# Automated reasoning about variability models with solvers

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

UNIVERSITÉ DE RENNES 1

istic Informatique Électronique

# Material

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

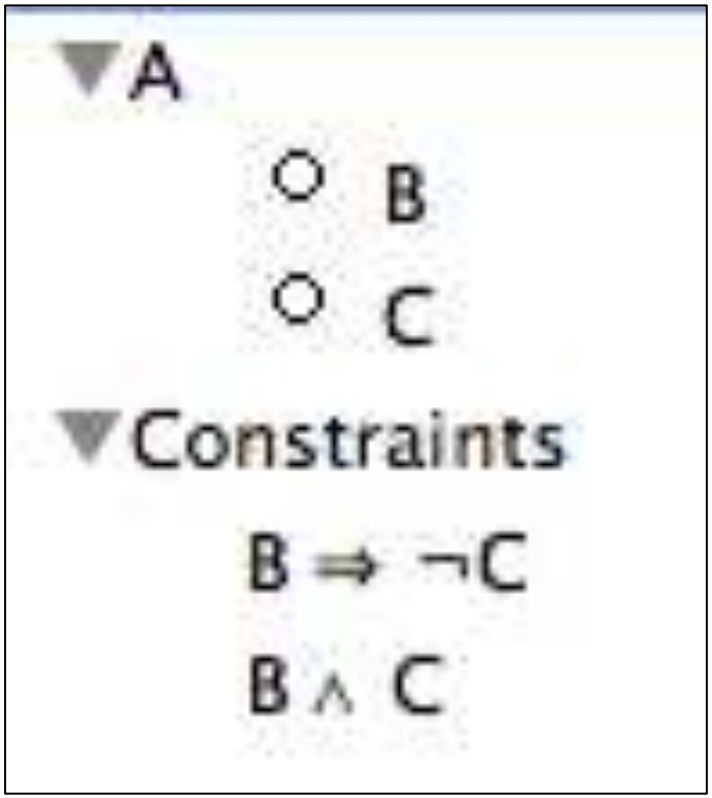
# Plan

- **Feature Models**
  - **Defacto standard for modeling product lines and variability**
  - **Syntax, semantics, automated reasoning, synthesis**

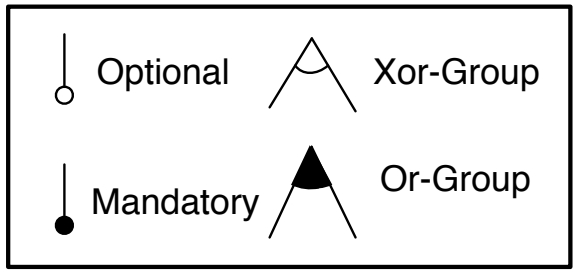
# Contract

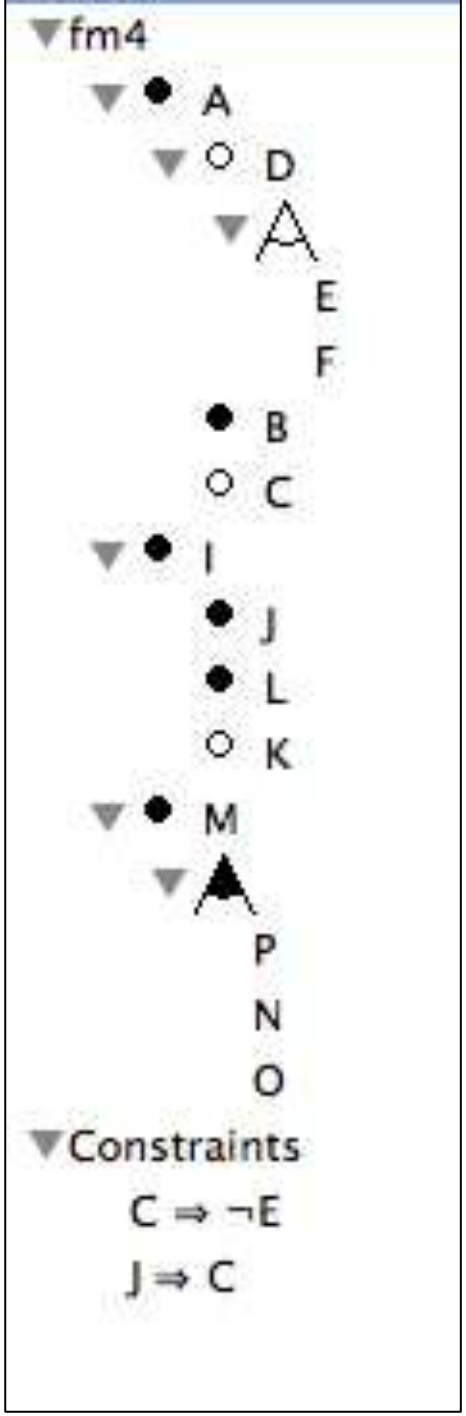
- The idea of software product lines and variability
  - You will be able to recognize this class of systems
  - Aware of the complexity, the specific development process, and existing techniques
- **Feature modeling**
  - **A widely used formalism for modeling product lines and configurable systems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

I want to analyze and play with my specification!




**Empty set of configurations**






**Dead feature**

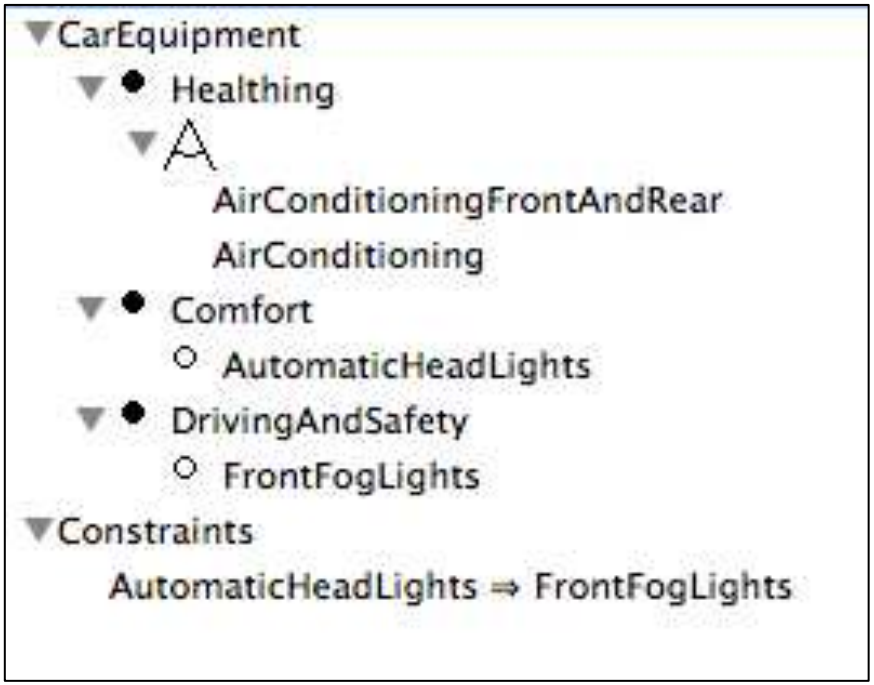
**False optional feature**

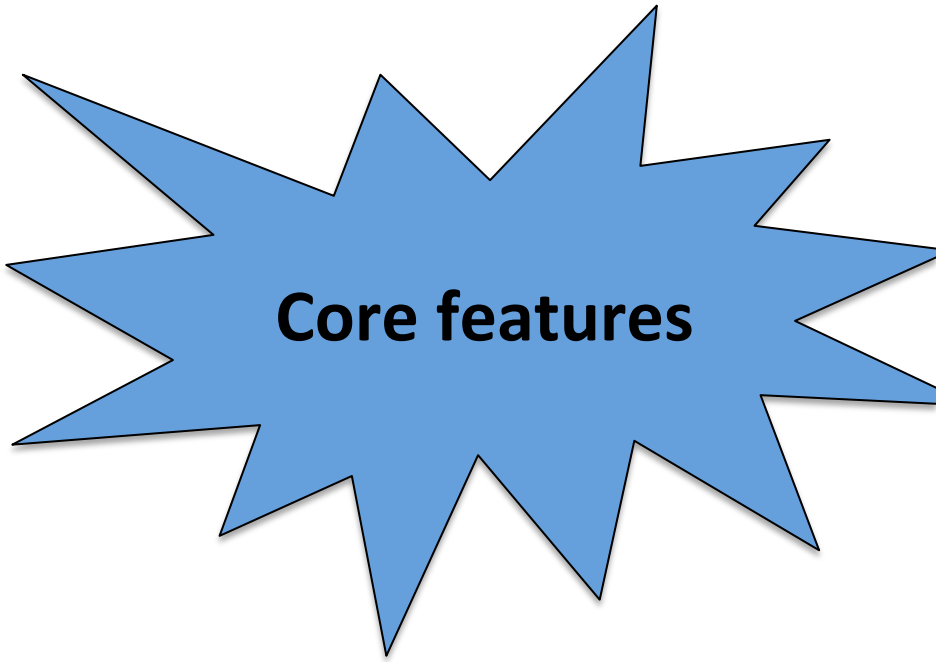
Optional

Xor-Group

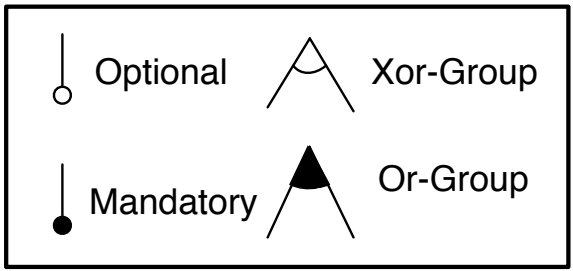
Mandatory

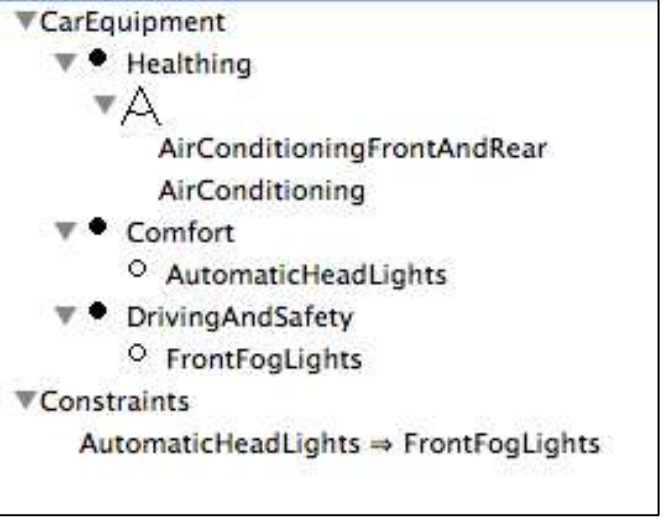
Or-Group

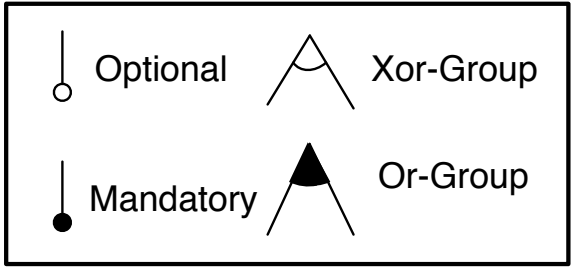


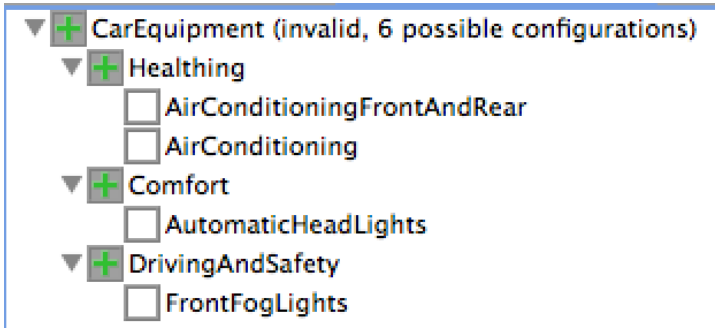



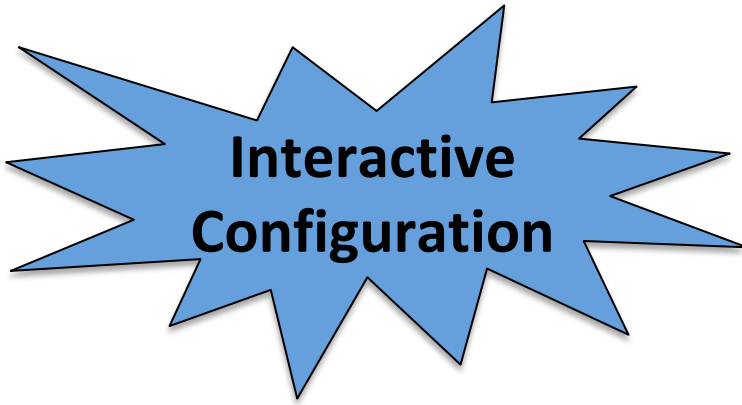

{CarEquipment, Comfort, DrivingAndSafety, Healthing}

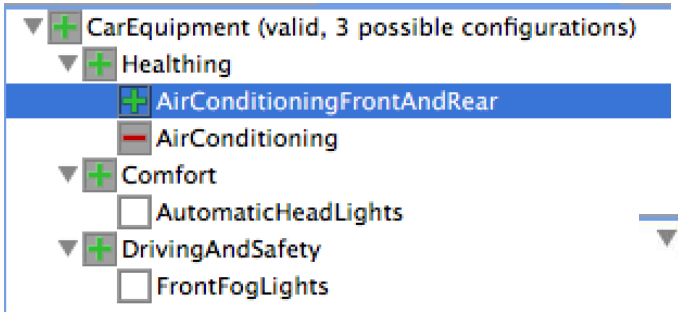


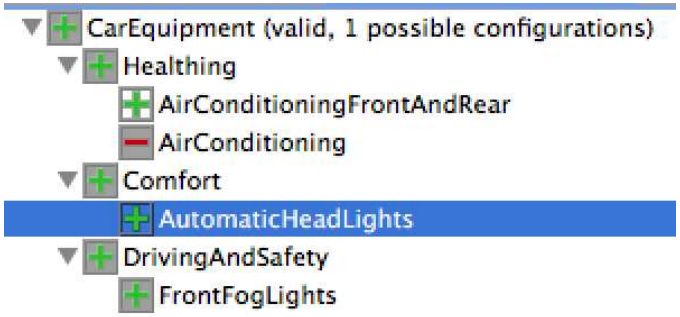

CarEquipment
Healthing
AirConditioningFrontAndRear
AirConditioning
Comfort
AutomaticHeadLights
DrivingAndSafety
FrontFogLights
Constraints
AutomaticHeadLights ⇒ FrontFogLights


Optional
Xor-Group
Mandatory
Or-Group


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


Interactive
Configuration


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


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

## Feature Models and Automated Reasoning

# Benavides et al. survey, 2010

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

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

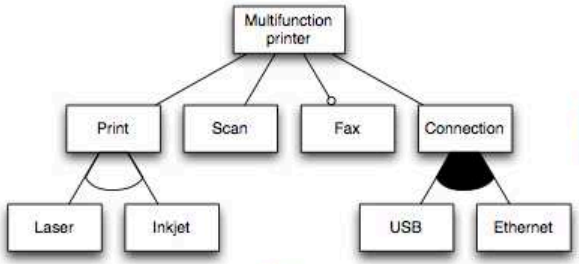
# Decision problems and complexity

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

# How to automate analysis of your feature models?

Binary Decision Diagram (BDD)
SAT solver

# Typical implementations





result

logics

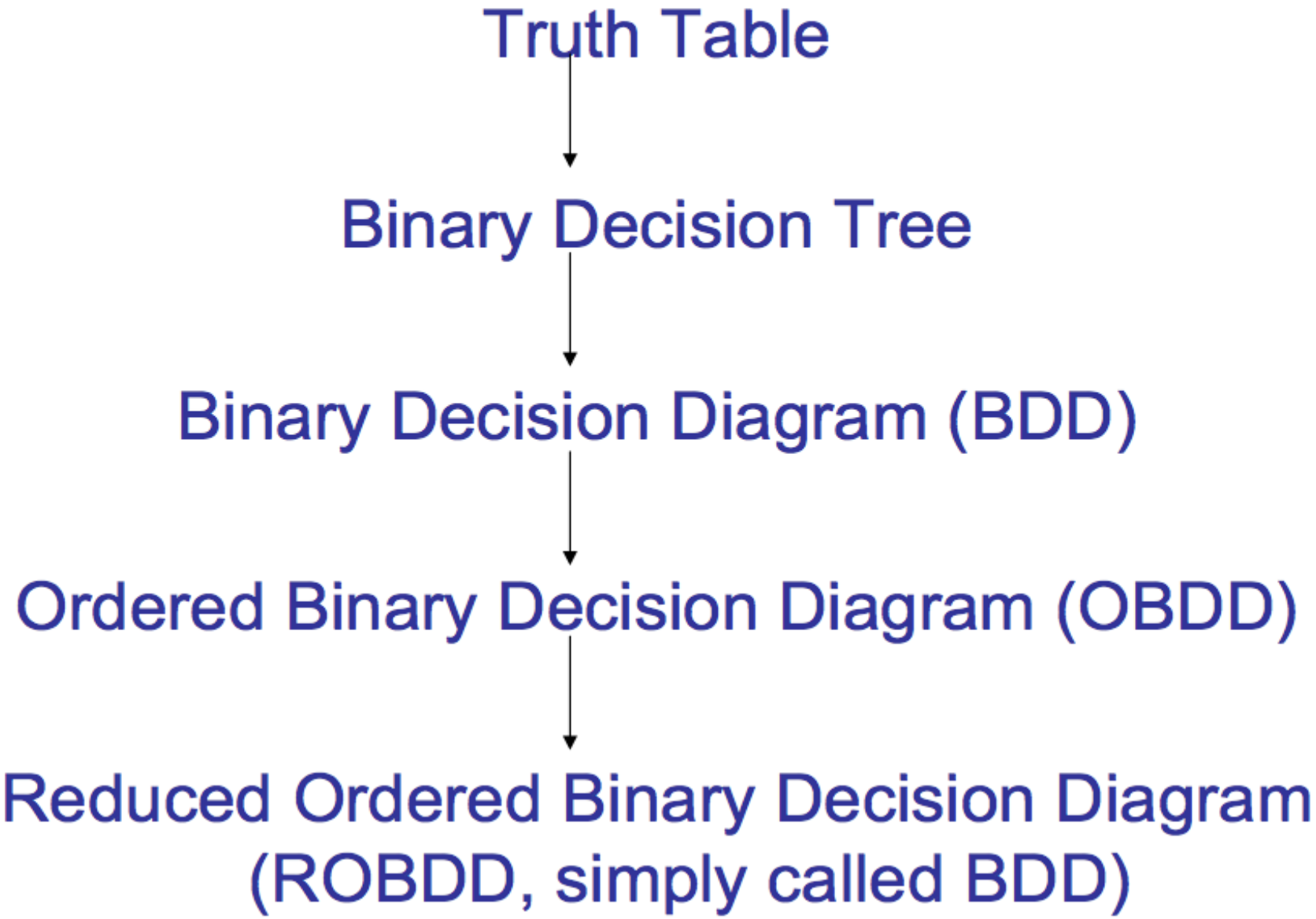
solvers



# 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 |

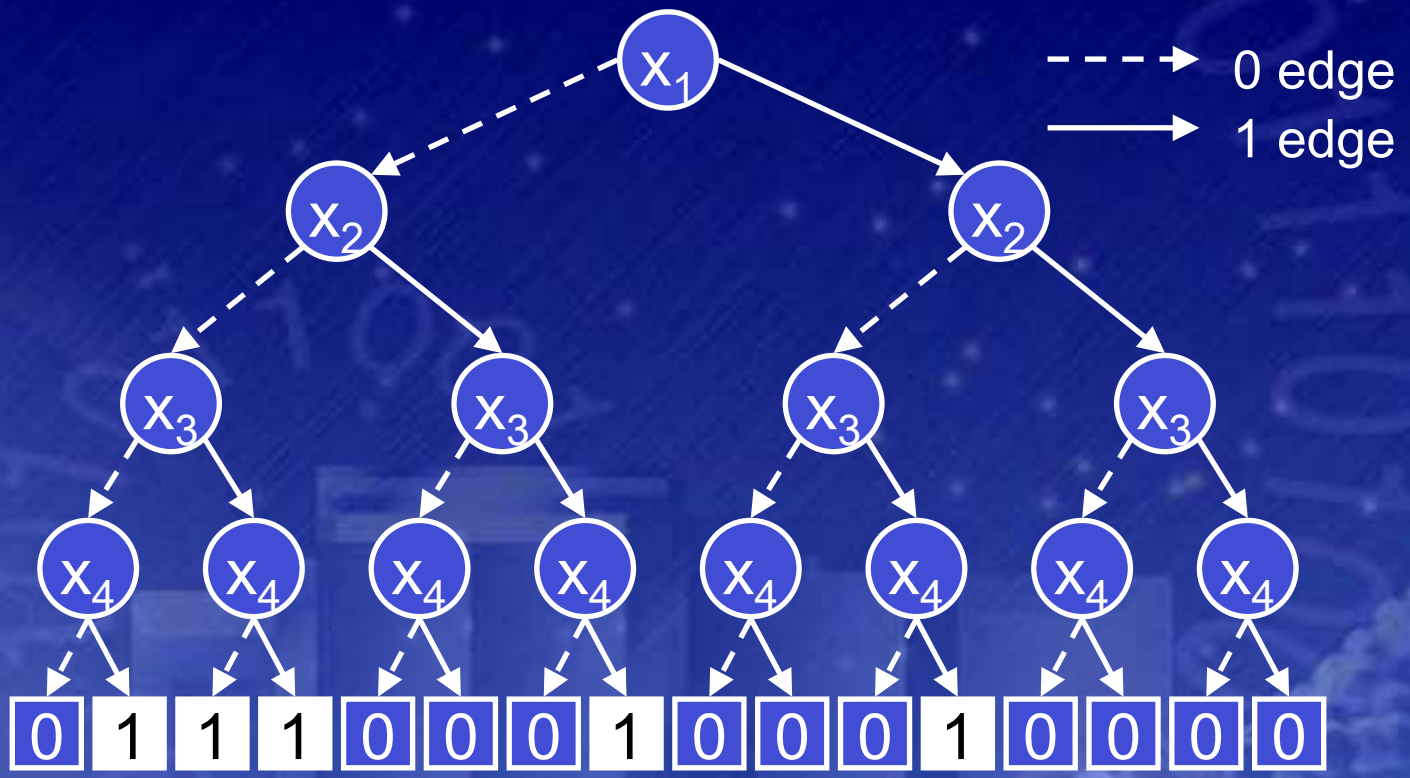
# BDDs from Truth Tables



# 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 |

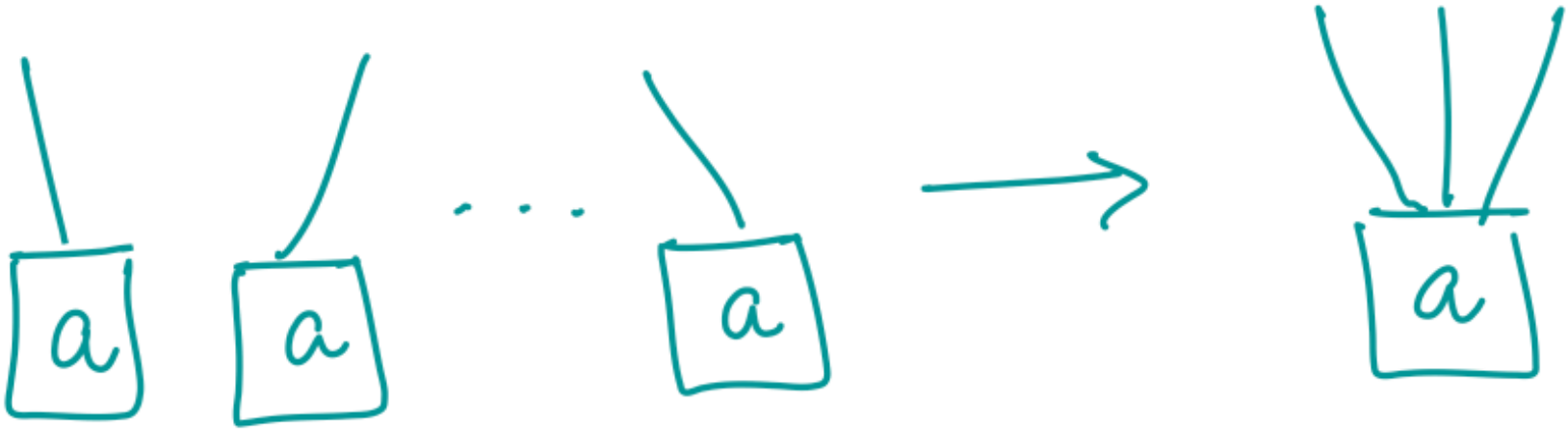


# Reduction

- Identify Redundancies

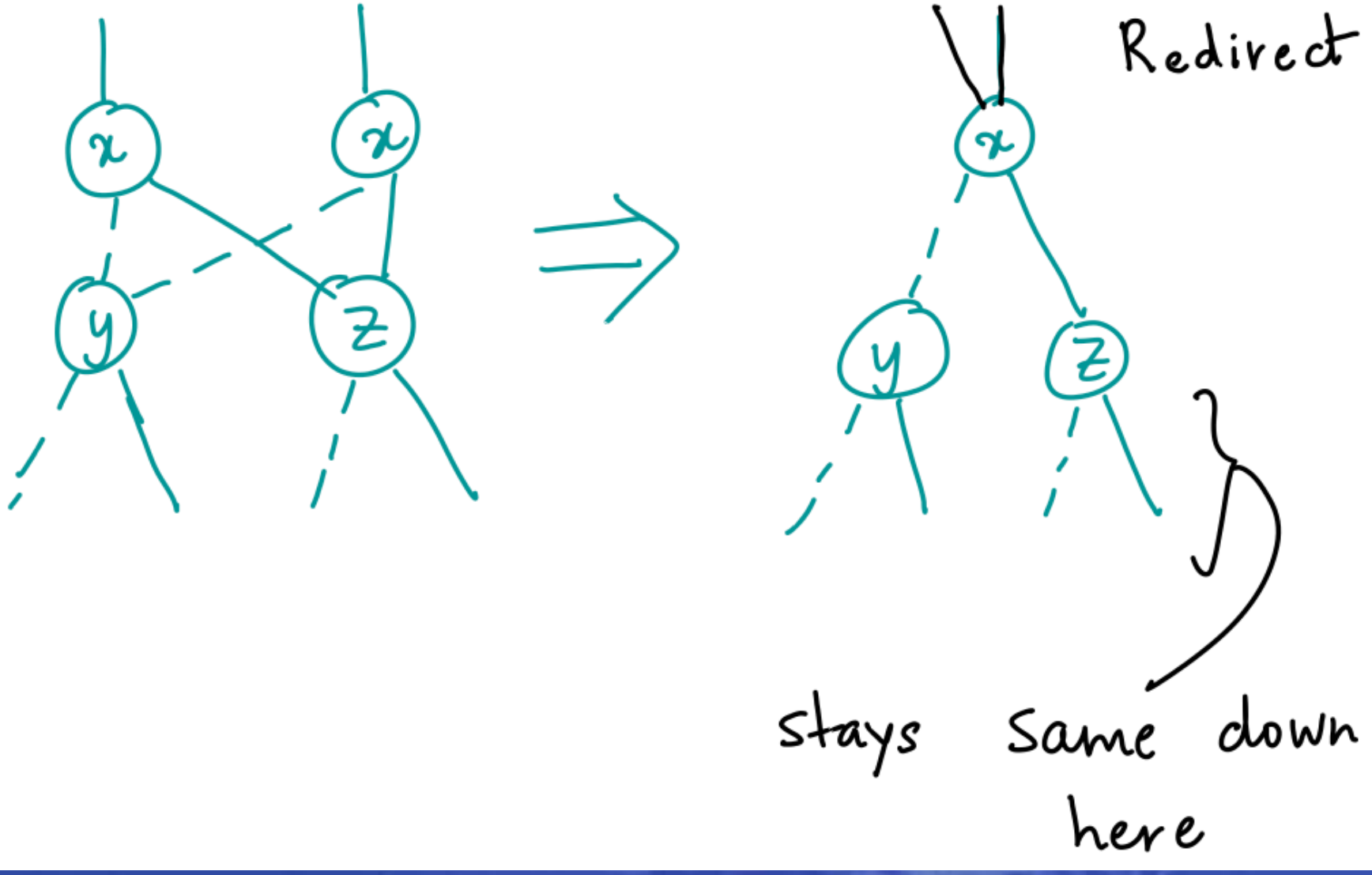
- 3 Rules
  - Merge equivalent leaves
  - Merge isomorphic nodes
  - Eliminate redundant tests

# Merge equivalent leaves

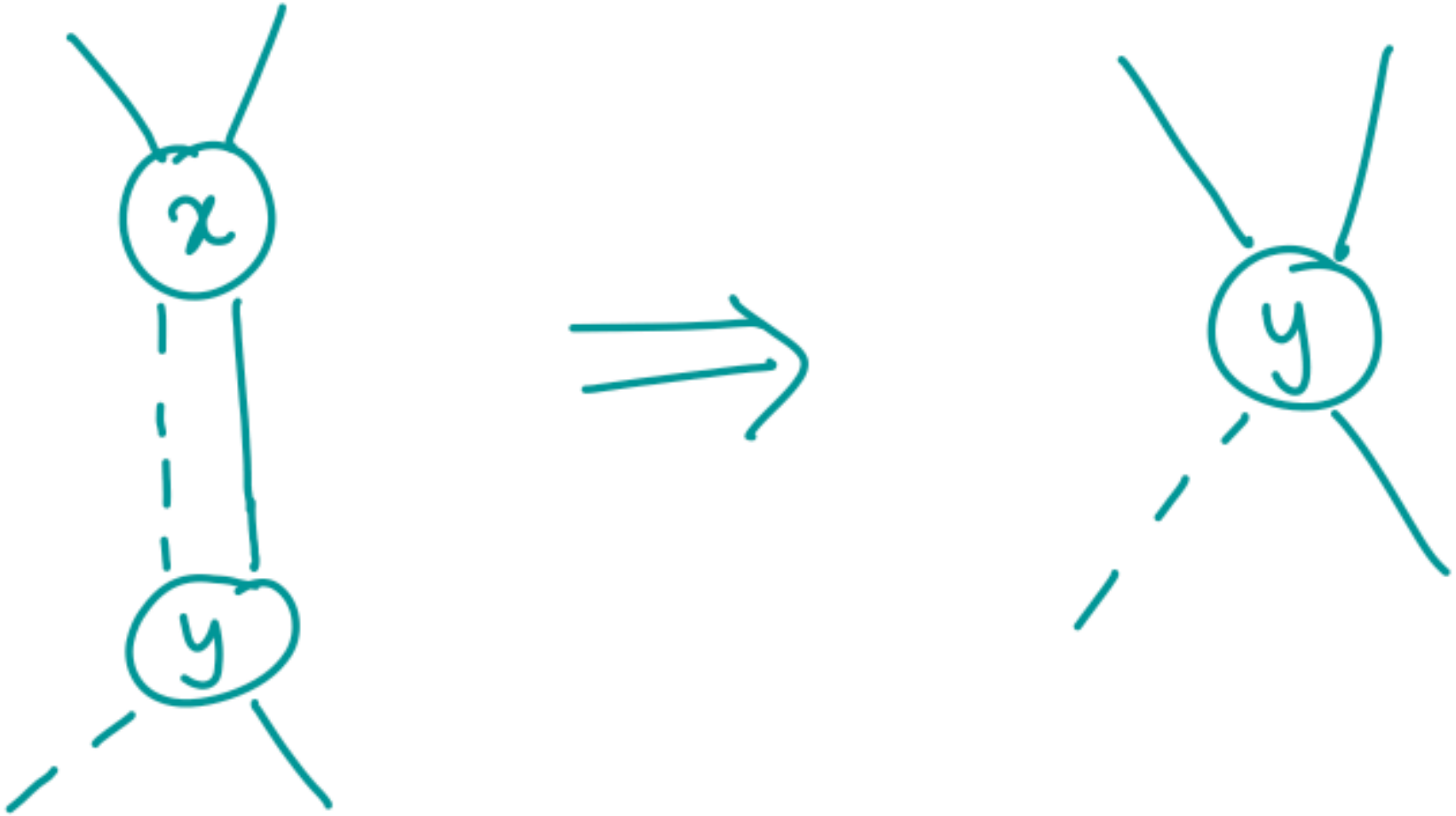



"a" is either 0 or 1

# Merge isomorphic nodes

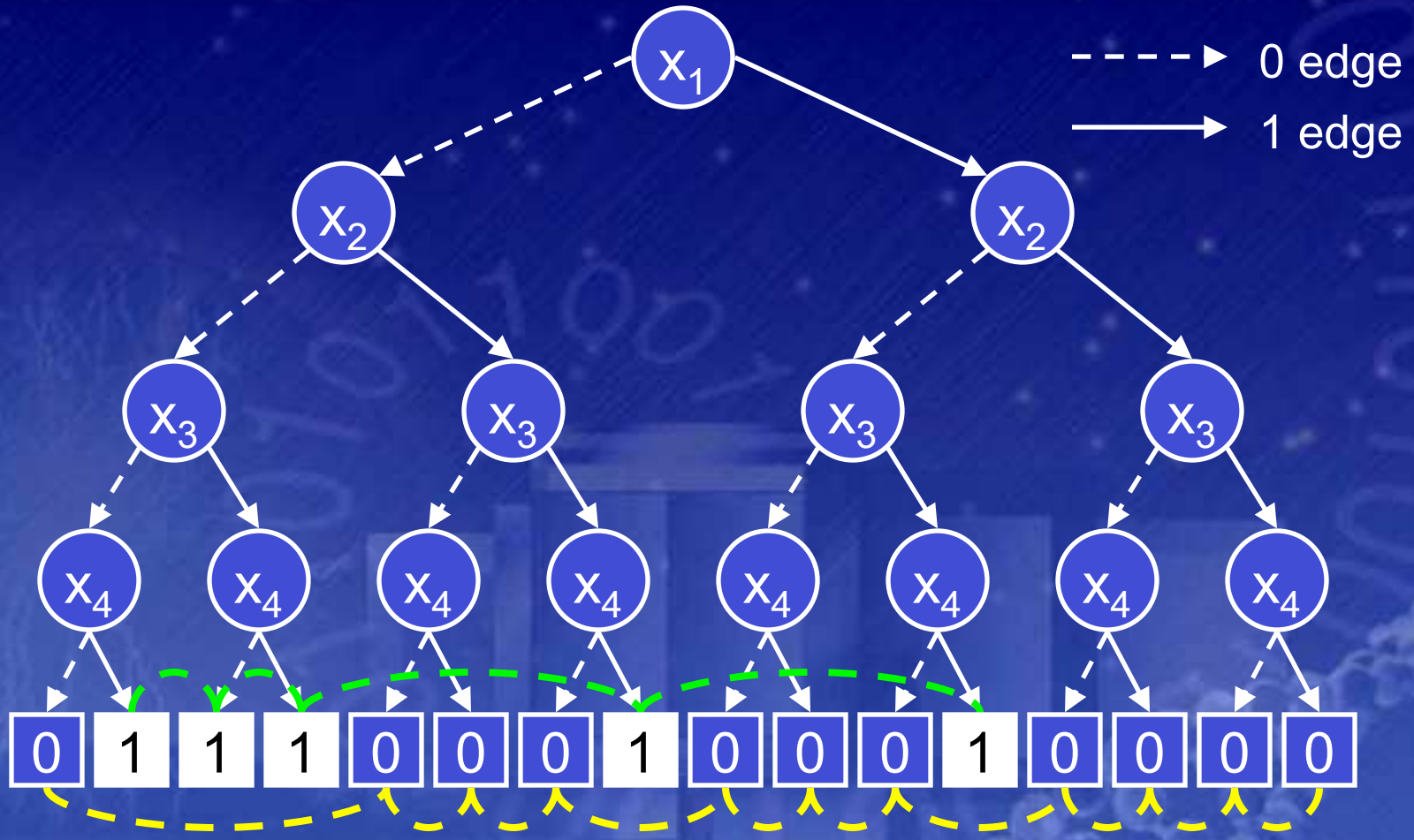


# Eliminate redundant tests



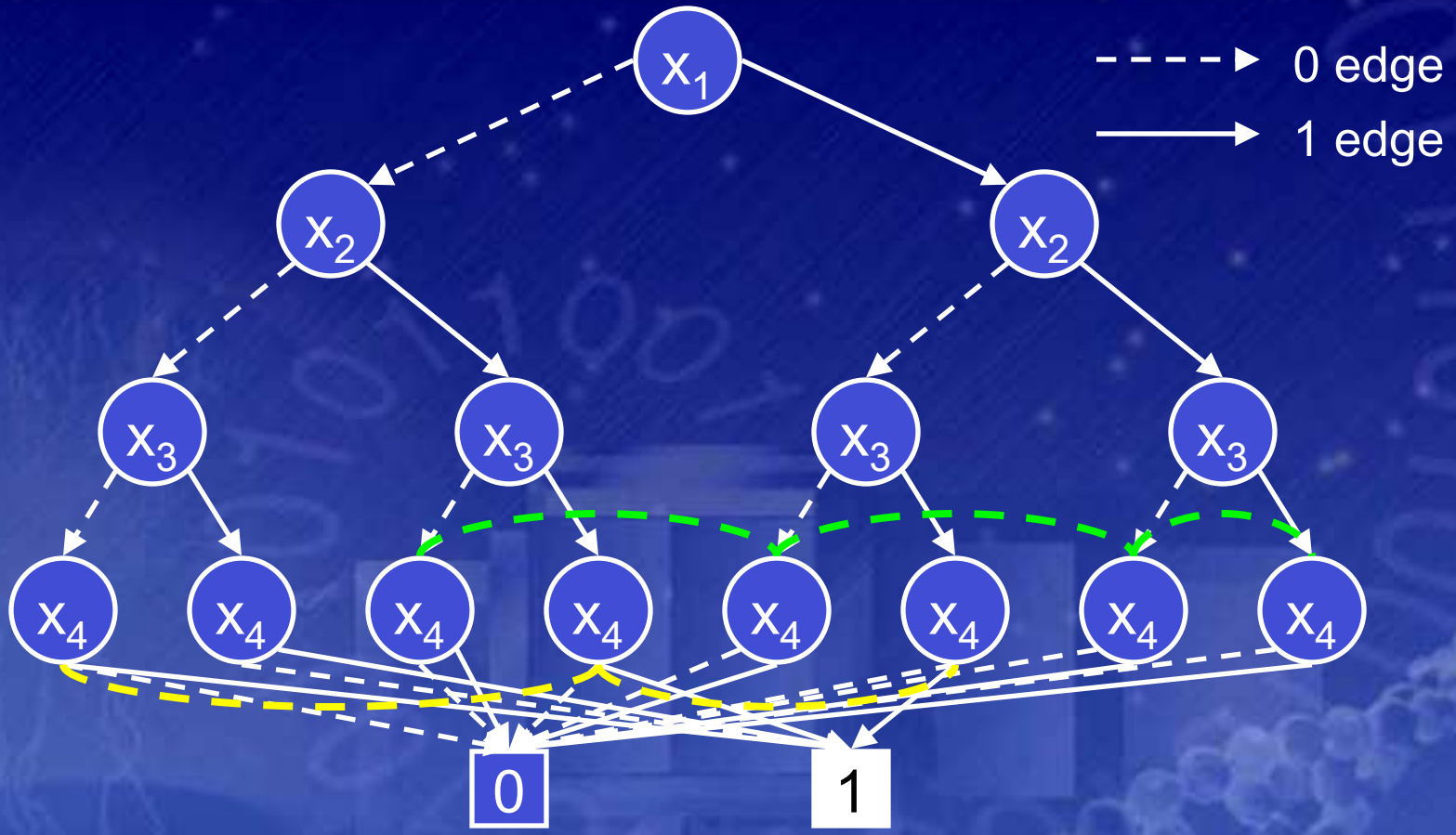
# Binary Decision Diagrams

- Collapse redundant nodes.



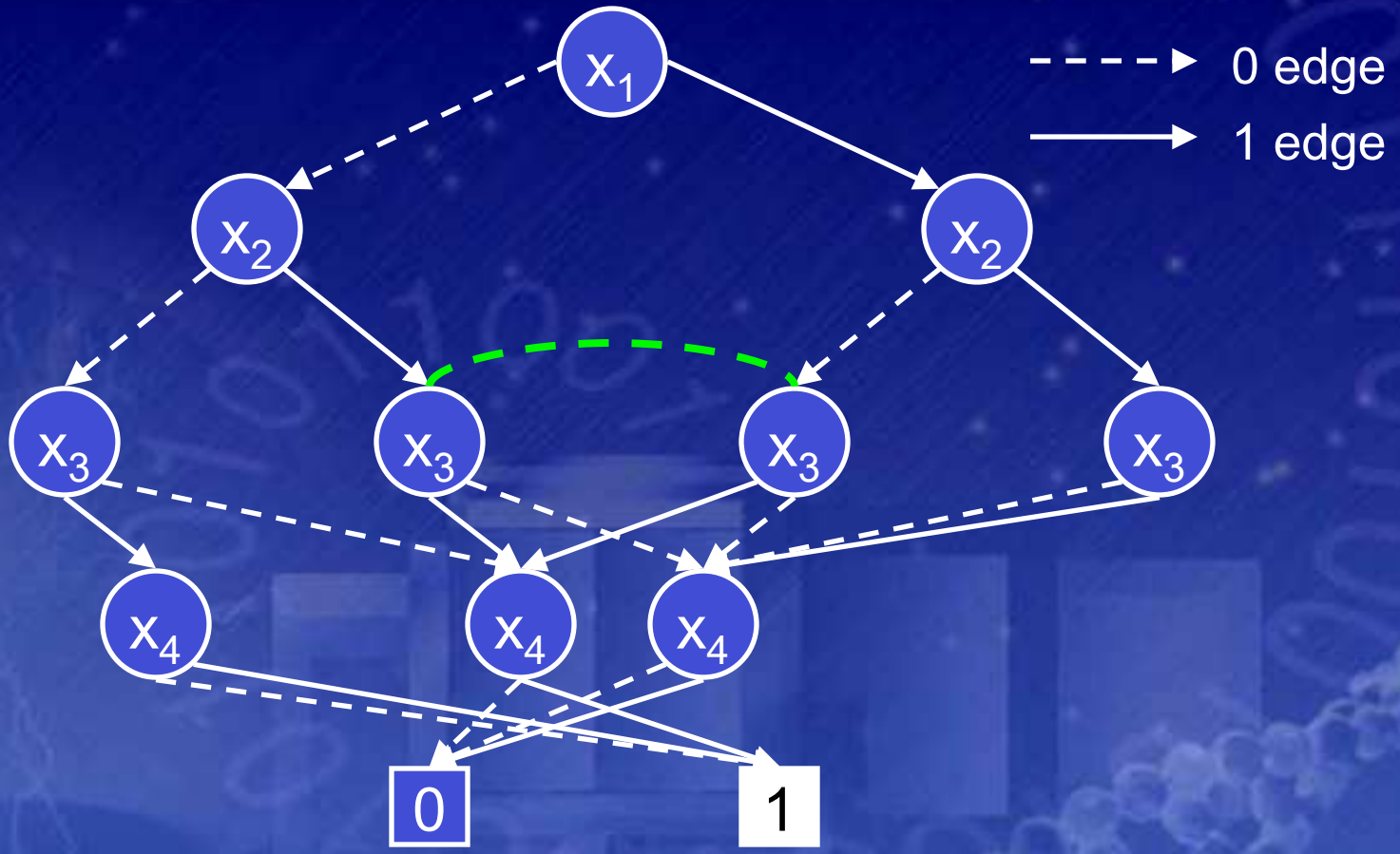
# Binary Decision Diagrams

- Collapse redundant nodes.



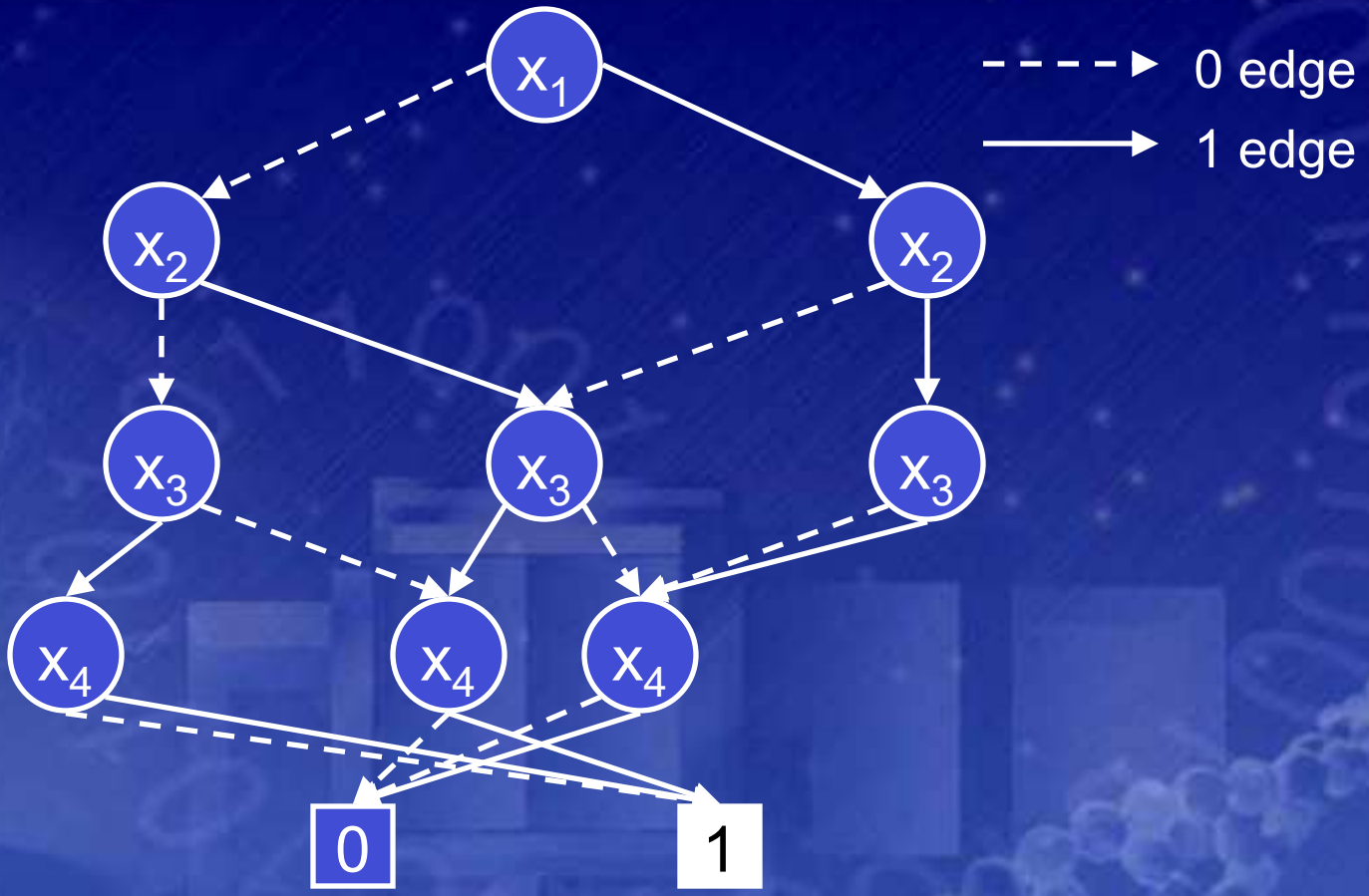
# Binary Decision Diagrams

- Collapse redundant nodes.



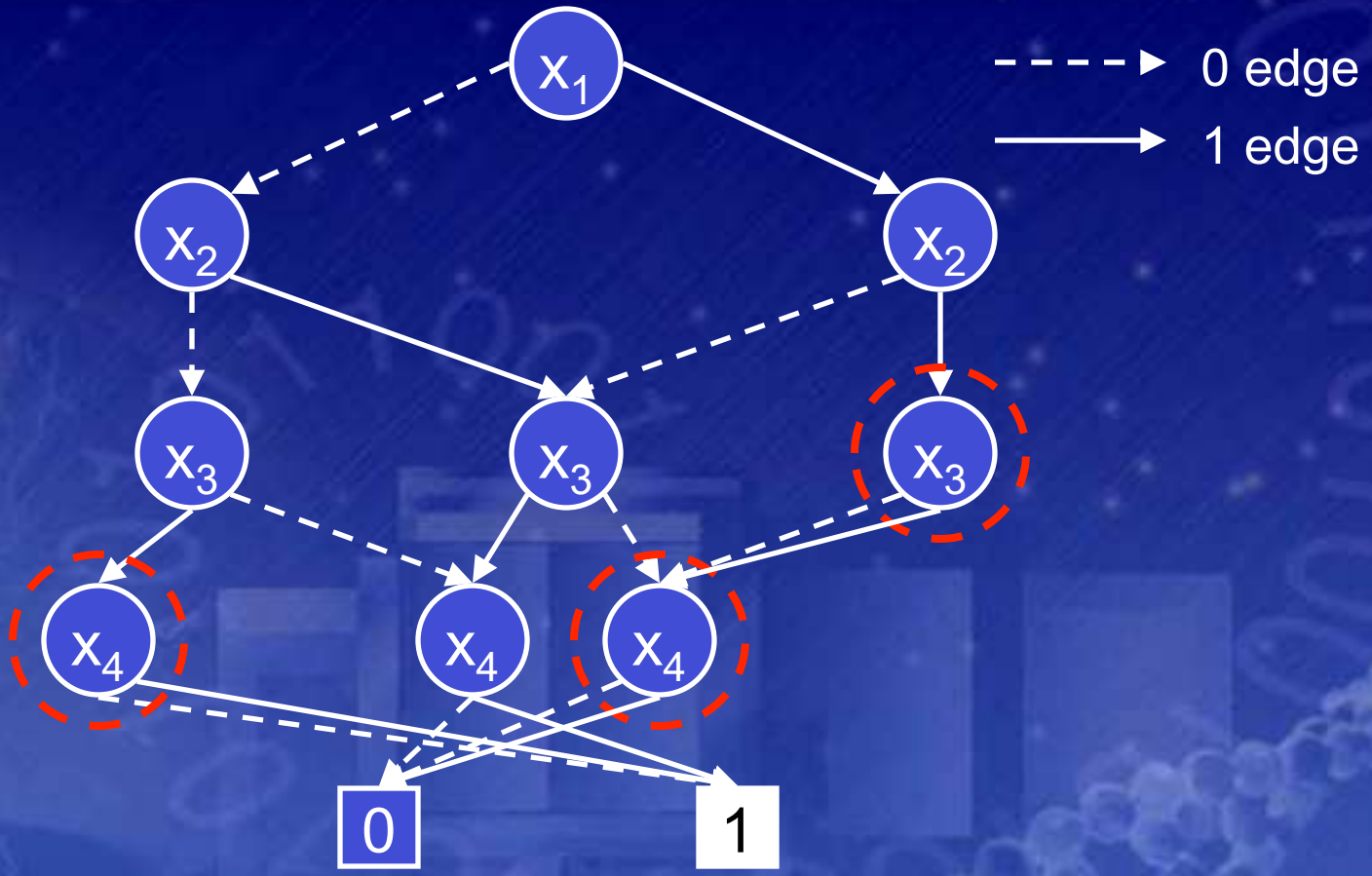
# Binary Decision Diagrams

- Collapse redundant nodes.



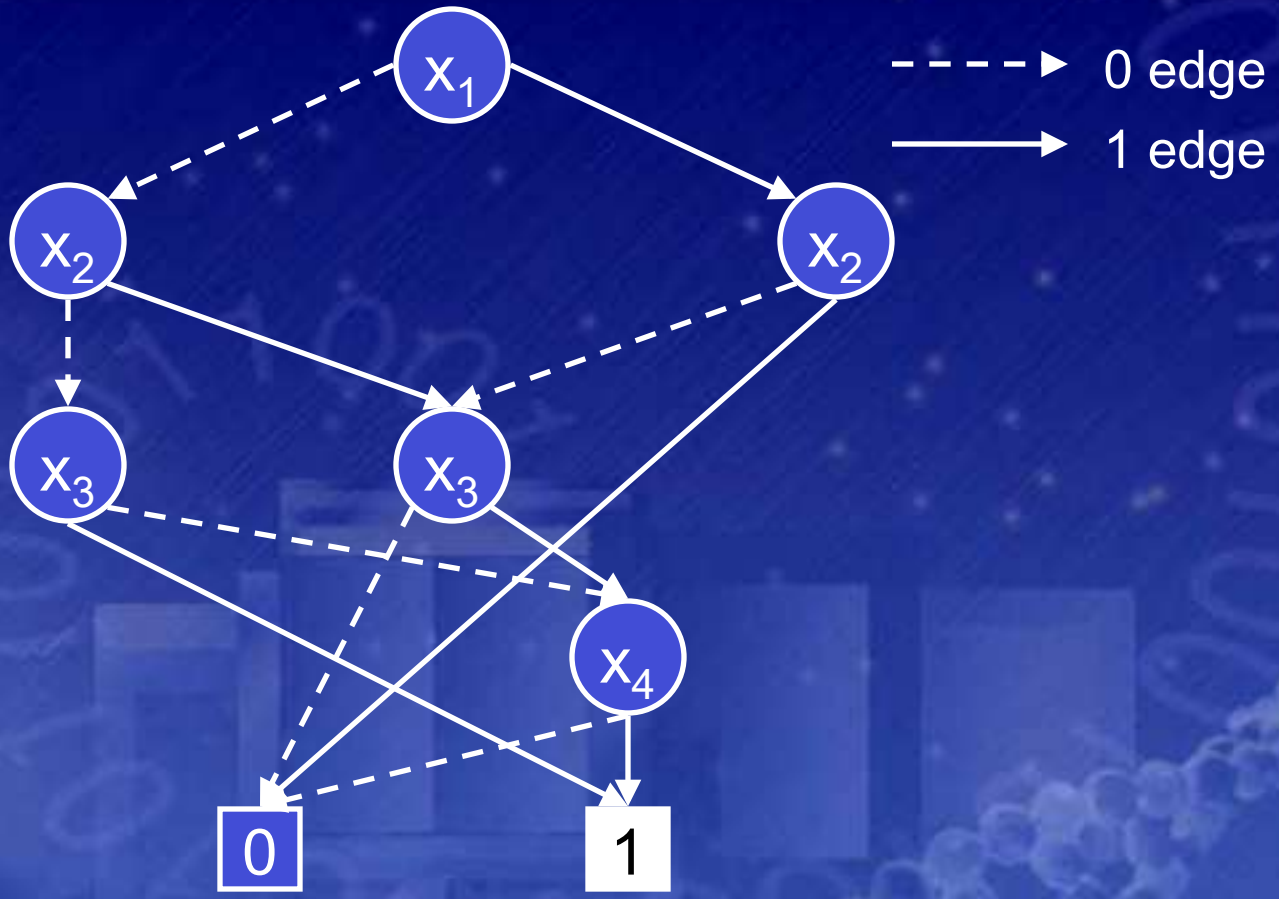
# Binary Decision Diagrams

- Eliminate unnecessary nodes.



# Binary Decision Diagrams

- Eliminate unnecessary nodes.

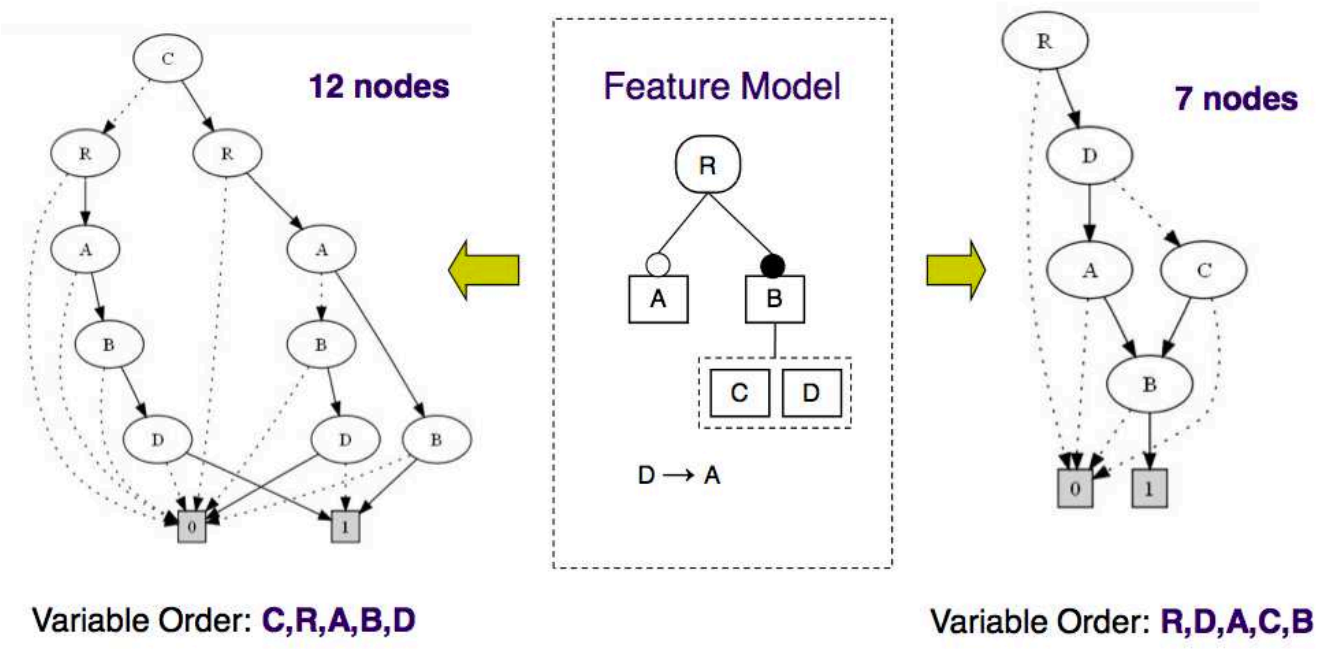


# Binary Decision Diagrams (BDDs)

- 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

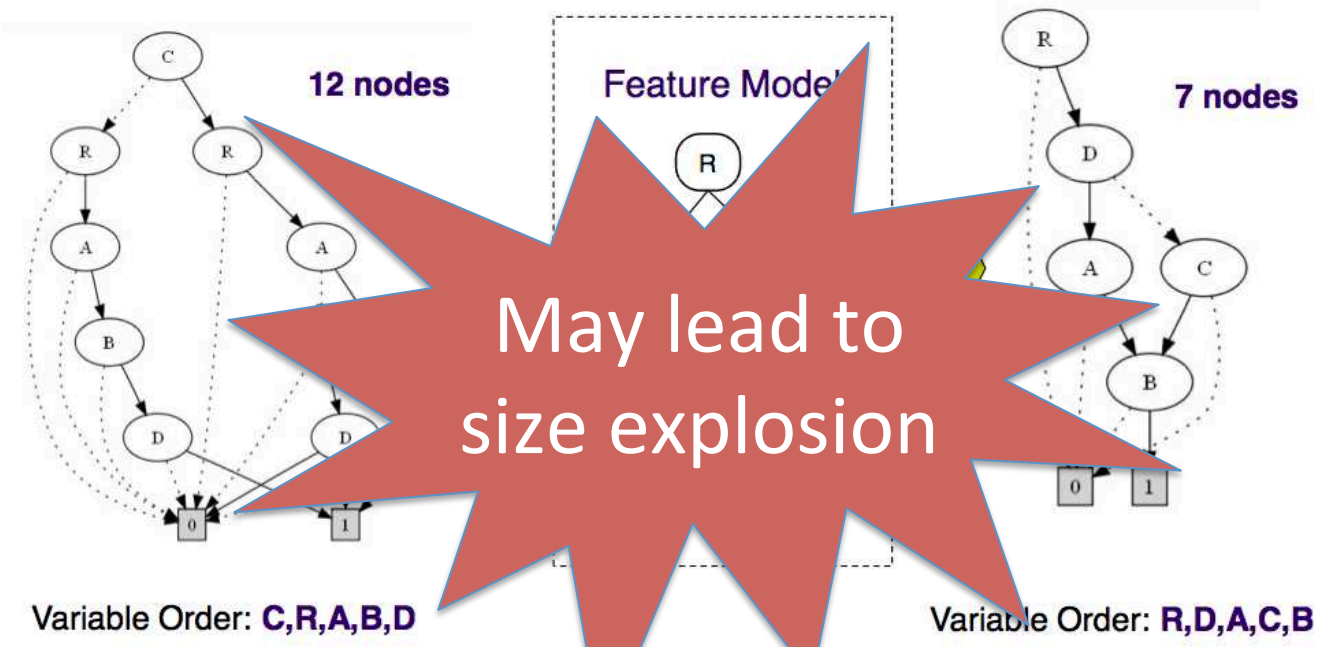
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**




*[Mendonca, slide]*

# Binary Decision Diagrams (BDDs): Theoretical Problem

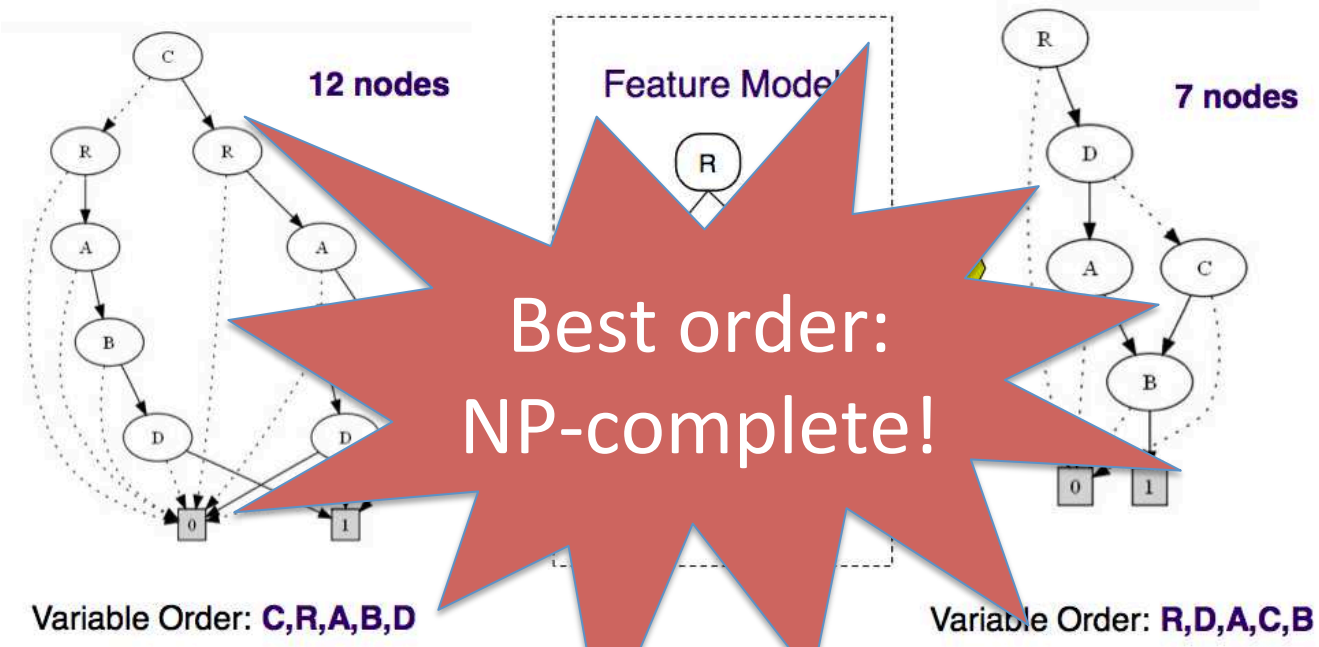
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**




*[Mendonca, slide]*

# Binary Decision Diagrams (BDDs): Theoretical Problem

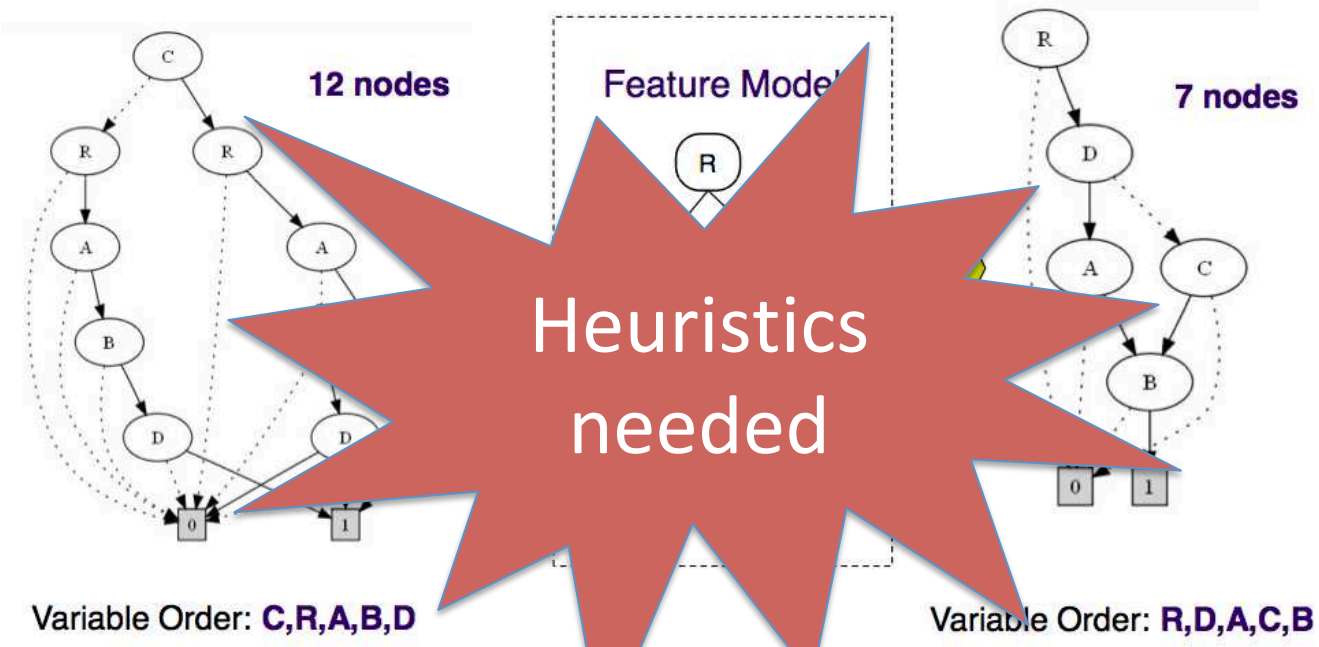
- The size of the BDD is very sensitive to the order of the BDD variables
  - **e.g. two equivalent BDDs for the same feature**




*[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**



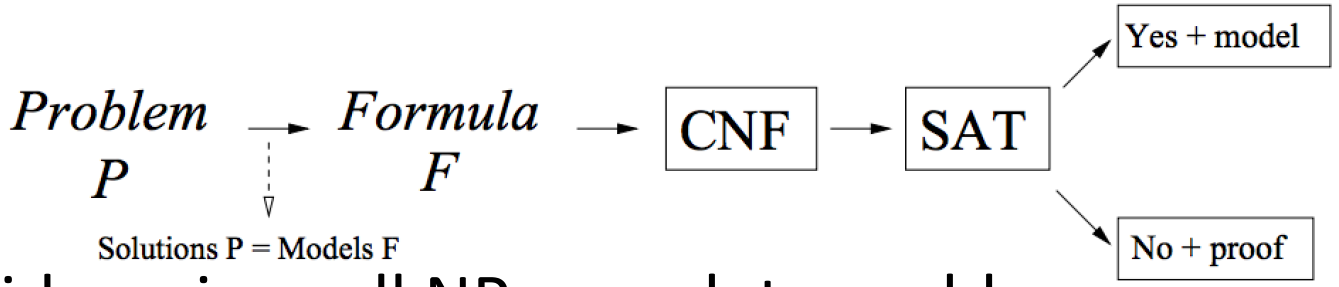

*[Mendonca, slide]*

# 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;

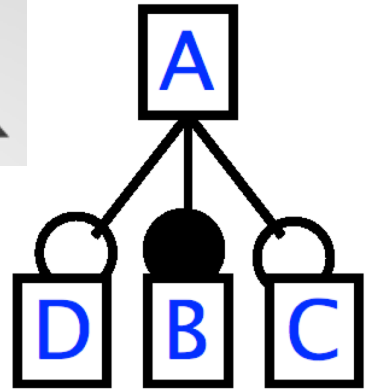
satisfy($\phi$) {
  if every clause of $\phi$ has a true literal, return T;
  if BCP() == F, return F;
  assign appropriate values to all pure literals;
  choose an $x \in V$ that is unassigned in $A$,
    and choose $v \in \{T, F\}$.
  $A(x) = v$;
  if satisfy($\phi$) return T;
  $A(x) = \neg v$;
 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

# How to automate analysis of your feature models?

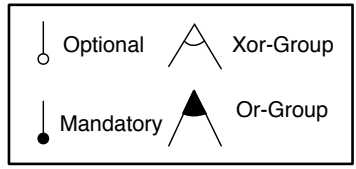
Let us use BDDs and SAT solvers

A ^
A ⇔ B ^
C => A ^
D => A

FAMiliAR




FM




φ

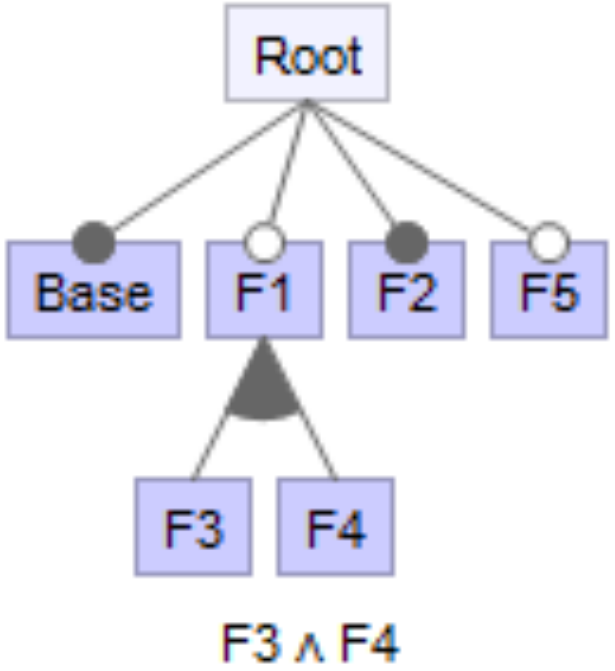
```
fml> fm1 = FM ("output/fm1.tvl")
root A {
    group [ 3..3 ] {
        opt D {
        },
        B {
        },
        opt C {
        }
    }
}
fm1: (FEATURE_MODEL) A: [D] B [C] ;
```

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

```
fml> c1 = cores fm1
c1: (SET) {B;A}
fml> c1bis = cores fm1bis
fml> compare fm1 fm1bis
res7: (STRING) REFACTORING
fml> compare fm1bis fm1bisbis
res8: (STRING) REFACTORING
fml> c1 eq c1bisbis
res6: (BOOLEAN) true
res4: (BOOLEAN) true
```

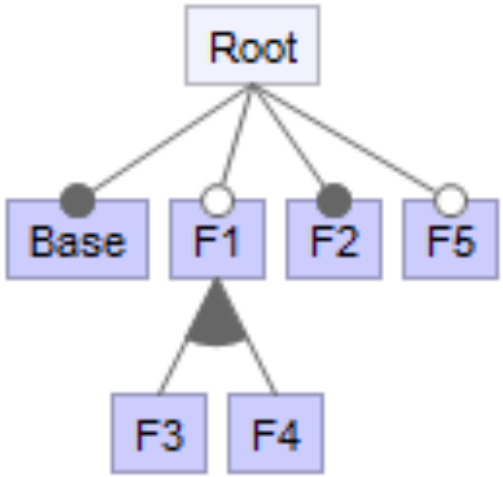
# Consistency

- SAT-Solver
  - SAT(FM)



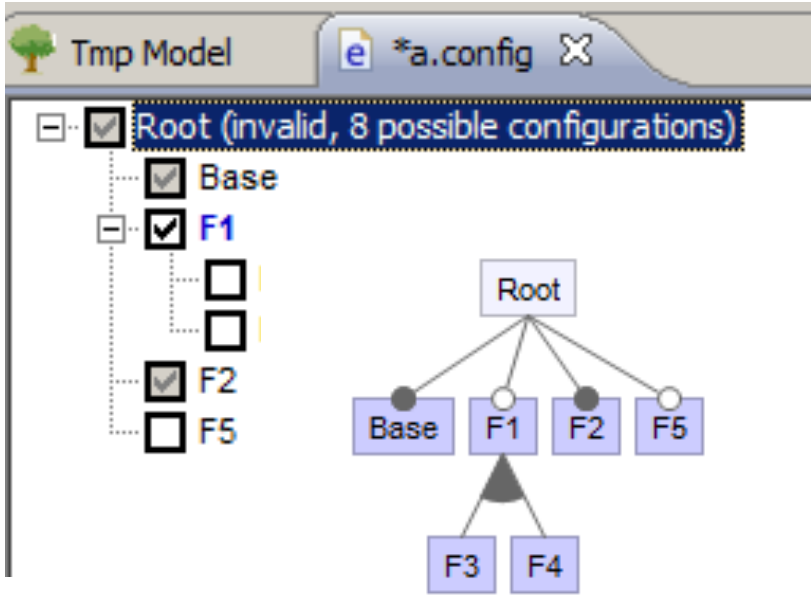
# Core and dead features

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

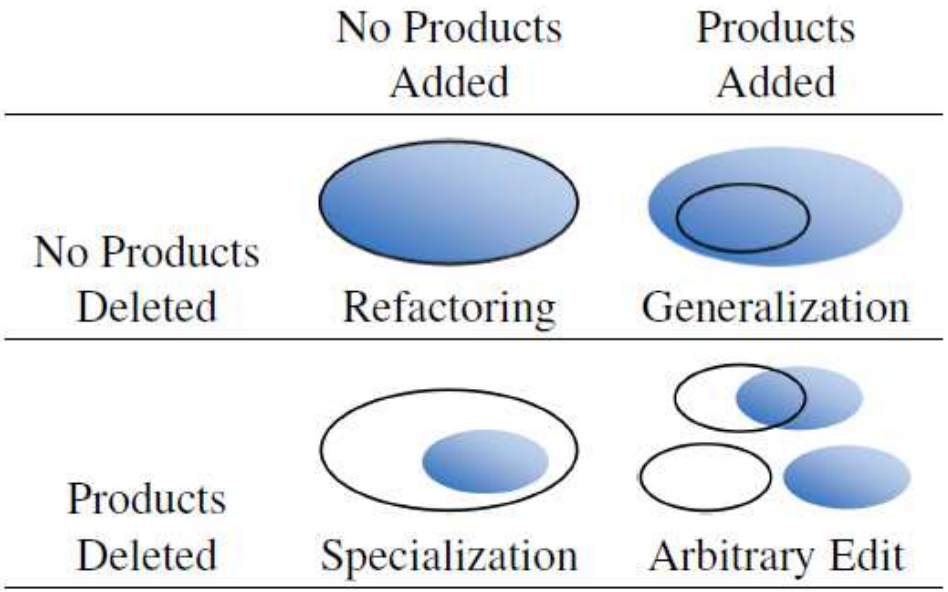


# Partial configuration

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



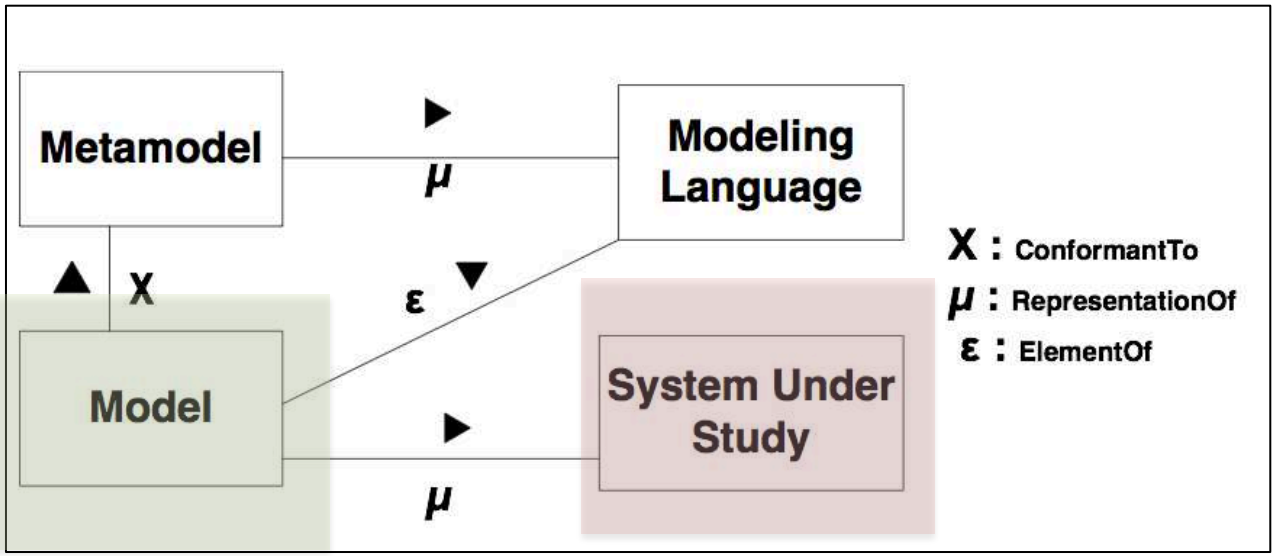
# Relationship between feature models




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

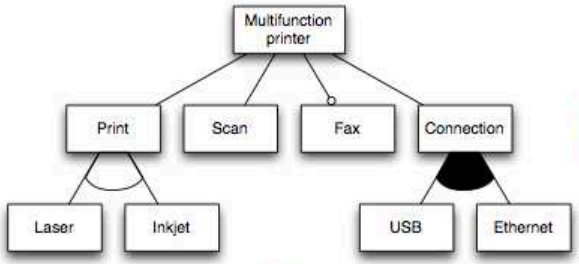
# Recap

# Feature Models




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

# Typical implementations





result

logics

solvers



# FAMiliAR

(FeAture Model scrIpt Language for manIpulation and Automatic Reasoning)

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

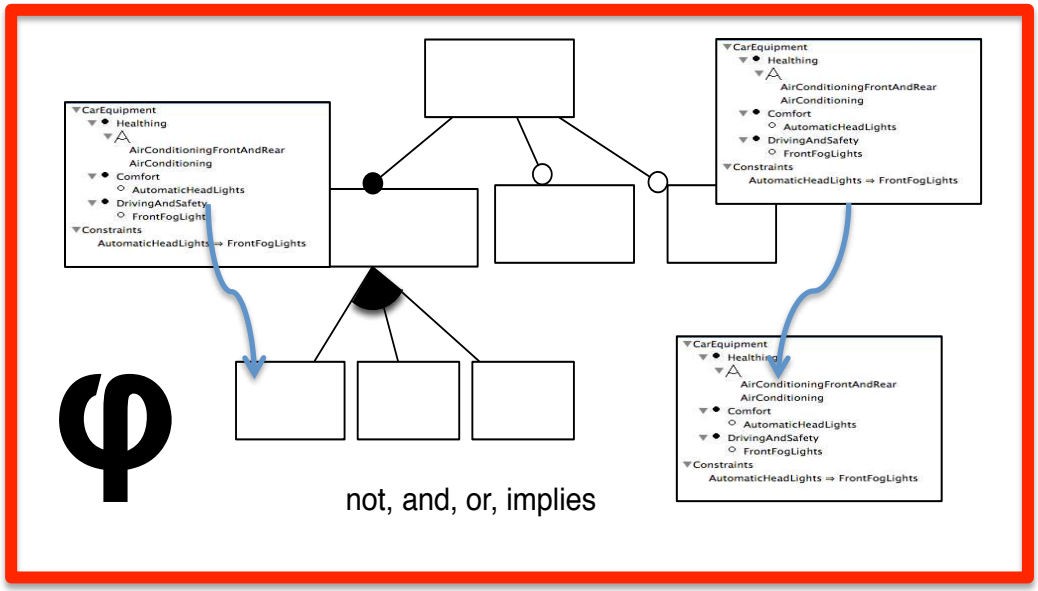
COMMON VARIABILITY LANGUAGE

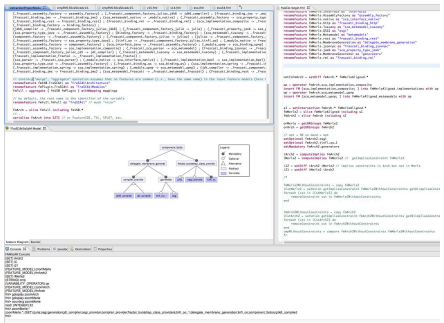
S.P.L.O.T. Software Product Lines Online Tools

IDE Feature

**TVL**

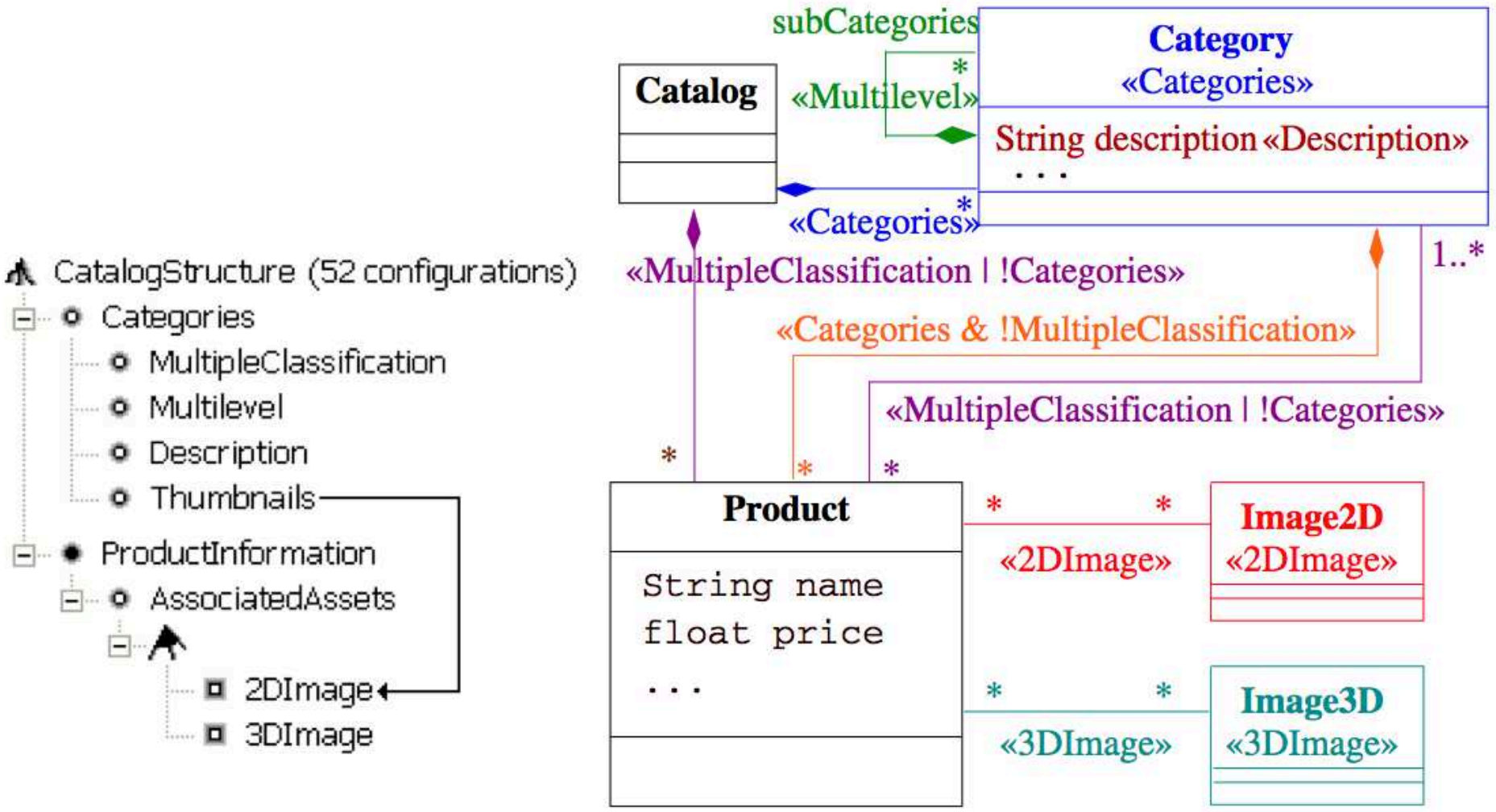
**DIMACS**

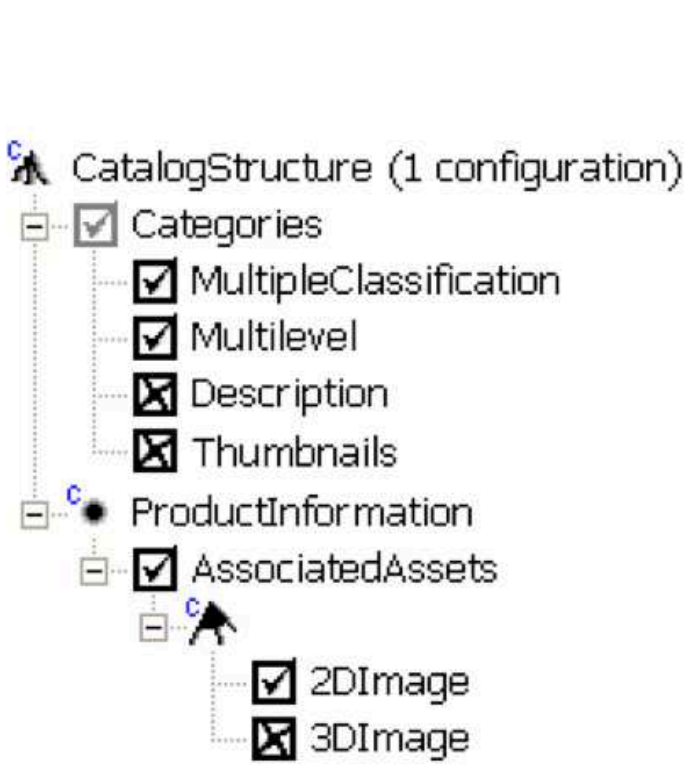


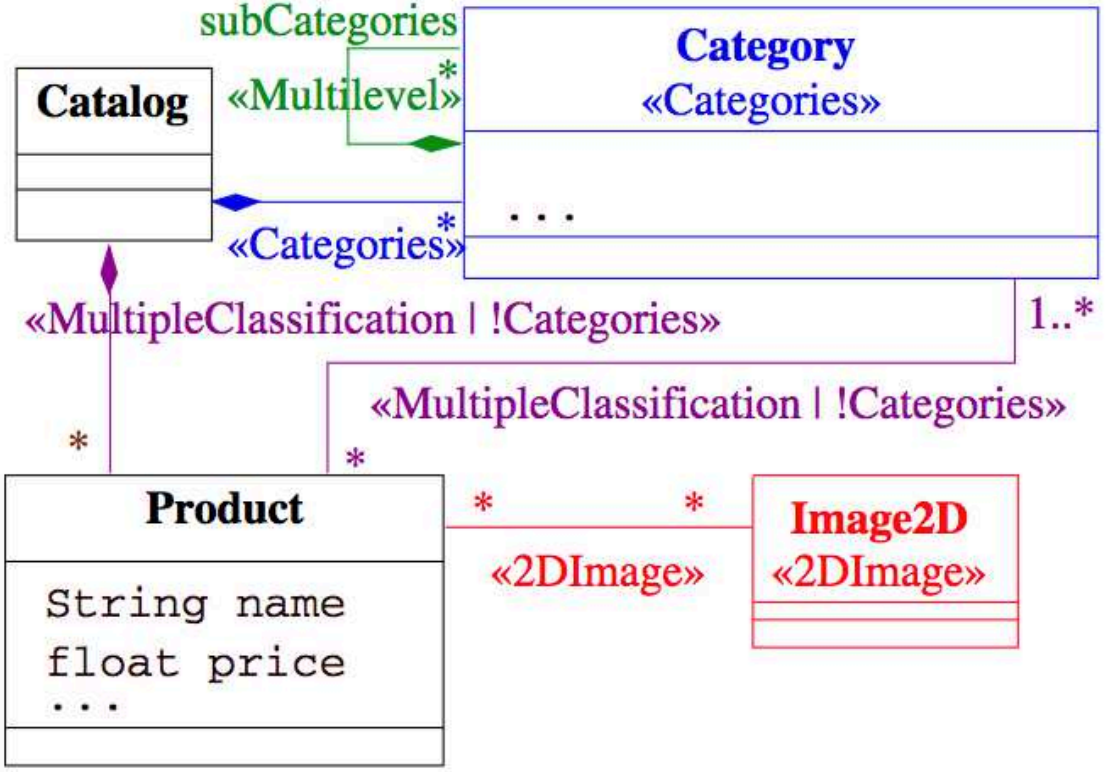



**importing, exporting, composing, decomposing, editing, configuring, reverse engineering, computing "diffs", refactoring, testing, and reasoning about (multiple) variability models**

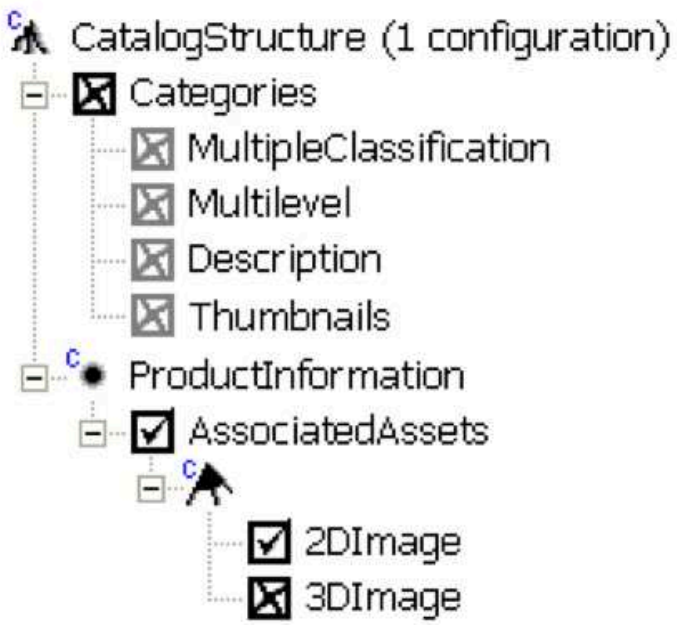
Mathieu Acher, Philippe Collet, Philippe Lahire, Robert B. France « A Domain-Specific Language for Large-Scale Management of Feature Models » Science of Computer Programming (SCP), 2013

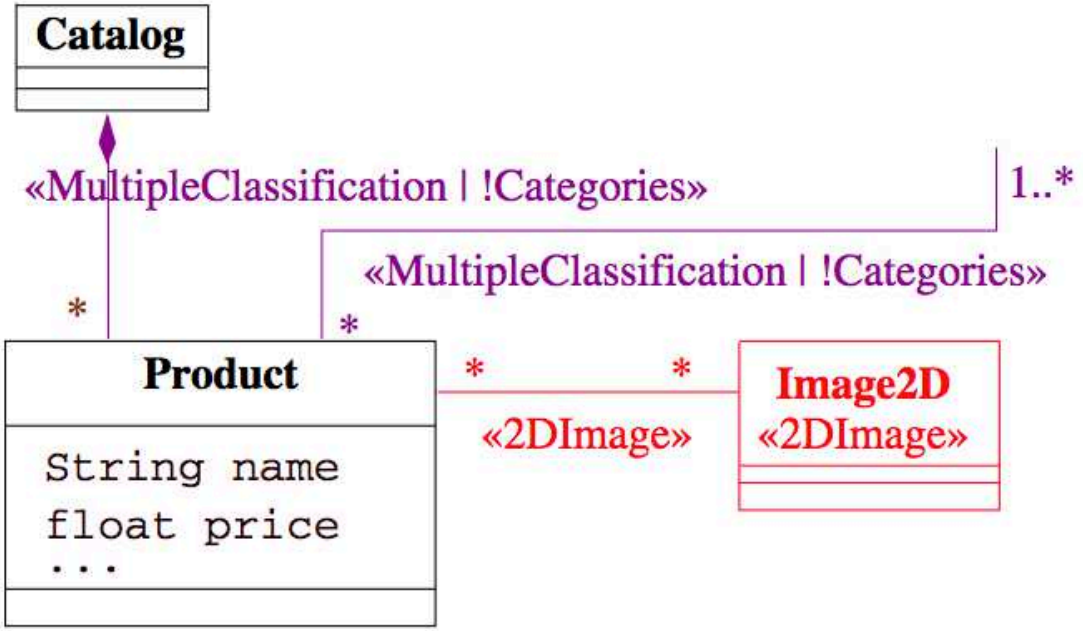

subCategories
Category
«Categories»
Catalog
«Multilevel»
String description «Description»
. . .
«Categories»
«MultipleClassification | !Categories»
1..*
«Categories & !MultipleClassification»
«MultipleClassification | !Categories»
Product
String name
float price
. . .
«2DImage»
Image2D
«2DImage»
«3DImage»
Image3D
«3DImage»
CatalogStructure (52 configurations)
Categories
MultipleClassification
Multilevel
Description
Thumbnails
ProductInformation
AssociatedAssets
2DImage
3DImage


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

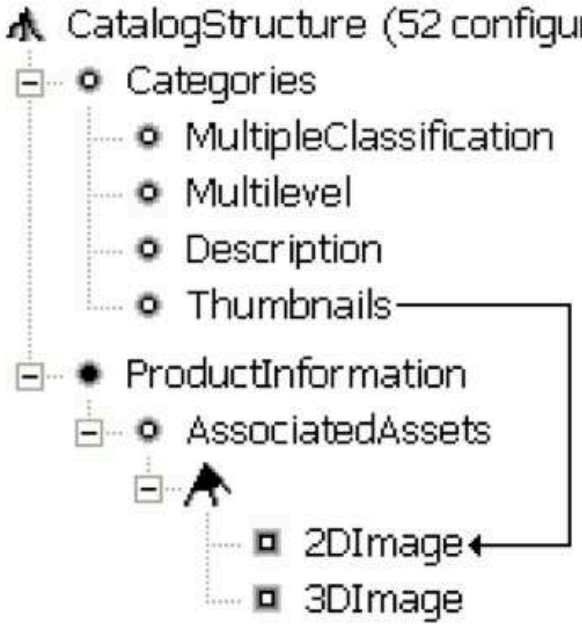

subCategories
Catalog
«Multilevel»
Category
«Categories»
...
«Categories»
«MultipleClassification | !Categories»
1..*
«MultipleClassification | !Categories»
Product
String name
float price
...
«2DImage»
Image2D
«2DImage»

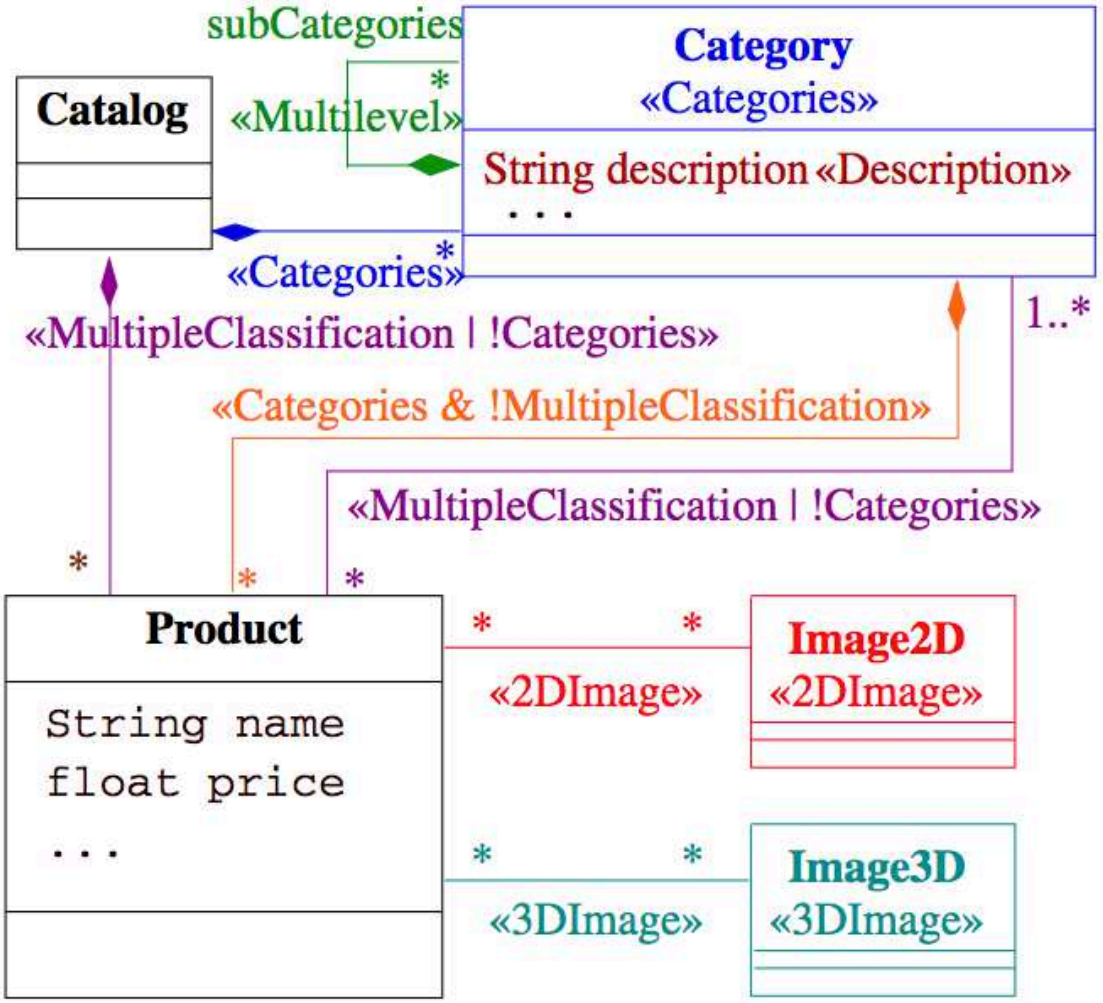
# Ooops


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


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

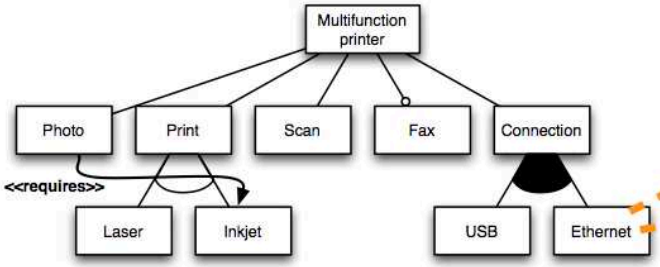
# Safe composition? No!



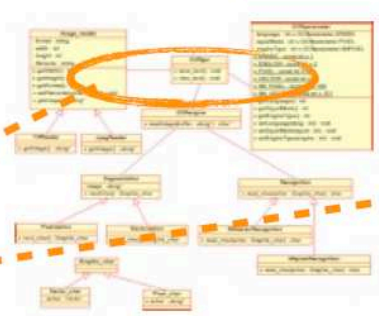



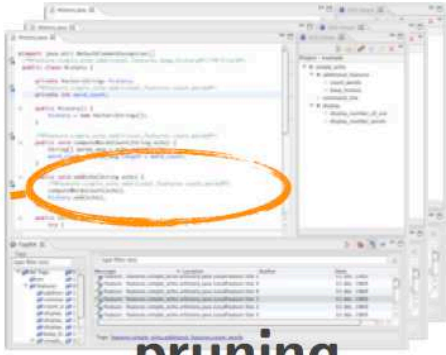
# Product Derivation

## feature model




## variable model and code assets

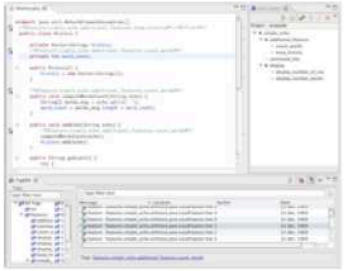




pruning,
composition,
weaving,
transformation
...

configuration

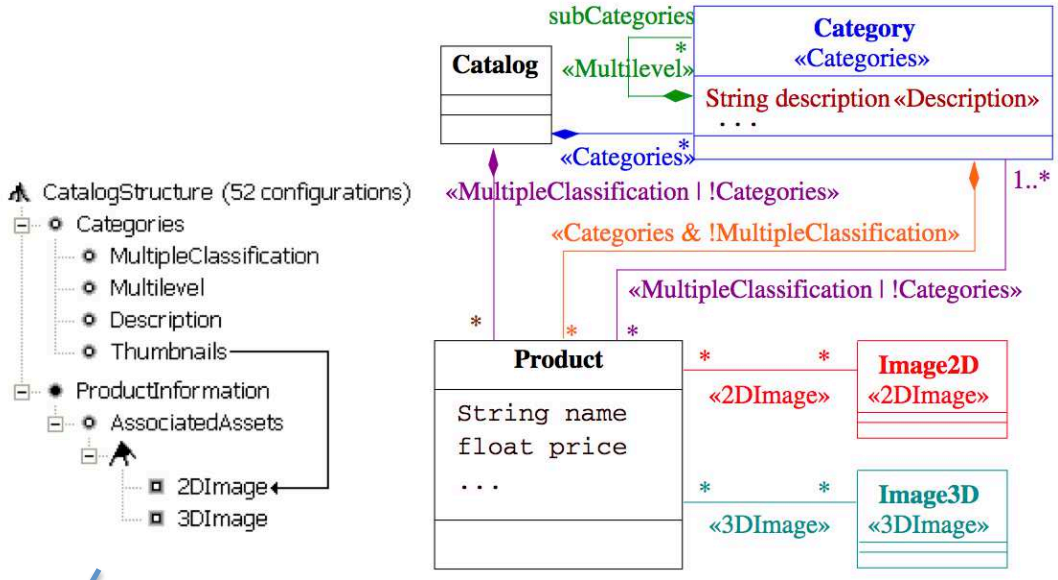
{ MP, Photo, Print, Inkjet, Scan,
Fax, Connection, USB, Ethernet }



product spec

product

# Safe composition: how does it work?




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

**Another propositional formula**