COMBINATORIAL TESTING FOR FEATURE MODELS USING CITLAB

Int. Workshop on Combinatorial Testing (IWCT) @ ICST 2013

Angelo Gargantini

Università di Bergamo - Italy

http://cs.unibg.it/gargantini

Joint work with Paolo Vavassori – Università di Bergamo

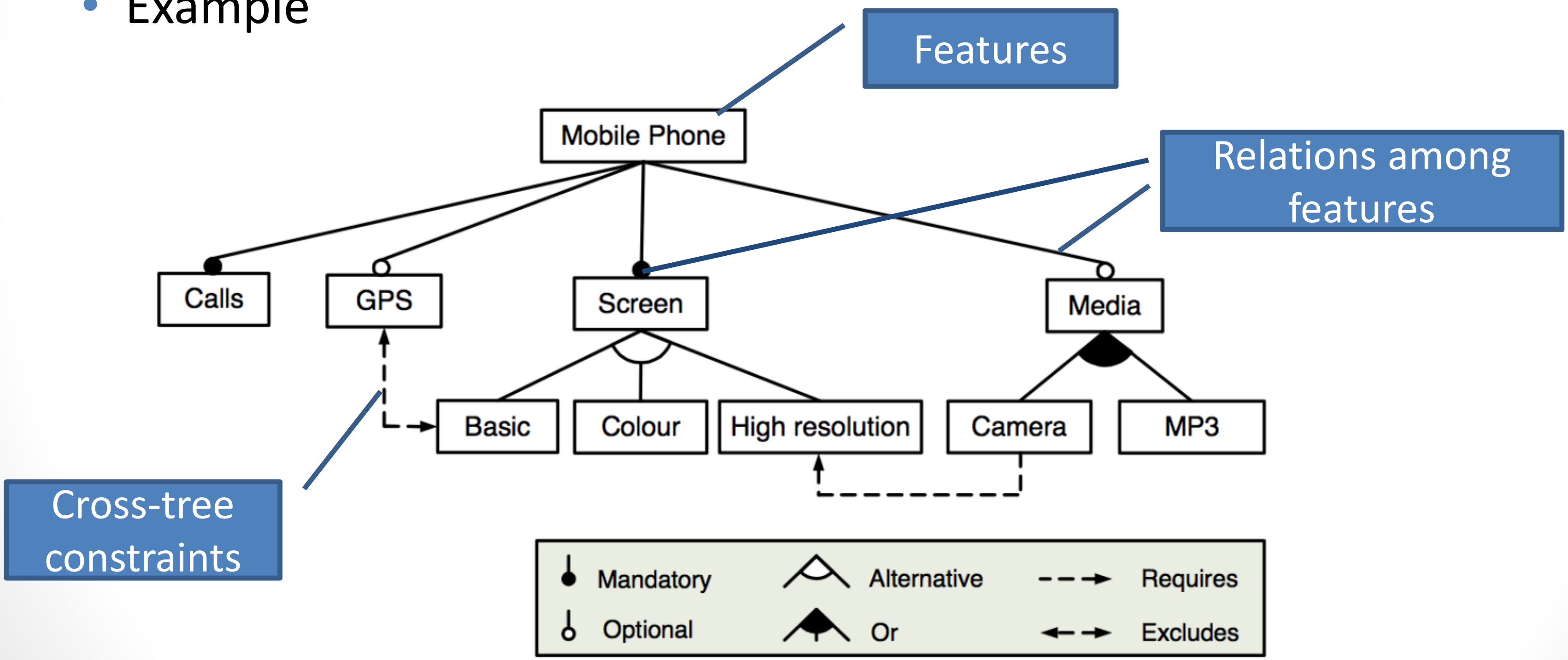
and Andrea Calvagna Università di Catania

SPLs, FM, and CIT

- Software Product Lines & Feature Models
  - SPLs and FMs are used to represent all the possible products of a software product line in terms of features and relationships among them.
- Combinatorial Interaction Testing
  - Often required for SPLs
- Current approach
  - Adapt CIT algorithms and tools for SPLs
- **OUR PROPOSAL** **FM2CitLab**
  - Use a tool for combinatorial testing (CitLab) for test generation starting from Feature Models

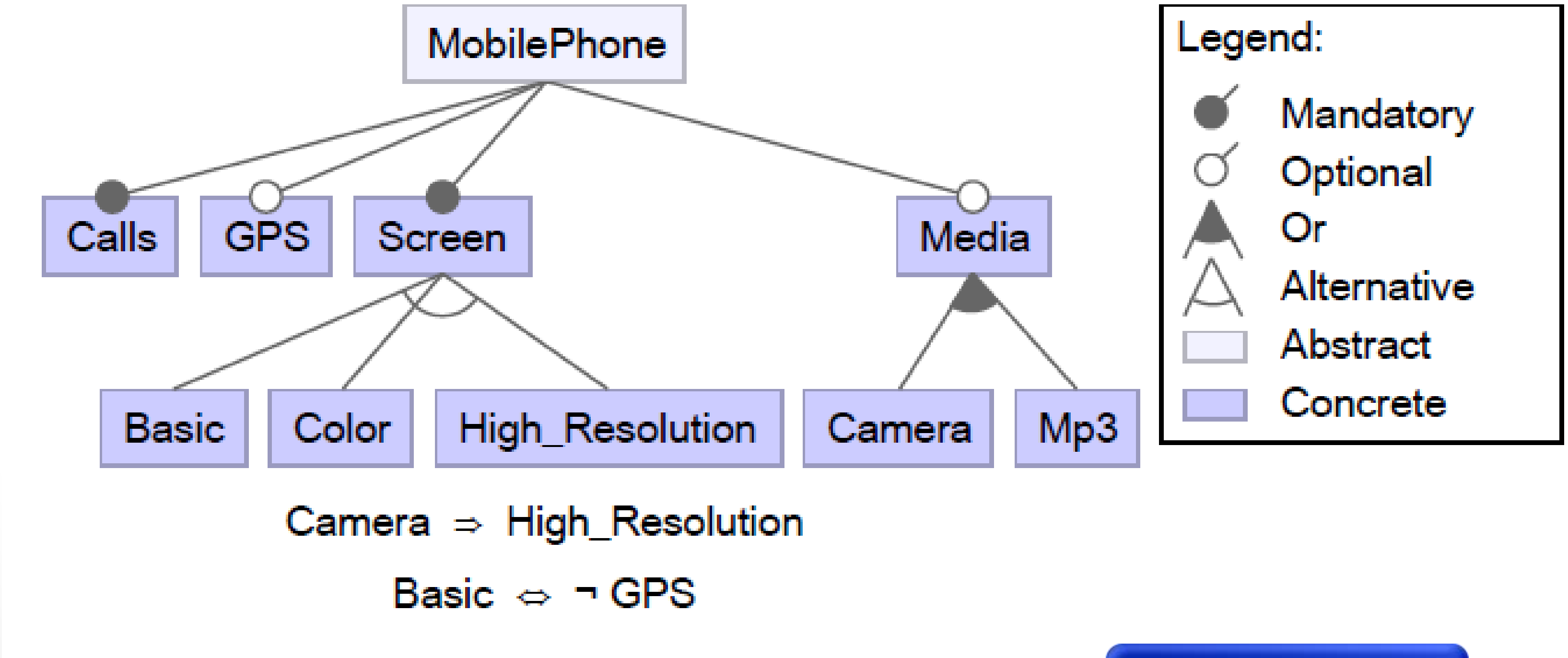
Feature models

- In software product line engineering, feature models represent all possible products of a software product line in terms of features and relationships among them.
- Example



Feature IDE

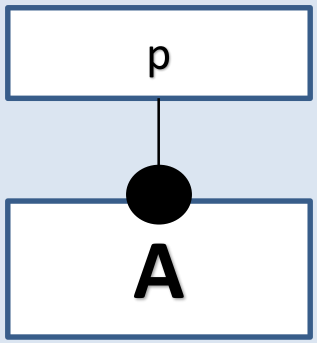
http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/



Features relationships in FMs

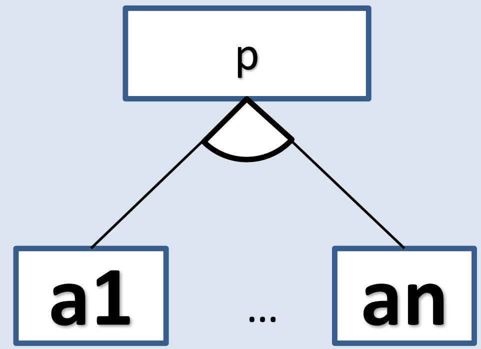
MANDATORY

child feature A is mandatory

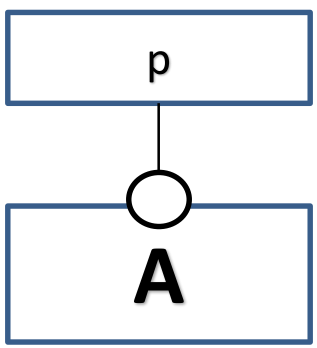


ALTERNATIVE

exactly one of the sub-features must be selected

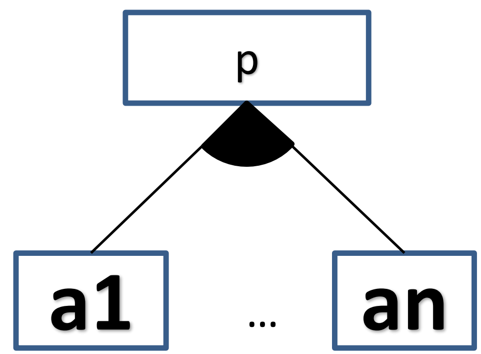


OPTIONAL



OR

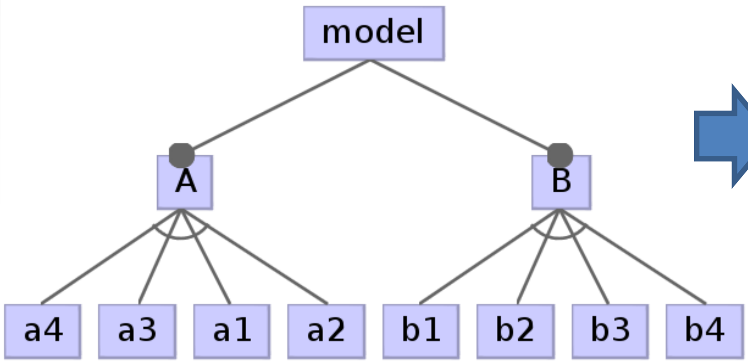
at least one of the sub-features must be selected



Standard semantics

- Feature models semantics can be rather simply expressed by using propositional logics
  - D. Batory. Feature models, grammars, and propositional formulas. Software Product Lines, pages 7–20, 2005.
- Every feature is translated to a **Boolean** input
- Add constraints for the relations among features (implicit constraints)
  - Alternative features are expressed as exclusive or
- Add constraints for cross-tree requirements

Disadvantages



10 Boolean variables
Model, A, B,
$a1, \dots, a4, b1, \dots, b4$

Constraints:
e.g. A is alternative:

For A:

$$(a1 \land \neg a2 \land \cdots \land \neg a4)$$
$$\lor$$
$$(\neg a1 \land a2 \land \cdots \land \neg a4)$$
$$\dots$$
$$\lor$$
$$(\neg a1 \land \neg a2 \land \cdots \land a4)$$

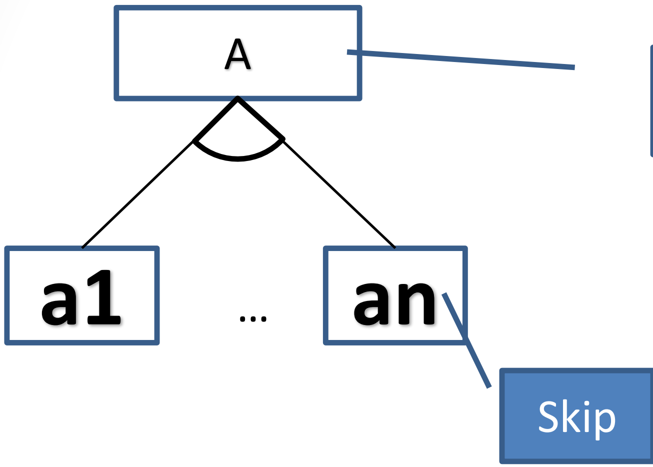
FM2CitLab

- A “better” way to translate FMs to combinatorial problems
- The translation to CitLab language is performed in the following steps

1. Every feature, starting from the root feature, is translated to an element (variable or literal constant) in the combinatorial problem.
   - Initialize also a function *isChosen* to be used when formalizing the constraints
2. Additional constraints are added in order to represent relationships among features as specified by the hierarchies in the future model.
3. Cross-tree constraints are translated and added to the model.
4. Apply some simplification

1. Parameters

Alternative

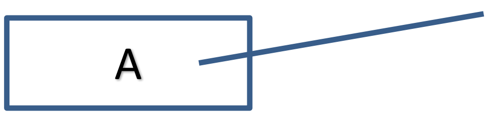



```
Enumerative A {a1 … an NONE};
```

isChosen$(A) \equiv$ **A!=NONE**

isChosen$(ai) \equiv$ **A==ai**

Everything else

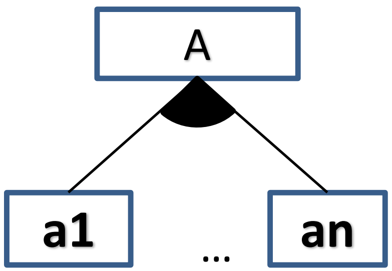



```
Boolean A;
```

isChosen$(A) \equiv$ **A==true**

2. Implicit constraints

Or



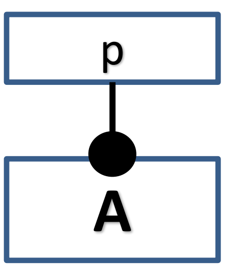
isChosen(
$\Longrightarrow$ isCho

For alternative no implicit constraints (unless...)

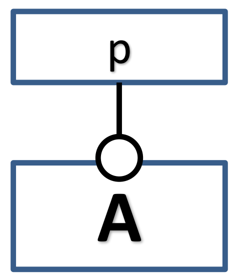
i=1..n

$\text{isChosen}(ai) \Longrightarrow \text{isChosen}(A)$

Mandatory



Optional



$\text{isChosen}(p) \Longleftrightarrow \text{isChosen}(A)$

$\text{isChosen}(A) \Longrightarrow \text{isChosen}(p)$

4. Simplification

- After translation, we simplify the model:

1. Simplify the constraints in a semantic preserving way (equivalence)
2. Remove unnecessary parameters and constraints.

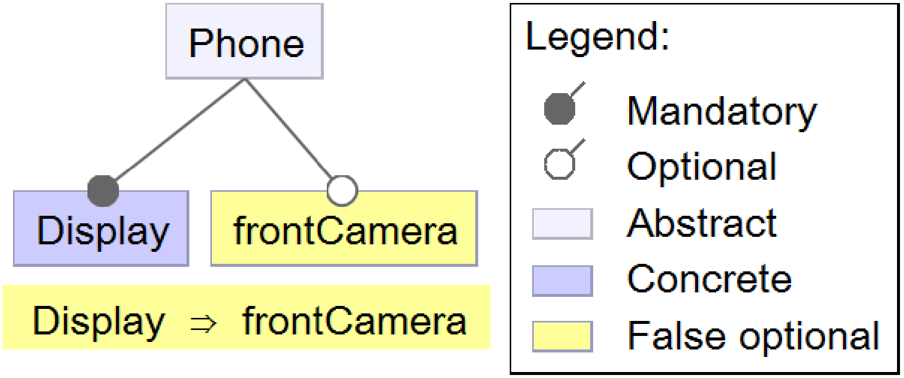
- The resulting model is equisatisifable as the original one
  - They allow the «same» family of products
  - Since some features are missing, products of the simplified model are more abstract.

It can be applied to any model, not only those coming from FMs

1. Constraints Simplification

Constraint	If already present	Replaced by
$a \Rightarrow b$	a	b
$a \Rightarrow b$	b	- remove
$a \Leftrightarrow b$	a	b
$a \Leftrightarrow b$	b	a

- In terms of FMs:



2. Parameter removal

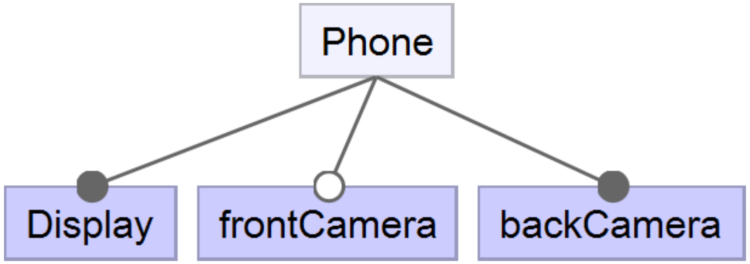
Parameter	If present	action
`Boolean A;`	A == true	Remove A and the constraint
	A == false	
`Enumerative A {a1 … an};`	A == a1	Remove A and the constraint
	A != a1	Remove ai and the constraint

- In terms of FMs:

Some features:

display, backCamera, Phone

are always present, can be ignored



Experiments

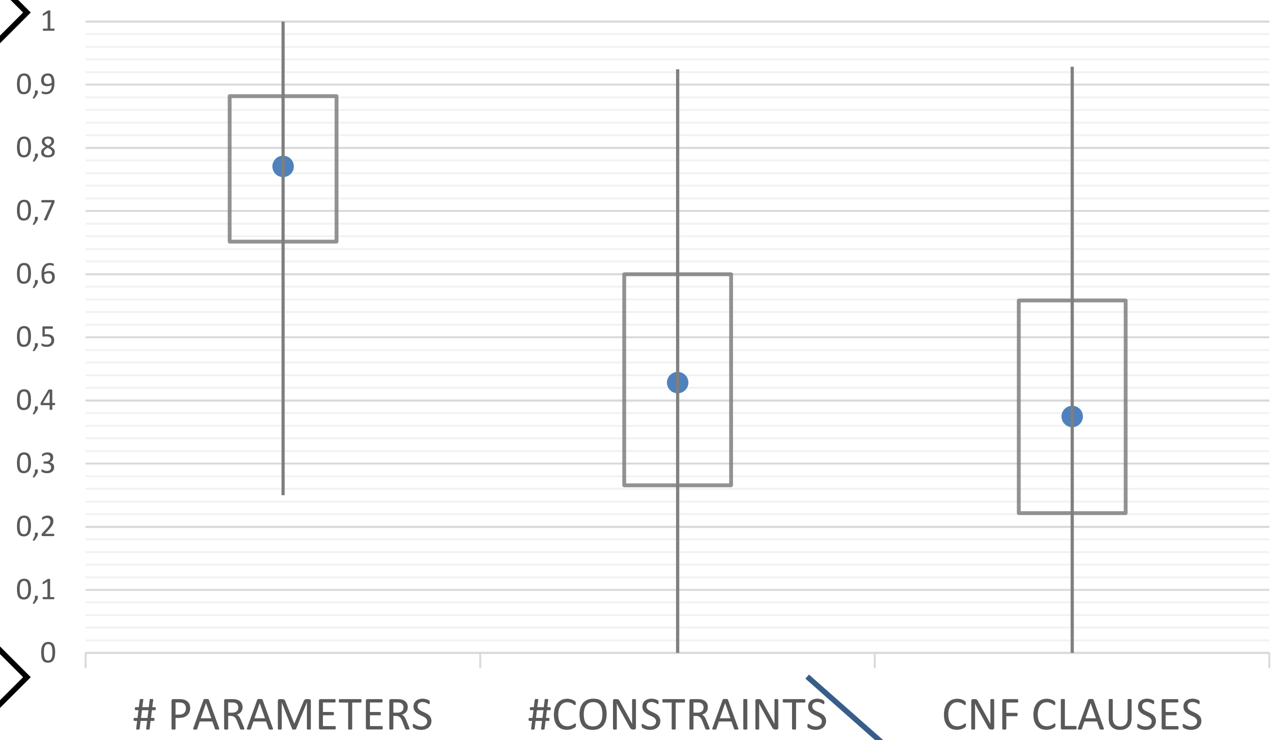
- Over 52 feature models from SPLOT repository
- Using the FeatureIde parser
  - We had to skip some models because of its faults
- Implemented the BOOL translation for comparison

1. **Testing the correctness** of the transformation
   - We have not proved that our translation is correct, but tested against the BOOL (by the number of valid products)
2. **Effect of the simplification** over the parameters and the constraints
3. Comparison with BOOL in terms of model
   1. # parameters: we should obtain **smaller** models
   2. # constraints: we should obtain **simpler** models
   3. variability: we should obtain **more compact** models
4. Test generation vs BOOL

2. Simplification effect

No reduction

Ratio: $\frac{x\ after}{x\ before}$



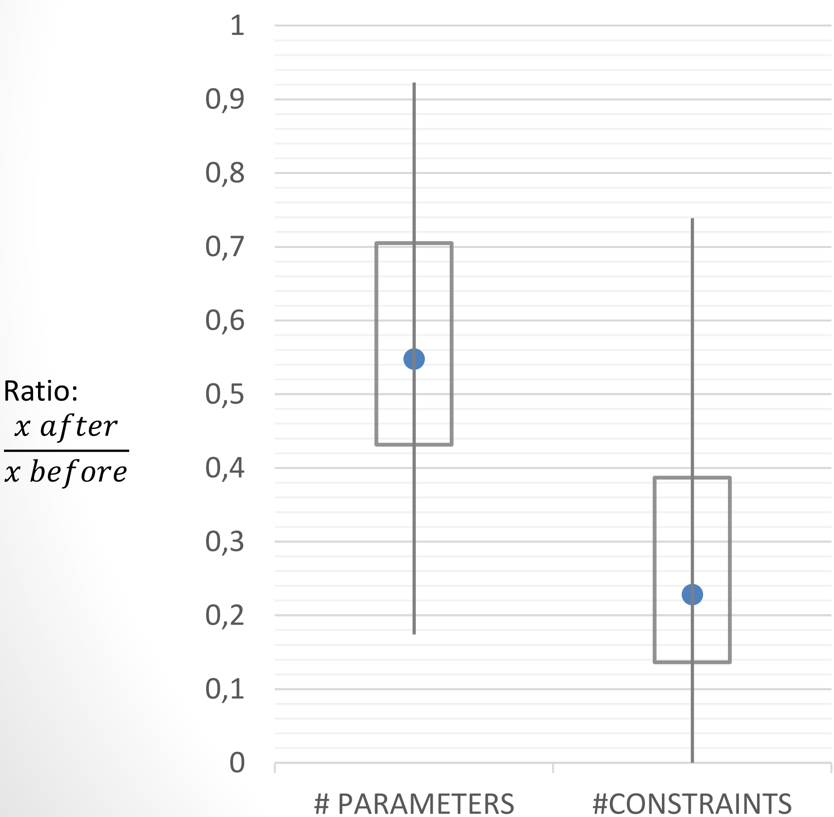
Reduction to 0

In 11% of cases complete reduction of constraints

Average:
77% of parameters
43% number of constraints
38% number of clauses in them

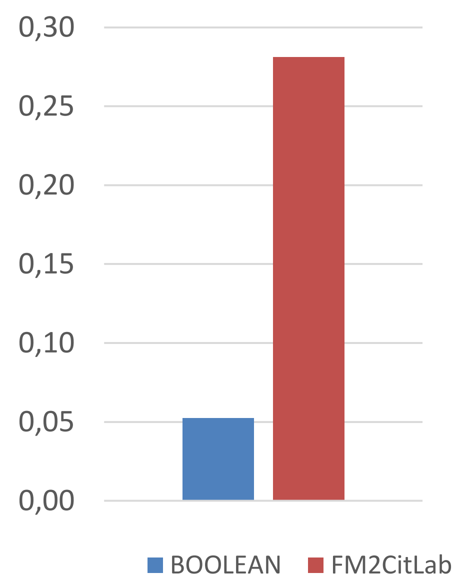
3 Vs. BOOL

Reduction of:
parameters
CNF clauses in constraints



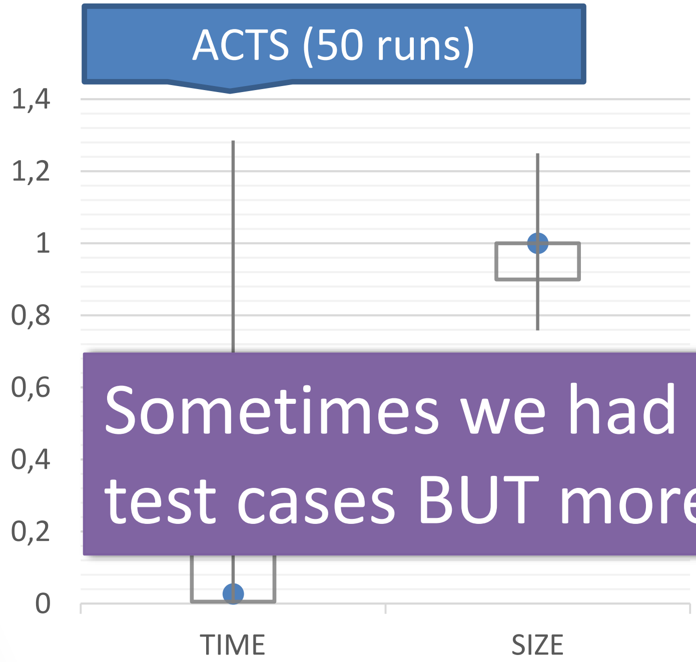
Increase of variability

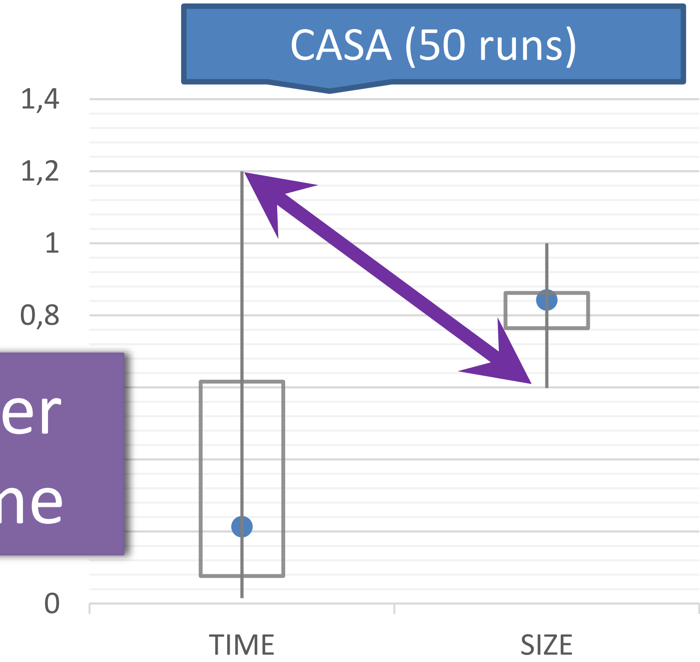
$$var = \frac{\#\ valid\ products}{\#\ all\ products}$$



4. Test generation time (vs. BOOL)

- Can test generators take advantage of our translation?
- vs BOOL (+ simpl) using





Sometimes we had fewer test cases BUT more time

BOOL		FM2CitLab	
Time	Size	Time	Size
0,785	39	2,034	37

+
c
tr
memory errors

Other results

- Tools for combinatorial testing performed much better than tools specifically developed for SPLs (PACOGEN, OSTER)

Conclusions

- A new better way to translate FMs to combinatorial problems
  - More compact
  - Fewer parameters and constraints
  - Increased variability
- Integrated into CitLab
  - Reuse of test generators

THANK YOU