted tree

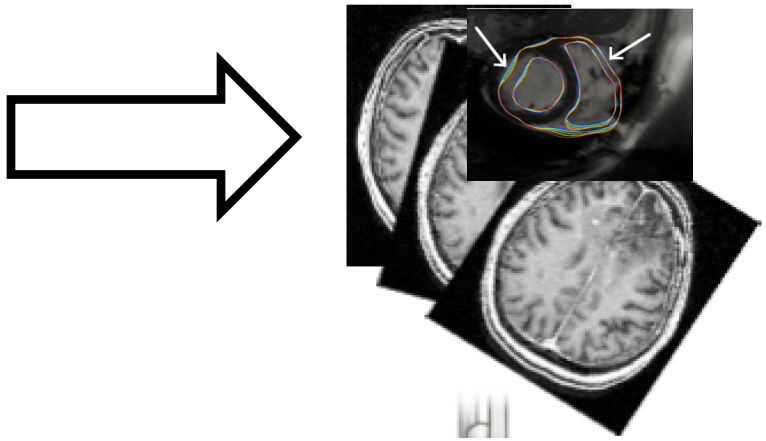
**Variability:**

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




**T1 or T2 excludes Anonymized**
**DICOM implies Anonymized**
**PET or Anonymized**



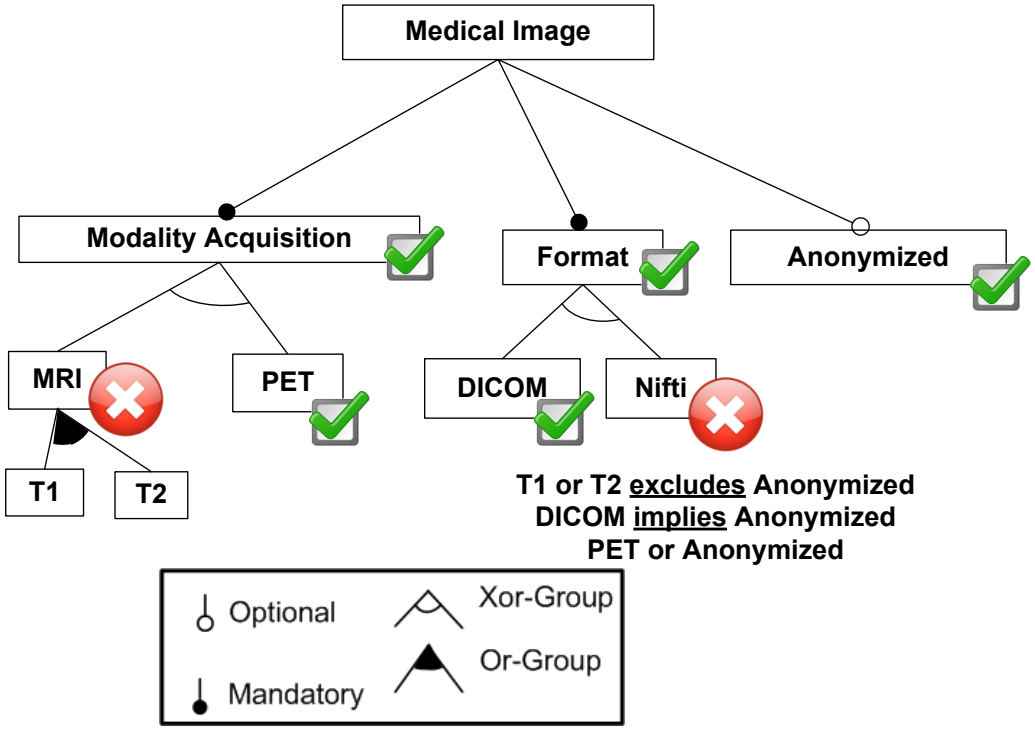



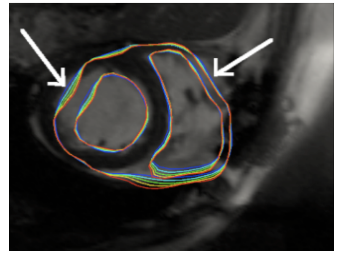
# Feature Models

# Semantics

# Feature Models

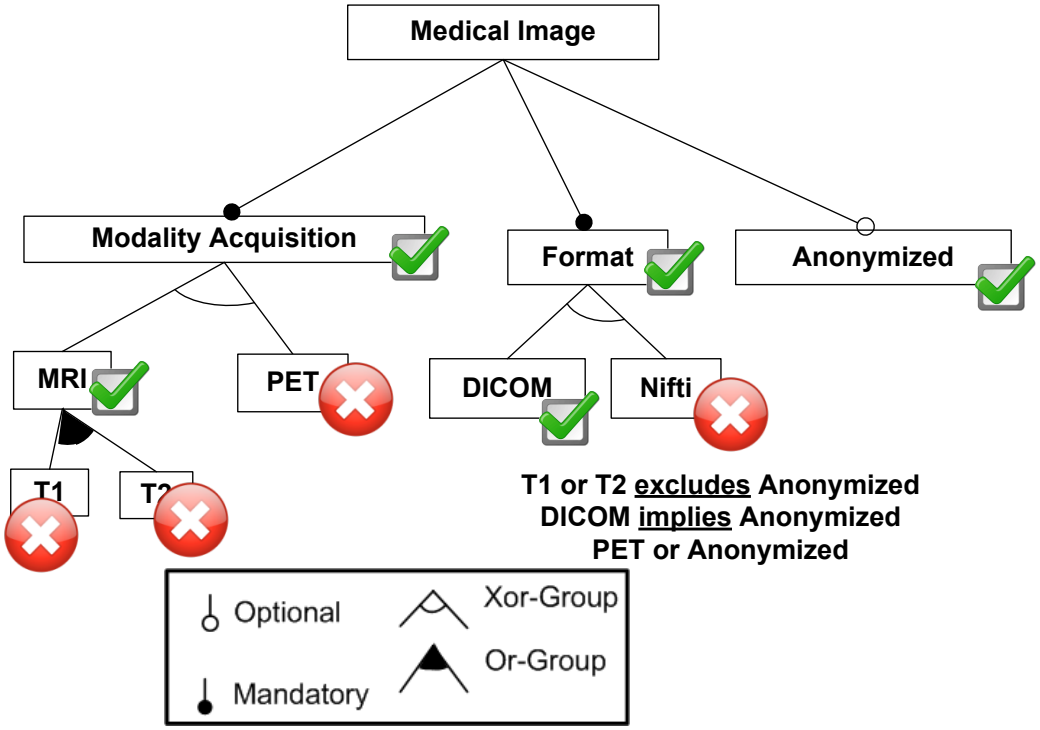
**Hierarchy + Variability = set of valid configurations**



# Feature Models

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

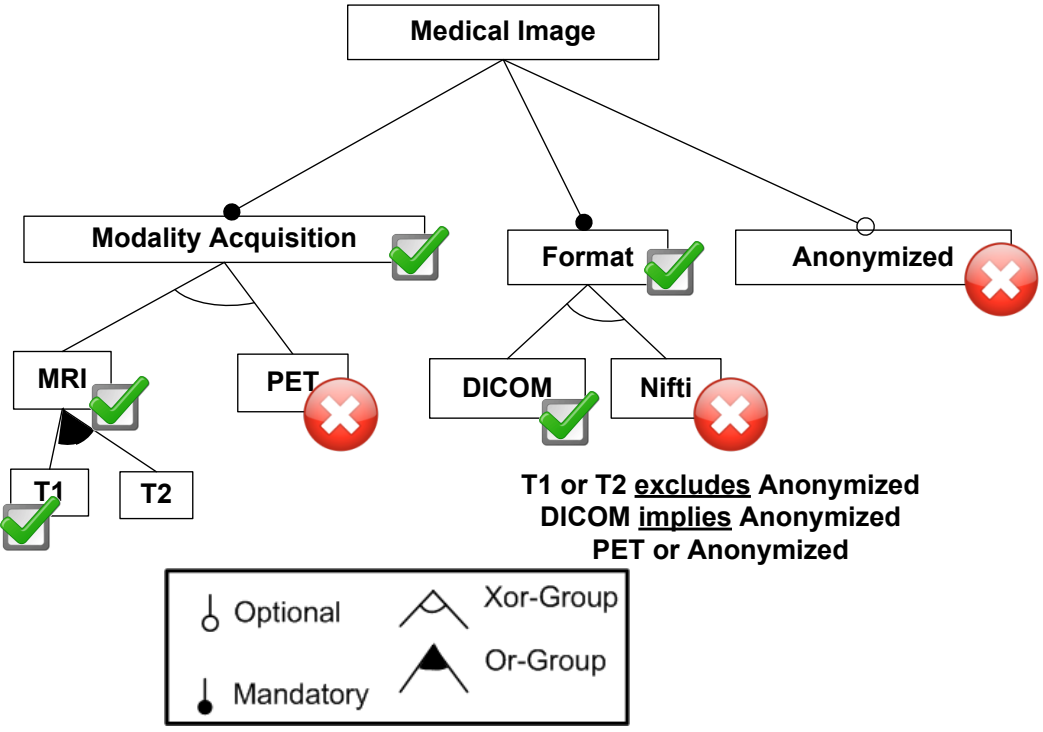



**Illegal configuration**

**see Or-group:**
**at least one feature should be selected**

# Feature Models

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

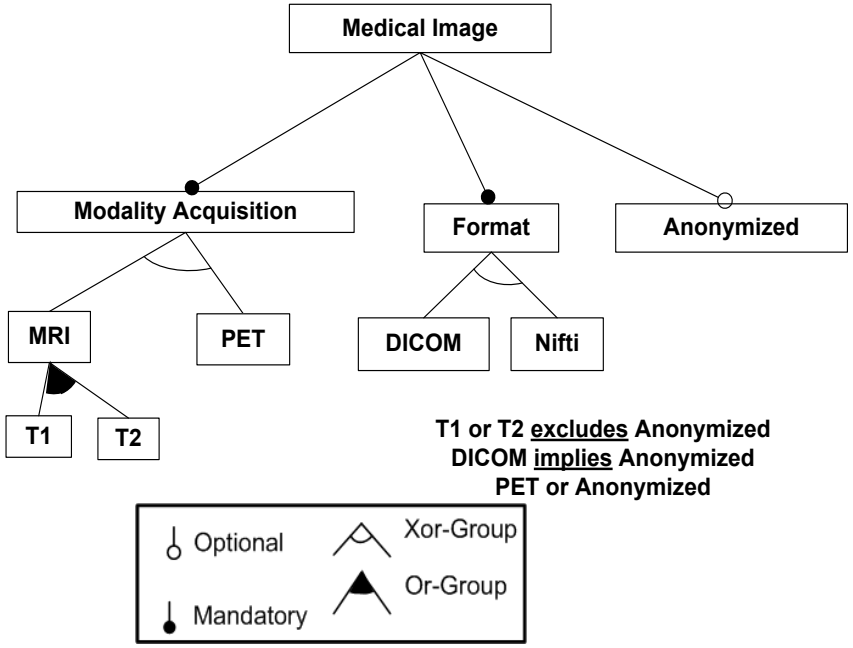



**Illegal configuration**

**see constraint "PET or Anonymized"**

# Feature Models

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




Satisfiability (NP-complete)

Configuration Checking

Dead features detection

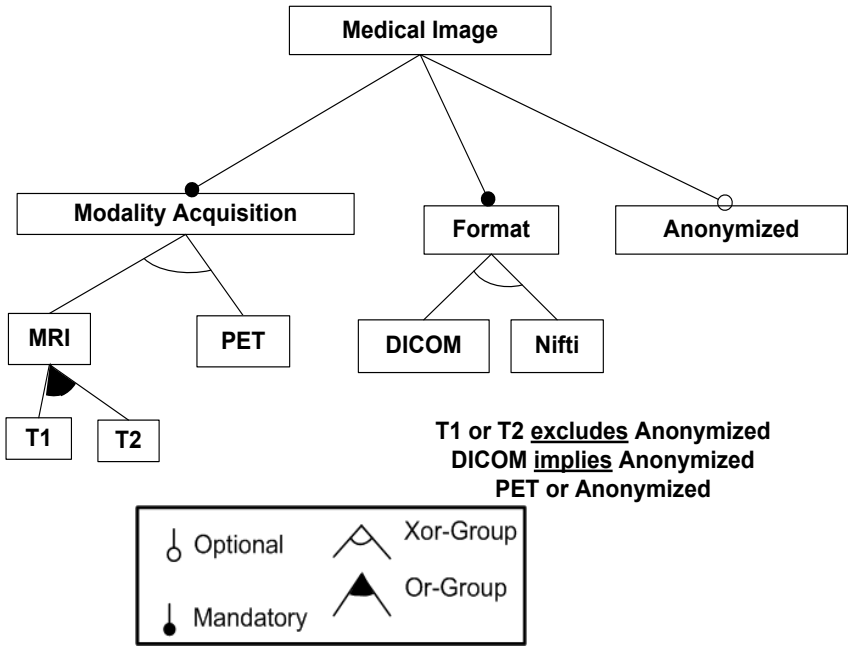
...

**Around 30 reasoning operations**

[Schobbens et al., 2007] [Benavides et al., 2010]

# Propositional Feature Models

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




**Propositional formula** (^, v, ~, ⇔, =>)

**Boolean variables**
**set of valid assignments**

**φ = Medical Image ^**

**Format ⇔ Medical Image^**

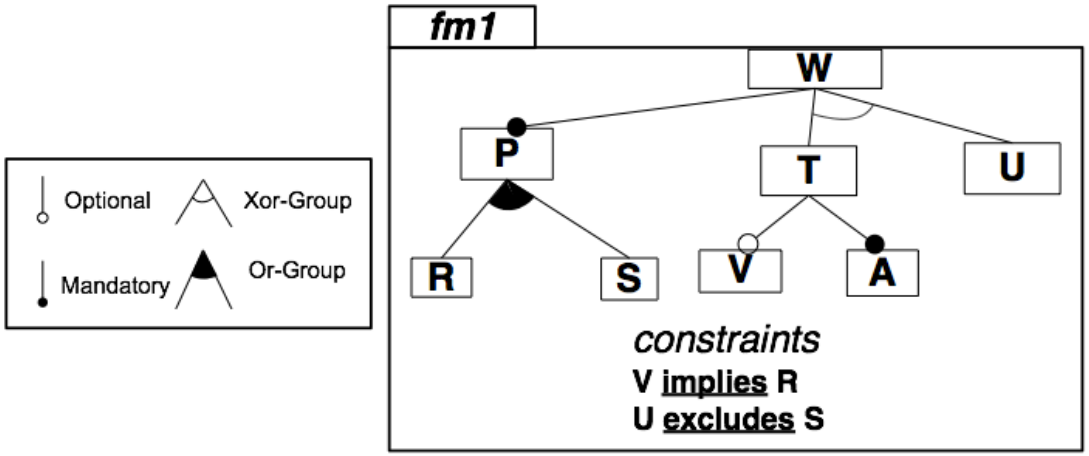
**Anonymized => Medical Image ^**

**…**

[Batory et al., 2005] [Czarnecki et al., 2007]

# Propositional Feature Models

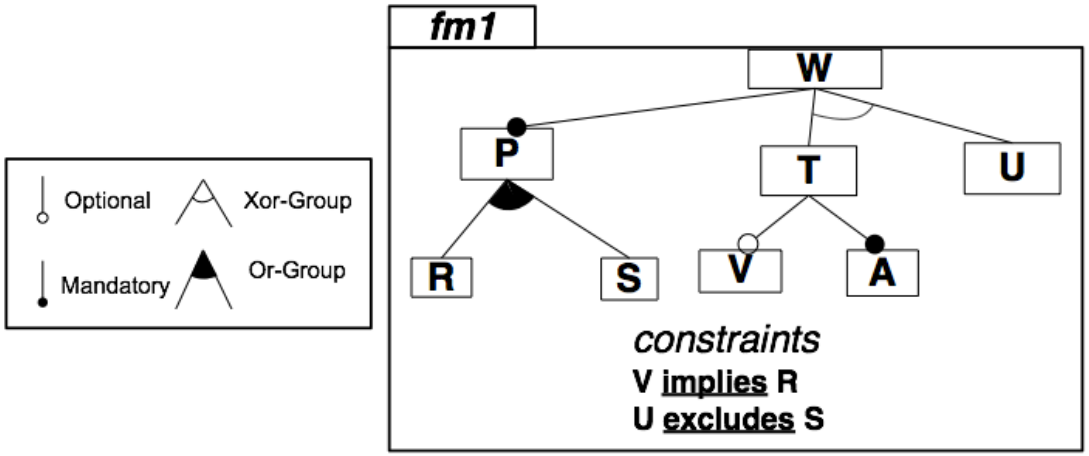
**Hierarchy + Variability = set of valid configurations**




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

# Propositional Feature Models

**~ propositional formula**




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

# FD formalism

★ **Formal semantics** which consists of

- ➢ a **formal syntax** (L) – clearcut syntactic rules defining all legal diagrams, a.k.a. syntactic domain
- ➢ a **semantic domain** (S) – a mathematical abstraction of the real-world concepts to be modelled
- ➢ a **semantic function** (M: L ⟶ S) – clearcut semantic rules defining the meaning of all legal diagrams

*[Harel & Rumpe, IEEE Computer, 2004]*

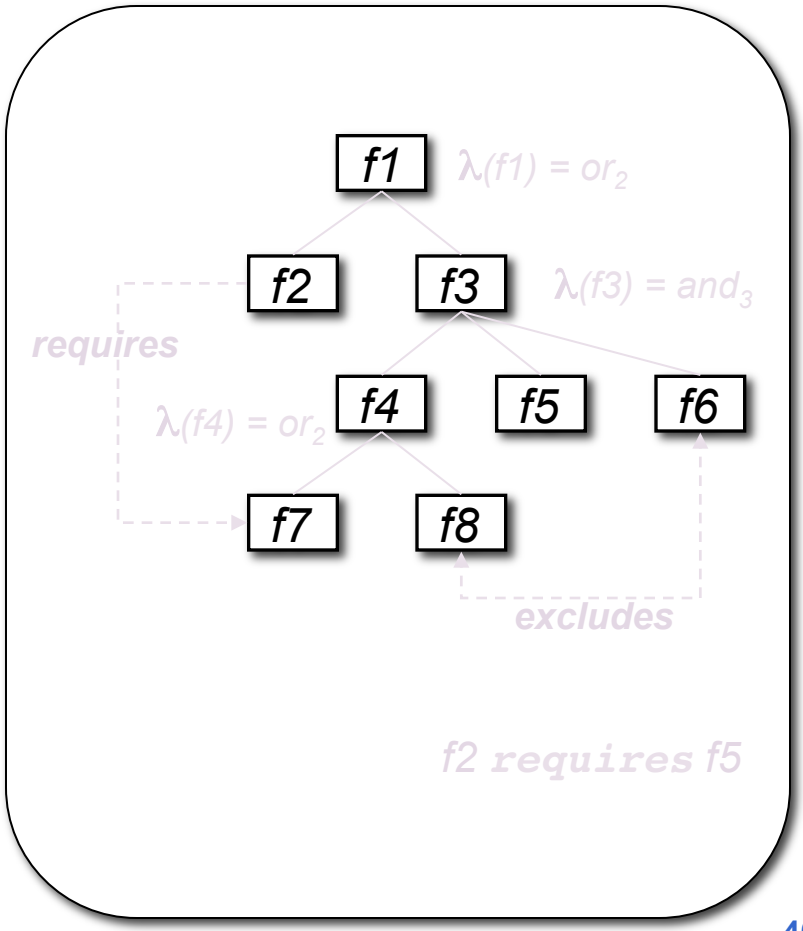


# The $L_{FFD}$ parametric abstract syntax

Every FD **d** $\in$ $\mathbf{L_{FFD}(GT,NT,GCT,TCL)}$ is a tuple **(N,r, λ,DE,CE,Φ)** such that

- ★ **N** *the set of nodes*
- ★ **r** $\in$ **N** *the root*
- ★ **DE** $\subseteq$ **N** × **N** *the decomposition edges (obeying GT)*
- ★ **λ : N→NT** *the function labelling nodes with boolean operators*
- ★ **CE** $\subseteq$ **N** × **GCT** × **N** *the constraint edges*
- ★ **Φ** $\subseteq$ **TCL** *the textual constraints*

- ★ + 4 well-formedness constraints e.g. *only r has no parent*

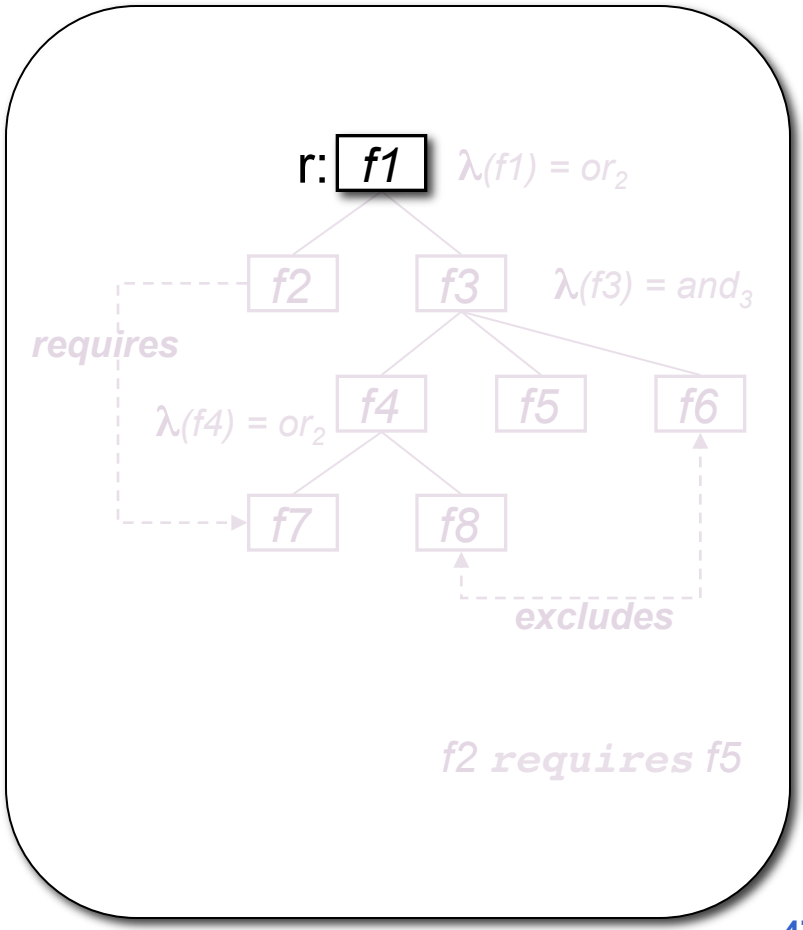


# The $L_{FFD}$ parametric abstract syntax

Every FD **d** $\in$ $\mathbf{L_{FFD}(GT,NT,GCT,TCL)}$ is a tuple **(N,r, λ,DE,CE,Φ)** such that

- ★ **N** *the set of nodes*
- ★ **r** $\in$ **N** *the root*
- ★ **DE** $\subseteq$ **N** × **N** *the decomposition edges (obeying GT)*
- ★ **λ : N→NT** *the function labelling nodes with boolean operators*
- ★ **CE** $\subseteq$ **N** × **GCT** × **N** *the constraint edges*
- ★ **Φ** $\subseteq$ **TCL** *the textual constraints*

- ★ + 4 well-formedness constraints
  e.g. *only r has no parent*

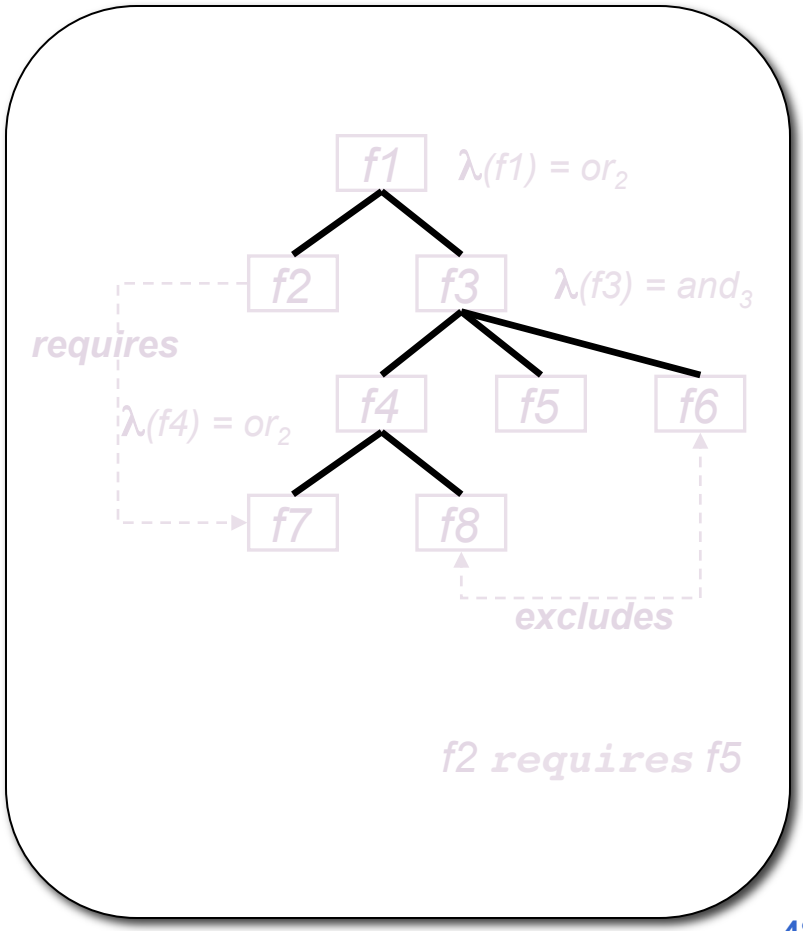


# The $L_{FFD}$ parametric abstract syntax

Every FD **d** $\in$ $\mathbf{L_{FFD}}$**(GT,NT,GCT,TCL)** is a tuple **(N,r, λ,DE,CE,Φ)** such that

- ★ **N** *the set of nodes*
- ★ **r** $\in$ **N** *the root*
- ★ **DE** $\subseteq$ **N** × **N** *the decomposition edges (obeying **GT**)*
- ★ **λ : N→NT** *the function labelling nodes with boolean operators*
- ★ **CE** $\subseteq$ **N** × **GCT** × **N** *the constraint edges*
- ★ **Φ** $\subseteq$ **TCL** *the textual constraints*

- ★ + 4 well-formedness constraints e.g. *only r has no parent*

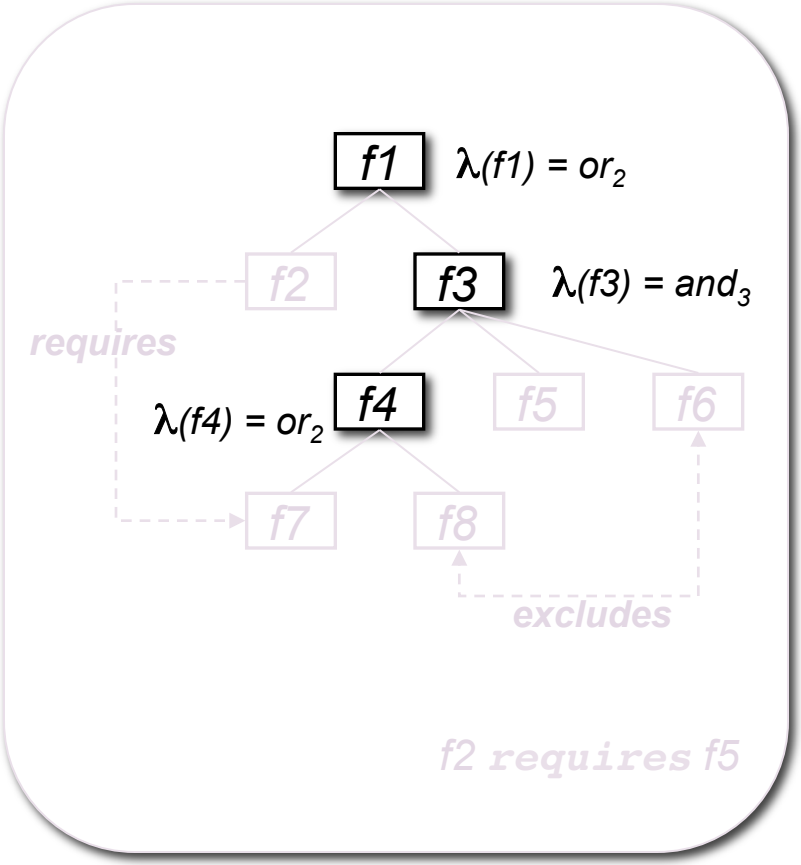


# The $L_{FFD}$ parametric abstract syntax

Every FD **d** $\in$ $\mathbf{L_{FFD}(GT,NT,GCT,TCL)}$ is a tuple **(N,r, λ,DE,CE,Φ)** such that

- ★ **N** *the set of nodes*
- ★ **r** $\in$ **N** *the root*
- ★ **DE** $\subseteq$ **N** × **N** *the decomposition edges (obeying GT)*
- ★ **λ : N→NT** *the function labelling nodes with boolean operators*
- ★ **CE** $\subseteq$ **N** × **GCT** × **N** *the constraint edges*
- ★ **Φ** $\subseteq$ **TCL** *the textual constraints*

- ★ + 4 well-formedness constraints e.g. *only r has no parent*

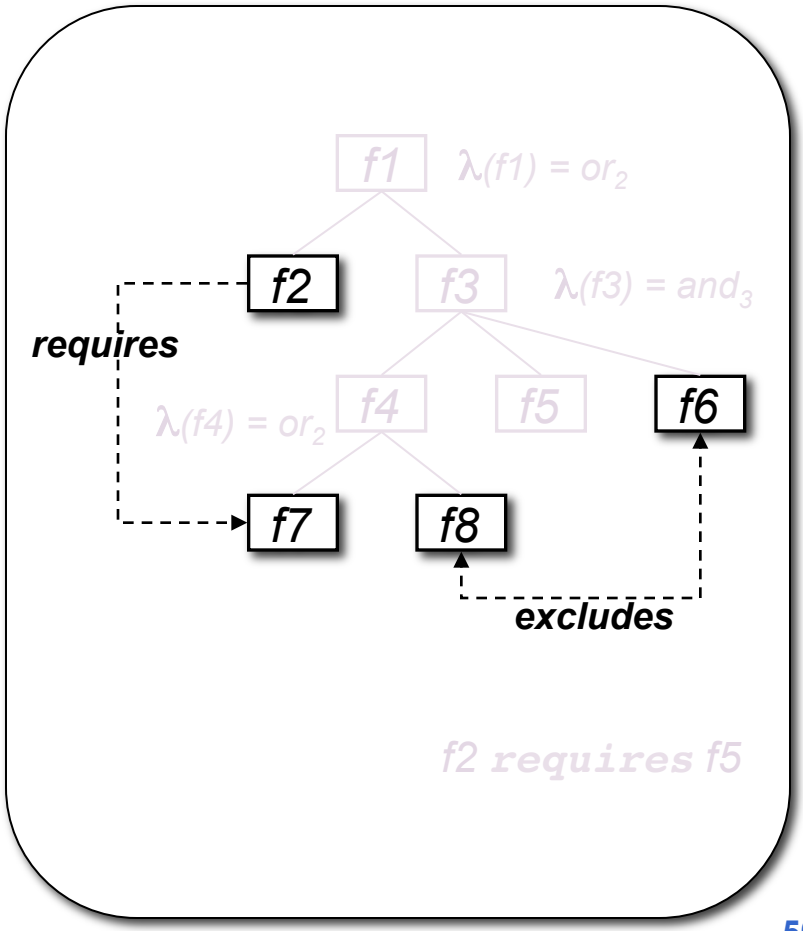



Conventionally, for a *leaf* node n, λ(n) =$and_0$

# The $L_{FFD}$ parametric abstract syntax

Every FD **d** $\in$ $\mathbf{L_{FFD}}$**(GT,NT,GCT,TCL)** is a tuple **(N,r,** $\lambda$**,DE,CE,**$\Phi$**)** such that

- ★ **N** *the set of nodes*
- ★ **r** $\in$ **N** *the root*
- ★ **DE** $\subseteq$ **N** $\times$ **N** *the decomposition edges (obeying GT)*
- ★ $\lambda$ **: N**$\rightarrow$**NT** *the function labelling nodes with boolean operators*
- ★ **CE** $\subseteq$ **N** $\times$ **GCT** $\times$ **N** *the constraint edges*
- ★ $\Phi$ $\subseteq$ **TCL** *the textual constraints*
- ★ + 4 well-formedness constraints e.g. *only r has no parent*

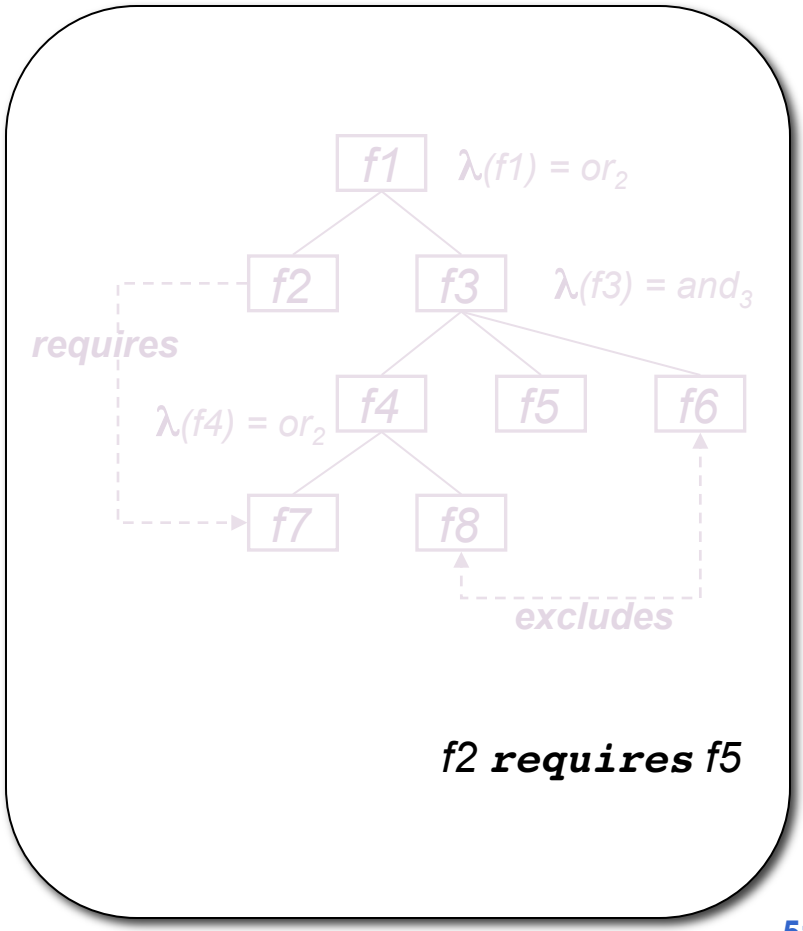


# The $L_{FFD}$ parametric abstract syntax

Every FD **d** $\in$ $\mathbf{L_{FFD}}$**(GT,NT,GCT,TCL)** is a tuple **(N,r,** $\lambda$**,DE,CE,**$\Phi$**)** such that

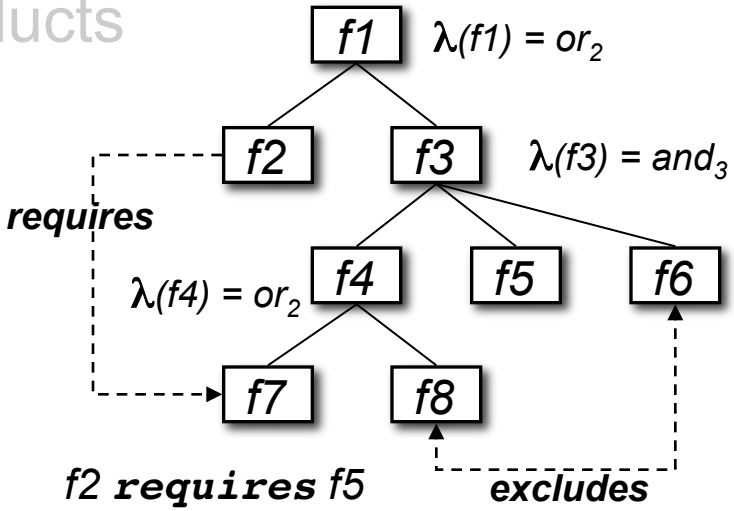
- ★ **N** *the set of nodes*
- ★ **r** $\in$ **N** *the root*
- ★ **DE** $\subseteq$ **N** $\times$ **N** *the decomposition edges (obeying GT)*
- ★ $\lambda$ **: N**$\rightarrow$**NT** *the function labelling nodes with boolean operators*
- ★ **CE** $\subseteq$ **N** $\times$ **GCT** $\times$ **N** *the constraint edges*
- ★ $\Phi$ $\subseteq$ **TCL** *the textual constraints*

- ★ + 4 well-formedness constraints e.g. *only r has no parent*



# Semantic domain

- ★ A **configuration** is a set of nodes/features
- ★ A **product** is a set of primitive nodes/features
- ★ A **product line** (PL) is a set of products

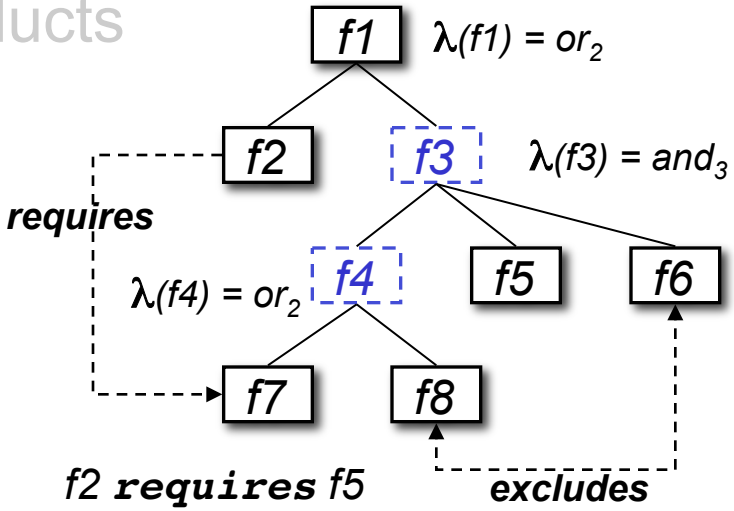



*f1 f3 f4 f5 f6 f7* $\in \wp \mathrm{N}$

# Semantic domain

- A **configuration** is a set of nodes/features
- A **product** is a set of primitive nodes/features
- A **product line** (PL) is a set of products

e.g., if *f3* and *f4* are "irrelevant" (non-primitive) for the stakeholders




*remove non-primitive features*

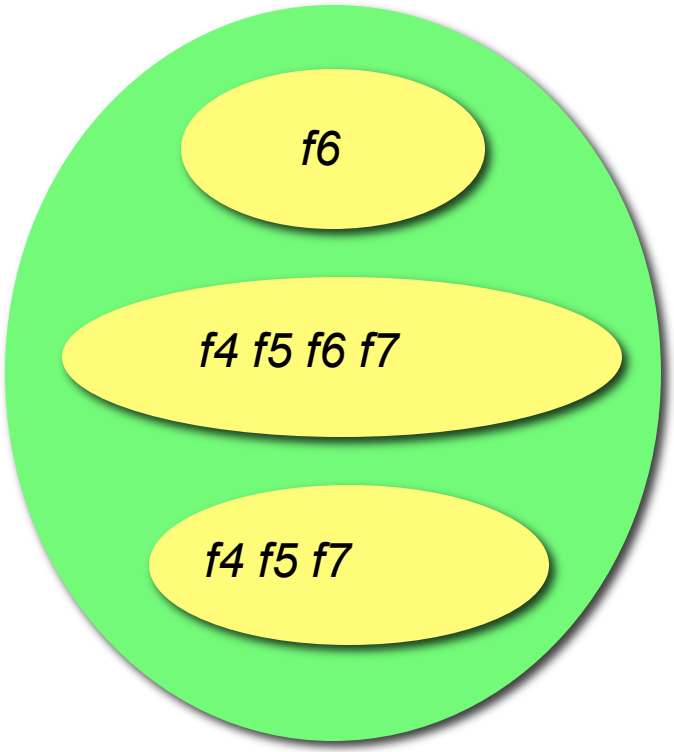
*f1 f3 f4 f5 f6 f7* $\in \wp$ N

*f1 f5 f6 f7* $\in \wp$ P

where P $\subseteq$ N

# Semantic domain

- ★ A **configuration** is a set of nodes/features
- ★ A **product** is a set of primitive nodes/features
- ★ A **product line** (**PL**) is a set of products




$$\in S_{FFD} = PL = \wp\,\wp\,P$$

# Feature Models

**#2 Language?**

**to elaborate/build variability models**

# Feature Models

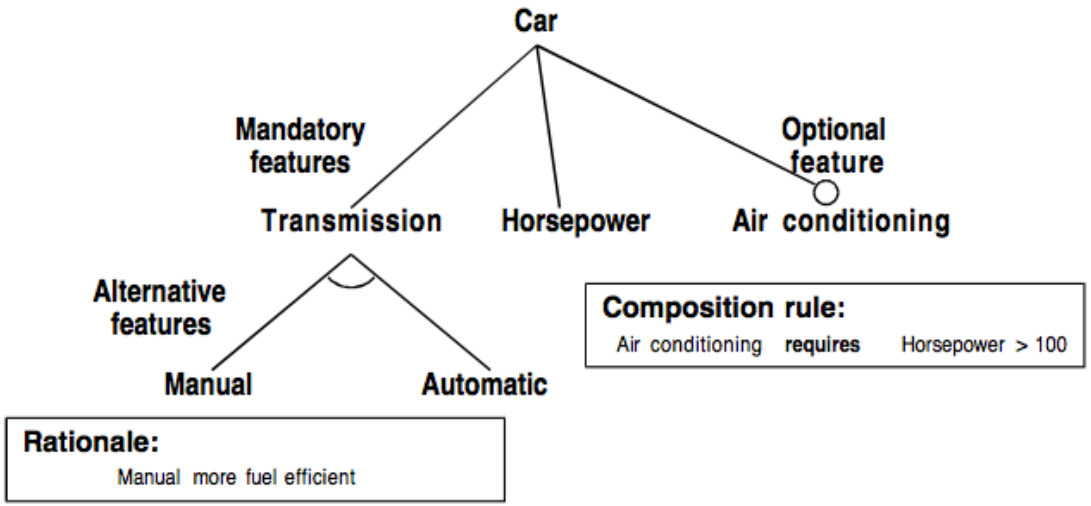
# existing languages
# vs
# TVL language

# Feature Models

# ~ Feature Diagrams (suggest a visual representation)

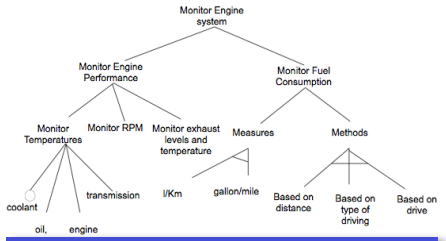
# graphical languages

# Feature Models

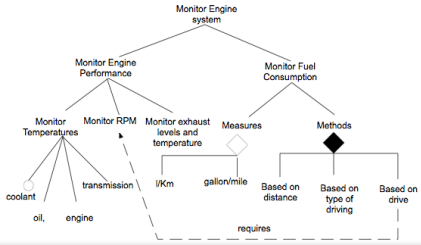



*Kang et al. (1990)*

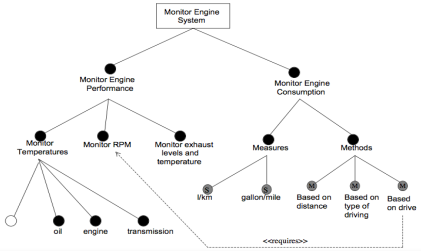
# FD dialects




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

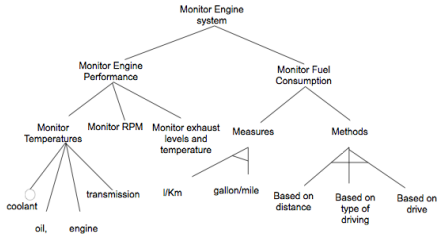



*FeatuRSEB (RFD)*
*[Griss et al., 1998]*

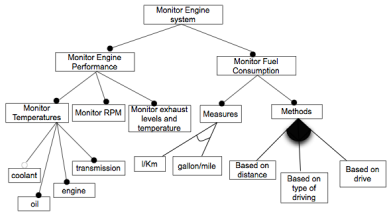



*PLUSS (PFT)*
*[Eriksson et al., 2005]*

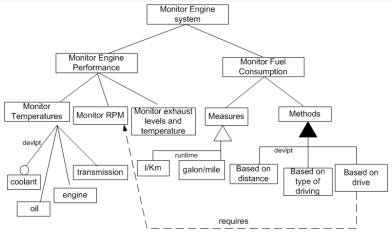
*FORM (OFD)*
*[Kang et al., 1998]*



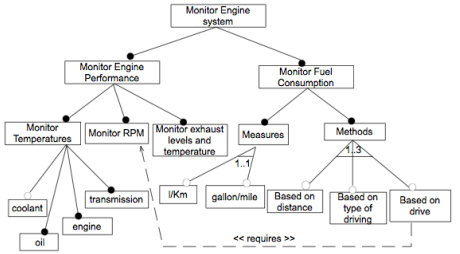

*Gen. Prog. (GPFT)*
*[Czarnecki et al., 2000]*



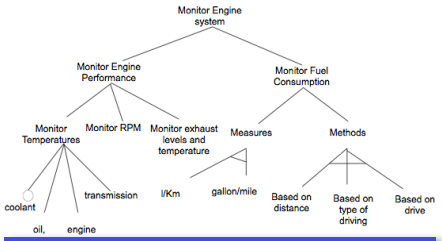

*VBFD*
*[van Gurp et al., 2001]*



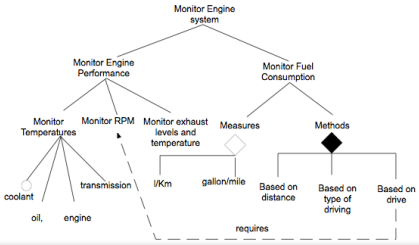

*EFD*
*[Riebisch et al., 2002]*



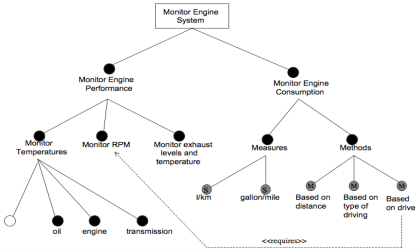
# FD dialects




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



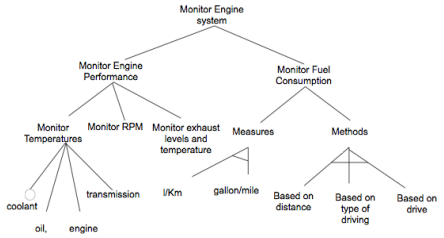

FeatuRSEB (RFD)
[Griss et al., 1998]



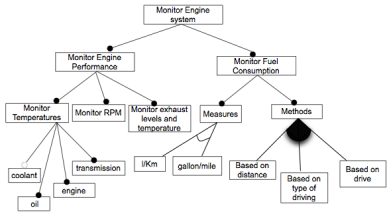

PLUSS (PFT)
[Eriksson et al., 2005]

Aesthetic differences

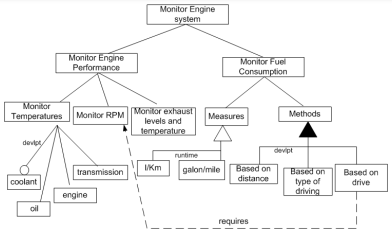
FORM (OFD)
[Kang et al., 1998]



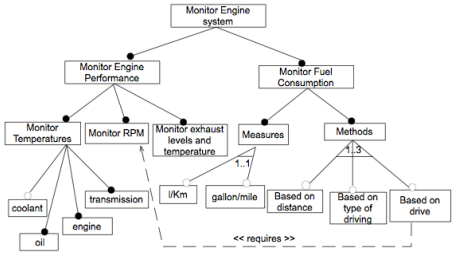

Gen. Prog. (GPFT)
[Czarnecki et al., 2000]



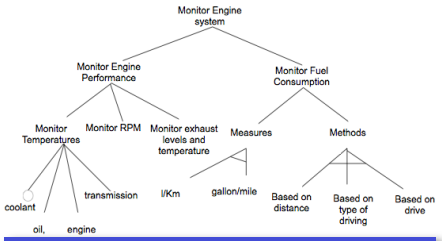

VBFD
[van Gurp et al., 2001]

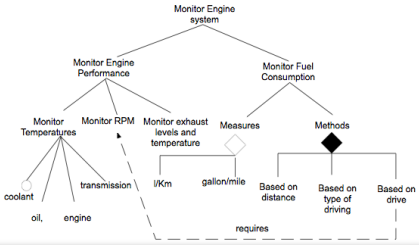


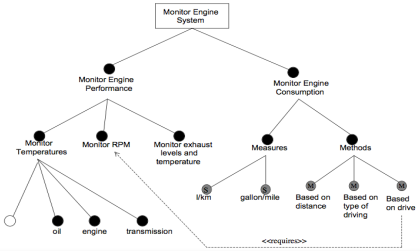

EFD
[Riebisch et al., 2002]



# FD dialects




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

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

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

Aesthetic differences

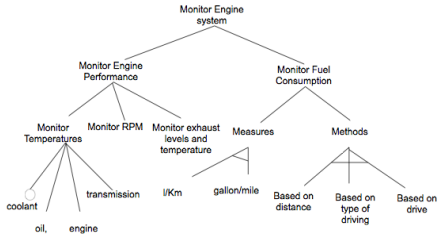
Stronger claims

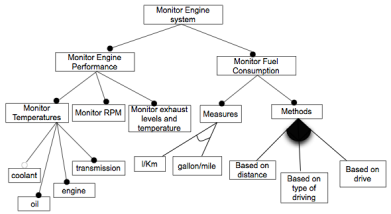
*FORM (OFD)*
*[Kang et al., 1998]*

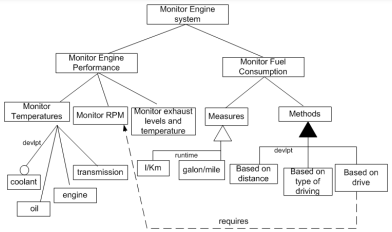
*Gen. Prog. (GPFT)*
*[Czarnecki et al., 2000]*

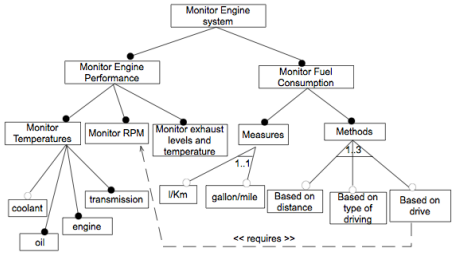
*VBFD*
*[van Gurp et al., 2001]*

*EFD*
*[Riebisch et al., 2002]*

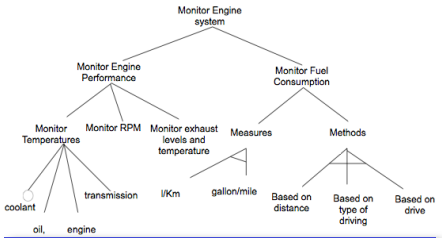


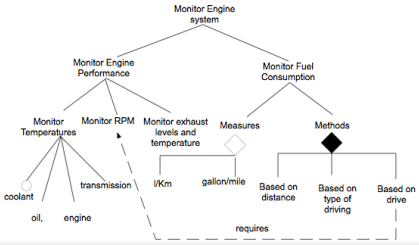



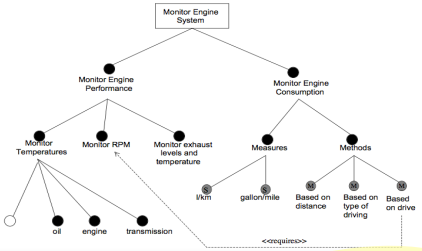



# FD dialects




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

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

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

Aesthetic differences

Stronger claims

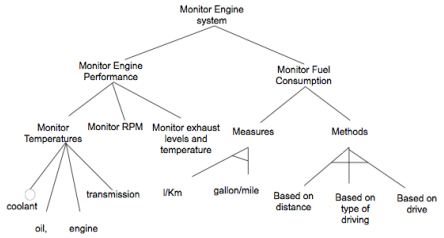
Vague use of terms
syntax, semantics, expressiveness, ambiguity…

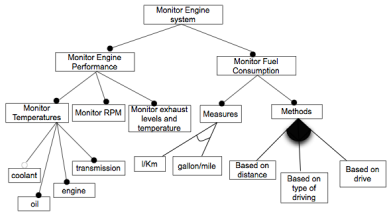
*FORM (OFD)*
*[Kang et al., 1998]*

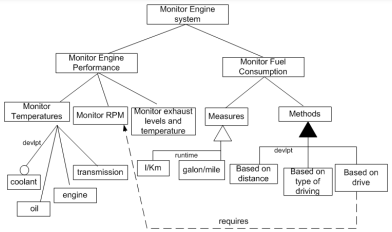
*Gen. Prog. (GPFT)*
*[Czarnecki et al., 2000]*

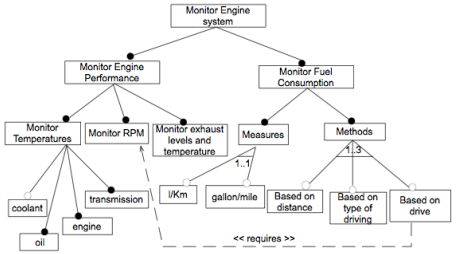
*VBFD*
*[van Gurp et al., 2001]*

*EFD*
*[Riebisch et al., 2002]*



# Attacking research fragmentation

- **Formal definition of FODA**

*[Bontemps et al., ICFI, 2004]*

- **General formal definition of FD dialects**

*[Schobbens et al., RE, 2006]*

- **Survey according to formal criteria**
  - Ambiguity – expressiveness – embeddability – succinctness
  - Computational complexity of decision problems

*[Schobbens et al., J. Computer Networks, 2007]*

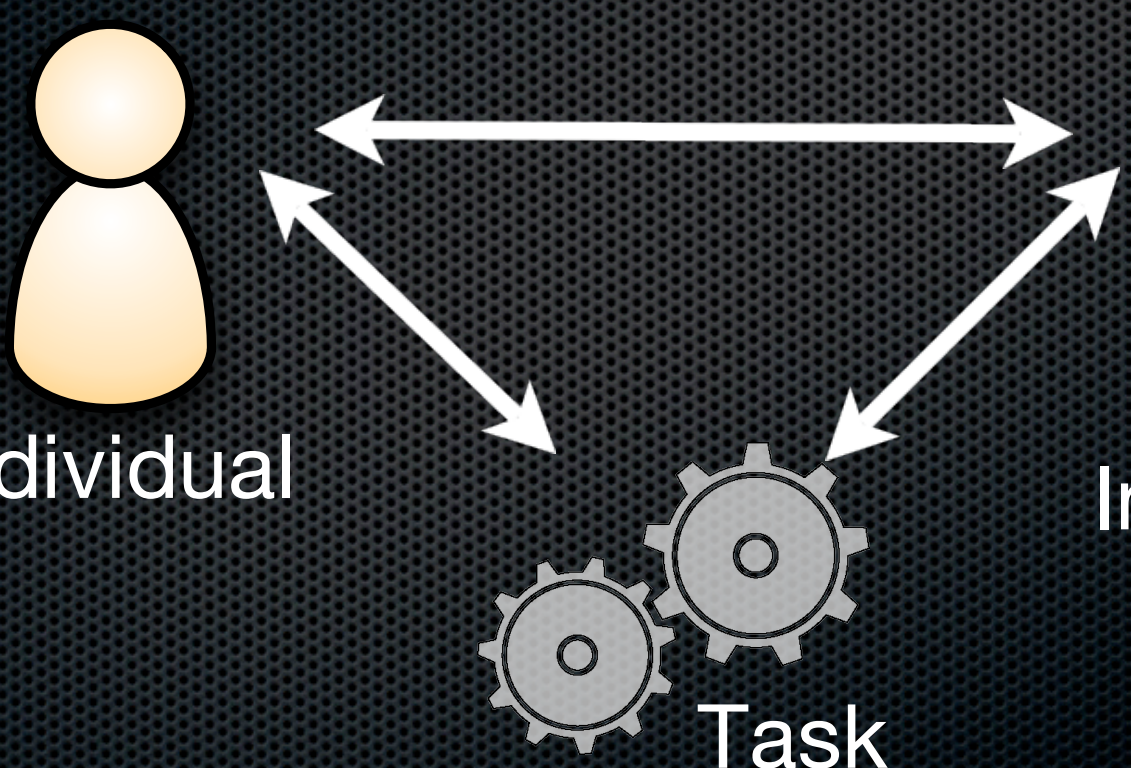
# Feature Models

# ~ Feature Diagrams (suggest a visual representation)

# but that's not mandatory!

# Cognitive fit theory

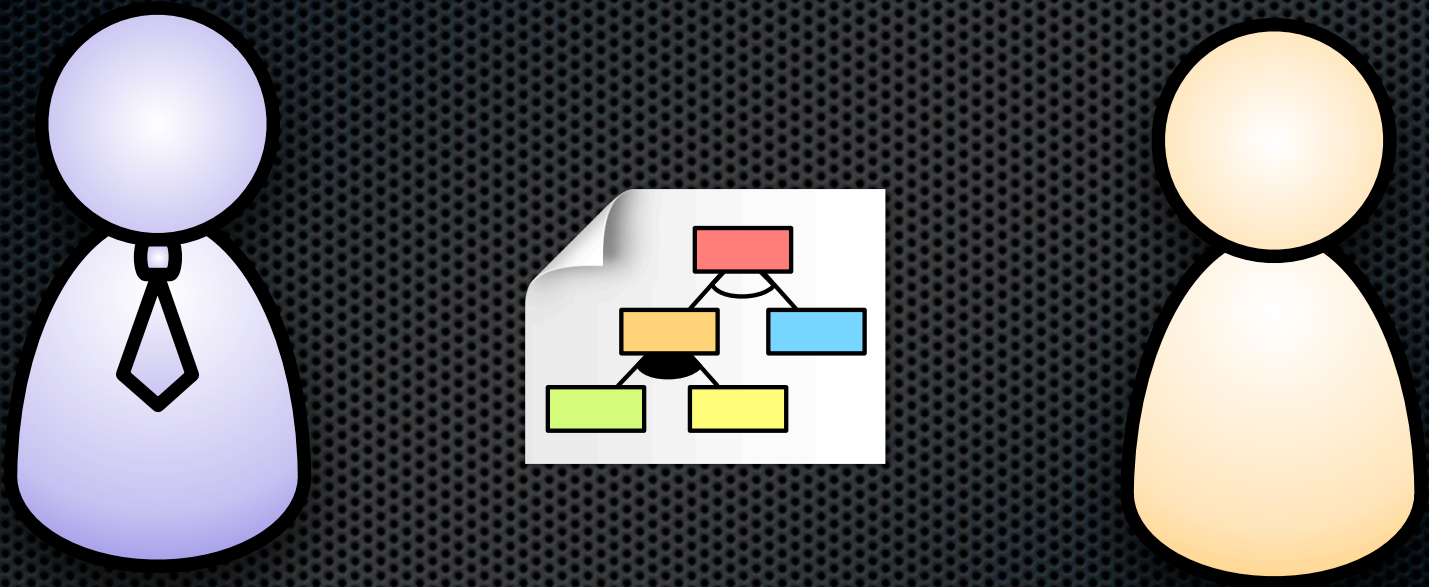
[Vessey1991]

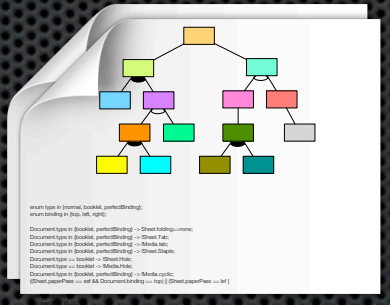


Individual

Information format

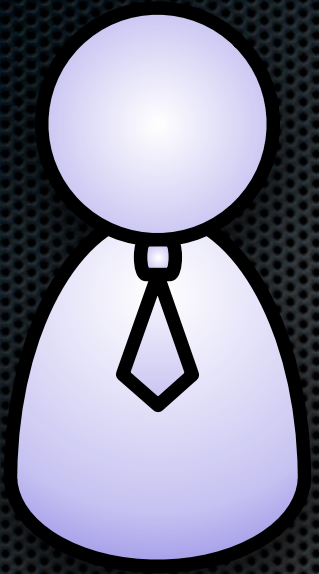
Task

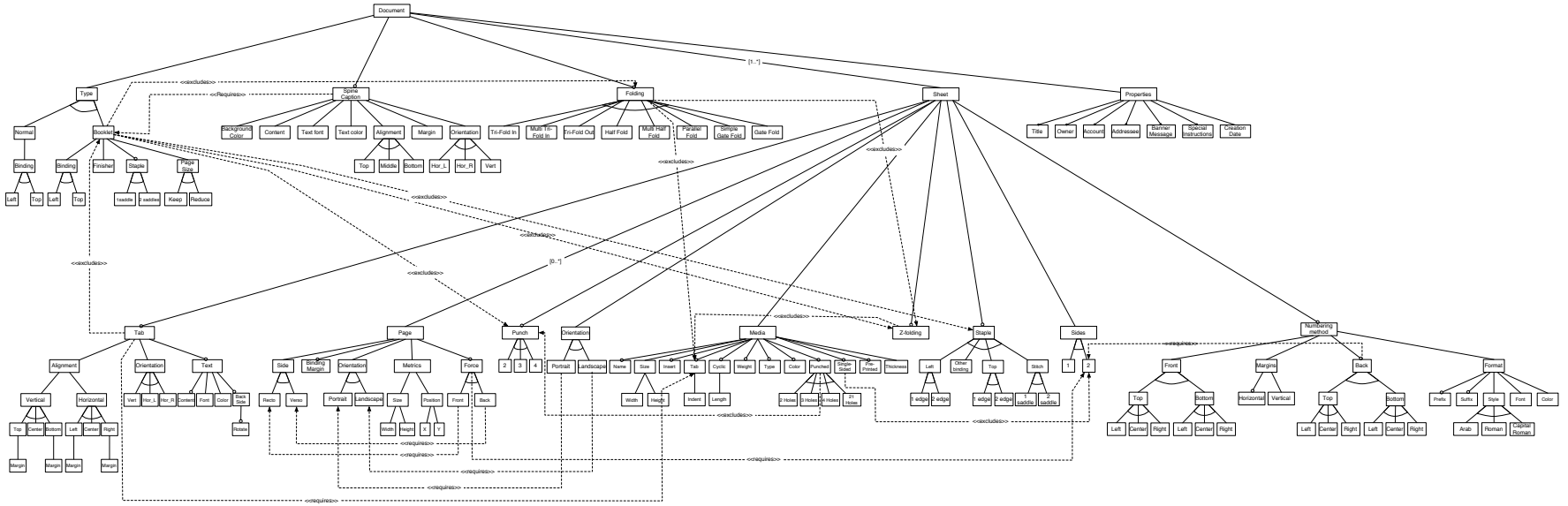




```
enum type in {normal, booklet, perfectBinding};
enum binding in {top, left, right};

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|}
```




Document
[1..*]
Type
<<excludes>>
<<Requires>>
Spine Caption
Folding
Sheet
Properties
Normal
Booklet
Background Color
Content
Text font
Text color
Alignment
Margin
Orientation
Tri-Fold In
Multi Tri-Fold In
Tri-Fold Out
Half Fold
Multi Half Fold
Parallel Fold
Simple Gate Fold
Gate Fold
Title
Owner
Account
Addressee
Banner Message
Special Instructions
Creation Date
Binding
Binding
Finisher
Staple
Page Size
Top
Middle
Bottom
Hor_L
Hor_R
Vert
Left
Top
Left
Top
Leaside
Faceted
Keep
Reduce
<<excludes>>
<<excludes>>
<<excludes>>
<<excludes>>
<<excludes>>
[0..*]
Tab
Page
Punch
Orientation
Media
Z-folding
Staple
Sides
Numbering method
<<requires>>
Alignment
Orientation
Text
Side
Binding Margin
Orientation
Metrics
Force
2
3
4
Portrait
Landscape
Name
Size
Insert
Tab
Cyclic
Weight
Type
Color
Punched
Single Sided
Pre-Printed
Thickness
Left
Other binding
Top
Stitch
1
2
Front
Margins
Back
Format
Vertical
Horizontal
Vert
Hor_L
Hor_R
Content
Font
Color
Back Side
Recto
Verso
Portrait
Landscape
Size
Position
Front
Back
Width
Height
Indent
Length
2 Holes
3 Holes
4 Holes
4+ Holes
1 edge
2 edge
1 edge
2 edge
1 saddle
2 saddle
Top
Bottom
Horizontal
Vertical
Top
Bottom
Prefix
Suffix
Style
Font
Color
Top
Center
Bottom
Left
Center
Right
Rotate
Width
Height
X
Y
Left
Center
Right
Left
Center
Right
Left
Center
Right
Left
Center
Right
Arab
Roman
Capital Roman
Margin
Margin
Margin
Margin
<<excludes>>
<<excludes>>
<<requires>>
<<requires>>
<<requires>>
<<requires>>
<<requires>>
<<requires>>
<<requires>>

# Visualisation will help, right?

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

# 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

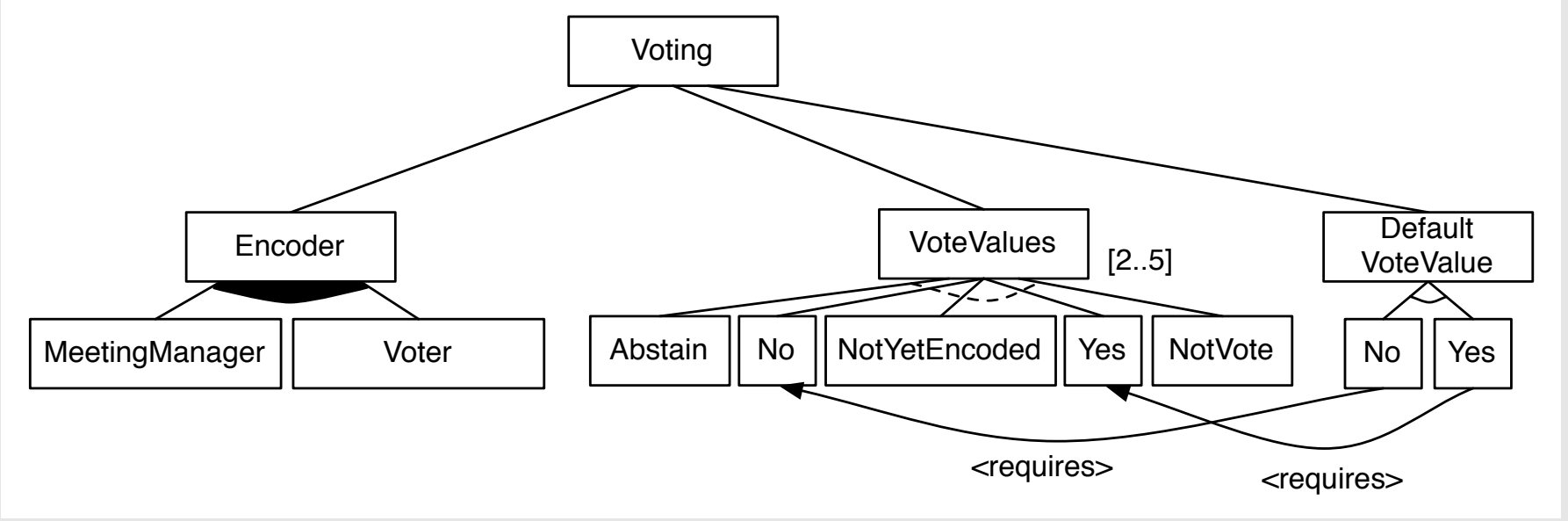
+
CML,
+ VSL

| Language | Userfriendly | Attributes | Cardinalities | Cross-tree constraints | Modularisation |
|---|---|---|---|---|---|
| Van Deursen et al. | ✅ | | | ✅ | |
| GUIDSL (AHEAD) | ✅ | | | ✅ | |
| SXFM | ✅ | | | ✅ | |
| XML-based (FAMA etc.) | | ✅ | ✅ | ✅ | |

# Furthermore

- ★ Intuitive C-like syntax
- ★ Formal semantics
- ★ Statically typed
- ★ Implemented

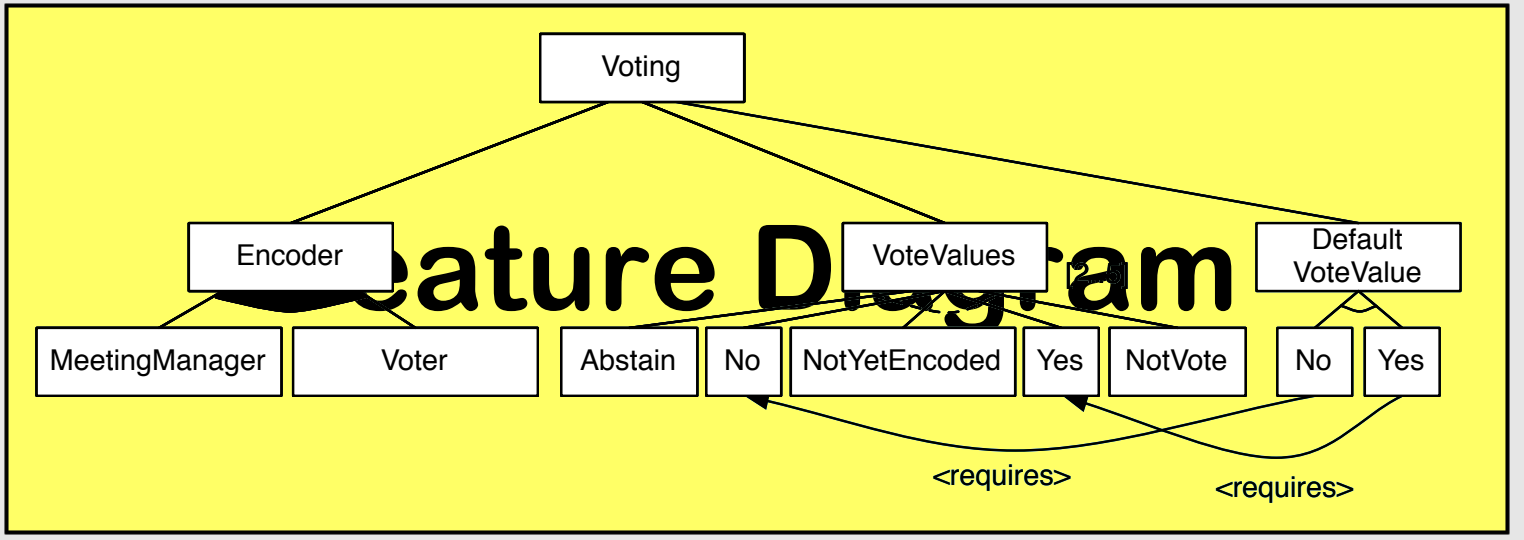
# Graphical feature models




**Legend**

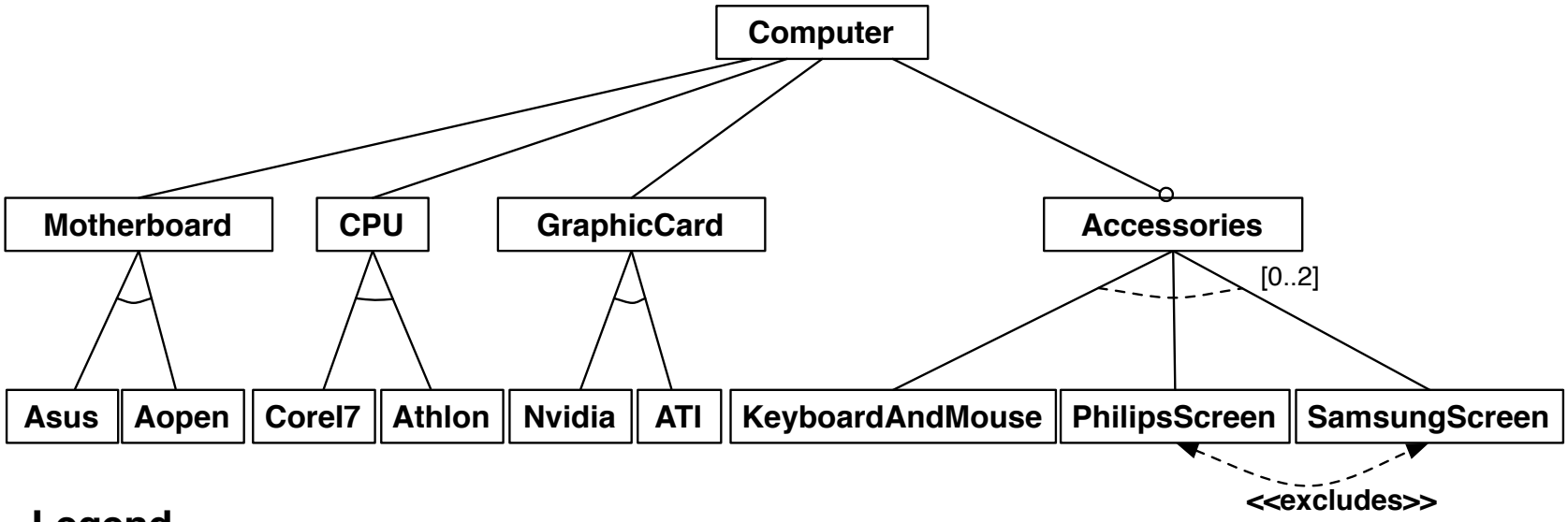
*And*-decomposition *Or*-decomposition Xor-decomposition [i..j] group cardinality

# TVL: Basic principles




```
root Voting {
enum defaultVoteValue in {no, yes};
(defaultVoteValue==no) -> No;
(defaultVoteValue==yes) -> Yes;
group allOf {
Encoder {group someOf {MeetingManager, Voter}},
VoteValues group [2..*] {
Abstain,
No,
NotYetEncoded,
Yes,
NotVote
}
}
}
```

# Example FD




**Legend**

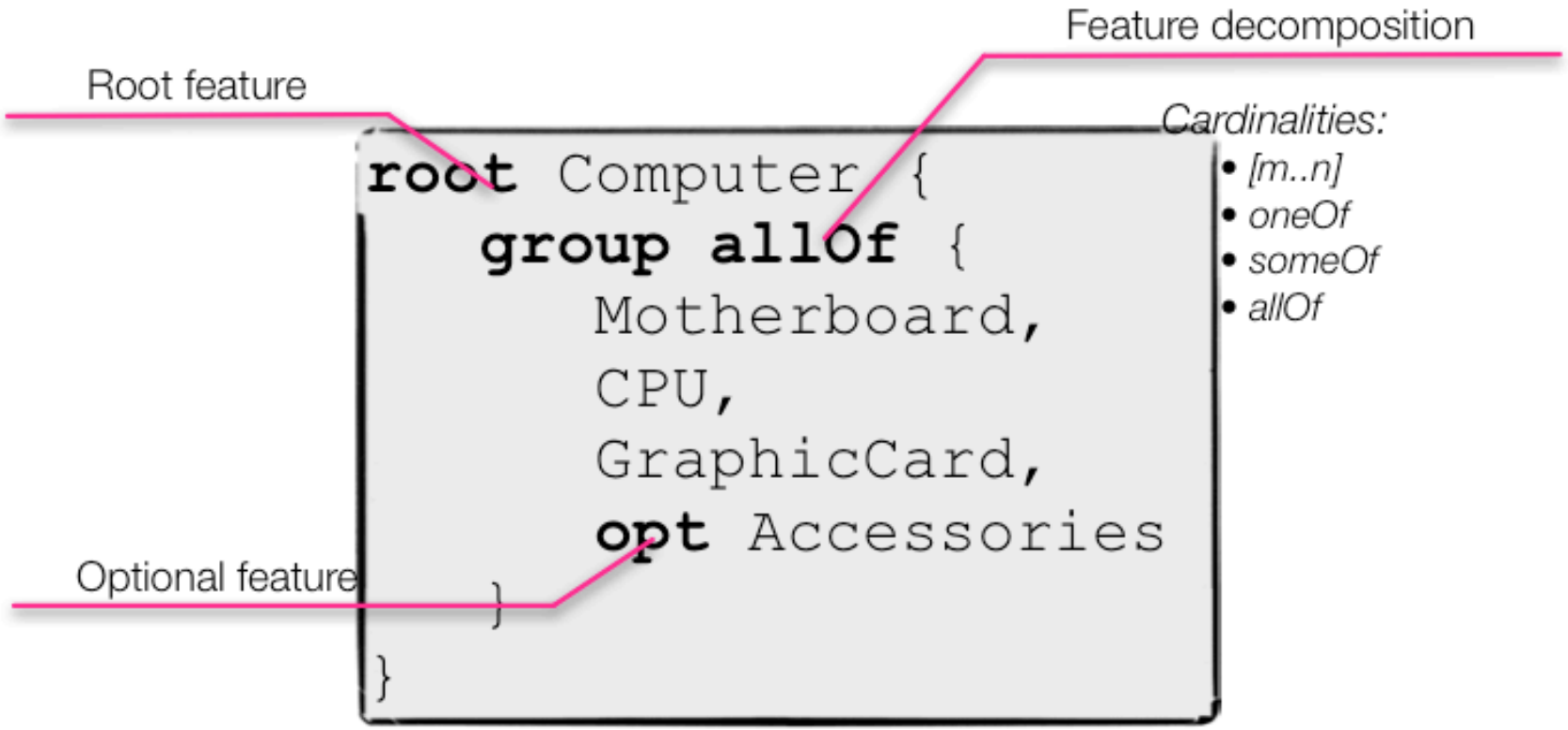
and-decomposition

xor-decomposition

[i..j] group cardinality [i..j]

optional feature

# TVL – Feature hierarchy



# TVL – Feature hierarchy (DAG)

```
root A {
  group oneOf {
     B group allOf {D},
     C group allOf {shared D}
  }
}
```

Shared feature

# TVL – Attributes (declaration)

4 attribute types

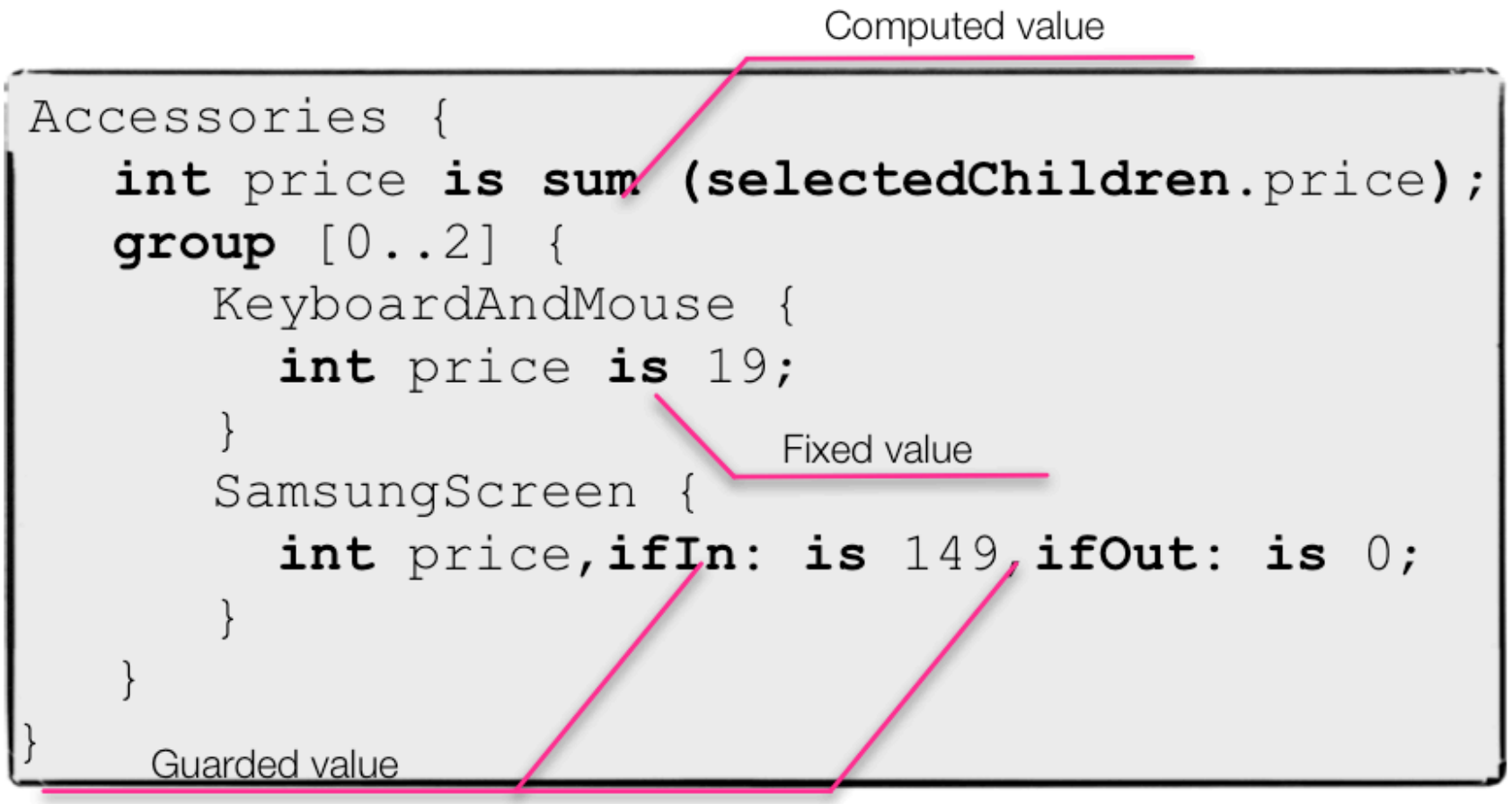
- *integer*
- *rational*
- *Boolean*
- *enumeration*

Domain interval

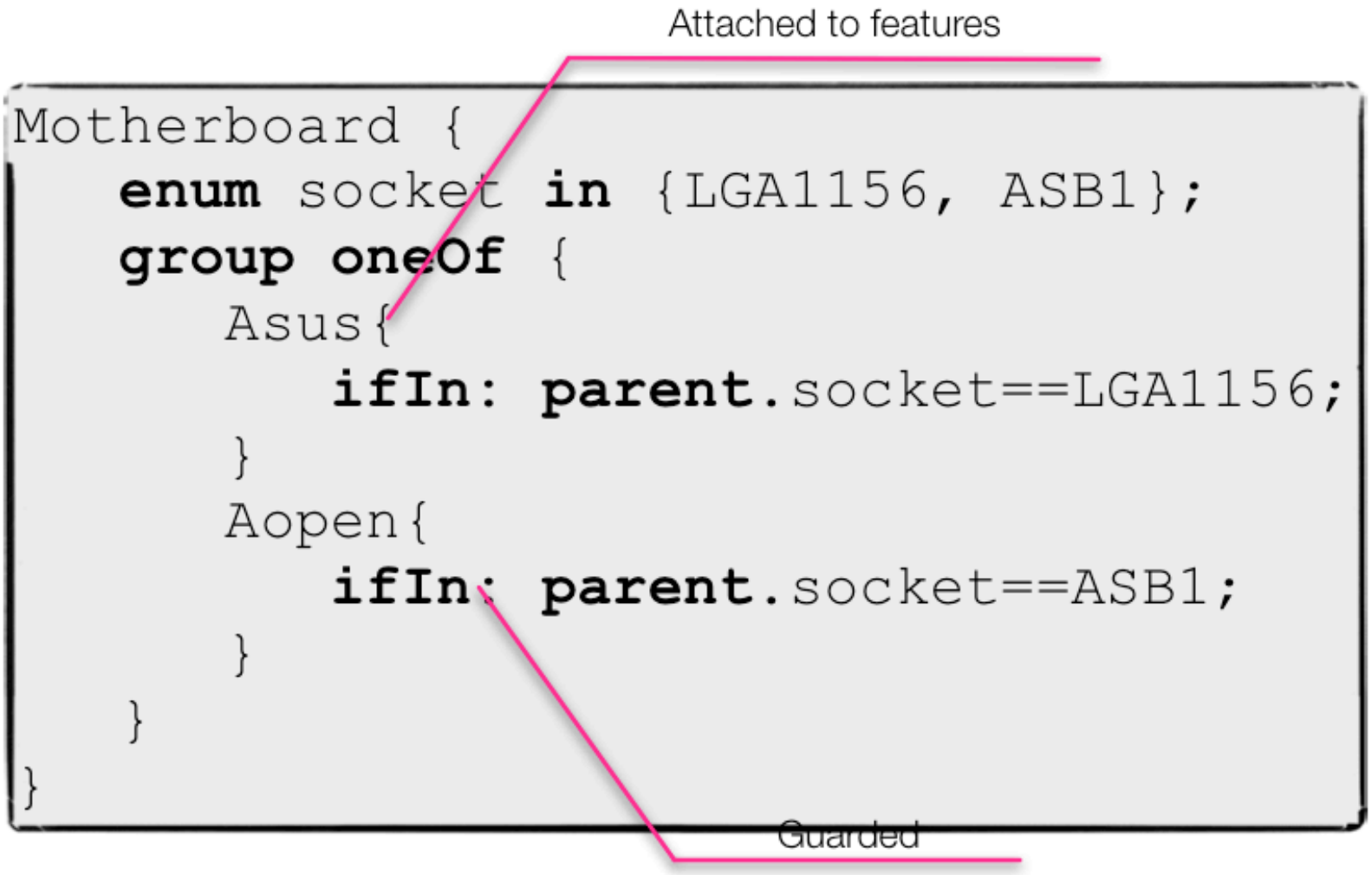
```
Computer {
    int price in [0..500];
    int width;
    int height;
    enum socket  in {LGA1156, ASB1};
}
```

Domain enumeration

# TVL – Attributes (assignment)



# TVL – Constraints



# TVL – Modularity

★ In the header

- Custom types definition for the entire feature model

```
enum cpuSocket {LGA1156, ASB1};
```

- Structured types definitition

```
struct motherBoardSize {
     int heigth;
     int width;
}
```

- Constants definition

```
const int maxRamBlocks 4;
```

# TVL – Modularity

★ In the feature model

- Other TVL files can be included

```
include (./some/other/file);
```

- Features can be extented anywhere in the code

```
root Computer {
   group allOf {
      Motherboard,
      CPU,
      GraphicCard,
      opt Accessories
   }
}
Computer {
   int price is sum(selectedChildren.price);
   ...
}
Motherboard {
   motherBoardSize size;
   ...
}
```

# More info online

- ★ Parser API (JAVA)
- ★ TVL to Boolean CNF implementation
- ★ Formal grammar
- ★ Formal semantics

www.info.fundp.ac.be/~acs/tvl

# Feature Models

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

# Feature Models

# and

# Automated Reasoning

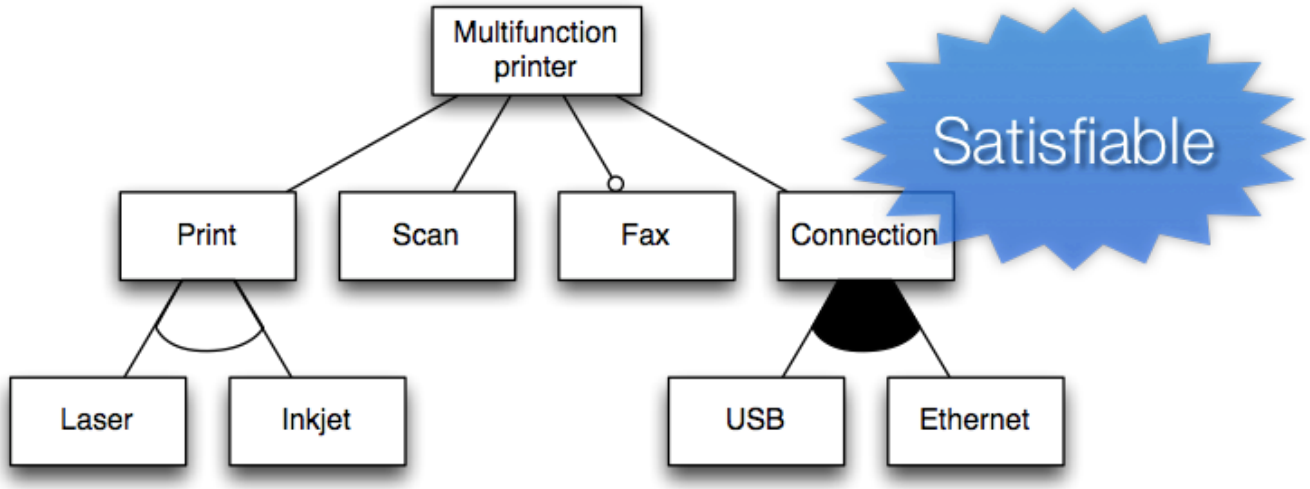
# Decision problems and complexity

- **Satisfiability**: $M_{FFD}(d) \neq \varnothing$
  - NP-complete
  - linear for many tree-shaped graphs
- **Product checking**: $\{f_1,...,f_n\} \in M_{FFD}(d)$
  - NP-complete
  - linear for many tree-shaped graphs, or if P = N
- **Equivalence**: $M_{FFD}(d_1) = M_{FFD}(d_2)$
  - $\Pi_1$-complete

- **Tractability issues** but **efficient algorithms in many common cases**
- Existing **algorithms** can now be **proved for correctness** and **optimality**

*[Schobbens et al., J. Computer Networks, 2007]*

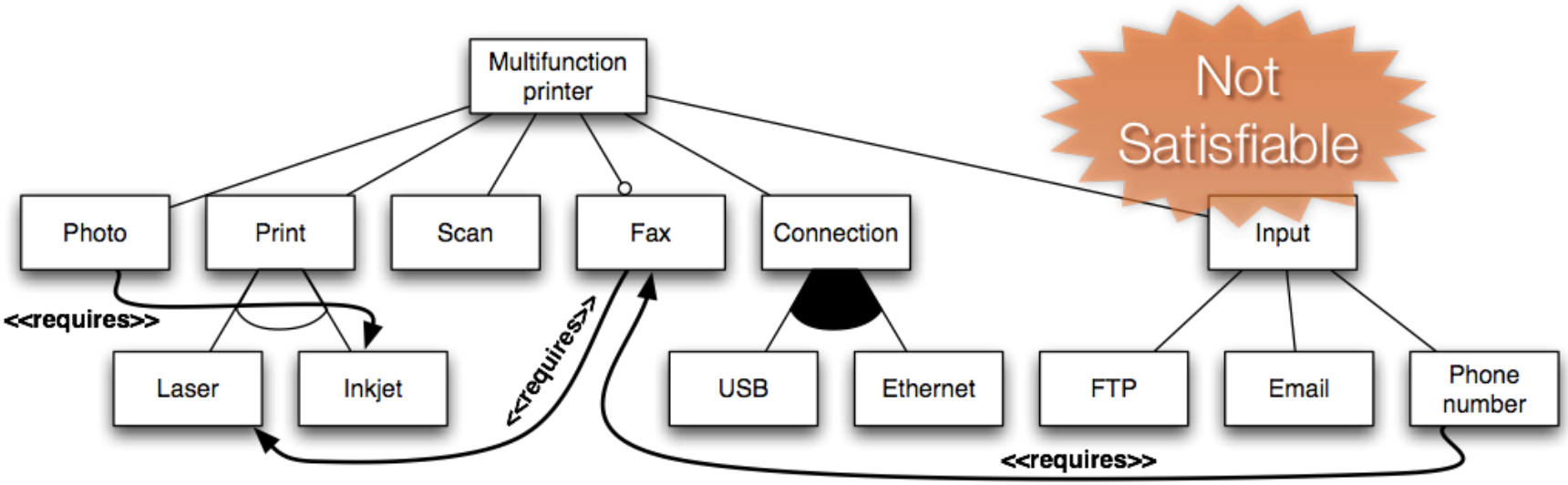
# Satisfiability – example




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

⇔ Not empty

# Satisfiability – example



# More decision problems

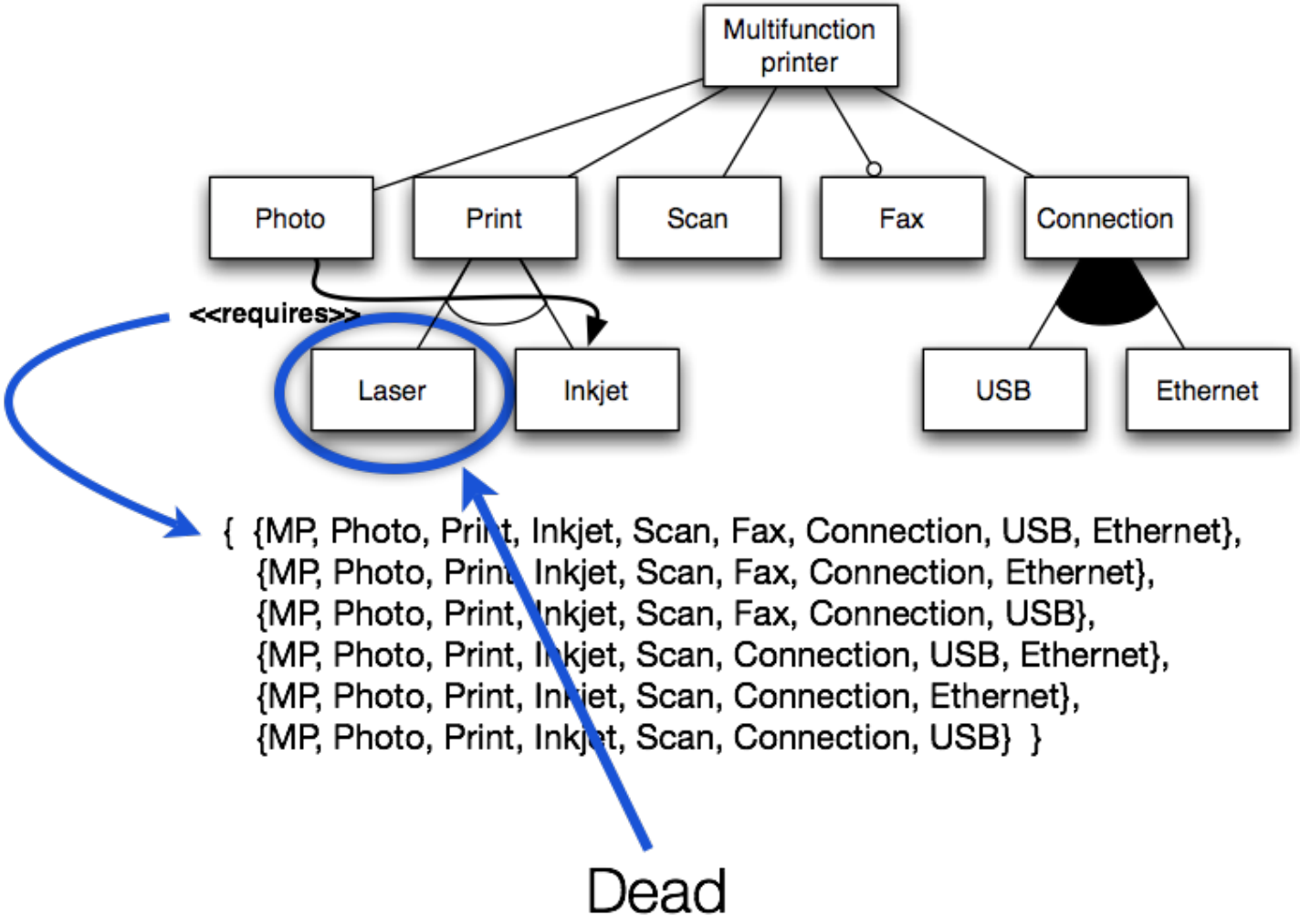
- **Dead features**: $P \setminus \cup M_{FFD}(d)$
- **Commonality**: $\cap M_{FFD}(d)$
- **Variability**: $P \setminus \cap M_{FFD}(d)$
- **"Merging"** FDs: construct $d_3$ from $d_1$ and $d_2$ s.t.
  - **Intersection**: $M_{FFD}(d_3) = M_{FFD}(d_1) \cap M_{FFD}(d_2)$
  - **Union**: $M_{FFD}(d_3) = M_{FFD}(d_1) \cup M_{FFD}(d_2)$
  - **Reduced product**:

$$M_{FFD}(d_3) = \{p_1 \cup p_2 \mid p_1 \in M_{FFD}(d_1) \wedge p_2 \in M_{FFD}(d_2)\}$$

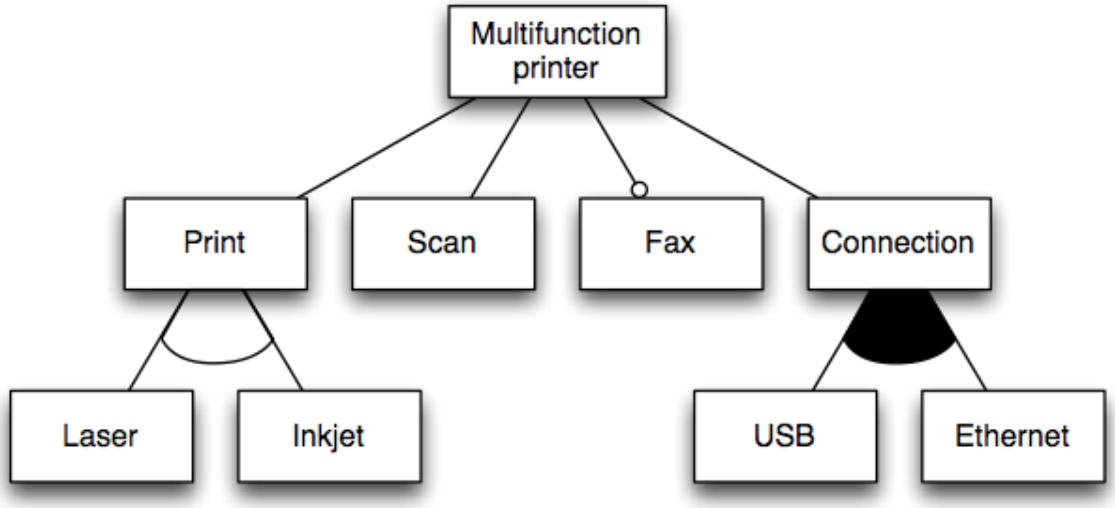
*[Schobbens et al., J. Computer Networks, 2007]*

- etc... *[Benavides et al., JISBD, 2006]*

# Dead feature – example

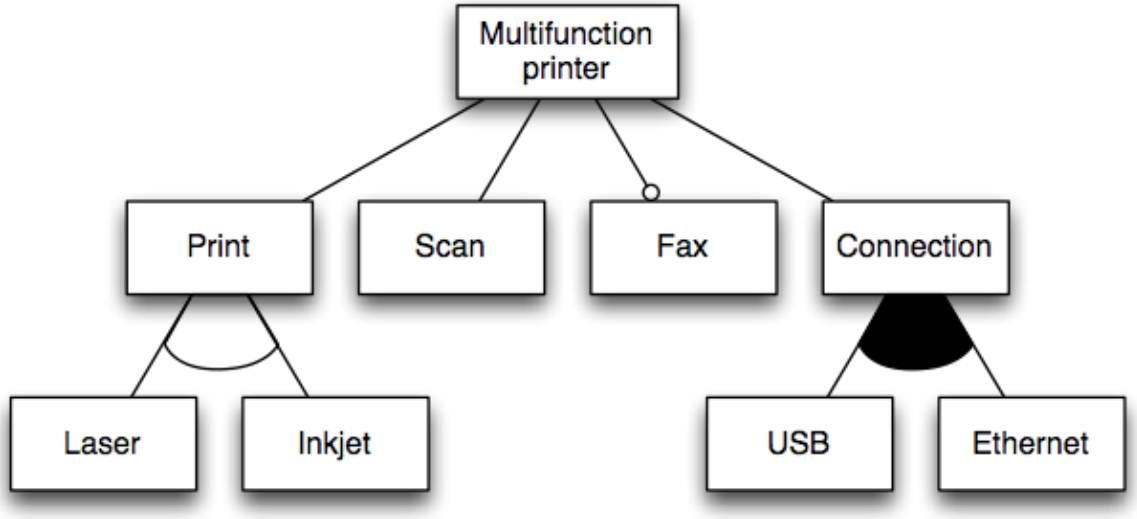


# Commonality – example




{ {**MP, Print, Laser, Scan**, Fax, **Connection**, USB, Ethernet},
{**MP, Print, Laser, Scan**, Fax, **Connection**, Ethernet},
{**MP, Print, Laser, Scan**, Fax, **Connection**, USB},
{**MP, Print, Laser, Scan**, **Connection**, USB, Ethernet},
{**MP, Print, Laser, Scan**, **Connection**, Ethernet},
{**MP, Print, Laser, Scan**, **Connection**, USB},
{**MP, Print, Laser, Scan**, Fax, **Connection**, USB, Ethernet},
{**MP, Print, Laser, Scan**, Fax, **Connection**, Ethernet},
{**MP, Print, Laser, Scan**, Fax, **Connection**, USB},
{**MP, Print, Laser, Scan**, **Connection**, USB, Ethernet},
{**MP, Print, Laser, Scan**, **Connection**, Ethernet},
{**MP, Print, Laser, Scan**, **Connection**, USB} }

# Variability – example




{ {MP, Print, Laser, Scan, **Fax**, Connection, **USB, Ethernet**},
{MP, Print, Laser, Scan, **Fax**, Connection, **Ethernet**},
{MP, Print, Laser, Scan, **Fax**, Connection, **USB**},
{MP, Print, Laser, Scan, Connection, **USB, Ethernet**},
{MP, Print, Laser, Scan, Connection, **Ethernet**},
{MP, Print, Laser, Scan, Connection, **USB**},
{MP, Print, Inkjet, Scan, **Fax**, Connection, **USB, Ethernet**},
{MP, Print, Inkjet, Scan, **Fax**, Connection, **Ethernet**},
{MP, Print, Inkjet, Scan, **Fax**, Connection, **USB**},
{MP, Print, Inkjet, Scan, Connection, **USB, Ethernet**},
{MP, Print, Inkjet, Scan, Connection, **Ethernet**},
{MP, Print, Inkjet, Scan, Connection, **USB**} }

## 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

# Feature Models

## SAT solvers
## Binary Decision Diagrams (BDDs)
## CSP 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.
  - indispensable component of many formal verification and (more recently) program analysis applications.
- 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

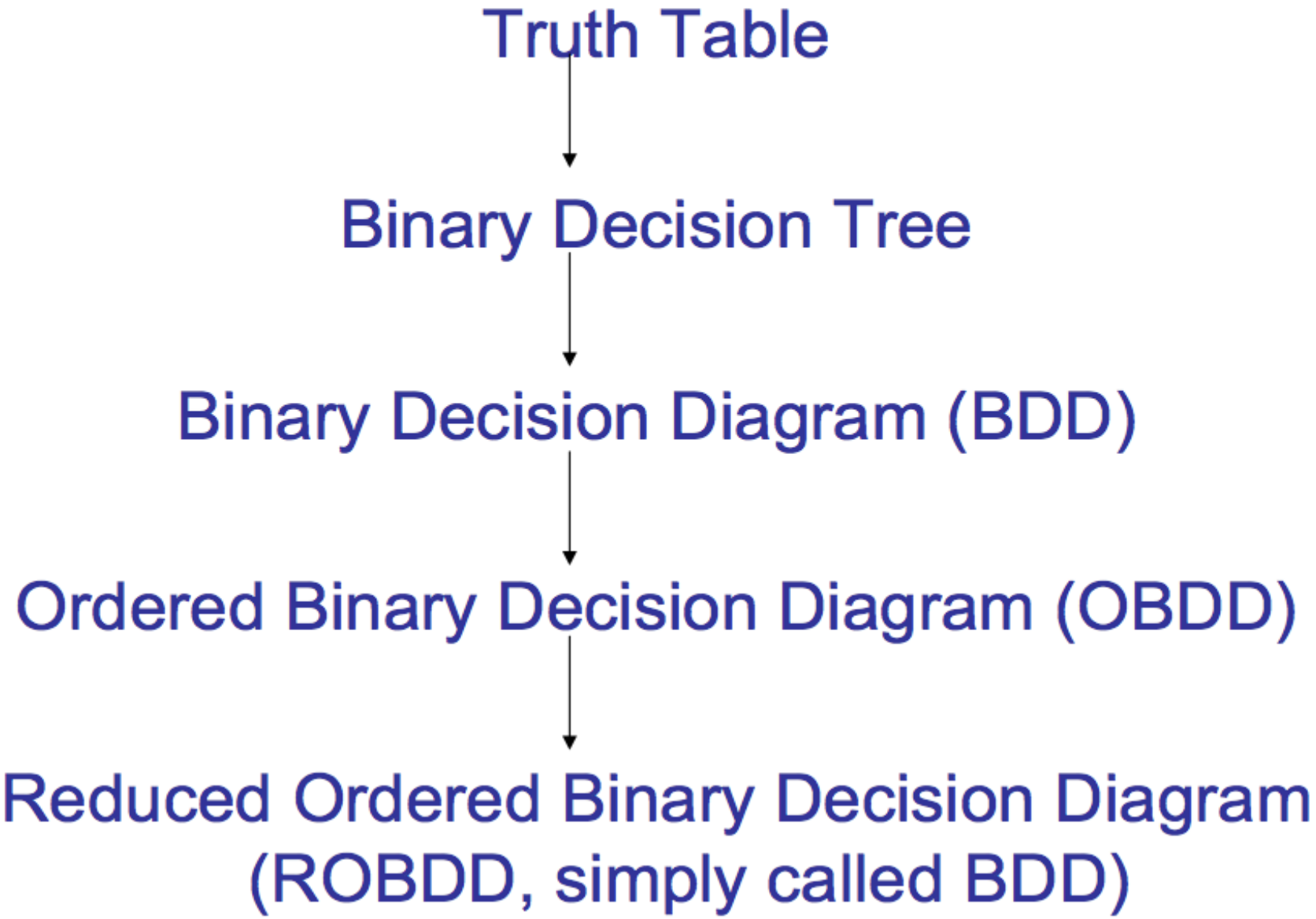
- Unit propagation (or “Boolean constraint propagation”, BCP for short) is the key component to fast SAT solving.
- Whenever all the literals in a clause are false except one, the remaining literal must be true in any satisfying assignment (such a clause is called a **unit clause**).
  - Therefore, the 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.
  - A good SAT solve often spends 80-90% of its time in unit propagation.

# SAT solver and Unit Propagation

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

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

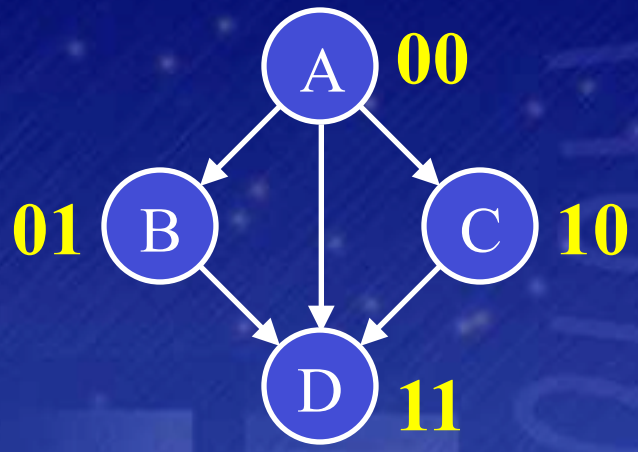
# BDDs from Truth Tables



# Truth table, boolean function

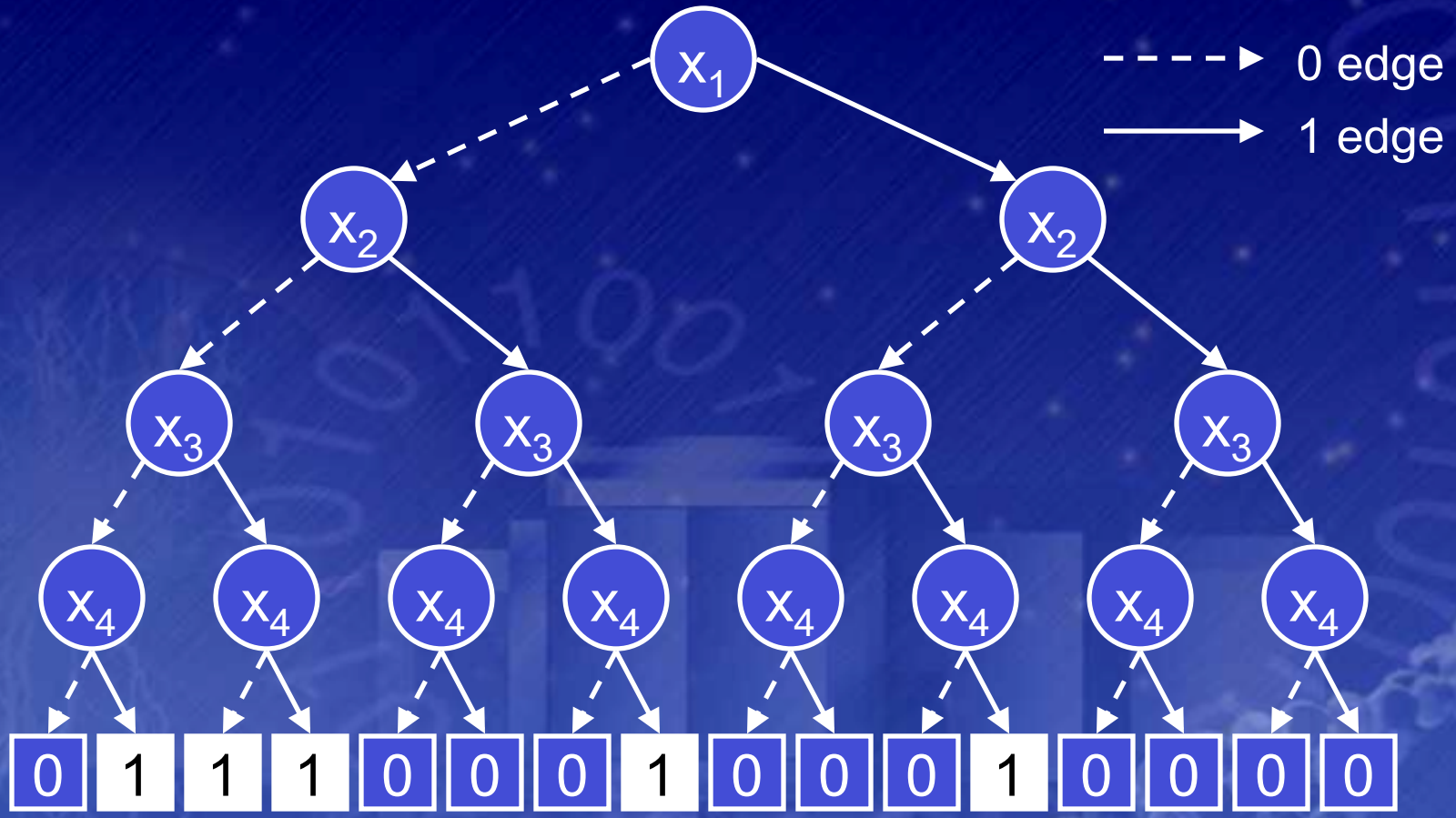
| *from* | | *to* | | |
|---|---|---|---|---|
| $x_1$ | $x_2$ | $x_3$ | $x_4$ | f |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0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

- Relation expressed as a binary function.
  - A=00, B=01, C=10, D=11



# Binary Decision Diagrams (Bryant 1986)

- Encoding of a truth table.



# Reduction

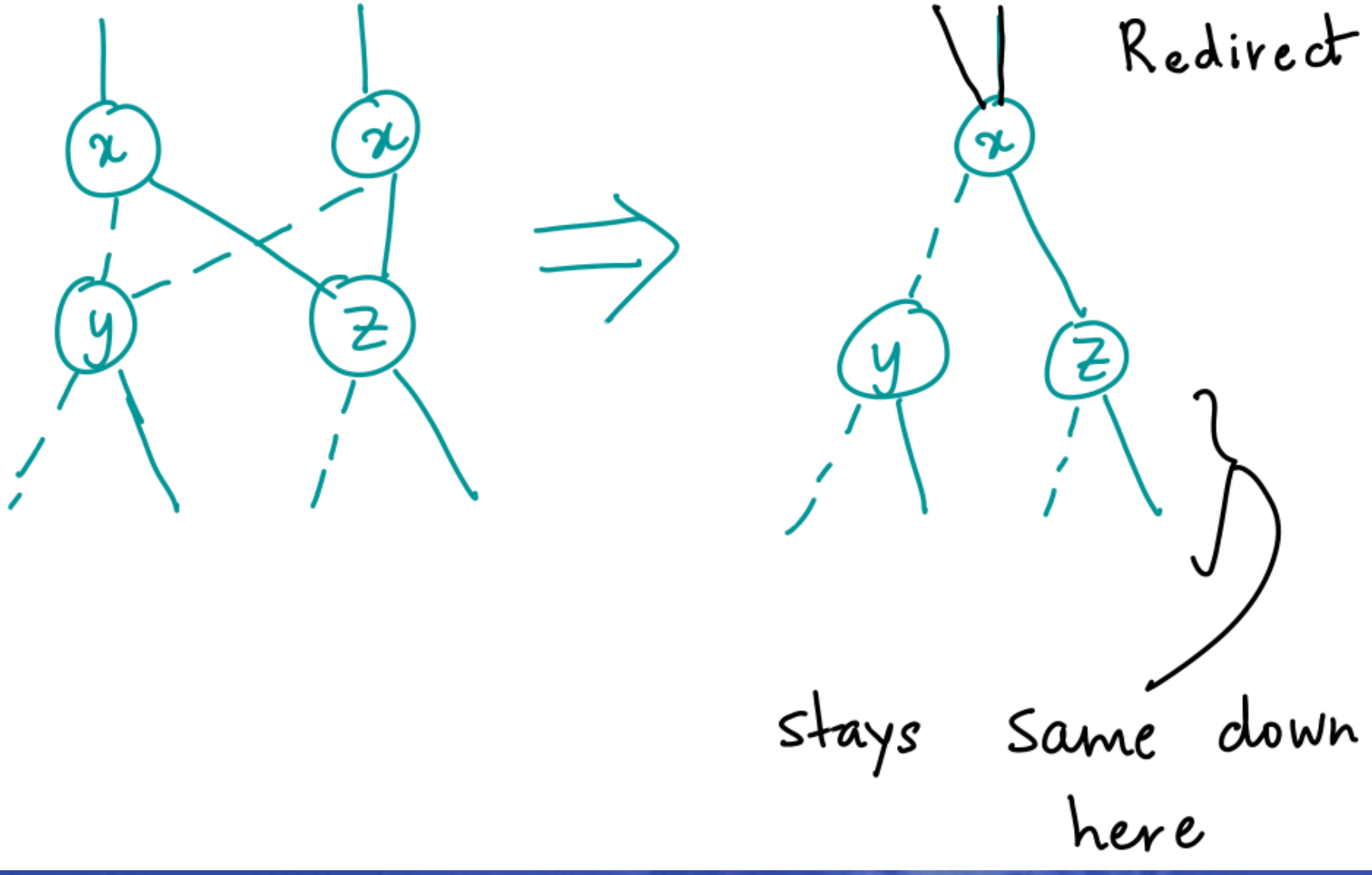
- Identify Redundancies

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

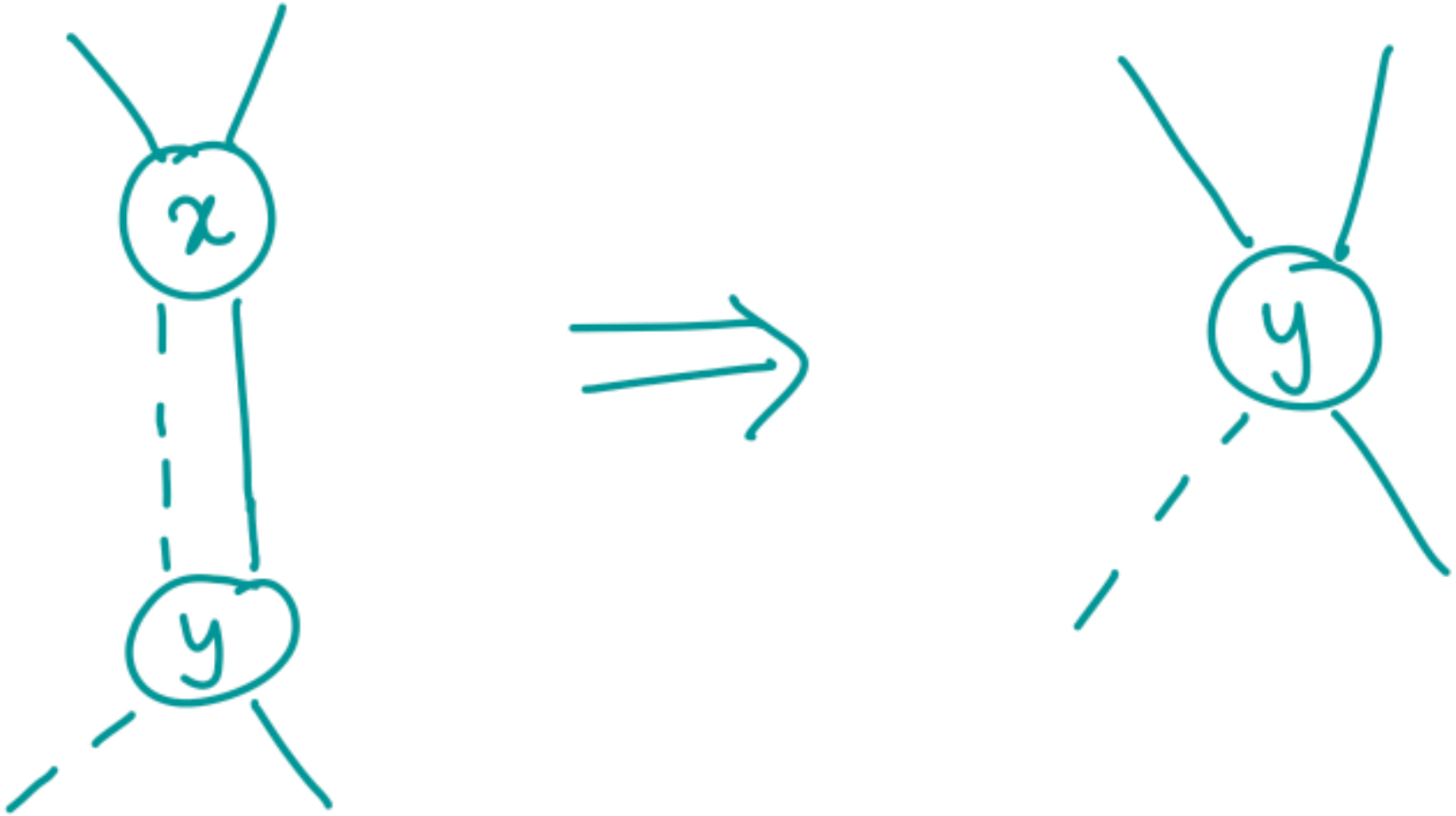
# Merge equivalent leaves


"a" is either 0 or 1

# Merge isomorphic nodes

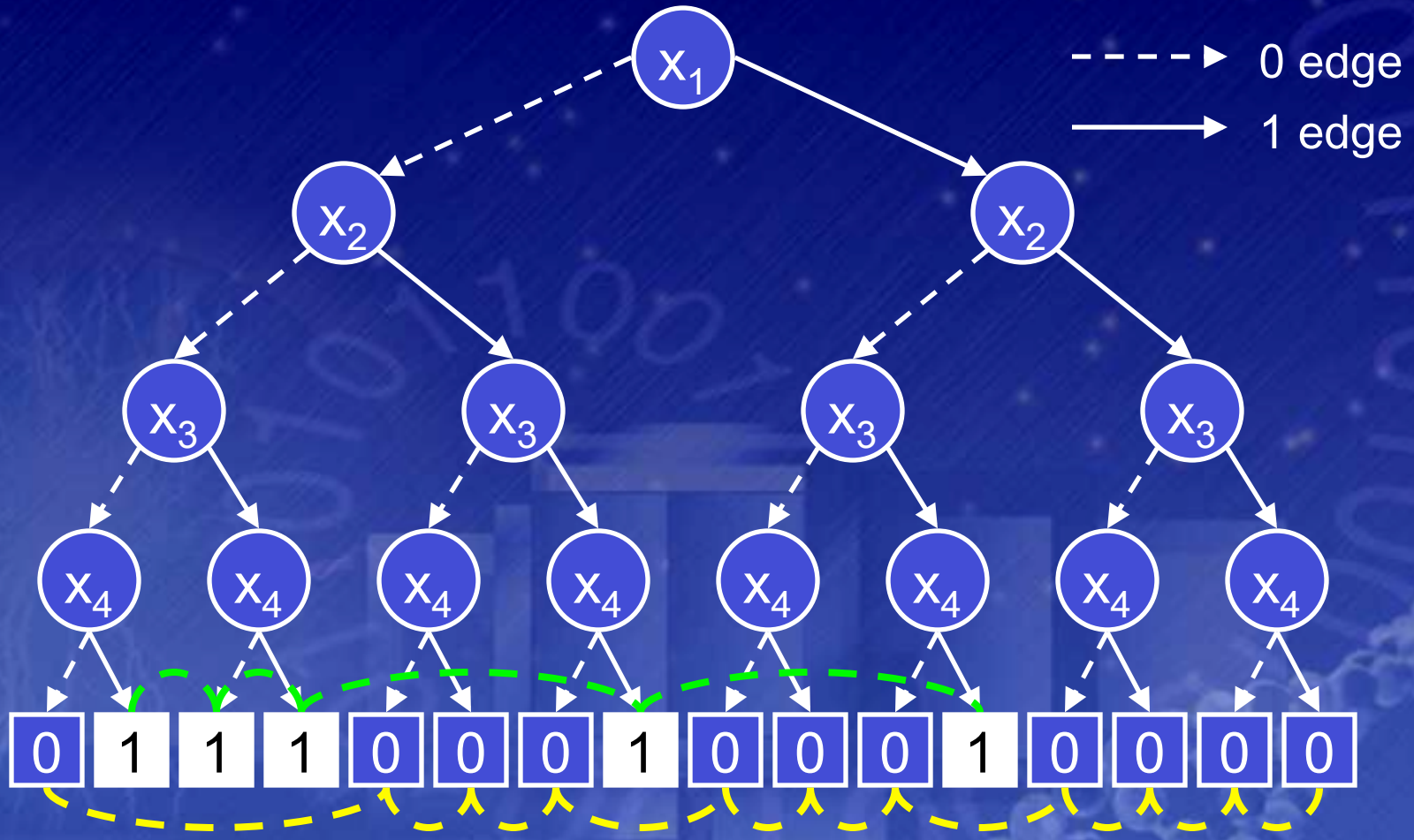


# Eliminate redundant tests



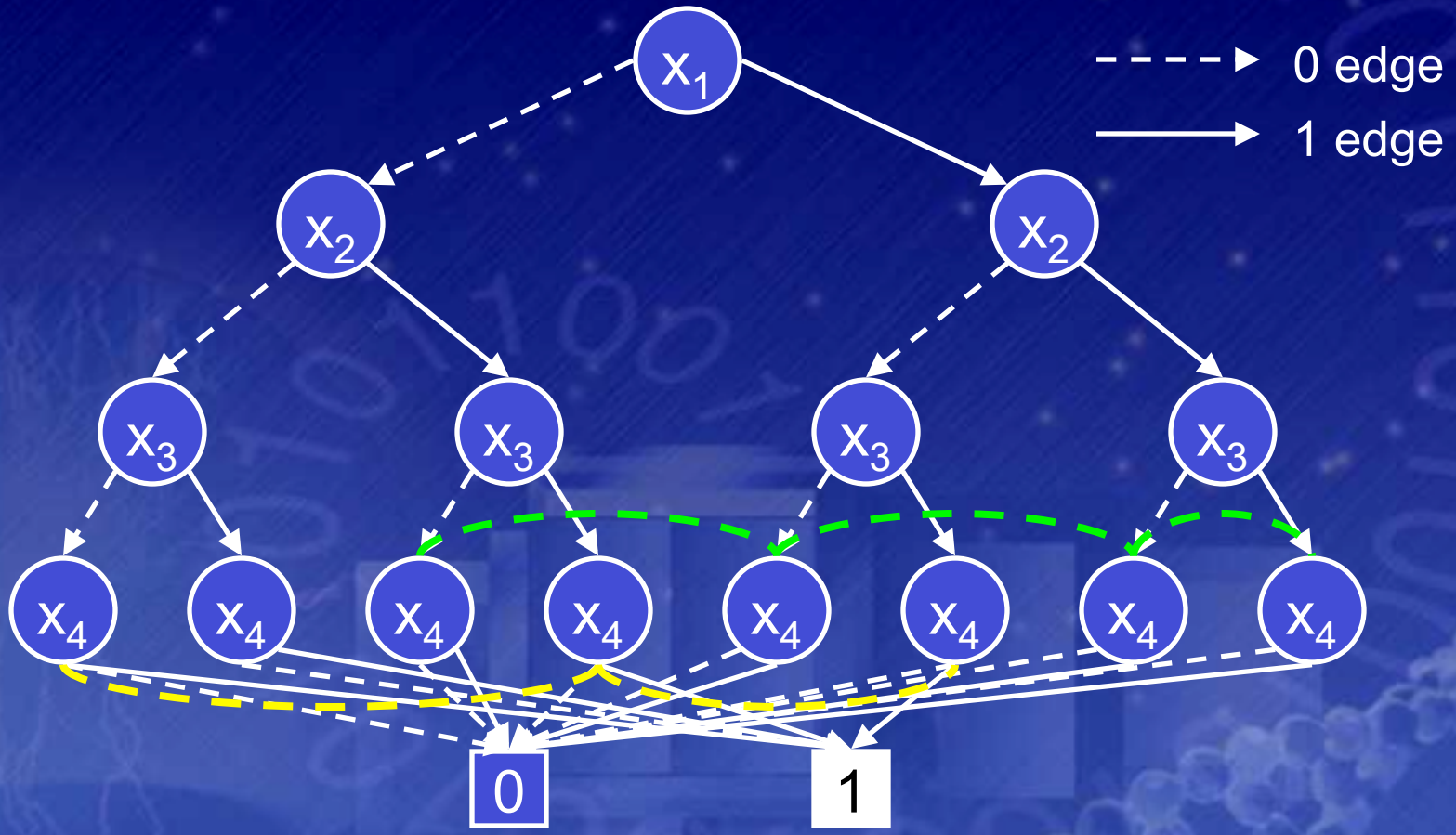
# Binary Decision Diagrams

- Collapse redundant nodes.



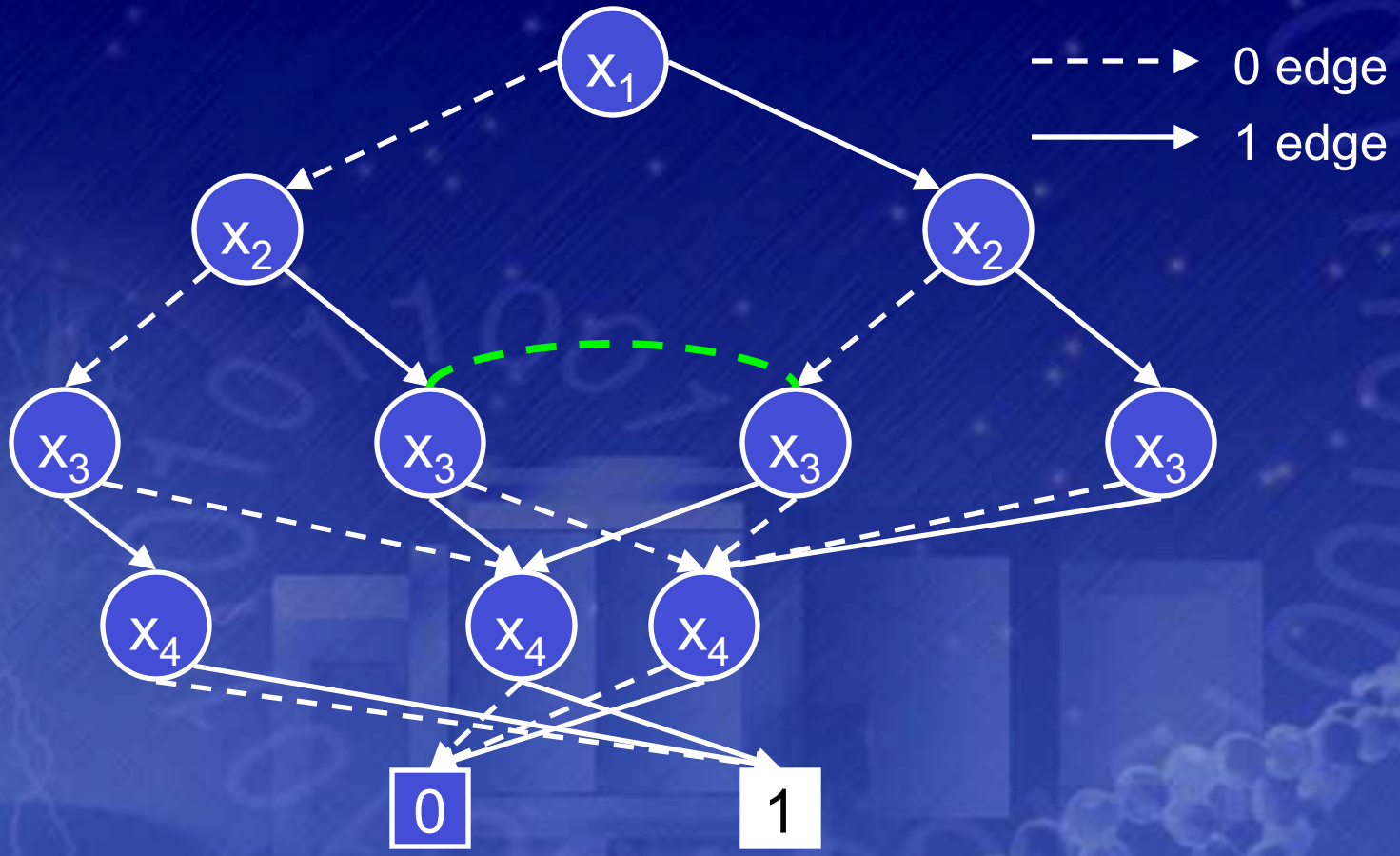
# Binary Decision Diagrams

- Collapse redundant nodes.



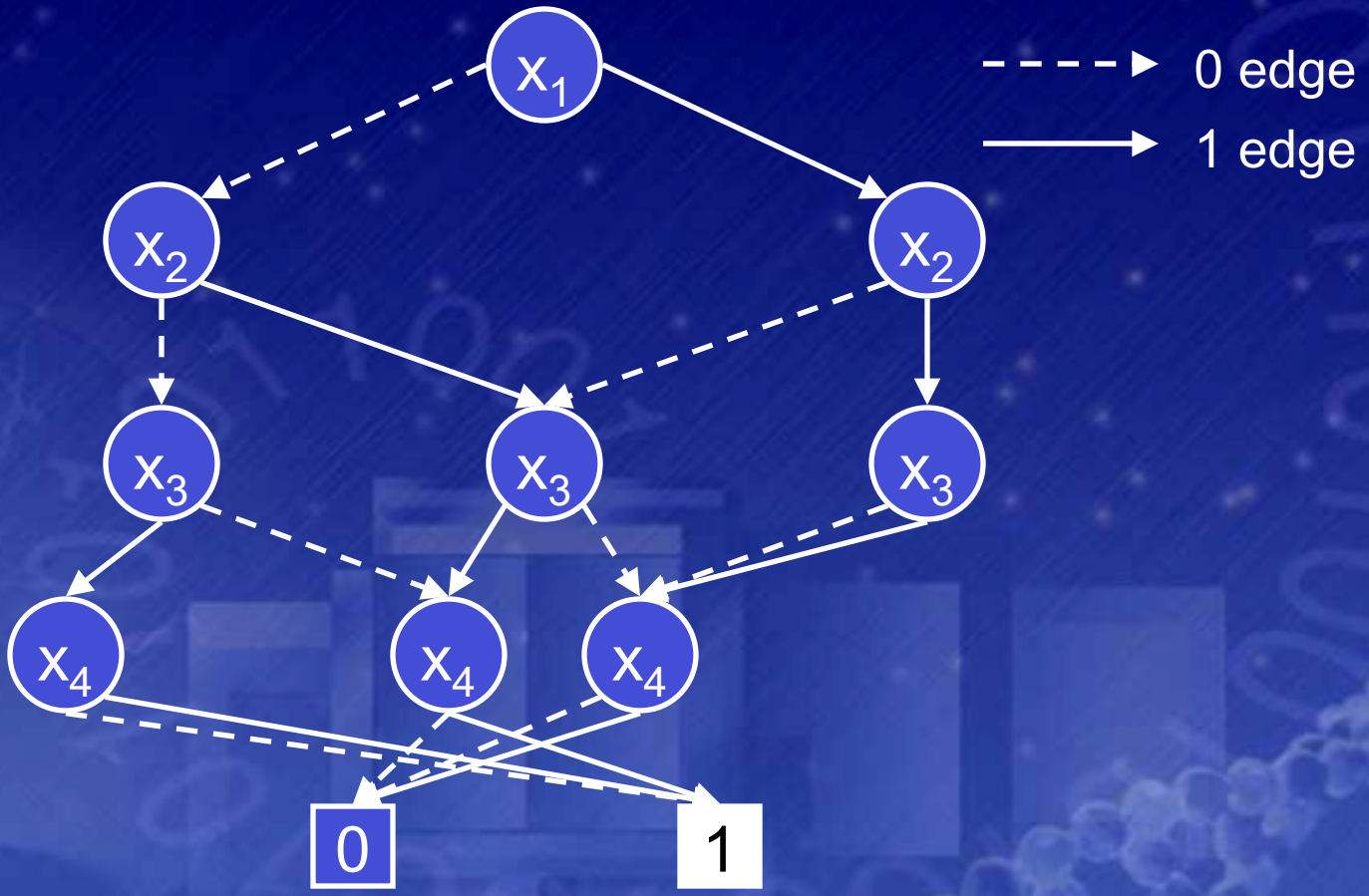
# Binary Decision Diagrams

- Collapse redundant nodes.



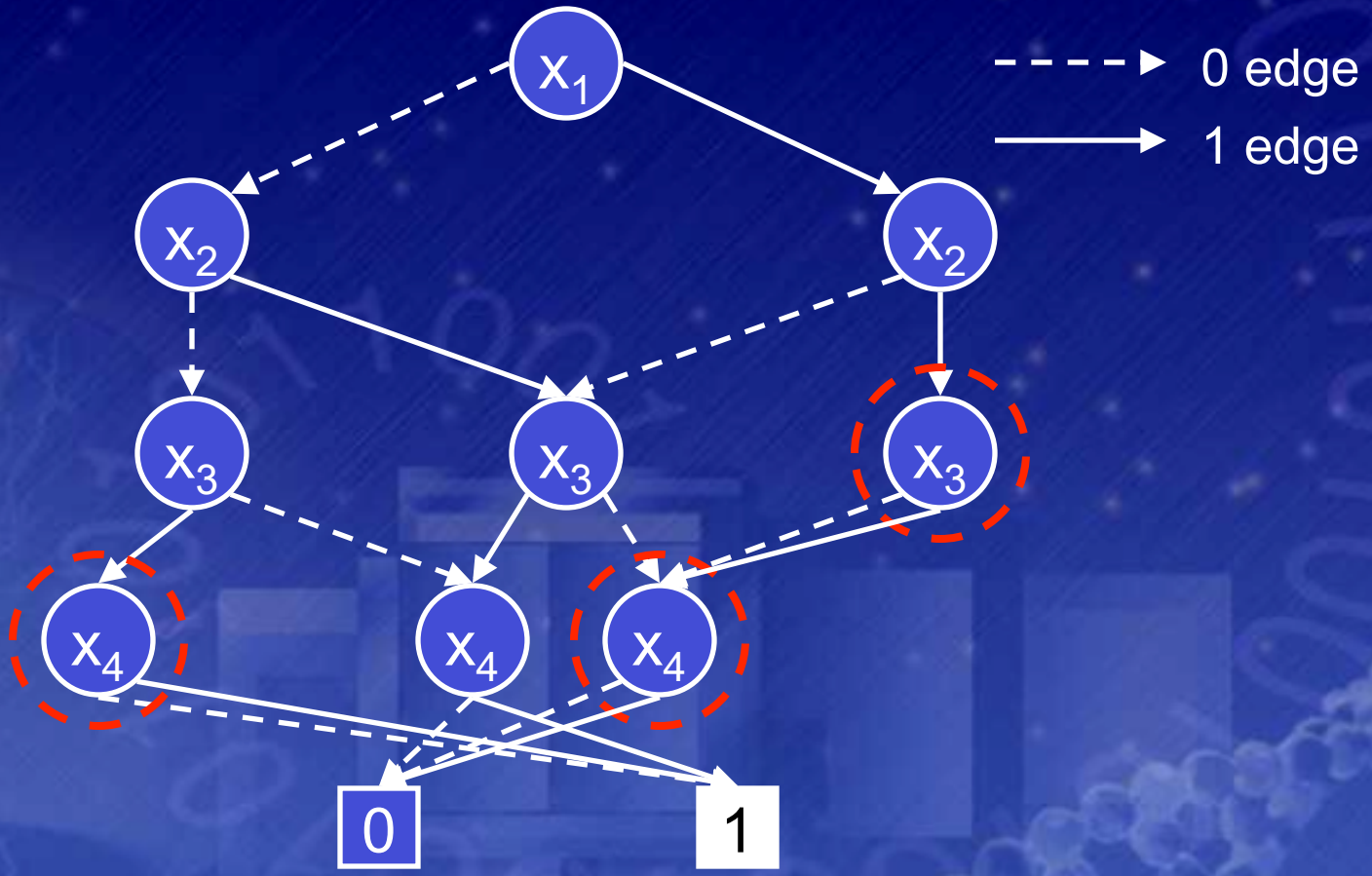
# Binary Decision Diagrams

- Collapse redundant nodes.



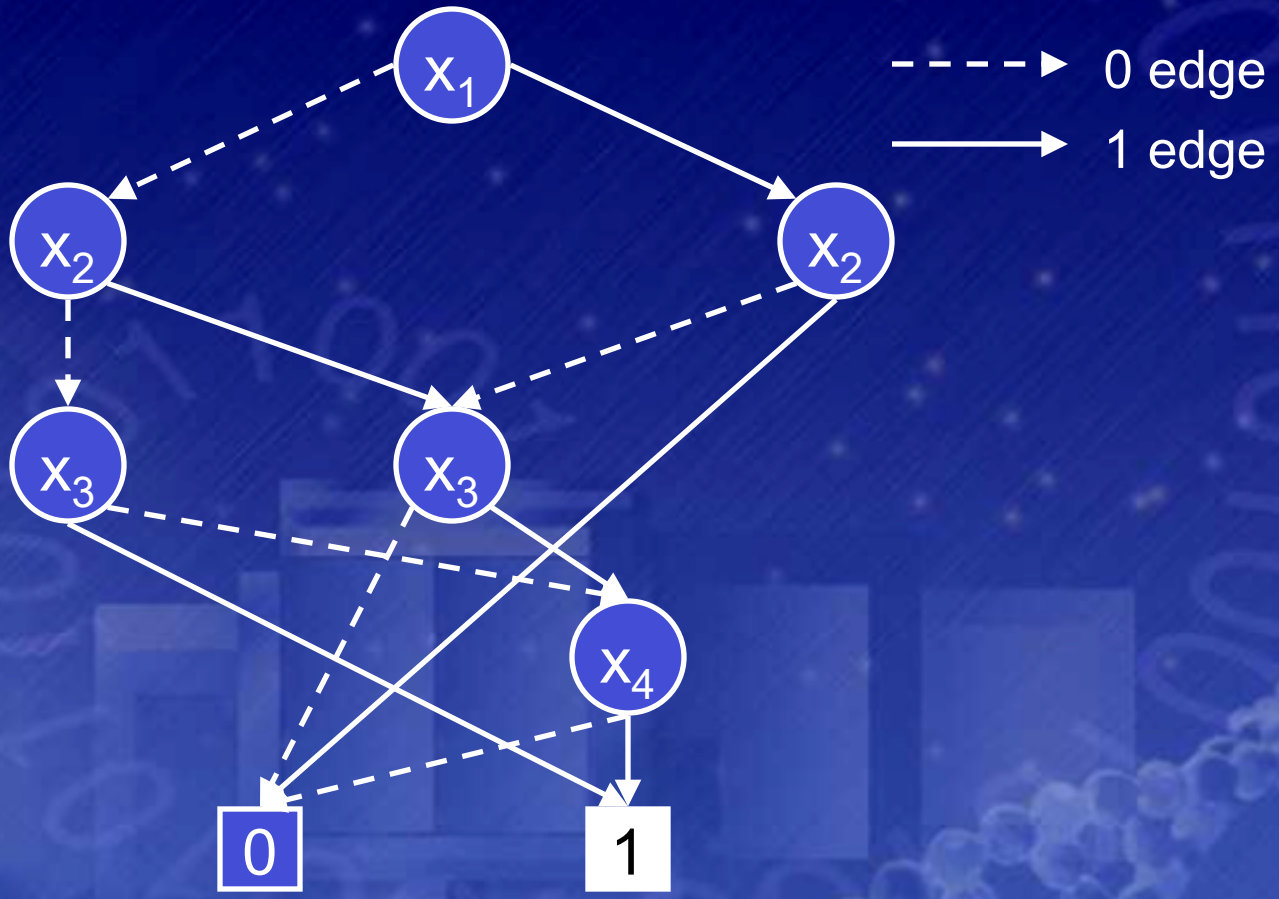
# Binary Decision Diagrams

- Eliminate unnecessary nodes.



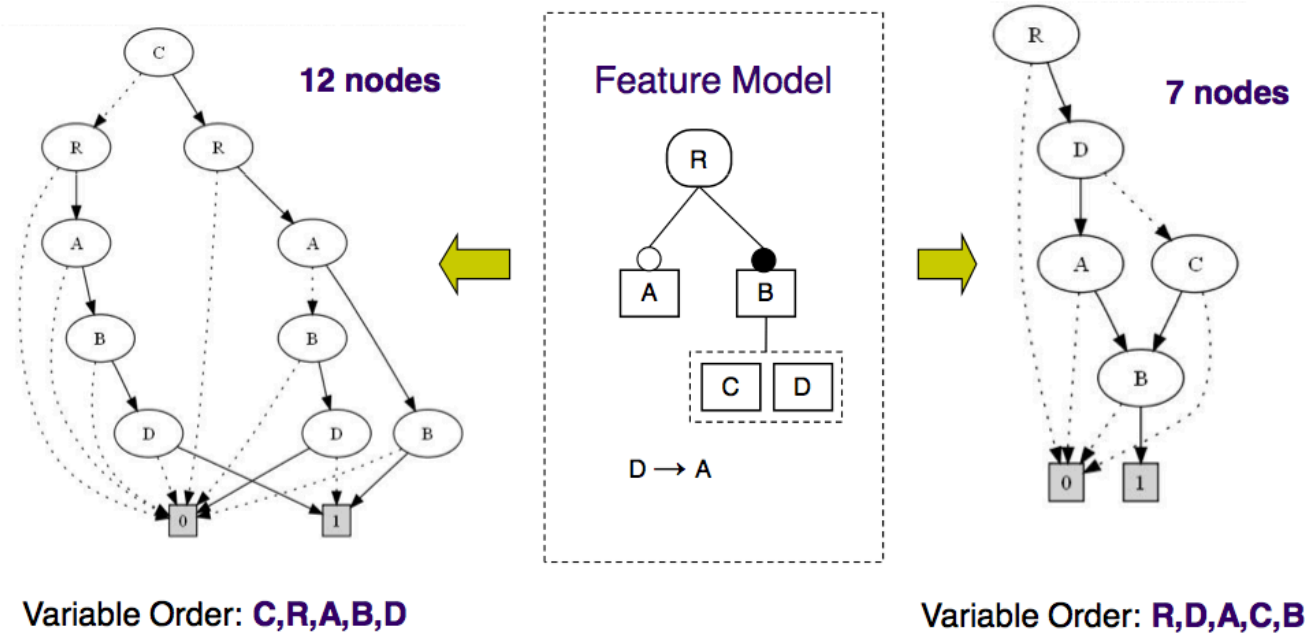
# Binary Decision Diagrams

- Eliminate unnecessary nodes.



# Binary Decision Diagrams

- The size of the BDD is very sensitive to the order of the BDD variables
  - **Two equivalent BDDs for the same feature model**




*[Mendonca, slide]*

# Constraint Satisfaction Problem

**Given**:

- **Variables** $x_1, \ldots, x_n$,
- **Domains** $D_1, \ldots, D_n$,

**Constraint Satisfaction Problem** (**CSP**):

$$\{\mathcal{C}\ ; x_1 \in D_1, \ldots, x_n \in D_n\}$$

$\mathcal{C}$ – constraints, each on a subsequence of $x_1, \ldots, x_n$.

$(d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n$ is a **solution** to

$$\{\mathcal{C}\ ; x_1 \in D_1, \ldots, x_n \in D_n\}$$

if for every constraint $C \in \mathcal{C}$ on $x_{i_1}, \ldots, x_{i_m}$

$$(d_{i_1}, \ldots, d_{i_m}) \in C.$$

# Constraint Satisfaction Problem: Feature Models

domain: {0, 1}
variables: features
constraints: see before

**Given**:

- **Variables** $x_1, \ldots, x_n$,
- **Domains** $D_1, \ldots, D_n$,

**Constraint Satisfaction Problem** (**CSP**):

$$\{\mathcal{C}\ ; x_1 \in D_1, \ldots, x_n \in D_n\}$$

$\mathcal{C}$ – constraints, each on a subsequence of $x_1, \ldots, x_n$.

$(d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n$ is a **solution** to

$$\{\mathcal{C}\ ; x_1 \in D_1, \ldots, x_n \in D_n\}$$

if for every constraint $C \in \mathcal{C}$ on $x_{i_1}, \ldots, x_{i_m}$

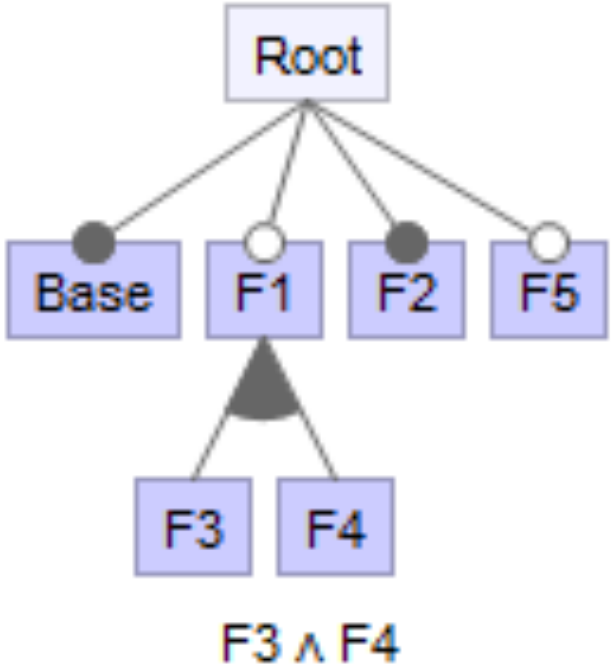
$$(d_{i_1}, \ldots, d_{i_m}) \in C.$$

## Feature Models
## and
## Automated Reasoning

# Operations can be realized using SAT solvers, BDDs or CSP (satisfiability problem)

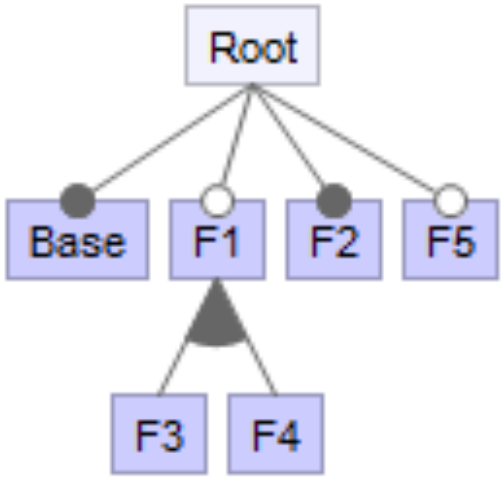
# Consistency

- SAT-Solver
  - SAT(FM)



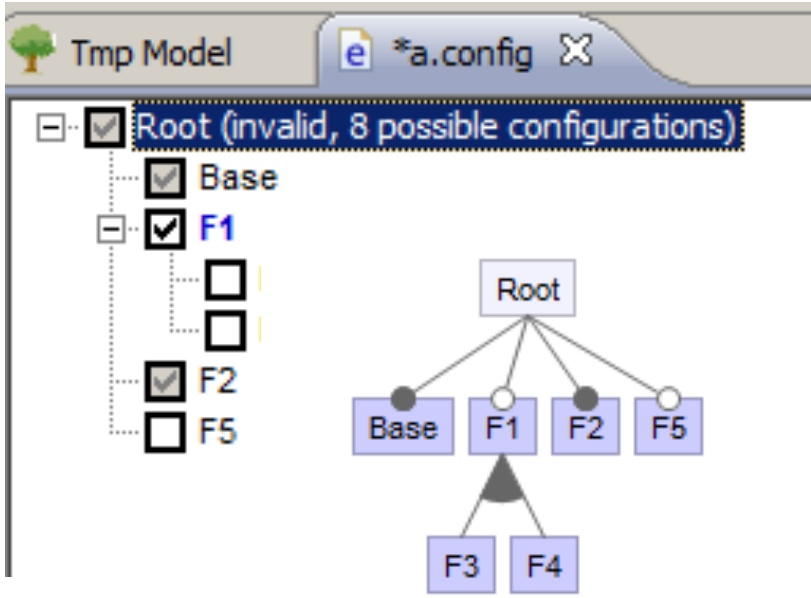
# Core and dead features

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

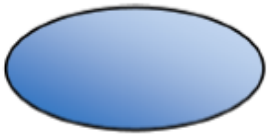
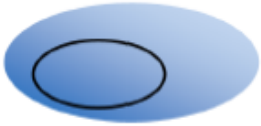
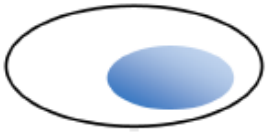
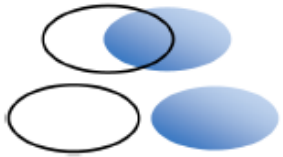


# Partial configuration

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



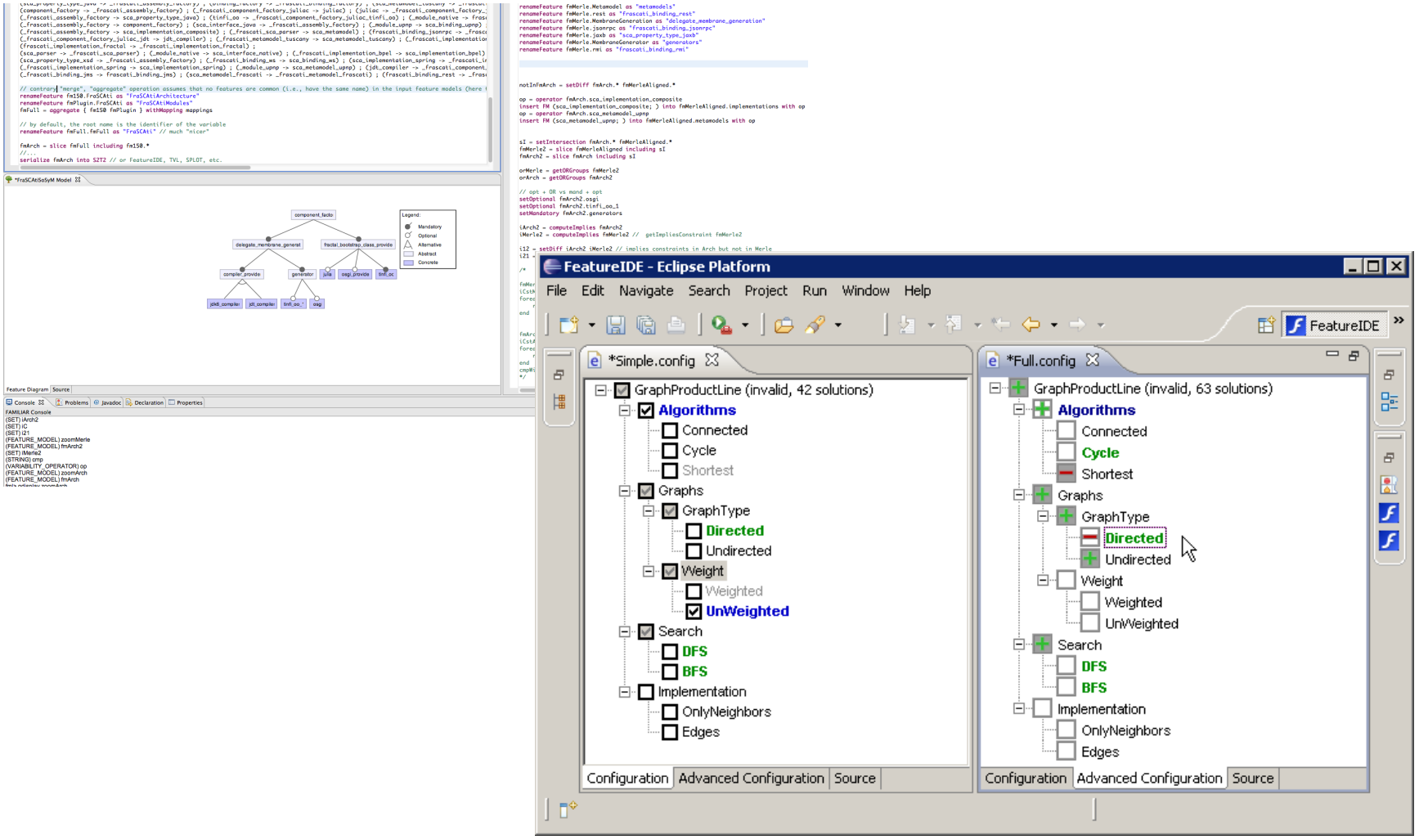
# Relationship between feature models

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

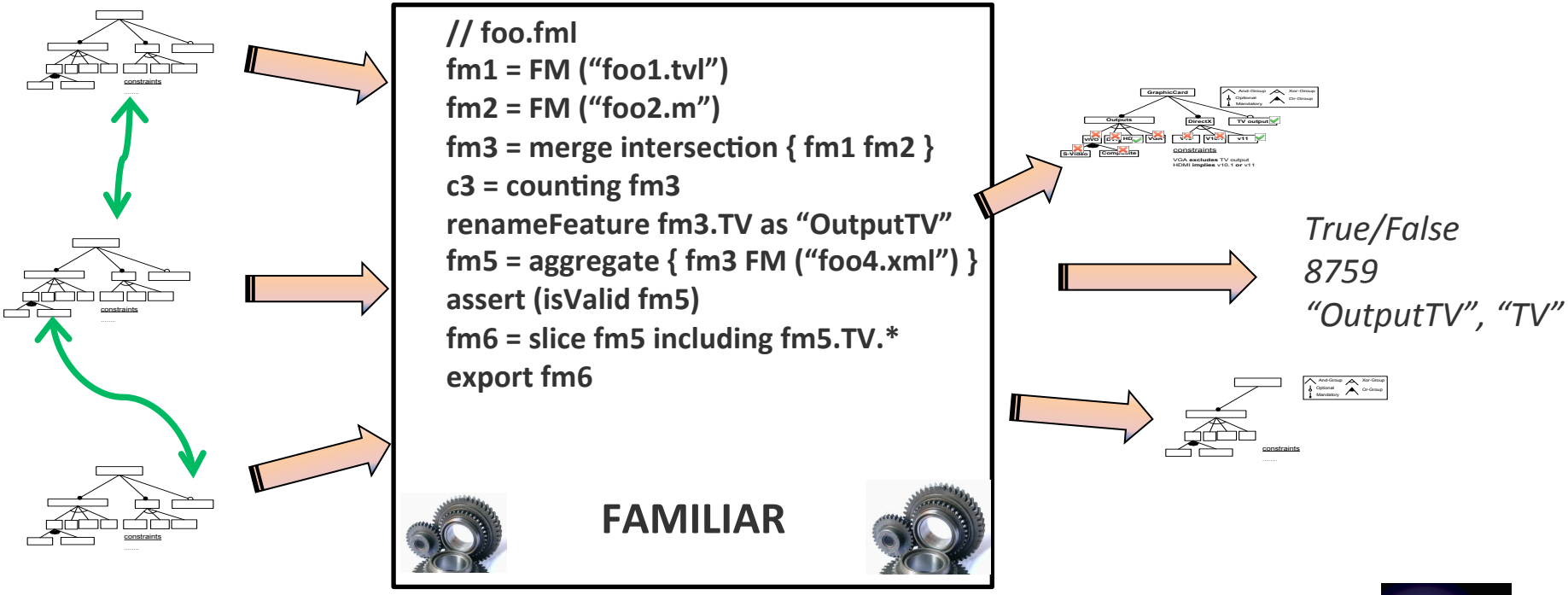
- Refactoring
  - Tautology: (FM1 <=> FM2)
    = not SAT(not (FM1 <=> FM2))

# FAMILIAR language and environment

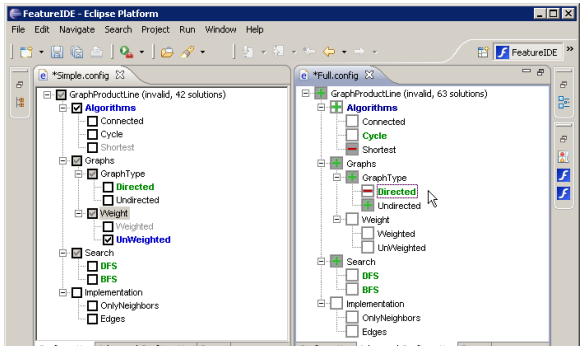
## Reasoning operations implemented in a dedicated language



# FAMILIAR language and environment




**Interoperability** **Language facilities** **Environment**



# Feature Models

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

# Feature model in the SPL framework

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

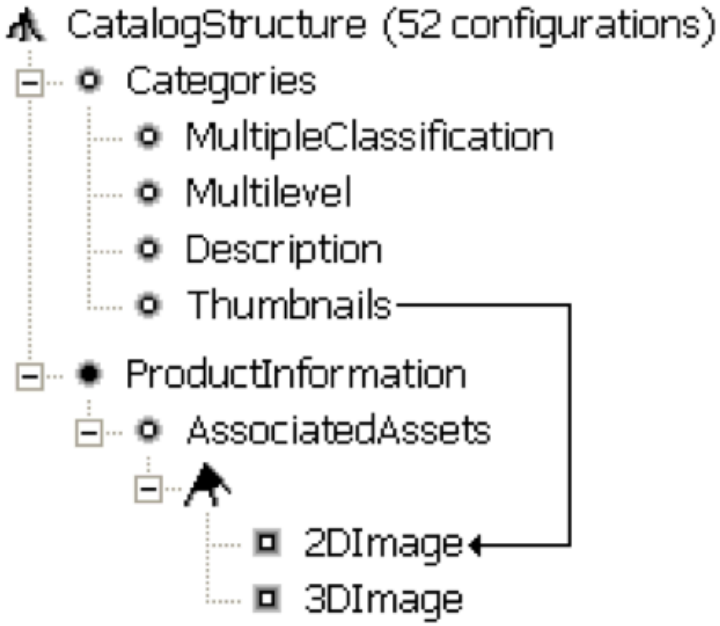
# Feature Models

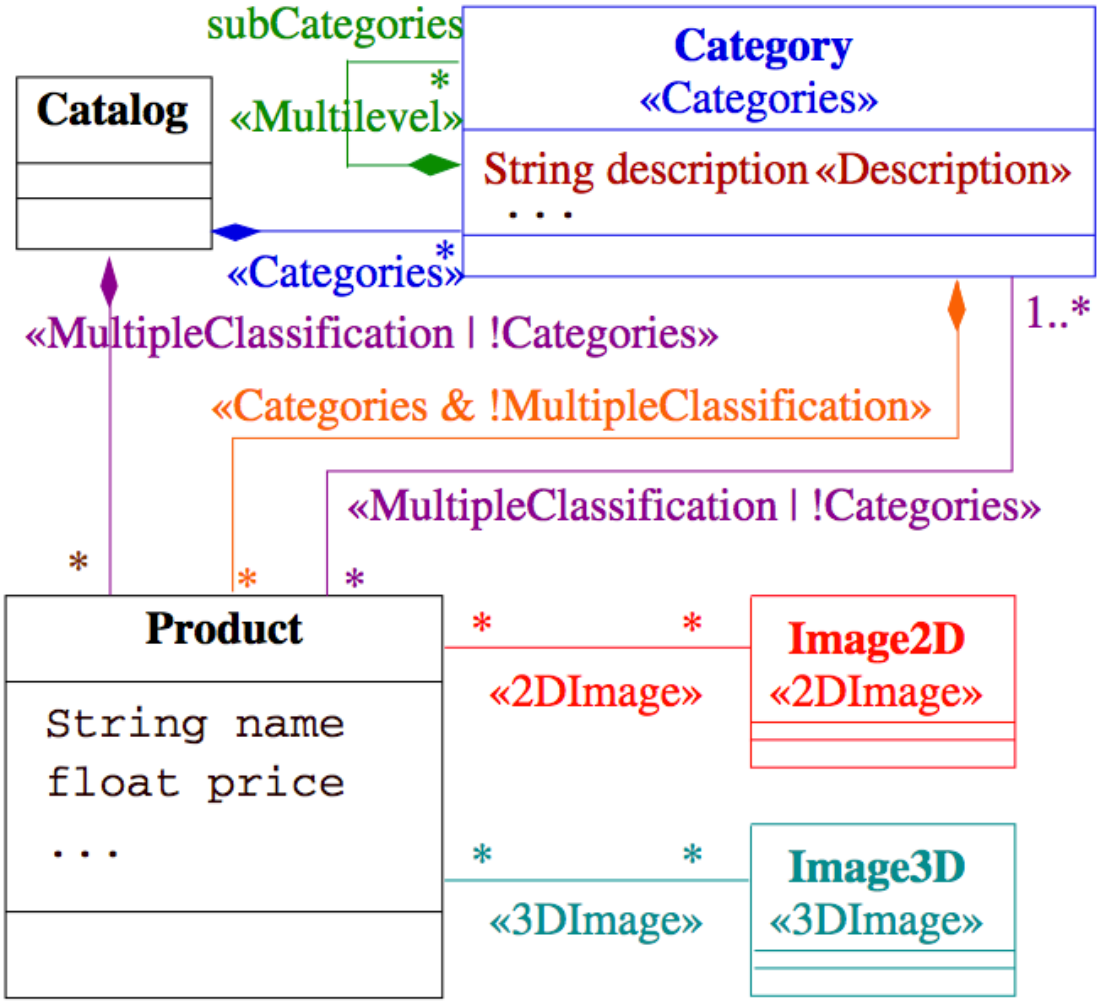
# +

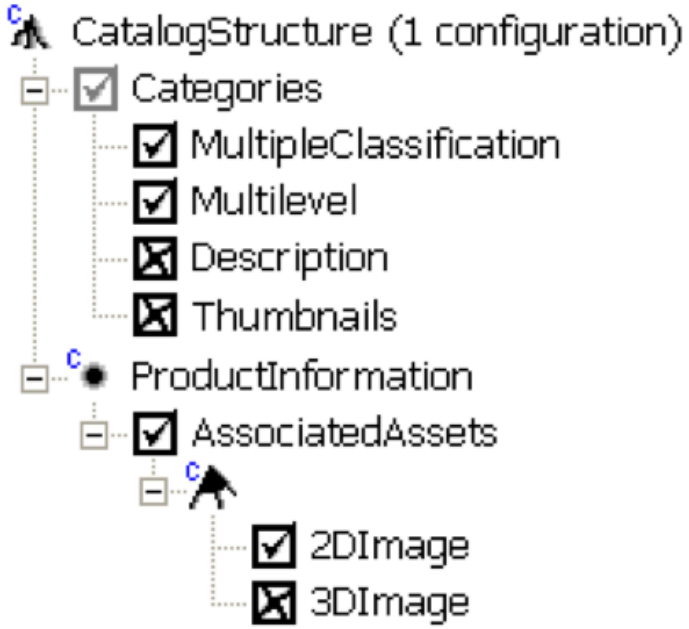
# other artefacs

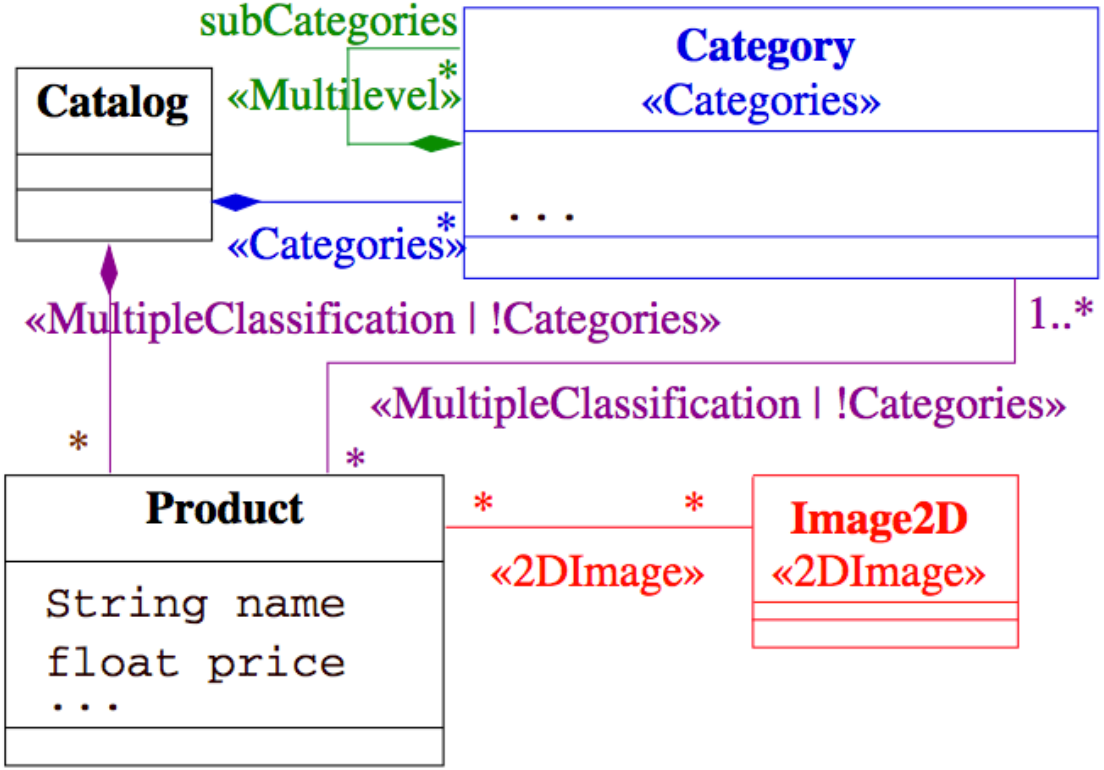
# (what we can do with feature models)

# “features” in feature models: a label

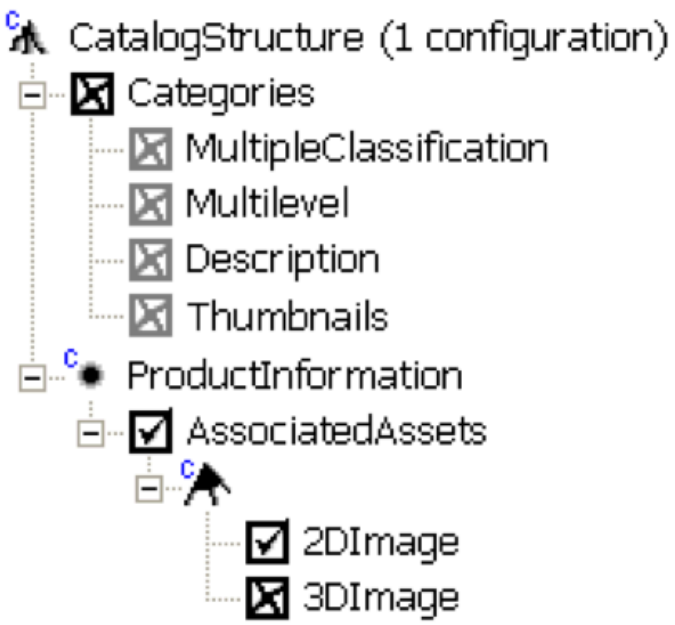

CatalogStructure (52 configurations)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage

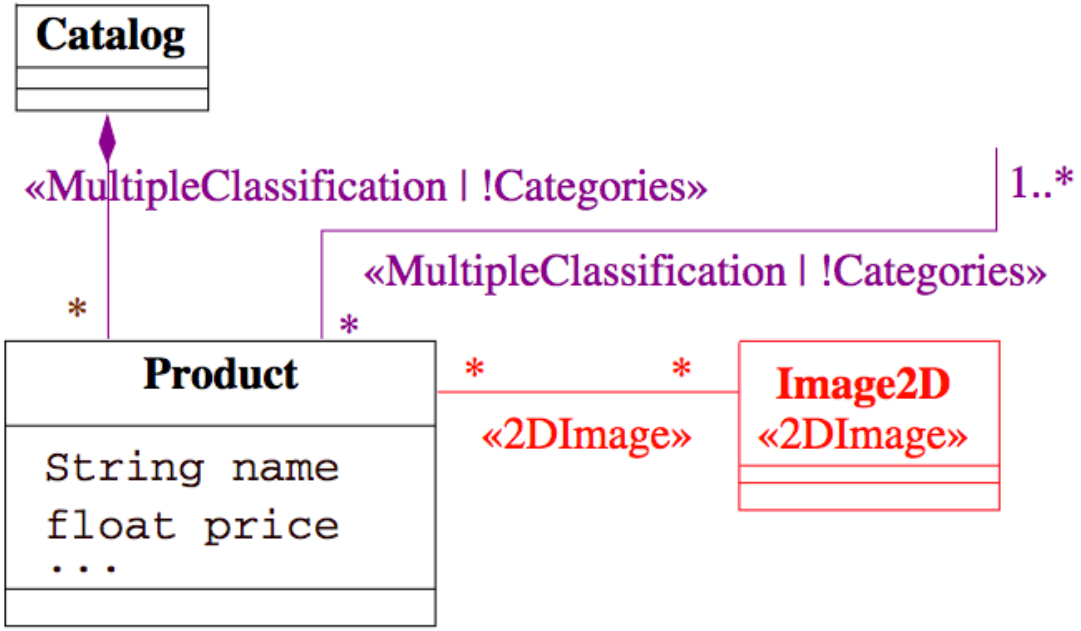

subCategories
Catalog
«Multilevel»
Category
«Categories»
String description «Description»
. . .
«Categories»
«MultipleClassification | !Categories»
1..*
«Categories & !MultipleClassification»
«MultipleClassification | !Categories»
Product
String name
float price
. . .
«2DImage»
Image2D
«2DImage»
«3DImage»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


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

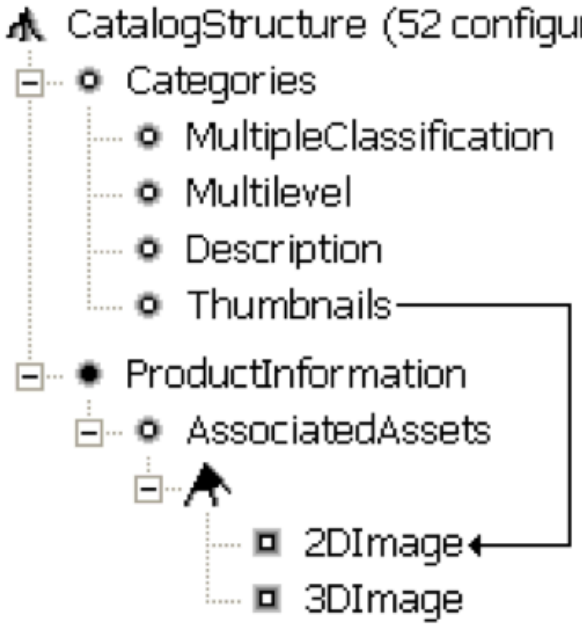

subCategories
Catalog
«Multilevel»
Category
«Categories»
...
«Categories»
«MultipleClassification | !Categories»
1..*
«MultipleClassification | !Categories»
Product
String name
float price
...
«2DImage»
Image2D
«2DImage»

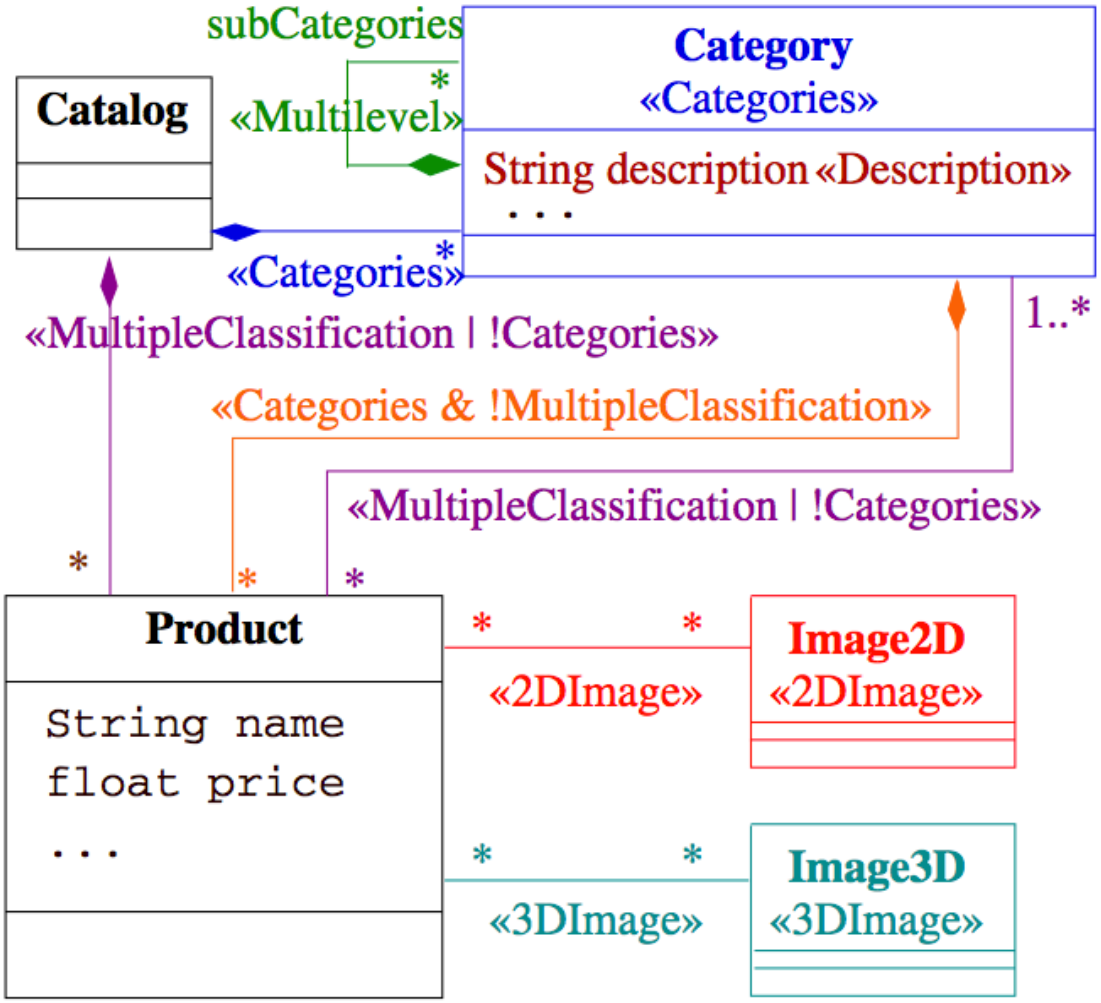
# Ooops


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

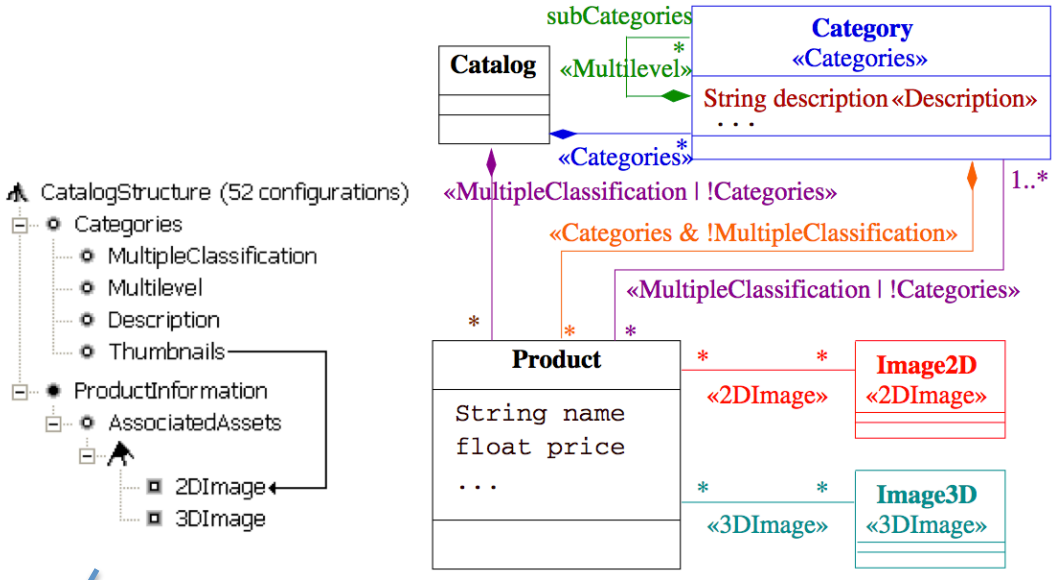

Catalog
«MultipleClassification | !Categories»
1..*
«MultipleClassification | !Categories»
*
*
Product
String name
float price
...
*
*
«2DImage»
Image2D
«2DImage»

# Safe composition? No!



# Safe composition: how does it work?



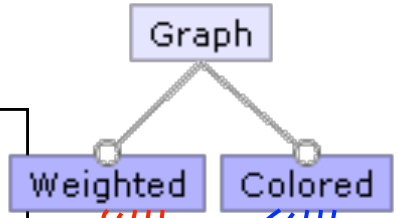

$$
\begin{array}{ll}
q_{FM} = & \\
\textit{root:} & cs \wedge \\
\textit{child-parent:} & (\texttt{ct} \Rightarrow \texttt{cs}) \wedge (\texttt{mc} \Rightarrow \texttt{ct}) \wedge (\texttt{ml} \Rightarrow \texttt{ct}) \wedge \\
 & (\texttt{ds} \Rightarrow \texttt{ct}) \wedge (\texttt{tn} \Rightarrow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

**Another propositional formula**

# 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

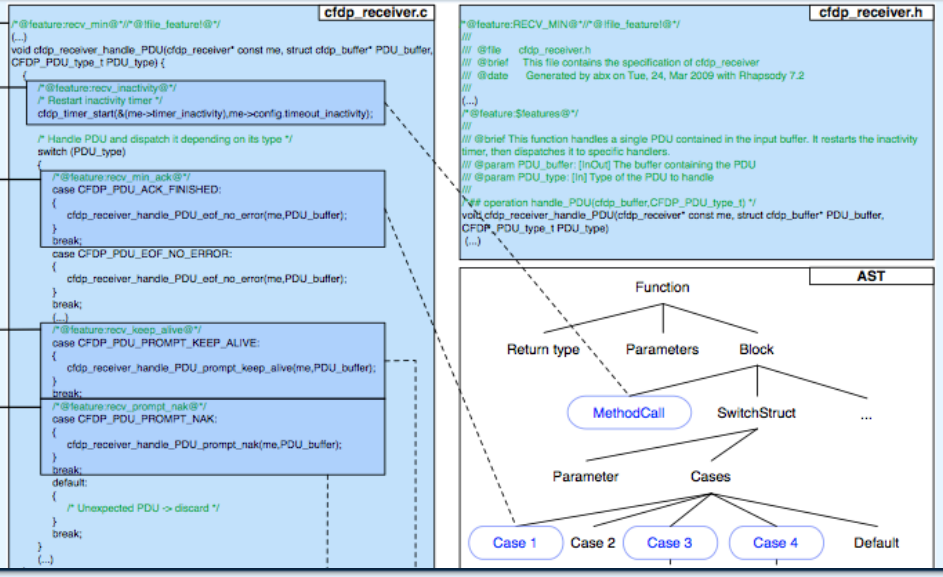
**Tag:** annotate blocks of code with *features*

**Prune:** remove blocks tagged with non-selected features

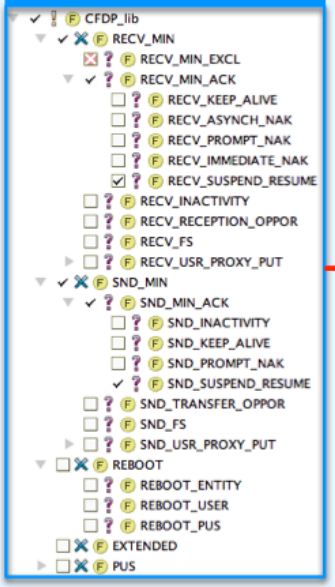
Tagging a whole file with feature *recv_min*

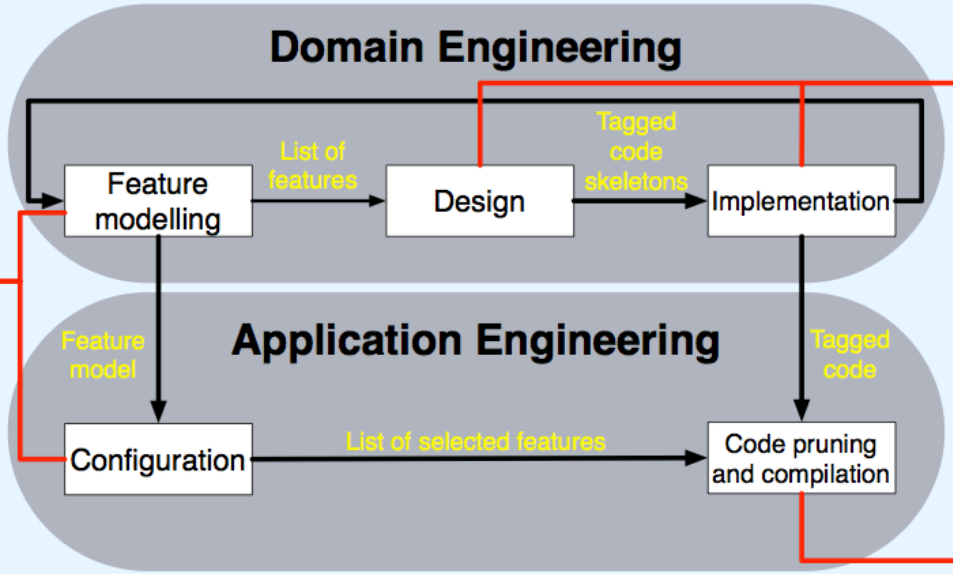
Tagging a function call with feature *recv_inactivity*

Tagging case blocks

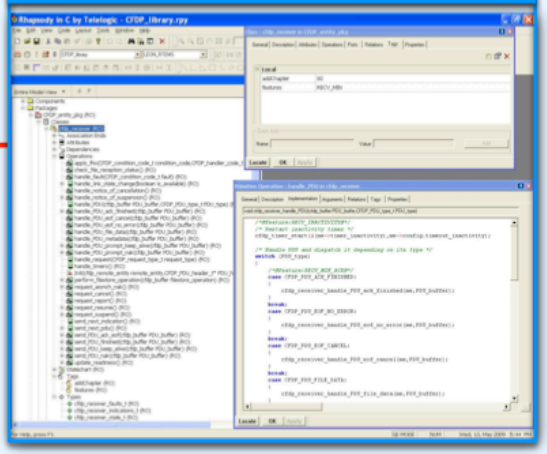



## Feature Modelling Tool




## UML Modelling Tool



## Source Code Pruner

# Feature models in the **real**

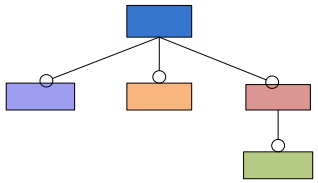
- SPLOT repository
  - more than 100 feature models reported from the literature (various domains)
- Linux feature model
  - worst case: more than 6300 features!
  - eCos, FreeBSD, BusyBox, etc.
- Automotive industry
  - thousands of features
- Wiki matrix
  - wiki engines: ~ 2000 features

# Summary

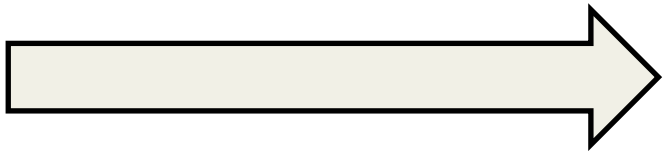
- Variability models

  … are needed for mastering complexity: variability is everywhere, precise specification, automated reasoning

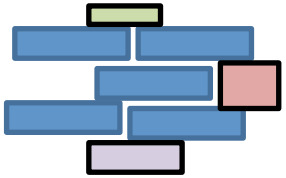
- Feature Models
  - **Formalism and theory**: modeling principles, semantics, logics
  - **Language** to elaborate/build variability models
  - **Reasoning techniques** to analyze properties of an SPL (scalable and automated way)
  - **Applications**: variability models and so what?



**Feature Model**



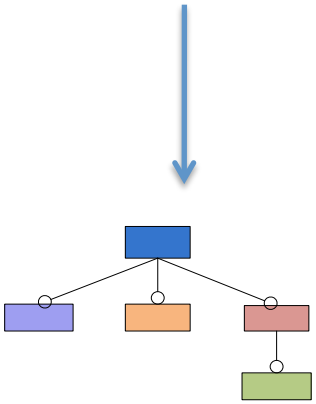
**Mapping**



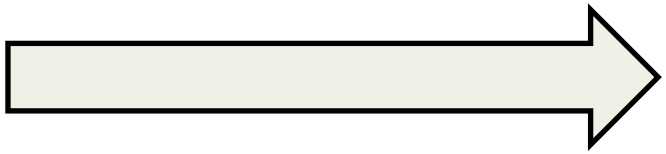
**Reusable Assets**
(e.g., models or source code)

# Summary

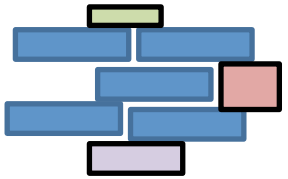
**Language (TVL, FAMILIAR)**



**Feature Model**



**Mapping**

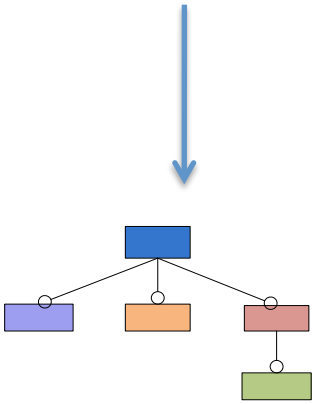


**Reusable Assets**
(e.g., models or source code)

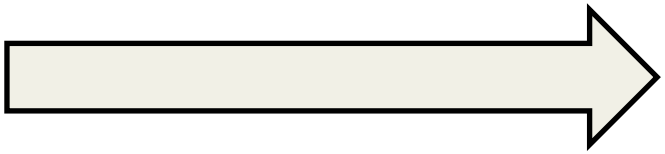
**Automated reasoning**

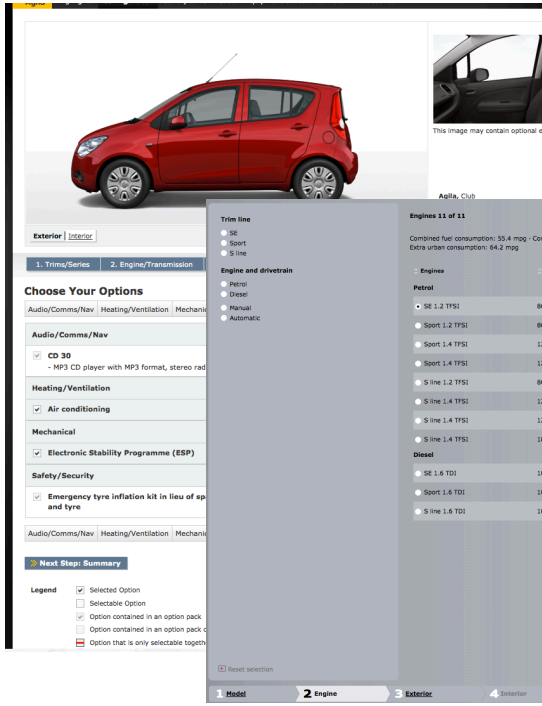
# Summary

**Language (TVL, FAMILIAR)**



**Feature Model**

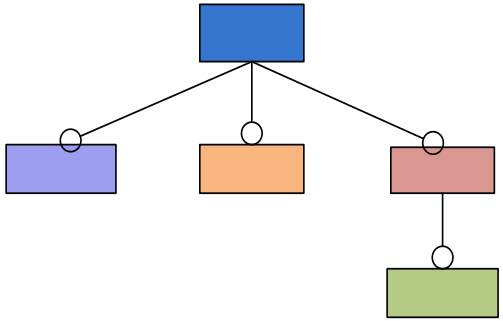




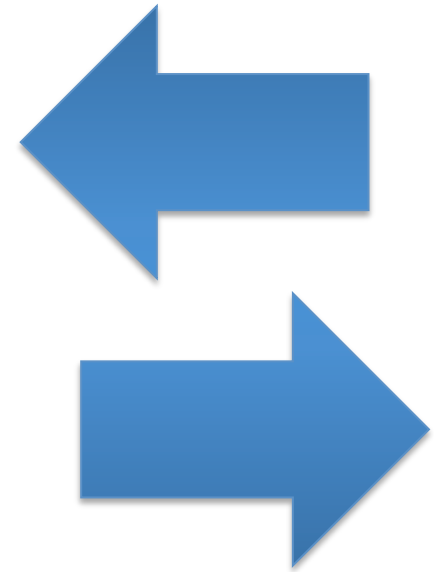

**Automated reasoning**

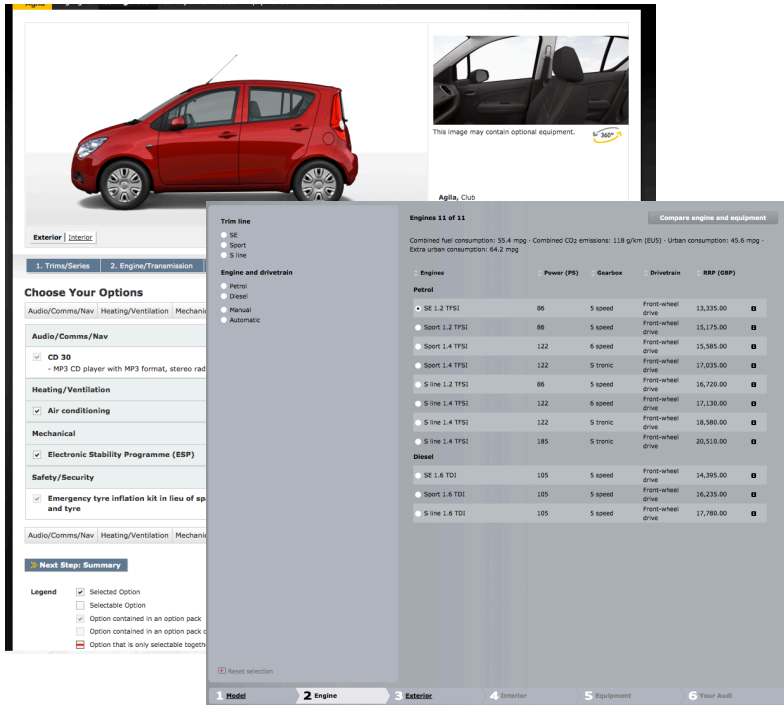
# Running project

- Re-engineer a car configurator
  - we are making some progress, right?



**Variability Model (Feature Model)**





# Homework for the next course

- No homework!
  - No “course” next week, but interactive session for learning feature models, TVL language, etc.
- Feature modeling in practice
  - Domain analysis: the domain of cars
  - Elaboration of a feature model
  - Use of TVL
  - Use of automated reasoning techniques
- First step towards a car configurator