

Automated reasoning about variability models with solvers

Mathieu Acher

Maître de Conférences

mathieu.acher@irisa.fr

Material

[https://github.com/FAMILIAR-project/
HackOurLanguages-SIF](https://github.com/FAMILIAR-project/HackOurLanguages-SIF)

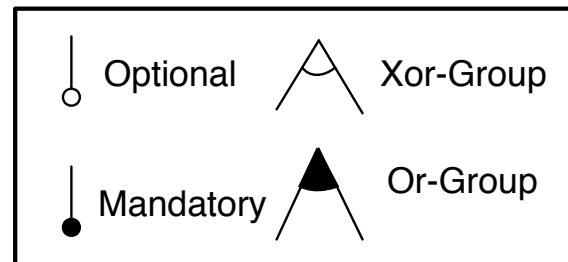
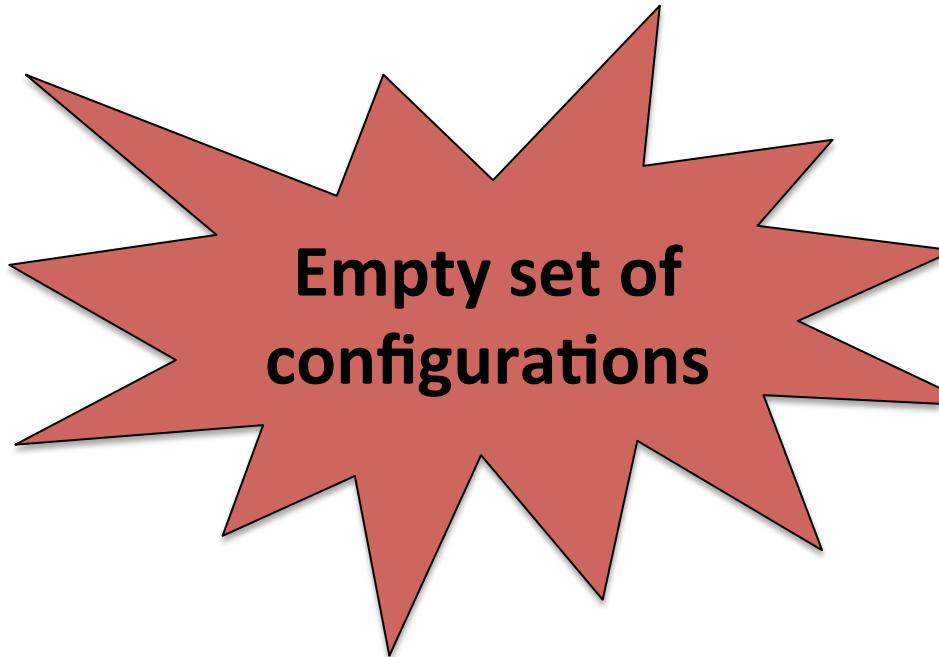
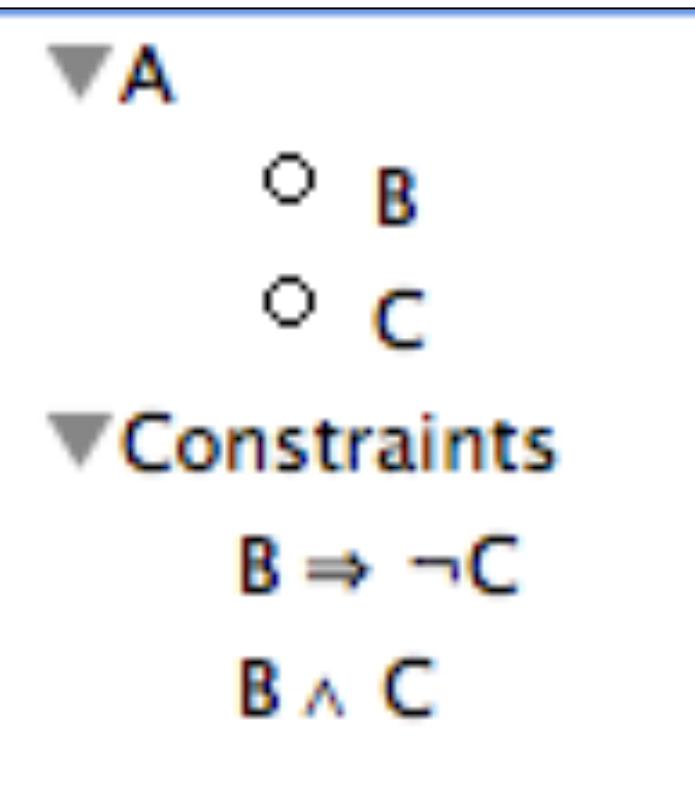
Plan

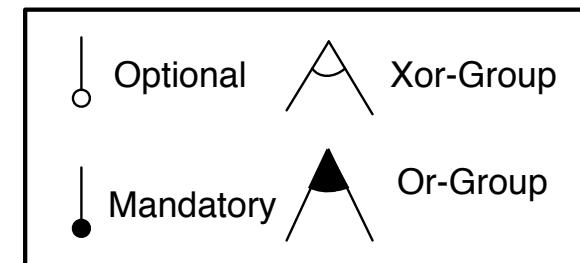
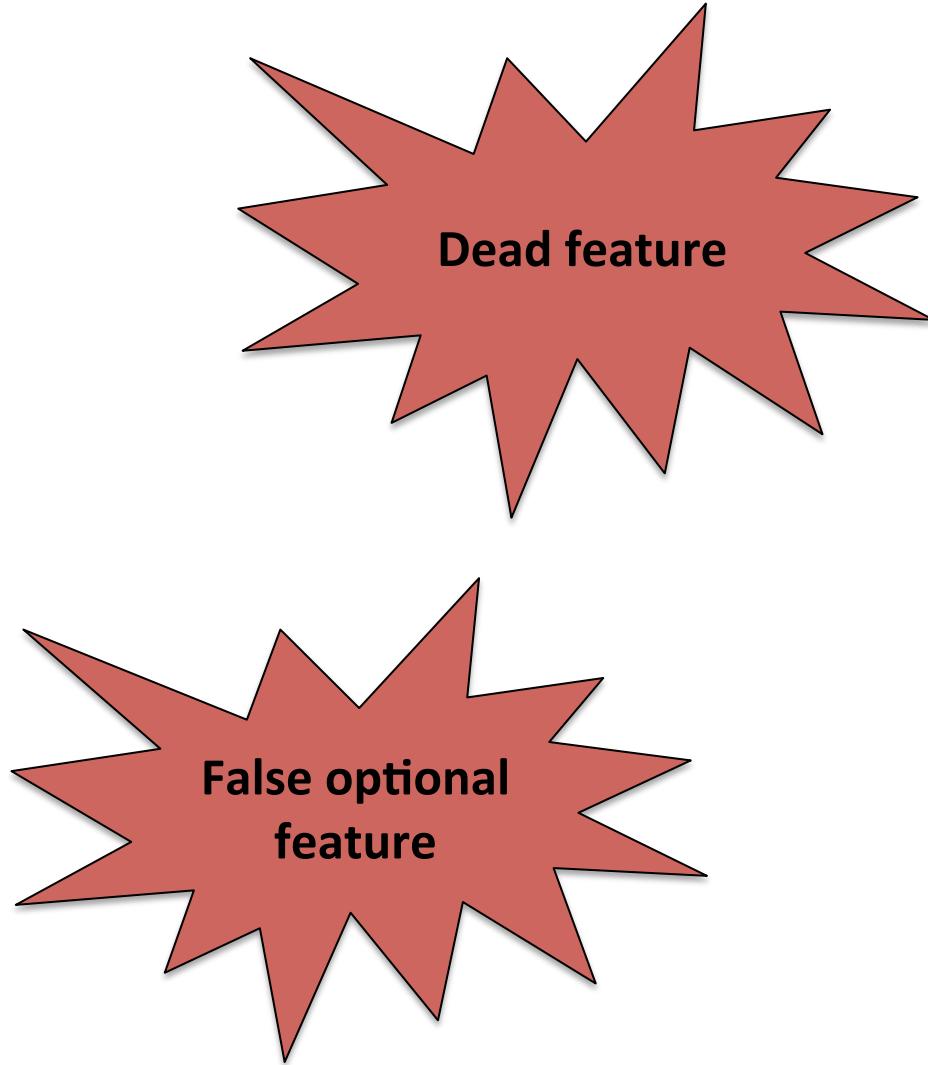
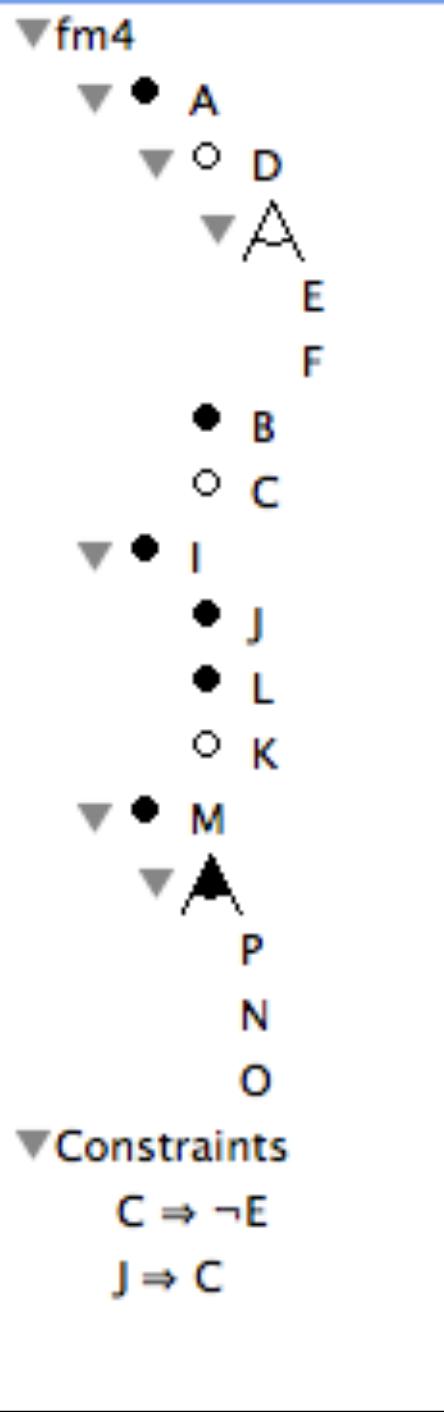
- **Feature Models**
 - Defacto standard for modeling product lines and variability
 - Syntax, semantics, automated reasoning, synthesis

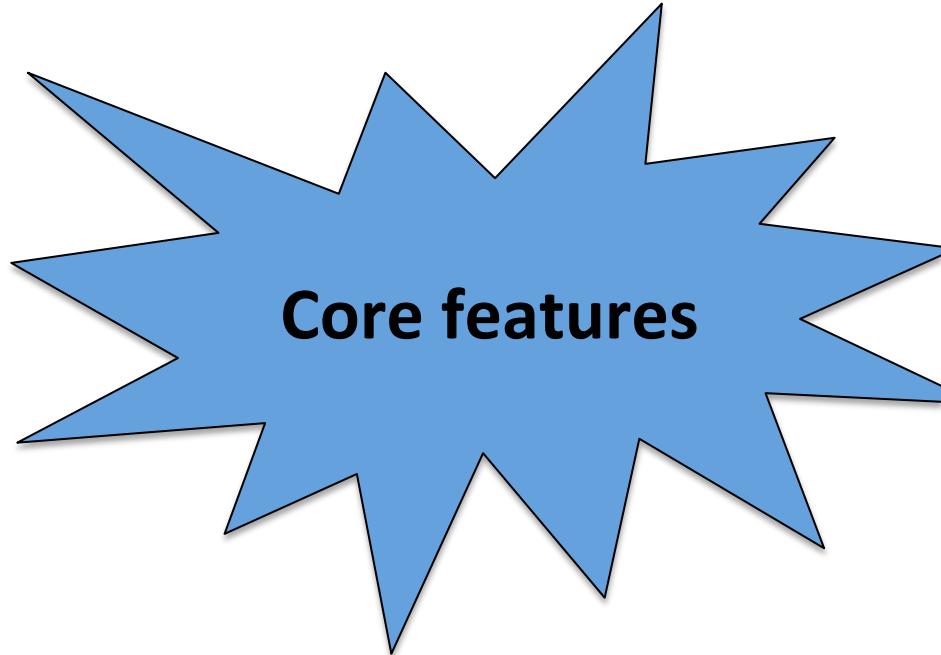
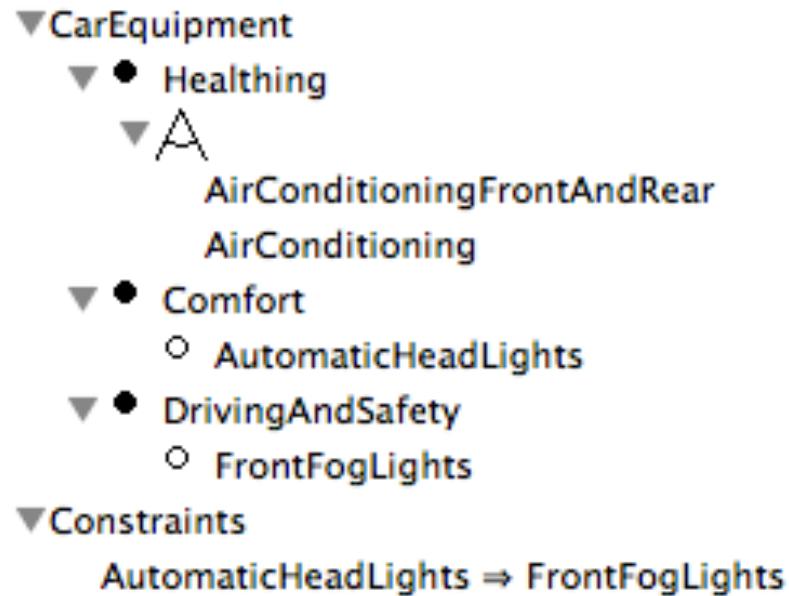
Contract

- 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**

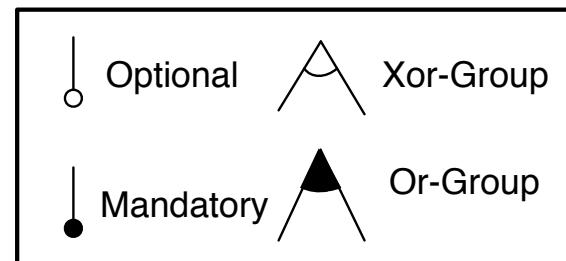
I want to analyze and
play with my specification!

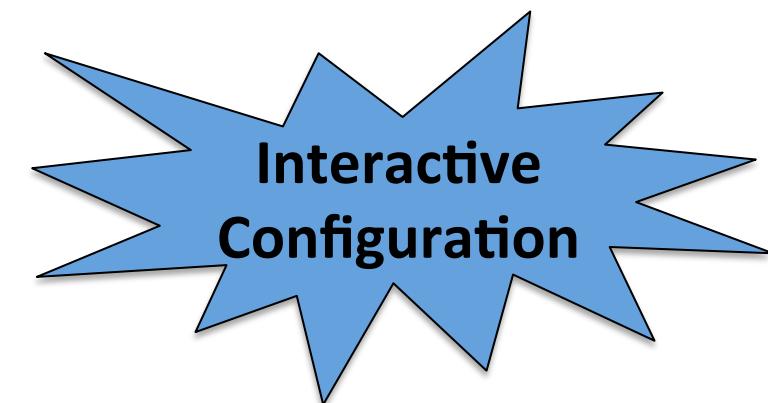
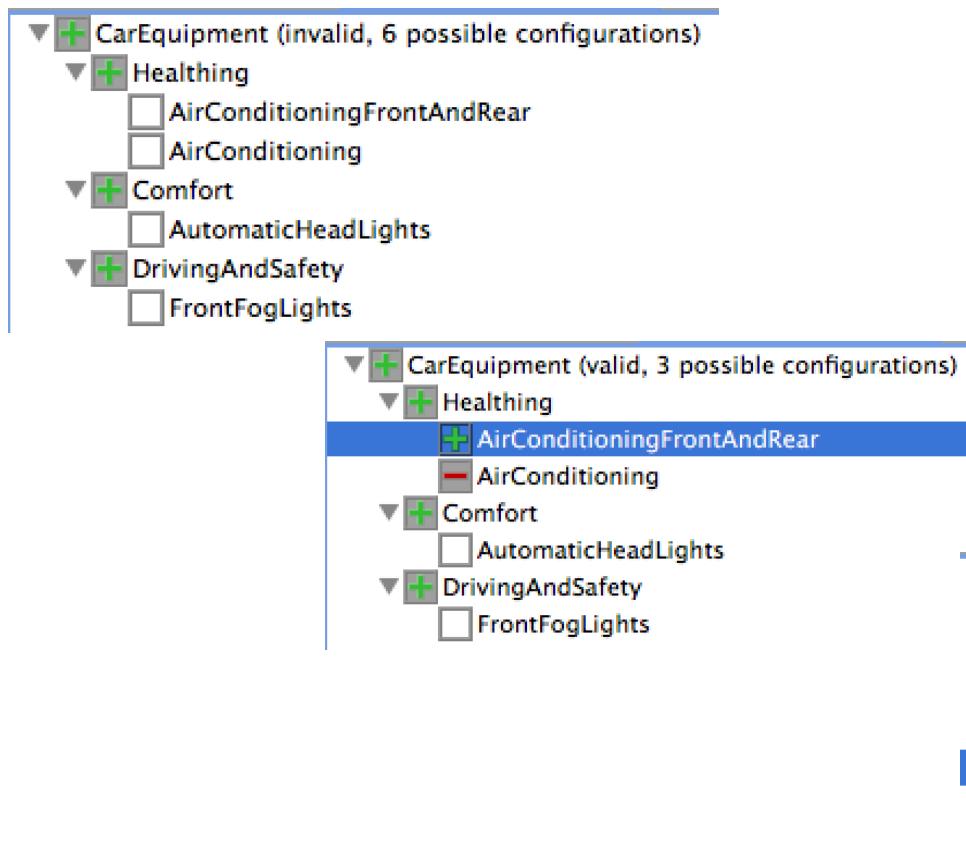
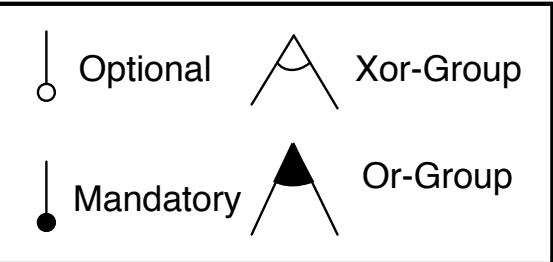






{CarEquipment, Comfort, DrivingAndSafety, Healthing}





Feature Models and Automated Reasoning

Benavides et al. survey, 2010

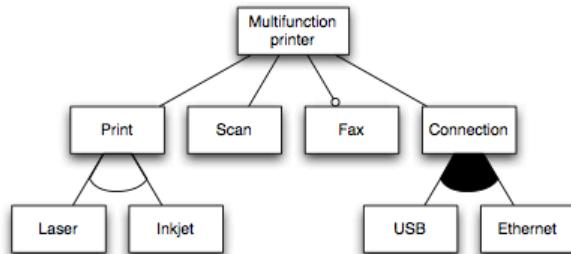
Decision problems and complexity

- 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

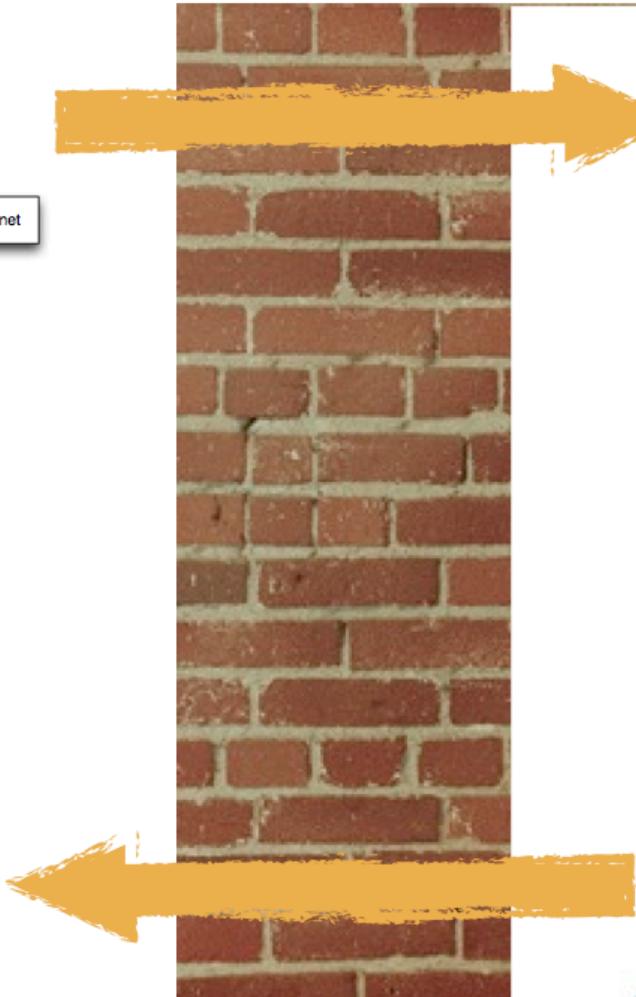
How to automate analysis of your feature models?

Binary Decision Diagram (BDD)
SAT solver

Typical implementations



Fontsource (Attributed - Free Processing 2012) (Attributed - Creative Commons)



logics
solvers

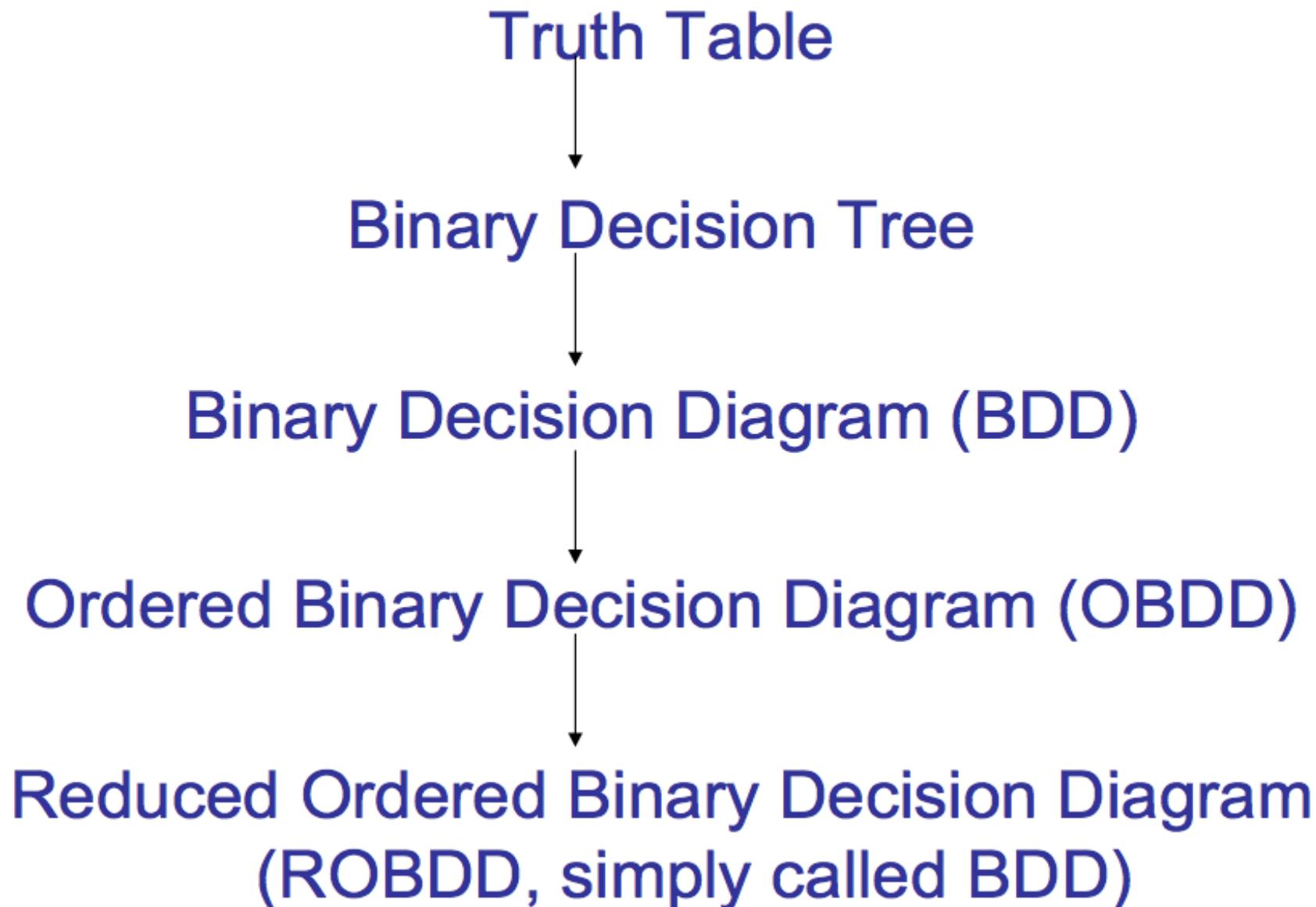


Z3

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

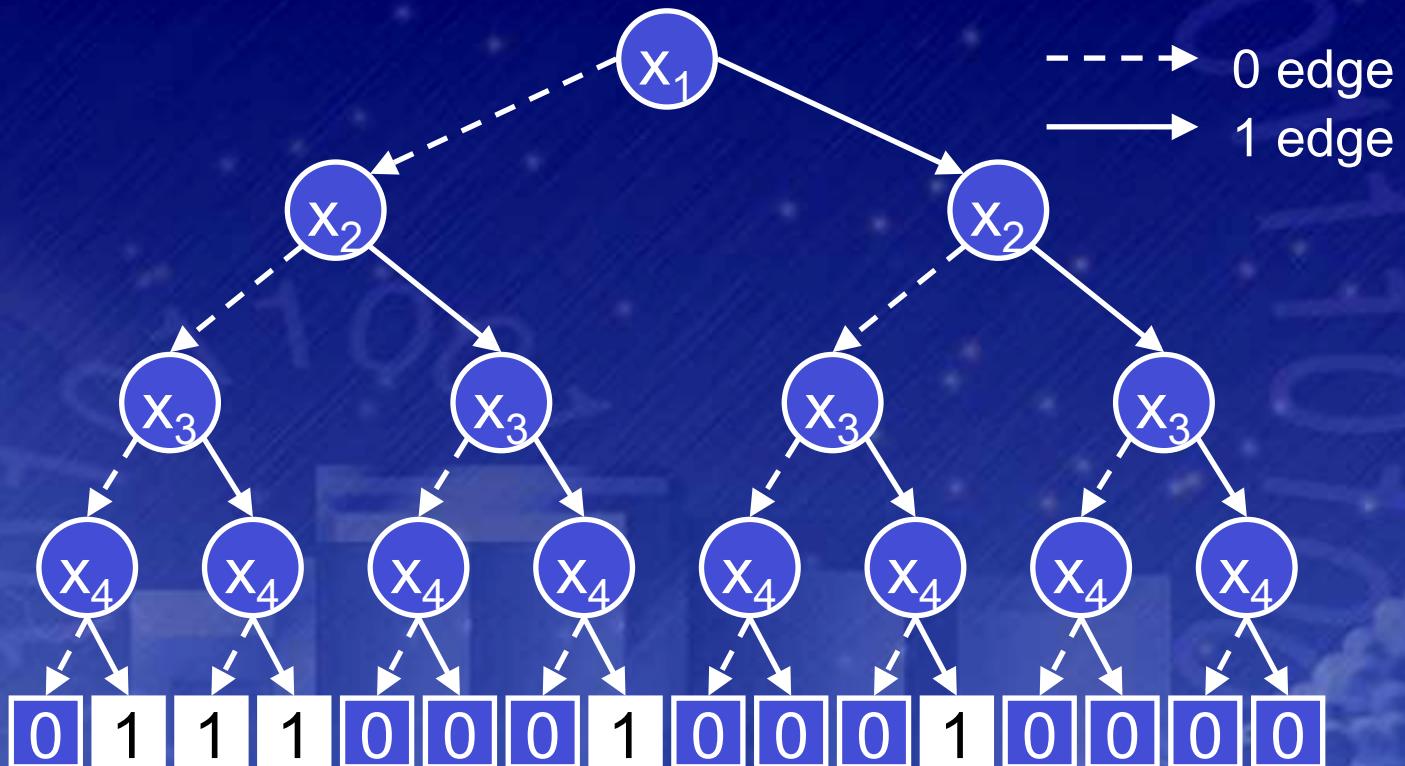
BDDs from Truth Tables



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

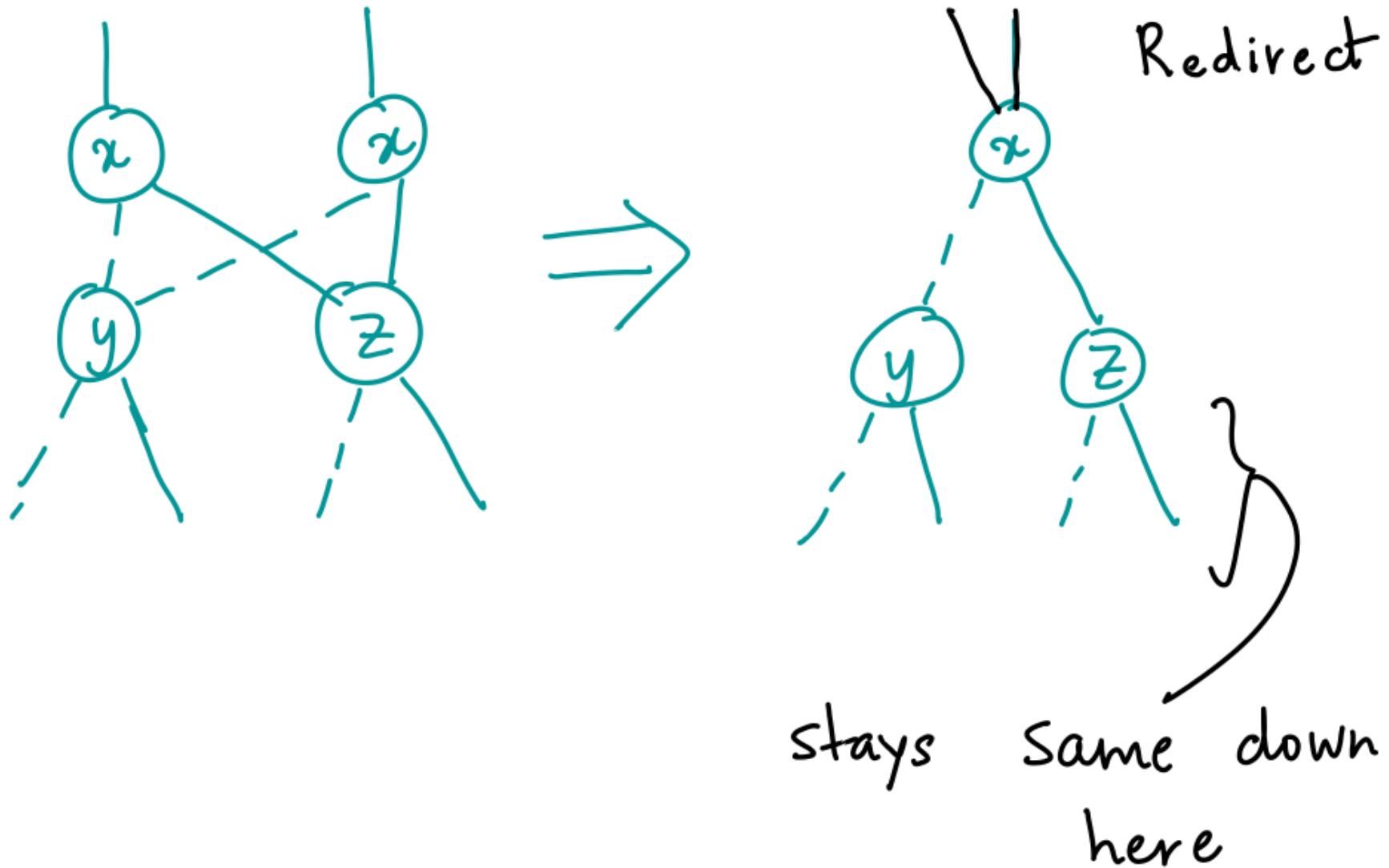
- Identify Redundancies
- 3 Rules
 - Merge equivalent leaves
 - Merge isomorphic nodes
 - Eliminate redundant tests

Merge equivalent leaves

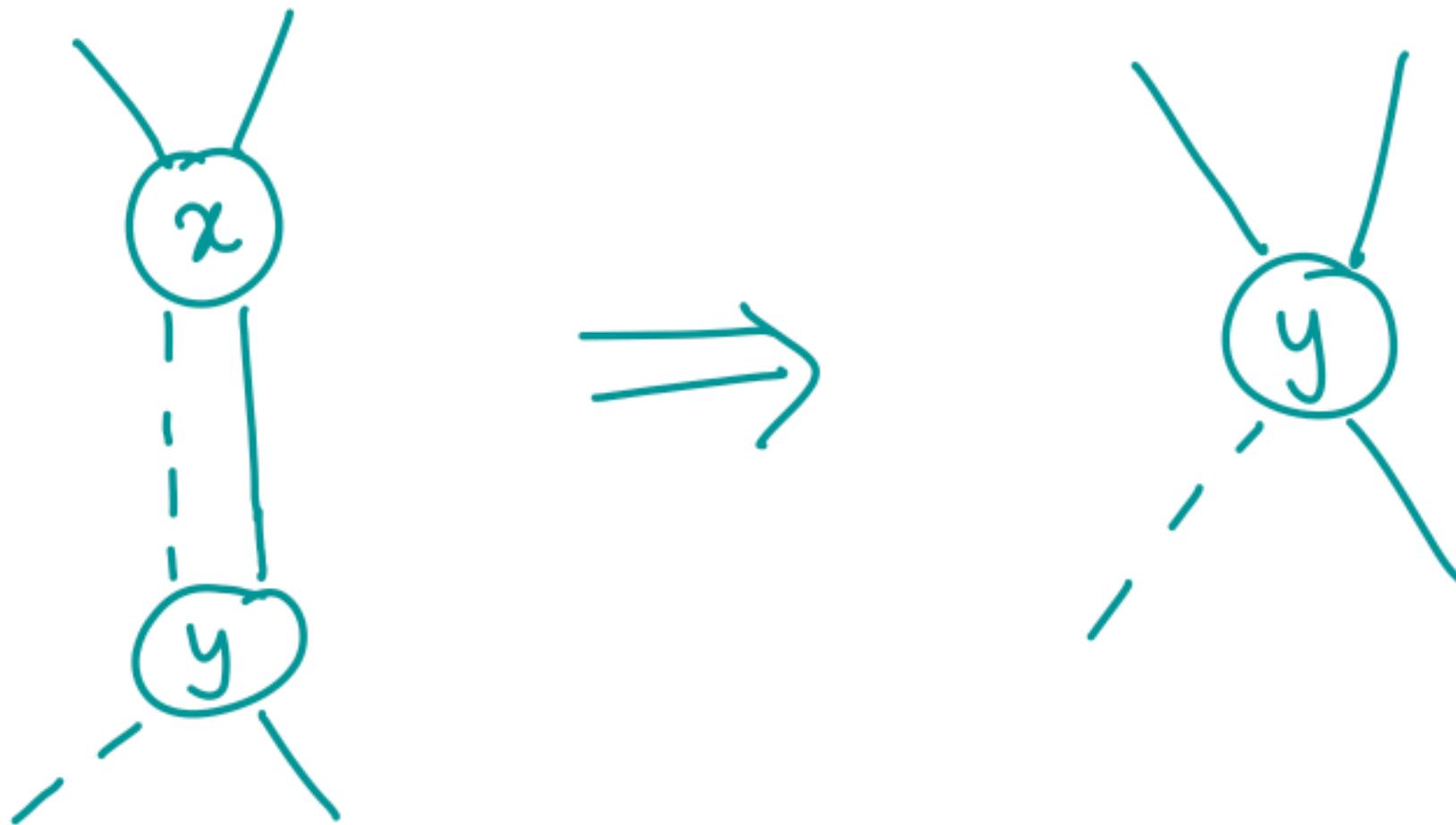


"a" is either 0 or 1

Merge isomorphic nodes

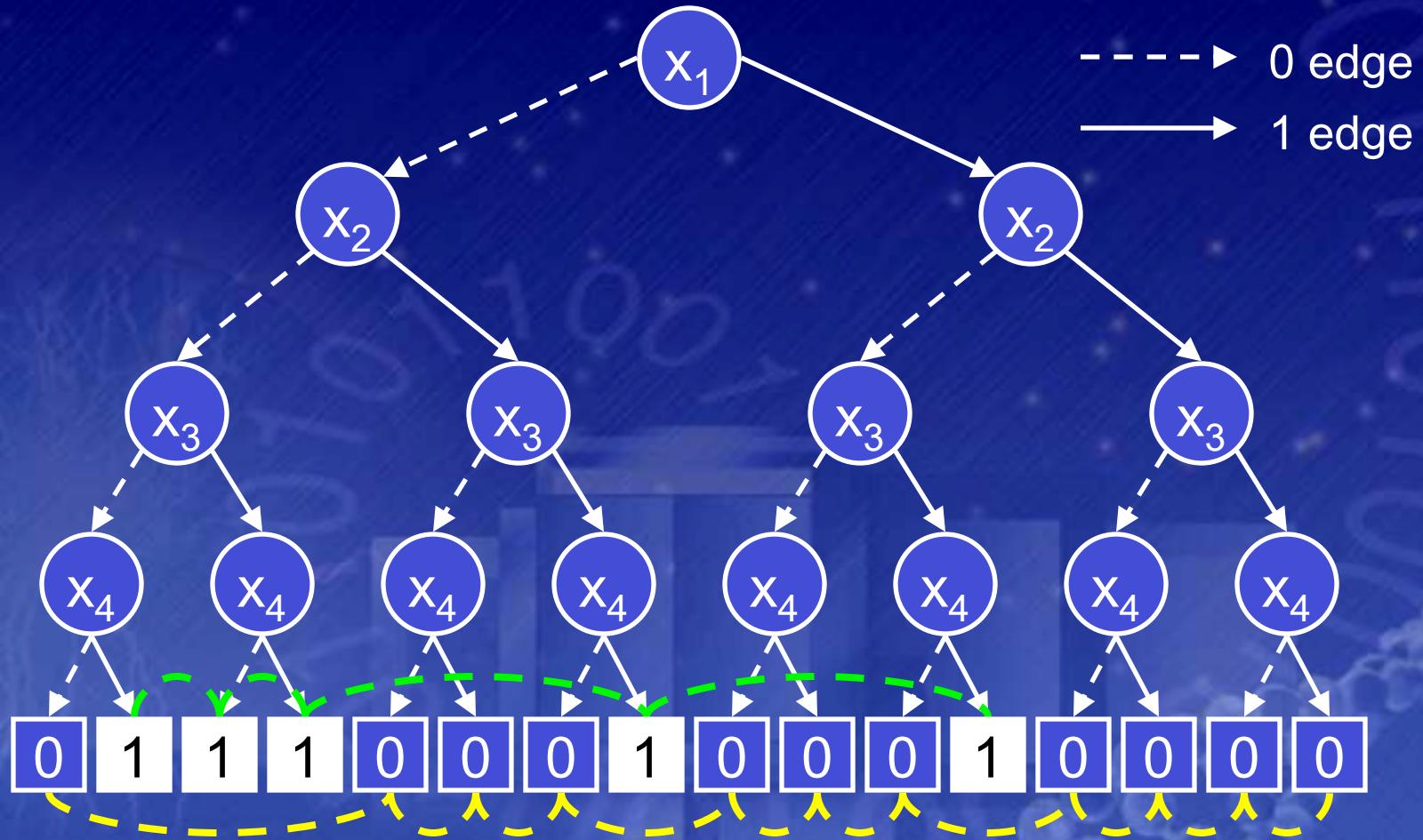


Eliminate redundant tests



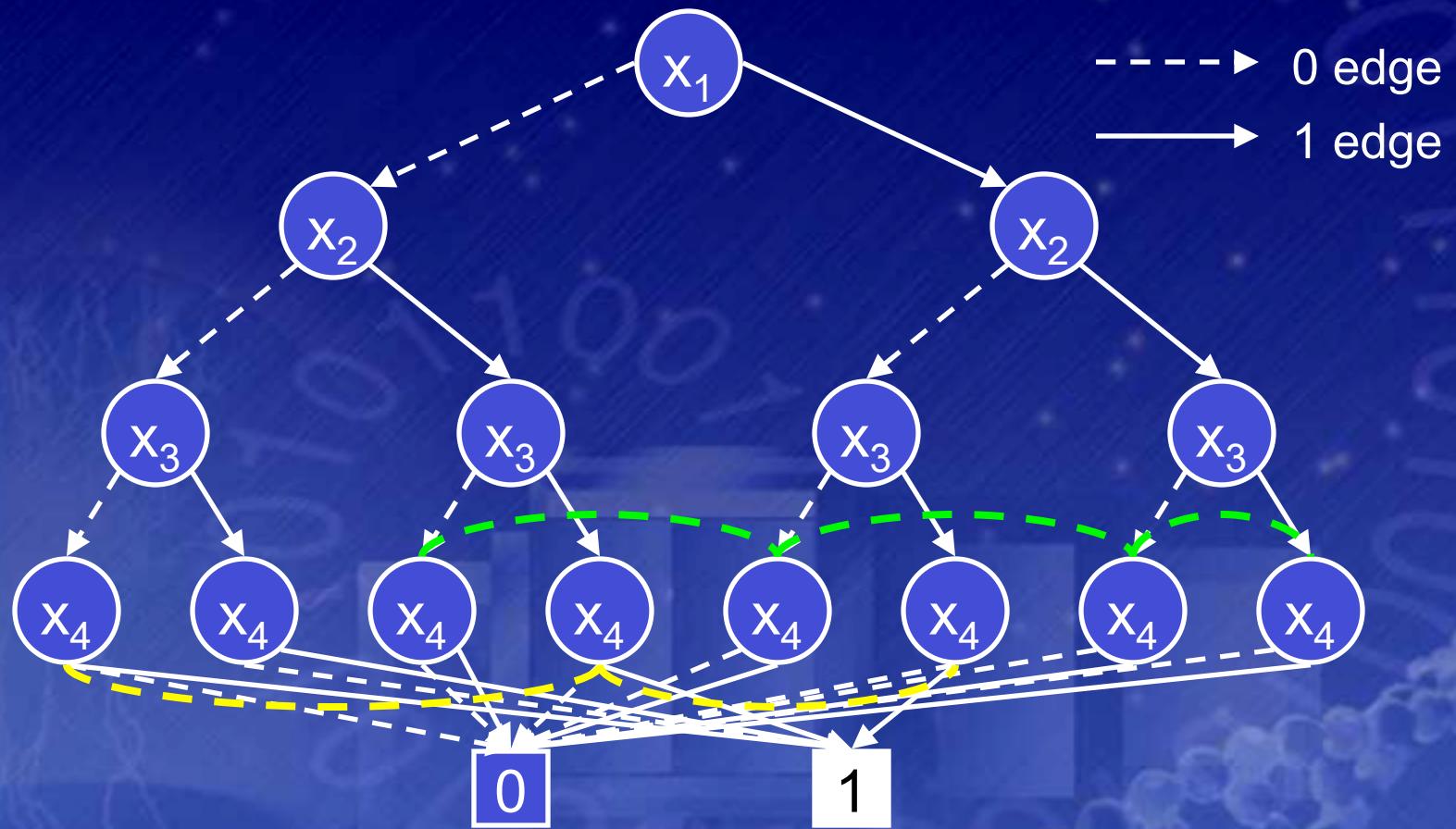
Binary Decision Diagrams

- Collapse redundant nodes.



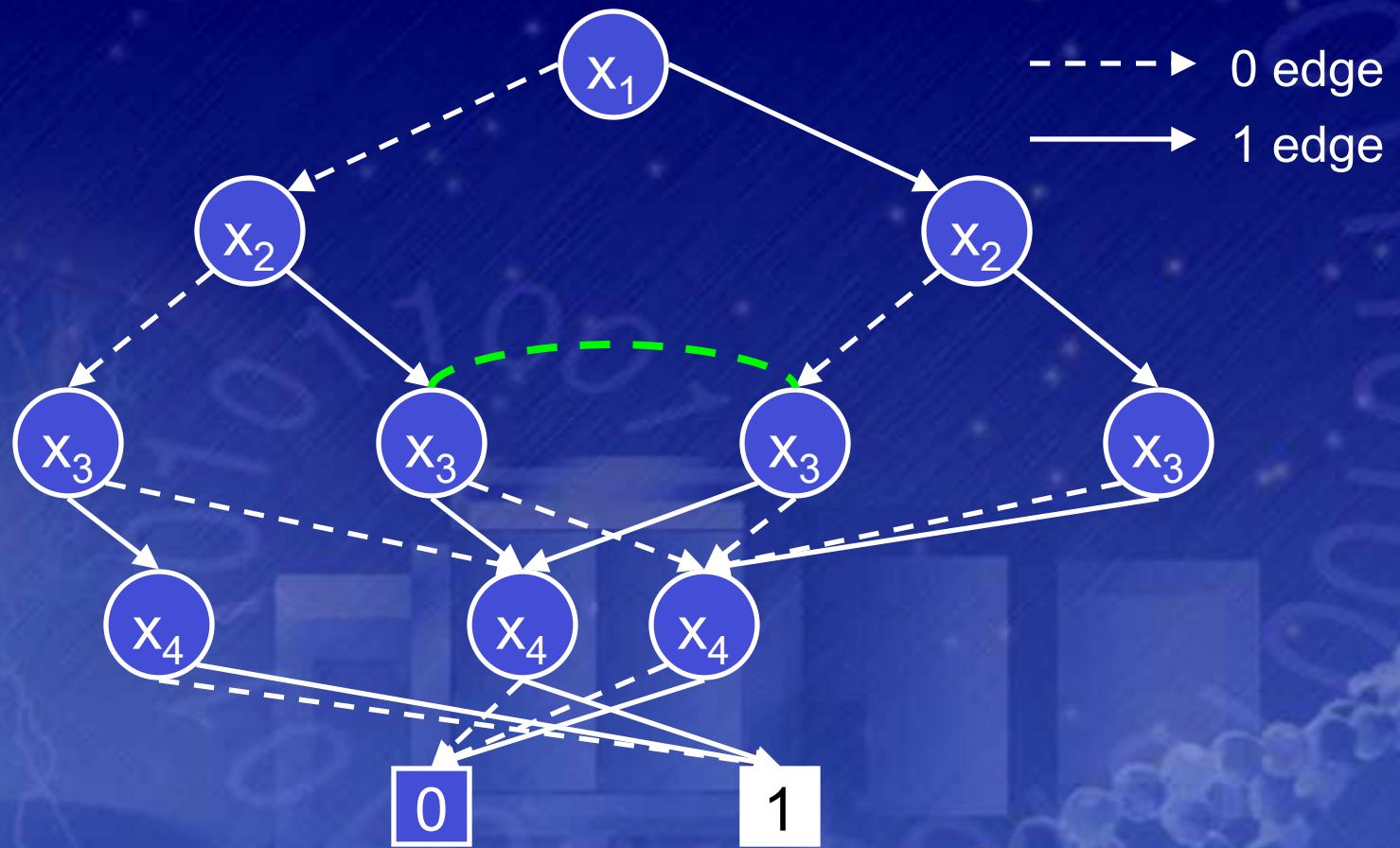
Binary Decision Diagrams

- Collapse redundant nodes.



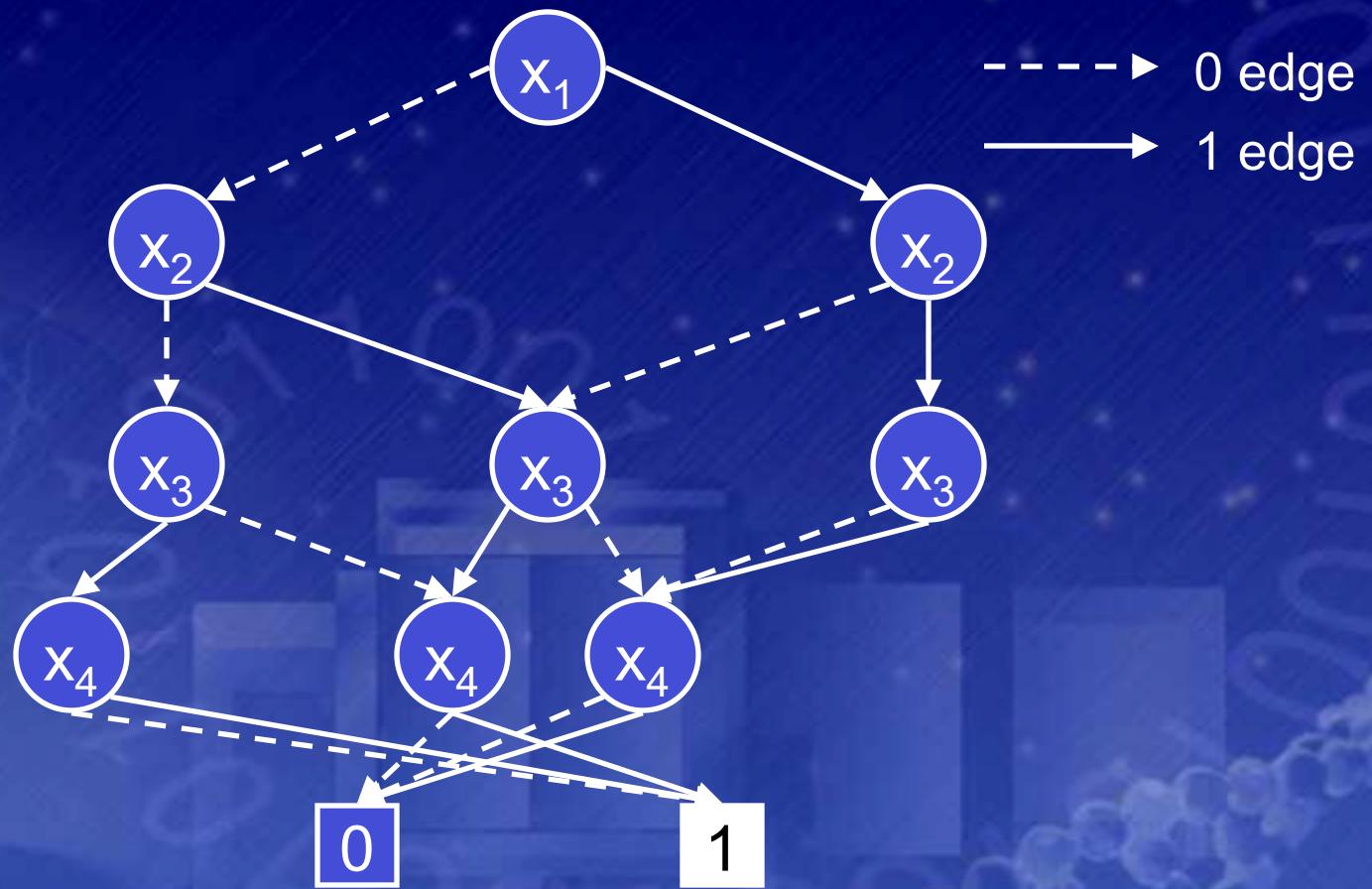
Binary Decision Diagrams

- Collapse redundant nodes.



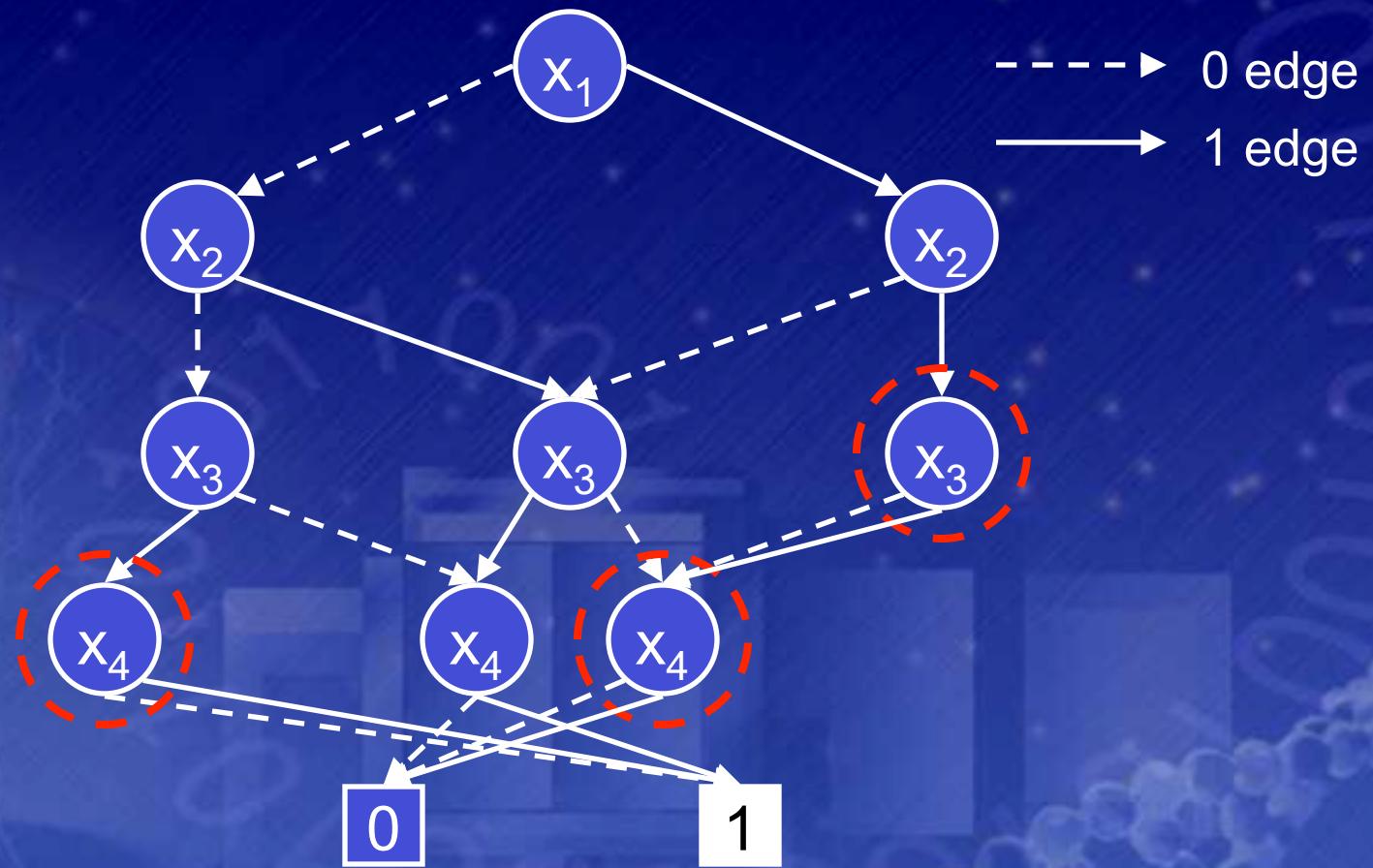
Binary Decision Diagrams

- Collapse redundant nodes.



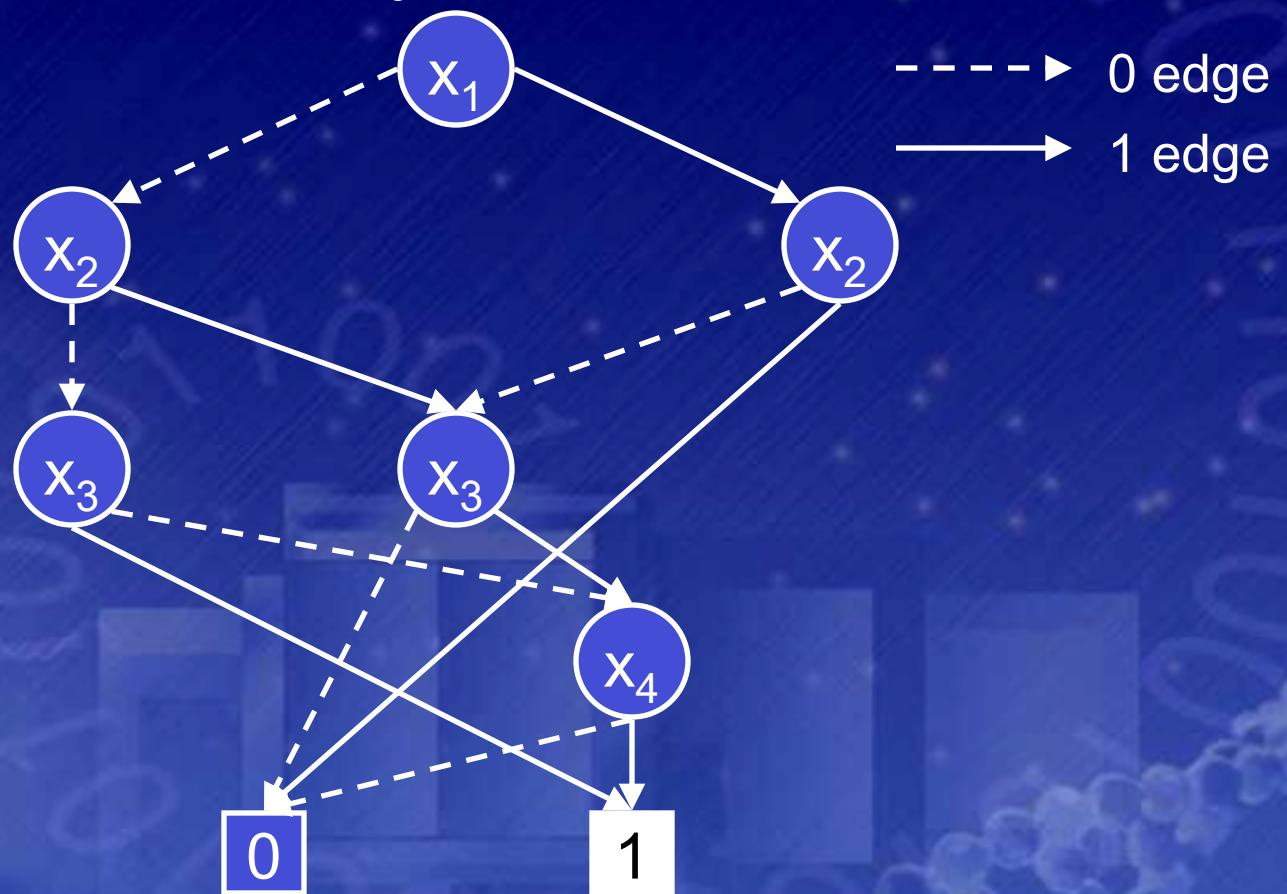
Binary Decision Diagrams

- Eliminate unnecessary nodes.



Binary Decision Diagrams

- Eliminate unnecessary nodes.

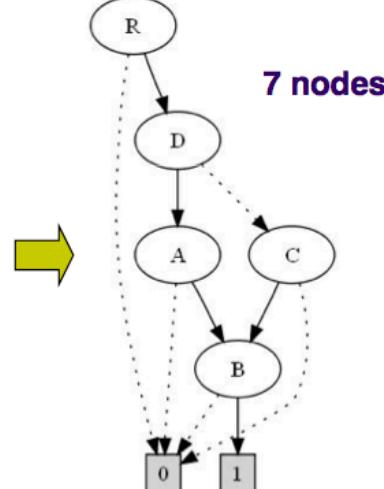
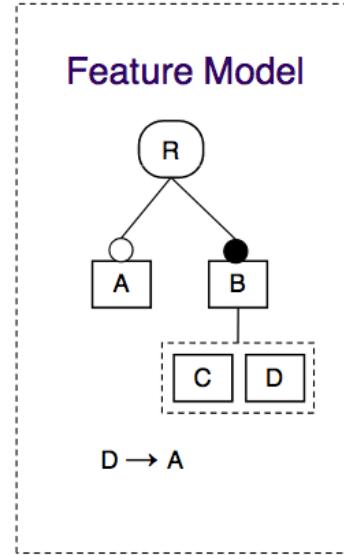
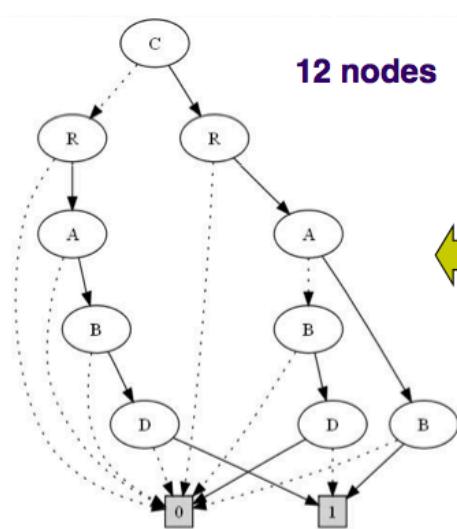


Binary Decision Diagrams (BDDs)

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

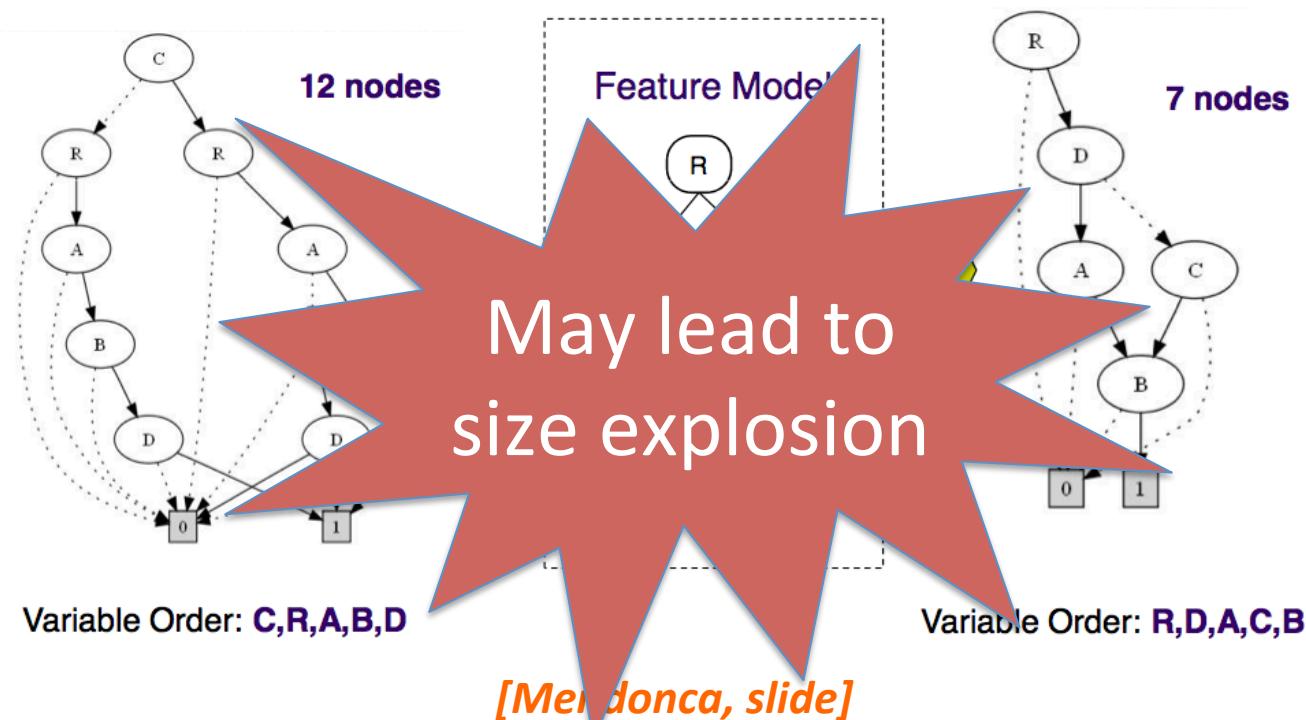


Variable Order: **C,R,A,B,D**

Variable Order: **R,D,A,C,B**

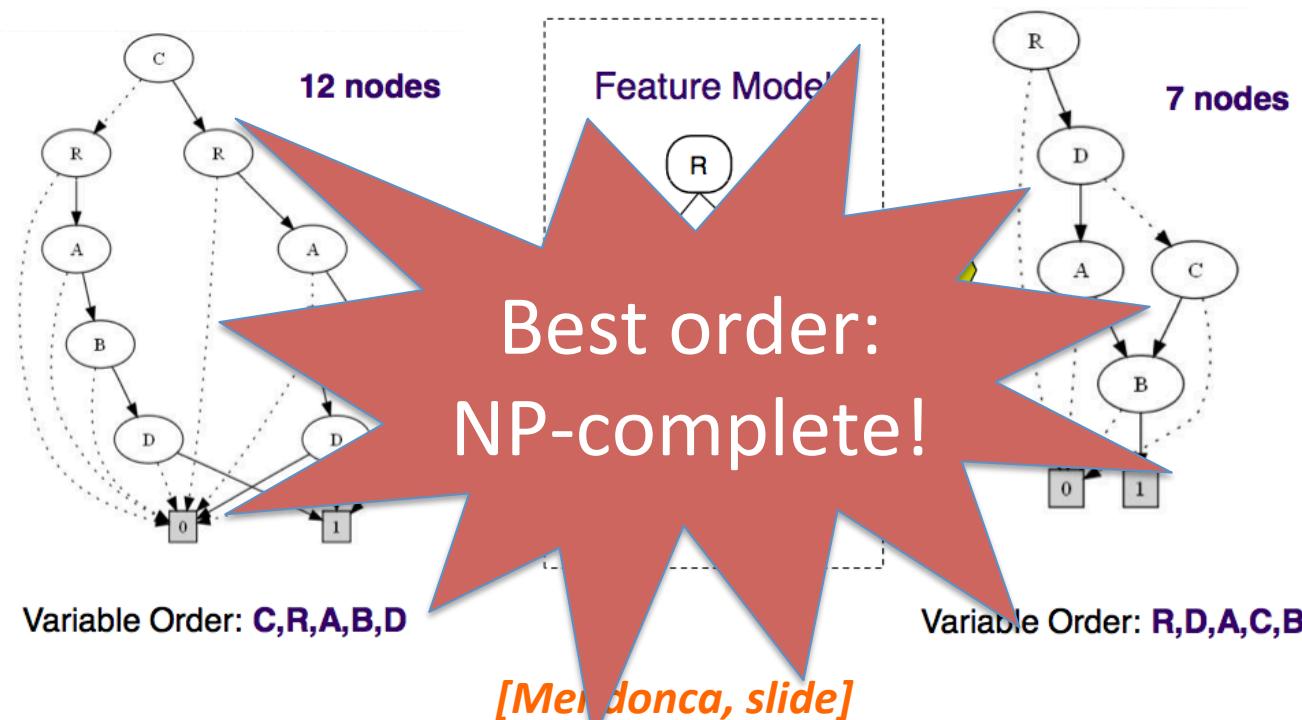
Binary Decision Diagrams (BDDs): Theoretical Problem

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



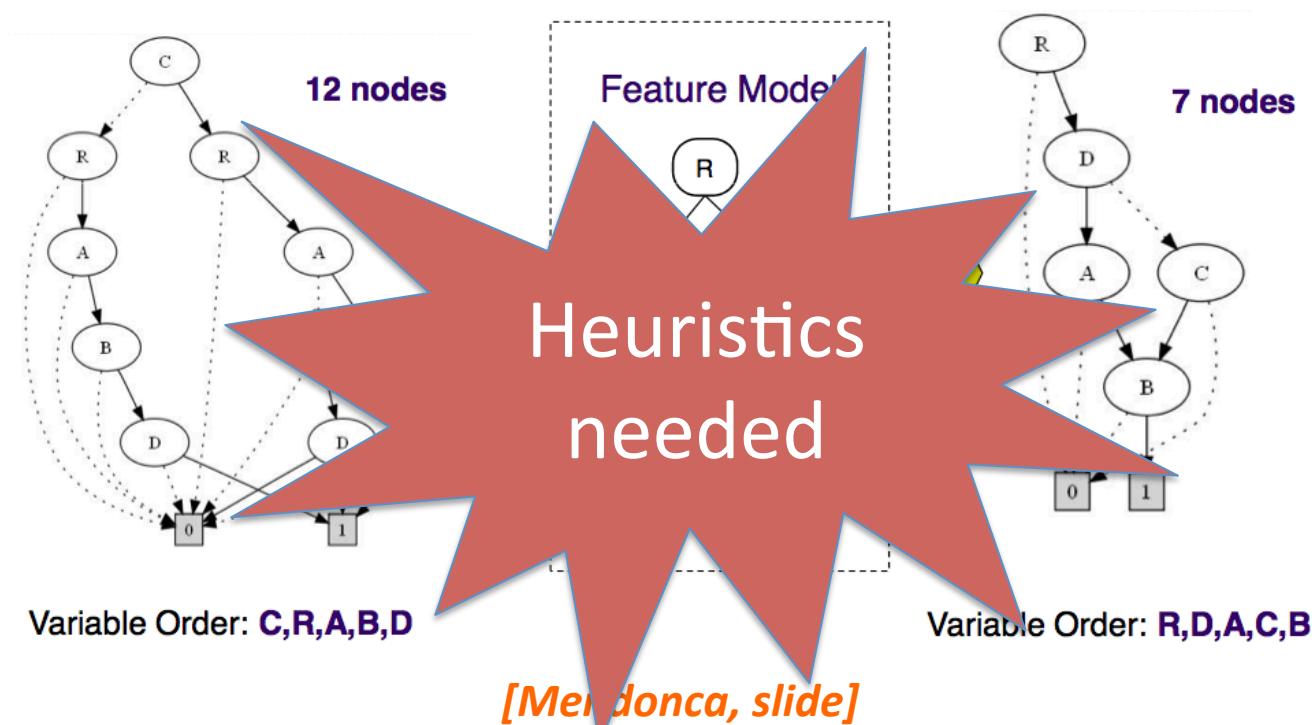
Binary Decision Diagrams (BDDs): Theoretical Problem

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



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**

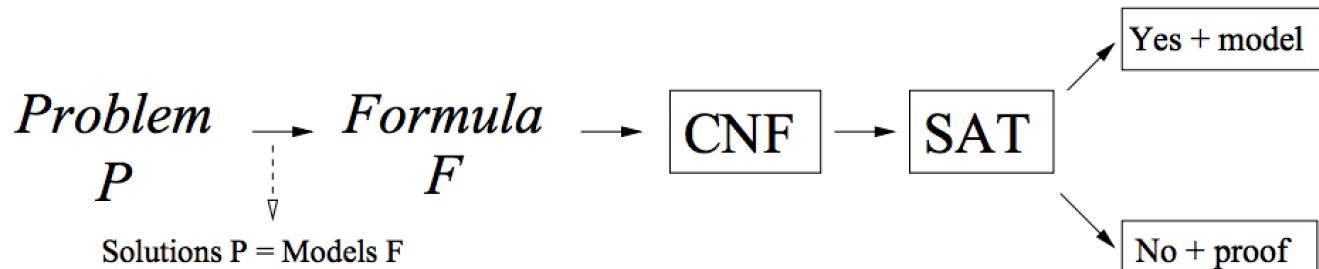


How to automate analysis of your feature models?

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 $\neg q$).
 - A clause is a disjunction of literals (e.g., $(p \vee \neg q \vee r)$). Since \vee 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., $(p \vee q \vee \neg r) \wedge (\neg p \vee s \vee t \vee \neg u)$

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.

$$\{\neg p \vee q, \neg p \vee \neg q \vee r, p, \neg r\}.$$

$$\begin{array}{c|c} \begin{array}{l} \neg p \vee q \\ \neg p \vee \neg q \vee r \\ p \\ \neg r \end{array} & \begin{array}{l} q \\ \neg q \vee r \\ \neg r \end{array} \end{array}$$

- A good SAT solver often spends 80-90% of its time in unit propagation.

$$\begin{aligned}
\mathcal{F}_{\text{unit}} := & (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \\
& (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \\
& (x_1 \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)
\end{aligned}$$

$$\begin{aligned}\mathcal{F}_{\text{unit}} := & (\neg \textcolor{red}{x}_1 \vee \neg x_3 \vee x_4) \wedge (\neg \textcolor{red}{x}_1 \vee \neg x_2 \vee x_3) \\ & (\neg \textcolor{red}{x}_1 \vee x_2) \wedge (\textcolor{green}{x}_1 \vee x_3 \vee x_6) \wedge (\neg \textcolor{red}{x}_1 \vee x_4 \vee \neg x_5) \\ & (\textcolor{green}{x}_1 \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)\end{aligned}$$

$$\varphi = \{x_1=1\}$$

$$\begin{aligned}\mathcal{F}_{\text{unit}} := & (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \\ & (\neg x_1 \vee x_2) \wedge (\textcolor{green}{x_1} \vee x_3 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \\ & (\textcolor{green}{x_1} \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)\end{aligned}$$

$$\varphi = \{x_1=1, \textcolor{teal}{x_2}=1\}$$

$$\begin{aligned}
\mathcal{F}_{\text{unit}} := & (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \\
& (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \\
& (x_1 \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)
\end{aligned}$$

$$\varphi = \{x_1=1, x_2=1, x_3=1\}$$

$$\begin{aligned}\mathcal{F}_{\text{unit}} := & (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \\ & (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_6) \wedge (\neg x_1 \vee x_4 \vee \neg x_5) \\ & (x_1 \vee \neg x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_5 \vee \neg x_6)\end{aligned}$$

$$\varphi = \{x_1=1, x_2=1, x_3=1, x_4=1\}$$

SAT solver and Unit Propagation

BCP():

Repeatedly search for unit clauses, and
set unassigned literal to required value.

If a literal is assigned conflicting values, return F
else return T;

satisfy(ϕ) {

if every clause of ϕ has a true literal, return T;

if BCP() == F, return F;

assign appropriate values to all pure literals;

choose an $x \in V$ that is unassigned in A ,

and choose $v \in \{T, F\}$.

$A(x) = v$;

if satisfy(ϕ) return T;

$A(x) = \neg v$;

if satisfy(ϕ) return T;

unassign $A(x)$; // undo assignment for backtracking.

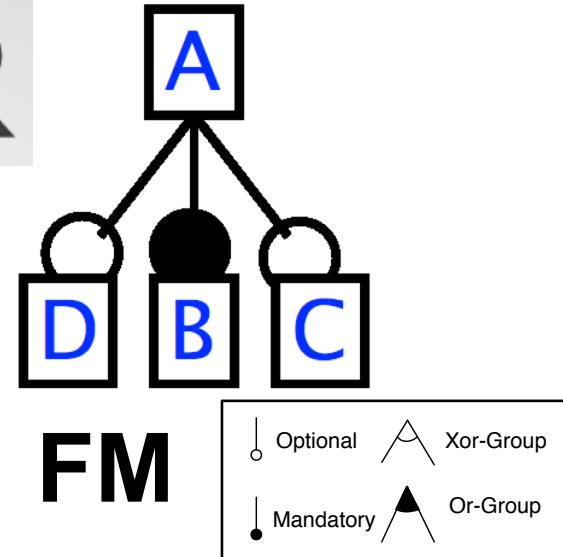
return F; }

How to automate analysis of your feature models?

Let us use BDDs and SAT solvers

$A \wedge$
 $A \Leftrightarrow B \wedge$
 $C \Rightarrow A \wedge$
 $D \Rightarrow A$

FAMiliAR



φ

```

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

```

```

fml> c1 = cores fm1
fml> s1c1: (SET) {B;A}
s1: (SET) {B;A}
fml> c1bis = cores fm1bis
fml> compare fm1 fm1bis
res7: (STRING) REFACTORING
fml> fml> compare fm1bis fm1bisbis
s1bis res8: (STRING) REFACTORING
fml> fml> compare fm1 fm1bisbis
res3: (BOOLEAN) true
fml> s1res6: (BOOLEAN) true
fml> res4: (BOOLEAN) true

```

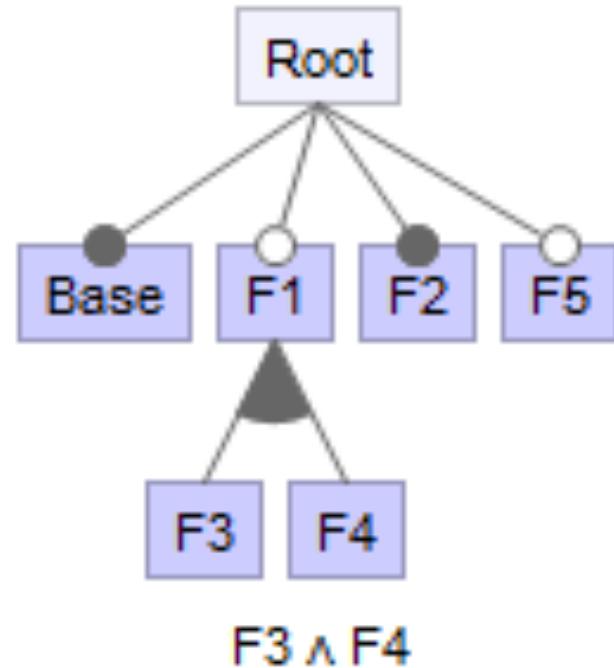
```

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

```

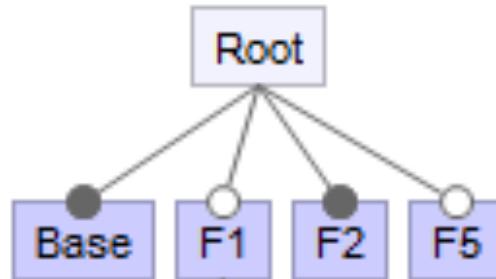
Consistency

- SAT-Solver
 - SAT(FM)



Core and dead features

- Dead : $\text{SAT}(\text{FM} \wedge F)$
- Core: $\text{SAT}(\text{FM} \wedge \text{not}(F))$



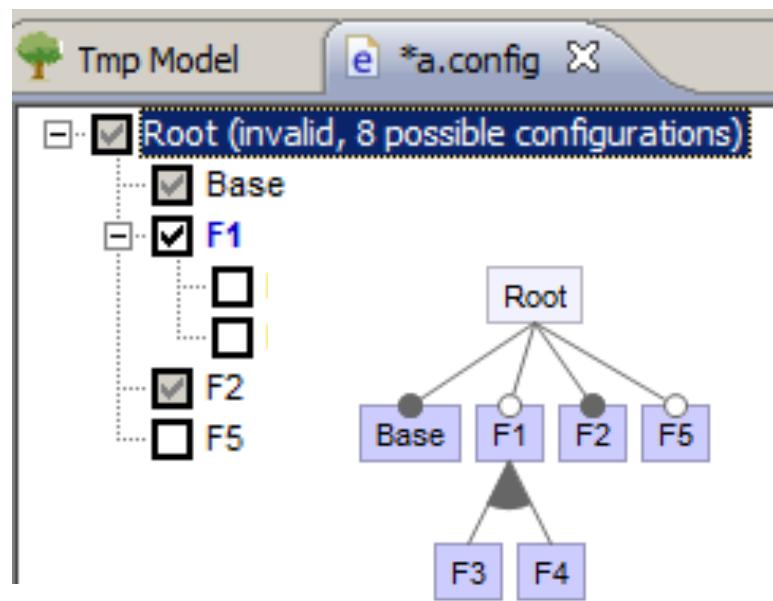
$$F5 \Rightarrow F4 \vee \text{Base}$$

$$F3 \Rightarrow F2 \wedge F5$$

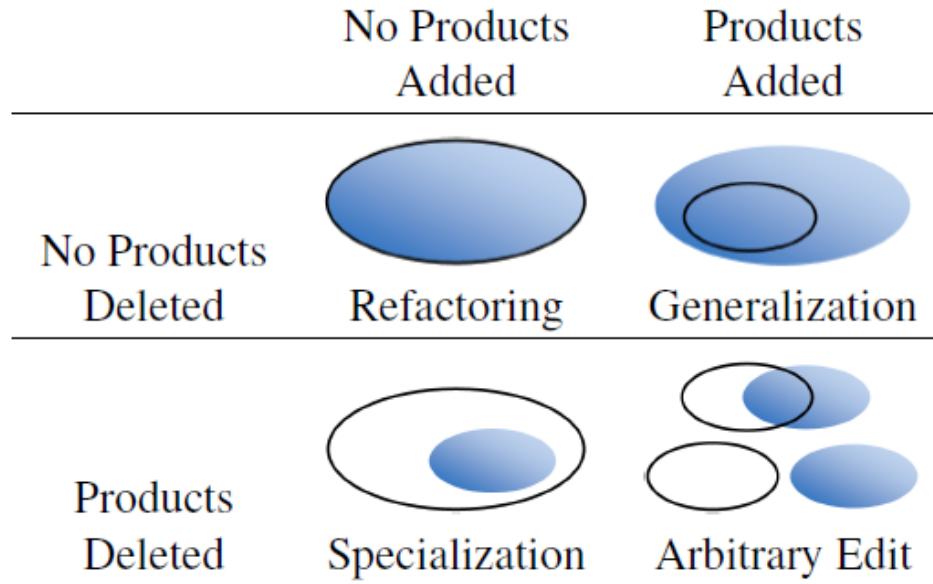
$$\neg(F4 \wedge F2)$$

Partial configuration

- $SAT(FM \wedge PK \wedge F)$
- $SAT(FM \wedge PK \wedge \neg(F))$



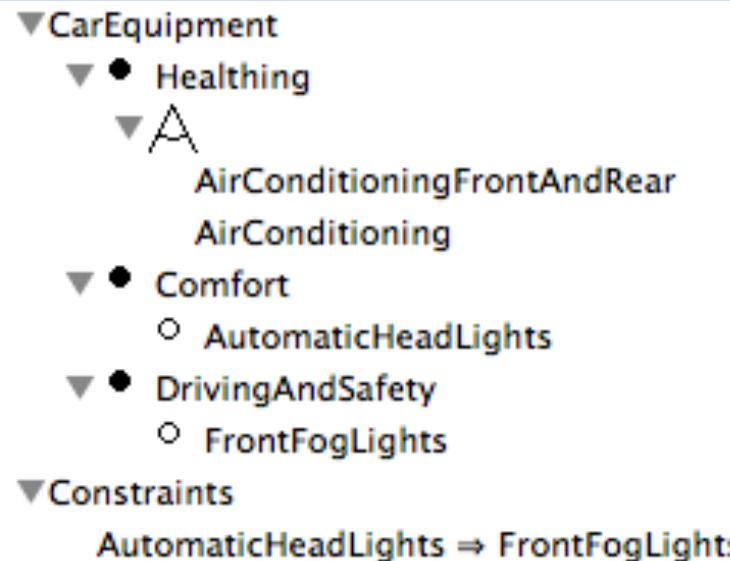
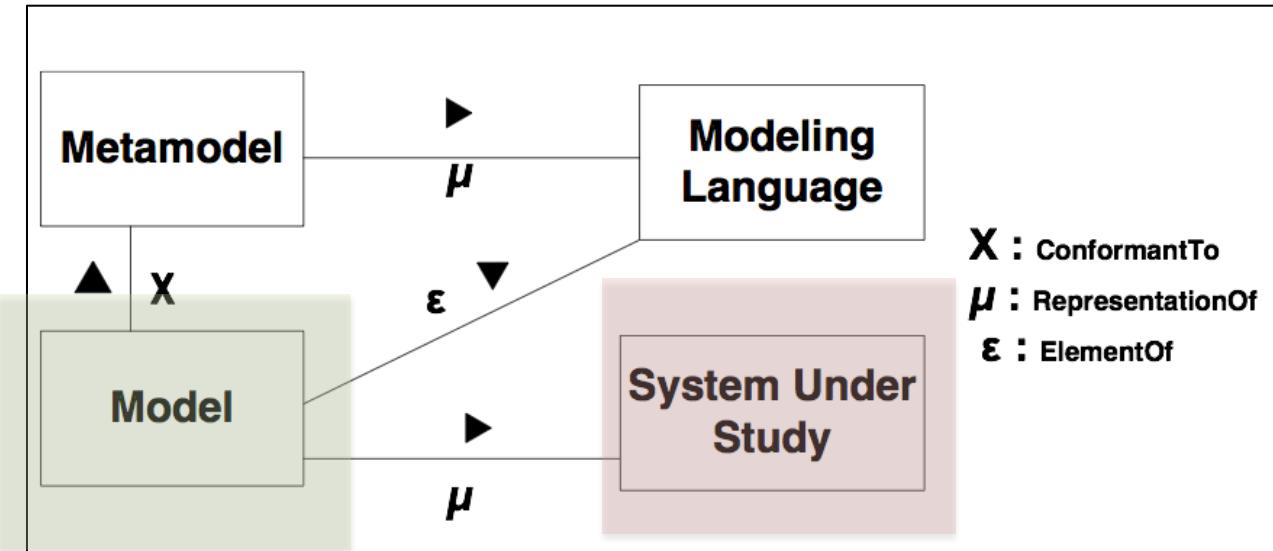
Relationship between feature models



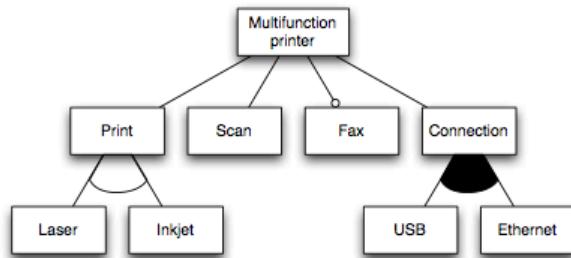
- Refactoring
 - Tautology: $(FM1 \Leftrightarrow FM2)$
 $= \text{not SAT}(\text{not } (FM1 \Leftrightarrow FM2))$

Recap

Feature Models



Typical implementations



Fontsource (Attributed - Free Processing 2012) (Attributed - Creative Commons)



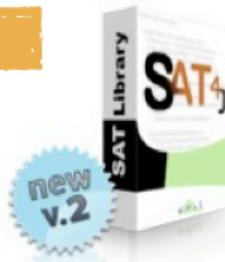
result



logics



solvers



Z3

FAMILIAR

(FeAture Model script Language for manipulation and Automatic Reasoning)

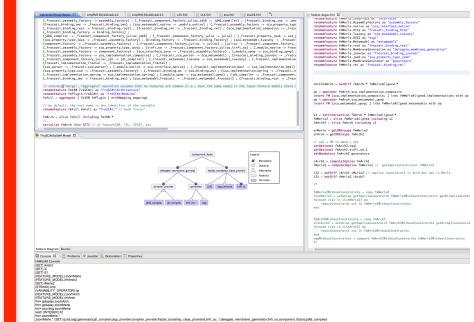
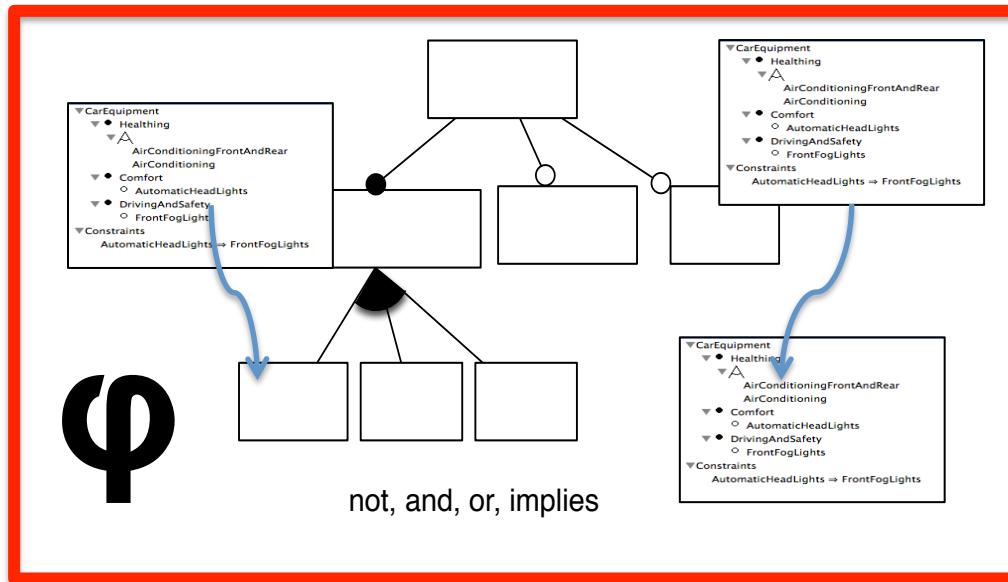
<http://familiar-project.github.com/>



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



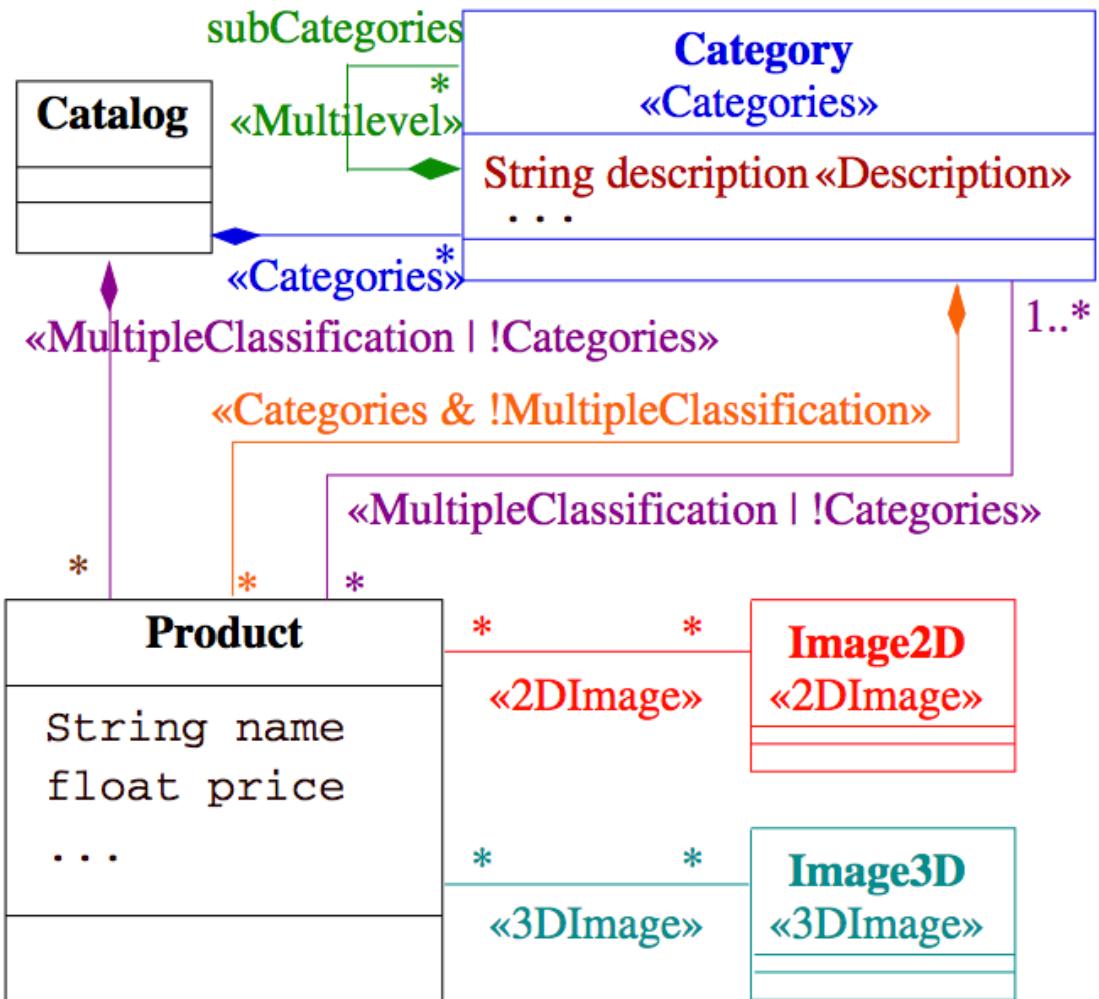
TVL
DIMACS

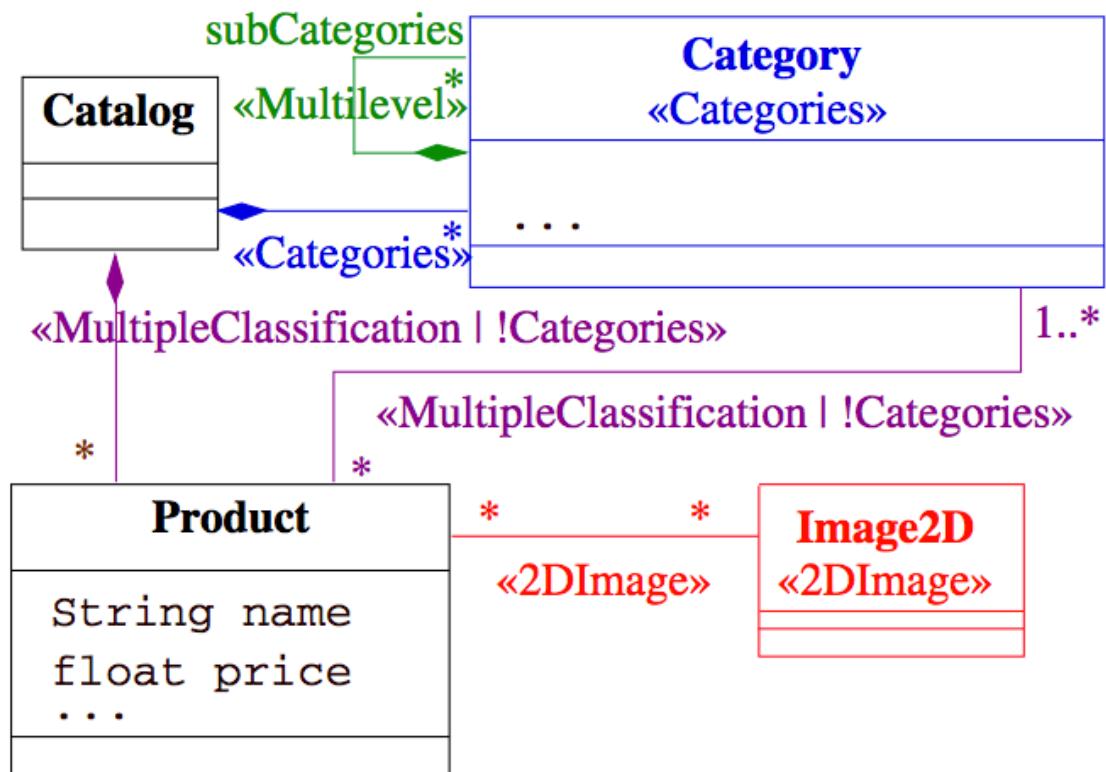
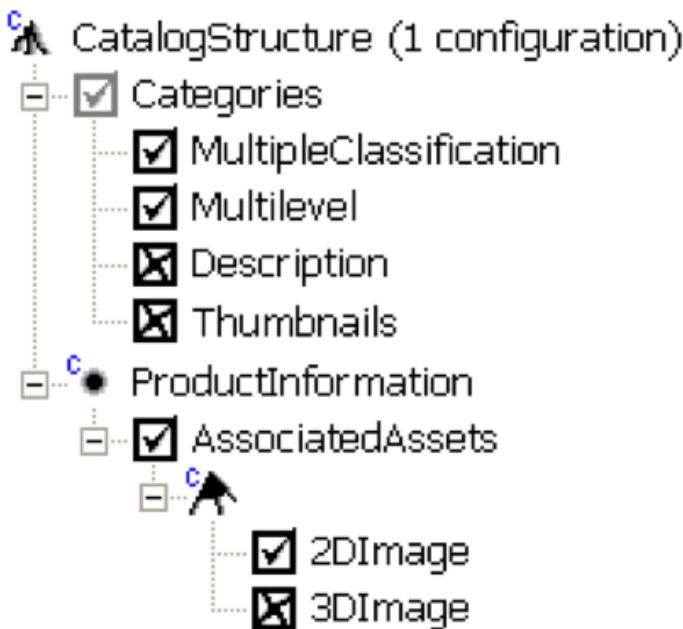


importing, exporting, composing, decomposing, editing, configuring,
reverse engineering, computing "diffs", refactoring, testing,
and reasoning about (multiple) variability models

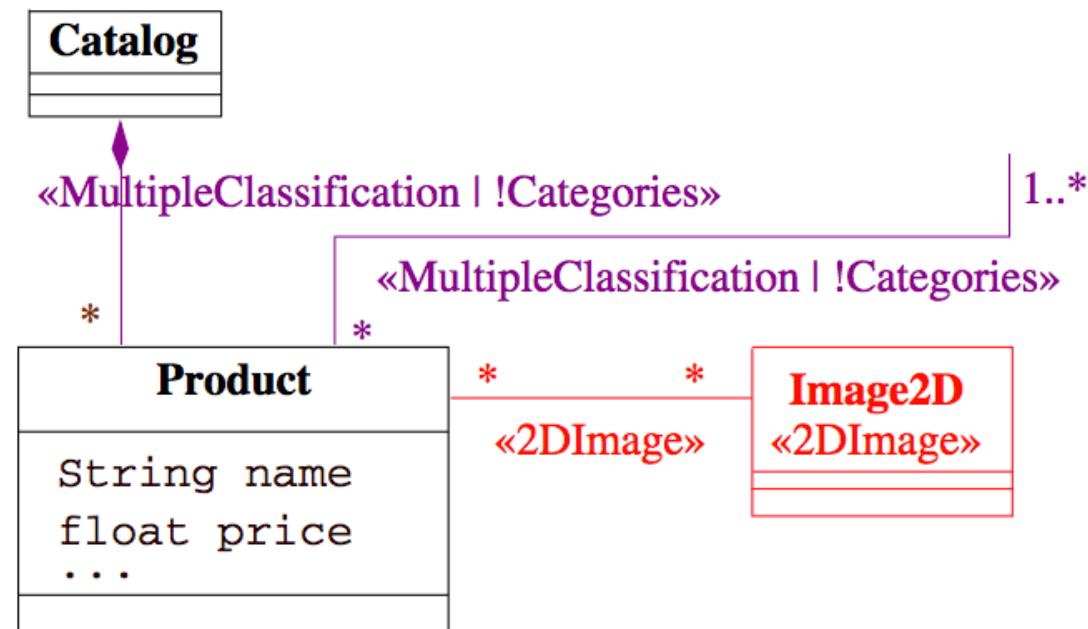
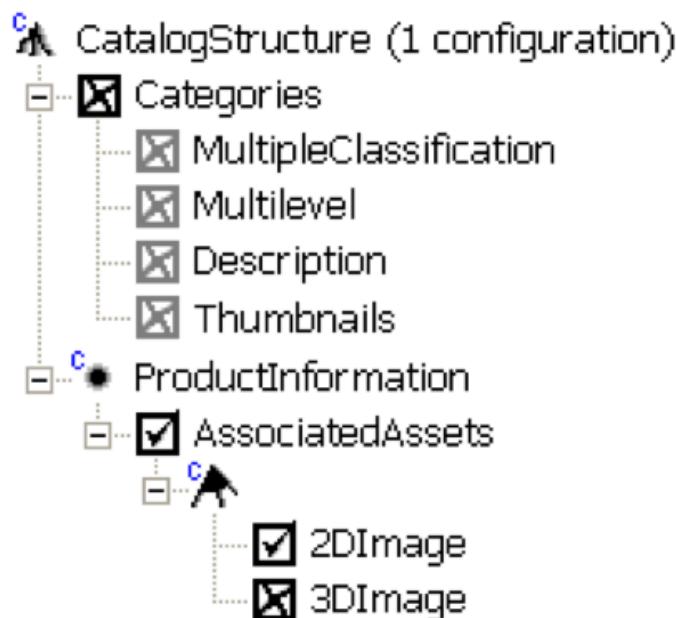
▲ CatalogStructure (52 configurations)

- Categories
 - MultipleClassification
 - Multilevel
 - Description
 - Thumbnails
- ProductInformation
 - AssociatedAssets
 - 2DImage
 - 3DImage



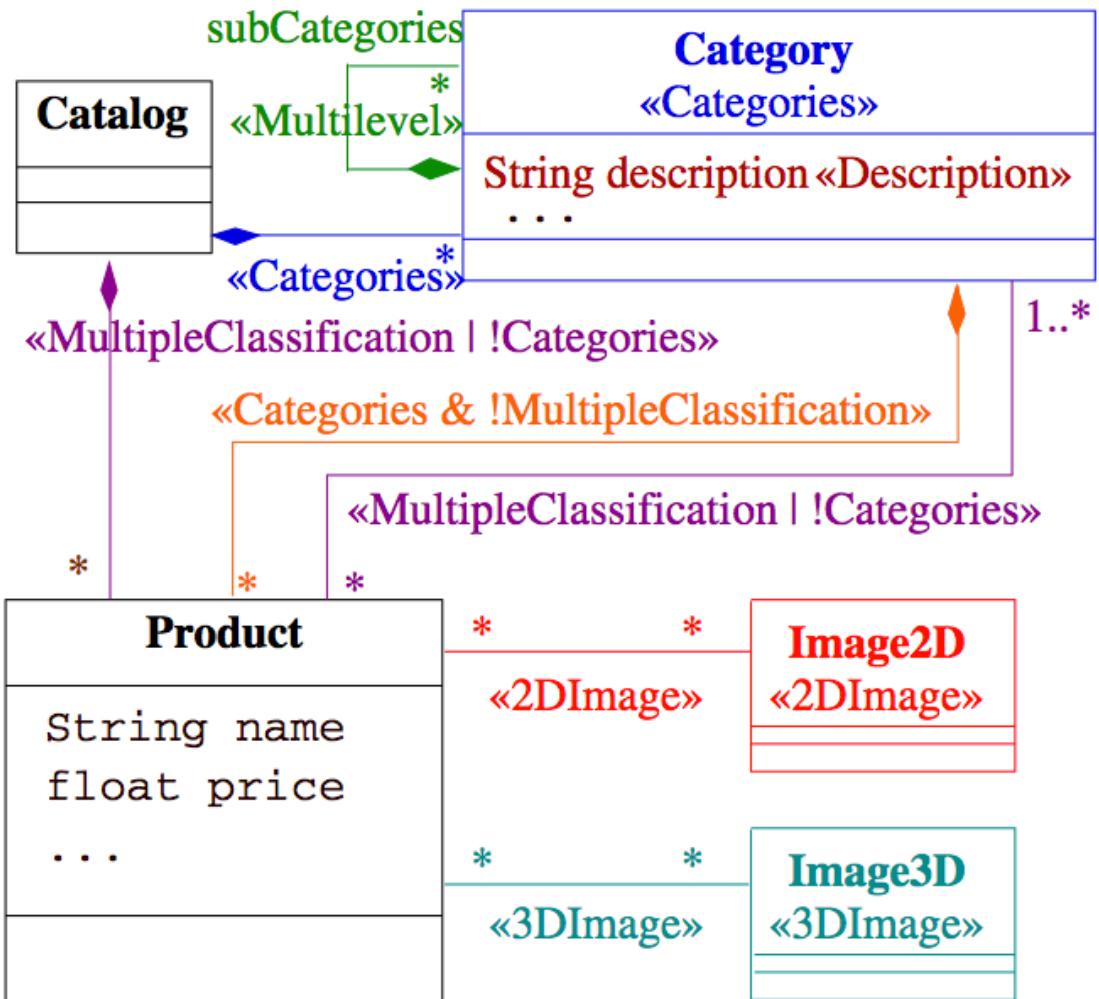
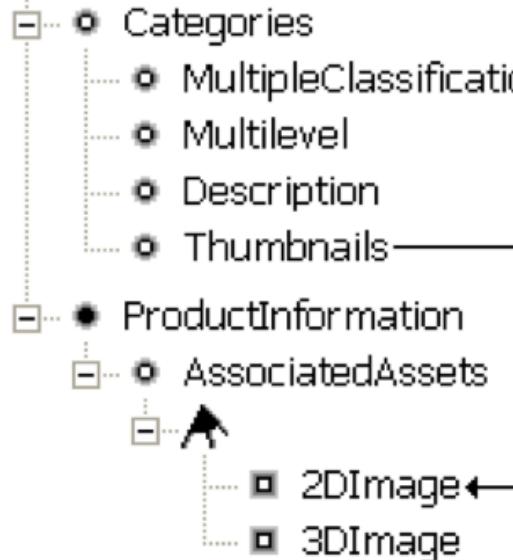


Ooops



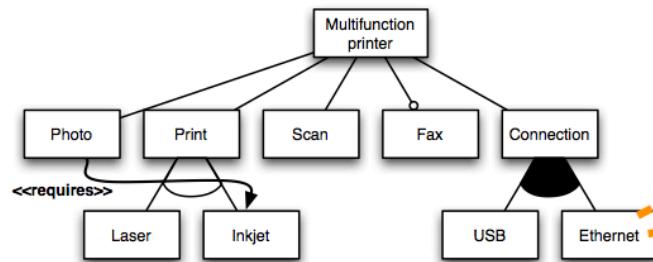
Safe composition? No!

CatalogStructure (52 configurations)



Product Derivation

feature model

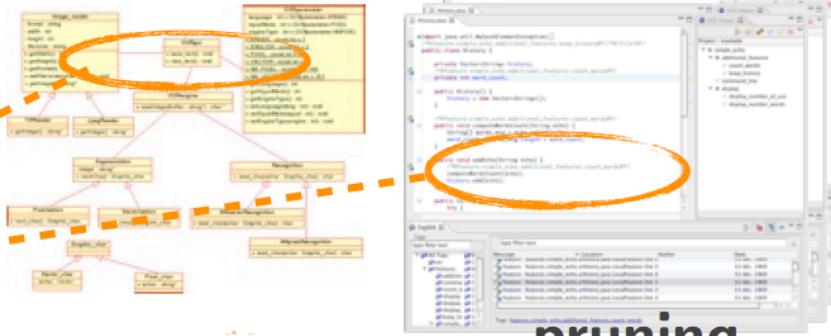


configuration

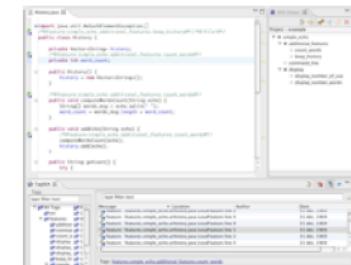
{ MP, Photo, Print, Inkjet, Scan,
Fax, Connection, USB, Ethernet }

product spec

variable model and
code assets

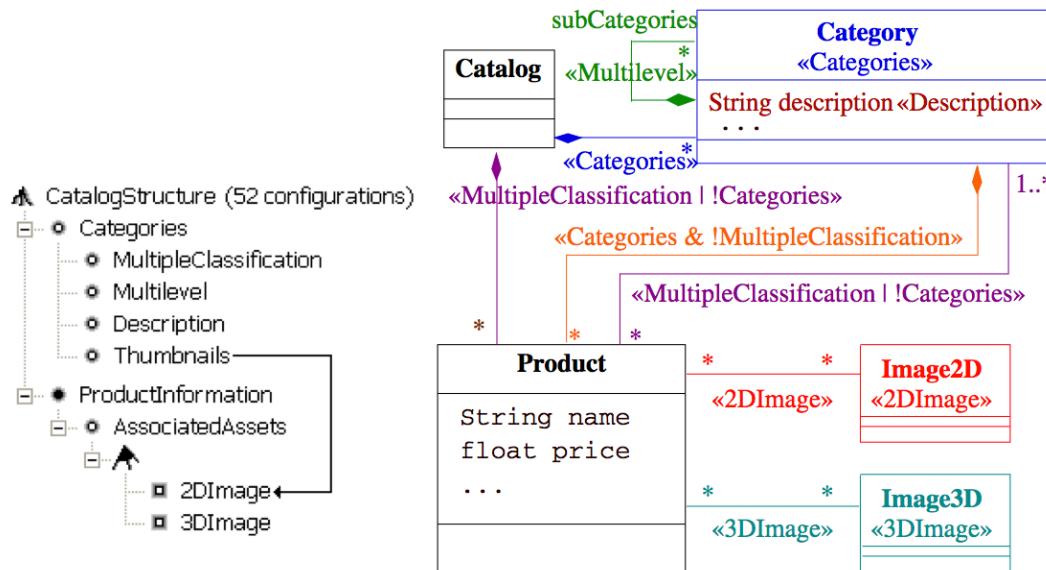


pruning,
composition,
weaving,
transformation



product

Safe composition: how does it work?



```

qFM          =
root:        cs ∧
child-parent: (ct ⇒ cs) ∧ (mc ⇒ ct) ∧ (ml ⇒ ct) ∧
              (ds ⇒ ct) ∧ (tn ⇒ ct) ∧ (pi ⇒ cs) ∧
              (aa ⇒ pi) ∧ (i2 ⇒ aa) ∧ (i3 ⇒ aa) ∧
              (aa ⇒ choice1,2(i2, i3)) ∧
              (cs ⇒ pi) ∧
              (tn ⇒ i2)
group:
mandatory:
additional:
  
```



**Another
propositional
formula**