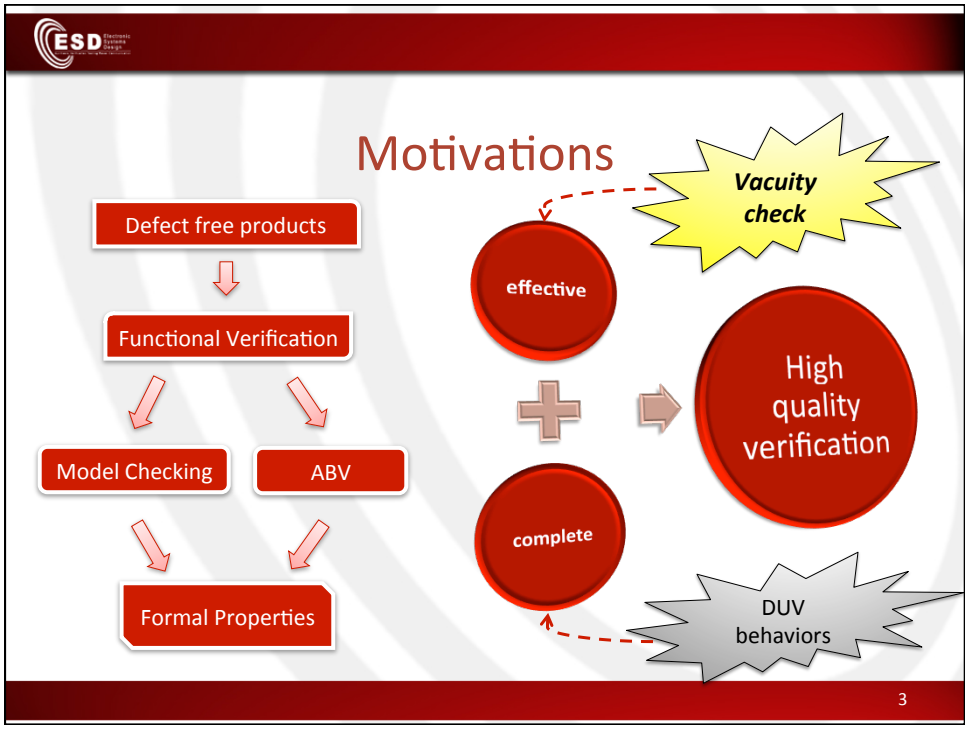
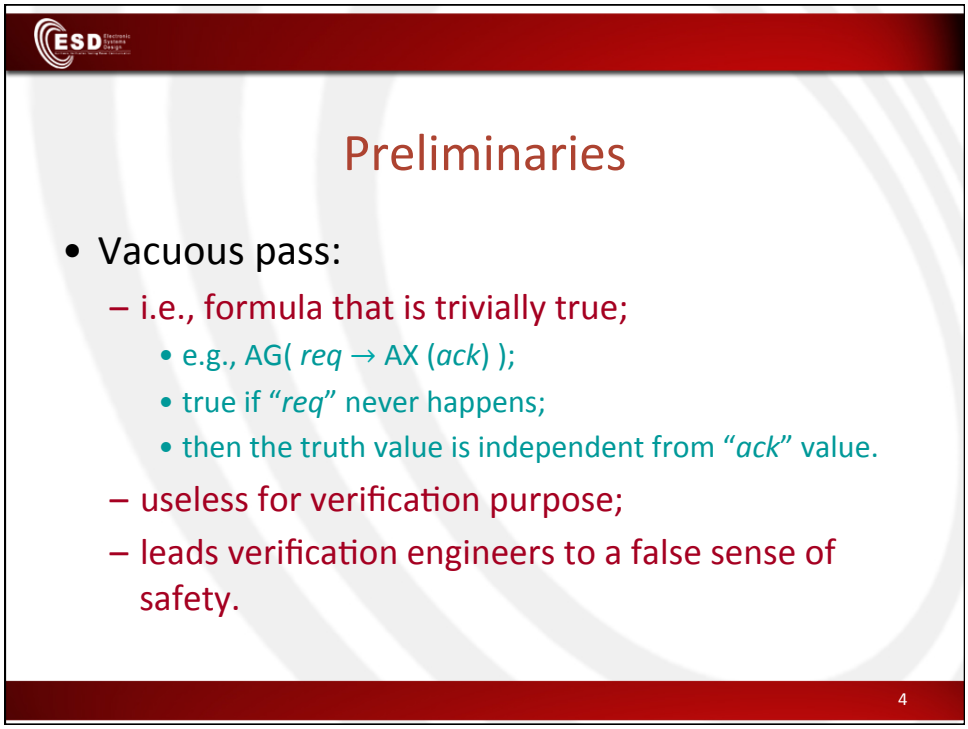
Vacuity Analysis

Graziano Pravadelli – Luigi Di Guglielmo
Dipartimento di Informatica Università di Verona

Agenda

- Motivations
- State of the art
- Goal
- Methodology
- Pros and cons
- Conclusions


Motivations
Defect free products
Functional Verification
Model Checking
ABV
Formal Properties
Vacuity check
effective
complete
High quality verification
DUV behaviors
3


Preliminaries
• Vacuous pass:
– i.e., formula that is trivially true;
• e.g., AG(req → AX (ack));
• true if "req" never happens;
• then the truth value is independent from "ack" value.
– useless for verification purpose;
– leads verification engineers to a false sense of safety.
4

Preliminaries: Affect and Vacuity

- Definition 1 (Affect) [Beer01]
  - A sub-formula ψ of a formula ϕ **affects** ϕ in model M if there is a formula ψ' such that the truth values of ϕ and ϕ[ψ ← ψ'] are different in M.
- Definition 2 (Vacuity) [Beer01]
  - A formula ϕ **passes vacuously** in model M if M⊨ϕ and ϕ includes a sub-formula ψ that does not affect ϕ in M.
    - In this case, we say that ϕ is ψ-vacuous in M

Preliminaries: Affect and Vacuity

- Example
  - AG(*req* → AX (*ack*))
    - The formula is true if *req* never happens independently from the value of AX(*ack)*
    - *AX(ack) does not affect the formula, which then passes vacuously in a model where req never happens*
- Previous definitions capture the intuitive notion of vacuity independently from the logic
  - But they are not practically useful

Preliminaries: Sub-formulas

- To limit the analysis to a small subset of sub-formulae
  - Definition 3 (**Minimal sub-formulas**) [Beer01]
    - Let S be a set of sub-formulas. The minimal sub-formulas of S is defined as:
      $\min(S) = \{ \psi \in S \mid \nexists\, \psi' \in S \text{ such that } \psi' \leq \psi \}$
      where $\psi' \leq \psi$ means that ψ' is a sub-formula of ψ
    - Assumption:
      - each sub-formula is unique

Preliminaries: Sub-formulas

- Example
  - $\phi = AG((\alpha \wedge \beta) \rightarrow AX(\gamma \vee \neg \alpha))$
  - Subformulae of ϕ
    - $S = \{\alpha_1 \wedge \beta_1, \alpha_1, \beta_1, \gamma_1 \vee \neg \alpha_2, \gamma_1, \neg \alpha_2, \alpha_2\}$
  - Minimal sub-formulae
    - $\min(s) = \{\alpha_1, \beta_1, \gamma_1, \alpha_2\}$

Preliminaries: vacuity

- Theorem 1 [Beer01]
  - In a logic with polarity, for a formula φ, and a set S of sub-formulas of φ, for every model M, **φ is S-vacuous in M** iff
    - there is $\psi \in min(S)$ such that
    - $M \models \varphi[\psi \leftarrow X]$, where $X=$***false*** if $M \models \varphi$ and ψ is of positive polarity, otherwise, $X=$***true***.
- minimal sub-formulae are substituted with either true or false, thus reducing the number of checks
  - $\phi[\psi \leftarrow X]$ is a witness formula

Preliminaries: vacuity

- From Theorem 1:
  - φ is not ψ-vacuous if $M \not\models \varphi[\psi \leftarrow X]$
  - In this case
    - The counterexample is an interesting witness proving the non vacuity of ϕ with respect to ψ

State of the Art

- Vacuity checking:
  - *witness formulas* obtained by the substitutions of sub-formulas with ***true*** or ***false*** according to their polarity;
  - *witness formulas* failure highlight how substituted sub-formula affects the original formula;
  - Theorem 1 allows to focus only on model checking φ where the minimal sub-formulas are substituted;
- Limits:
  - model checking;
  - verification of new set of formulas.

Increasing verification time

State of the Art

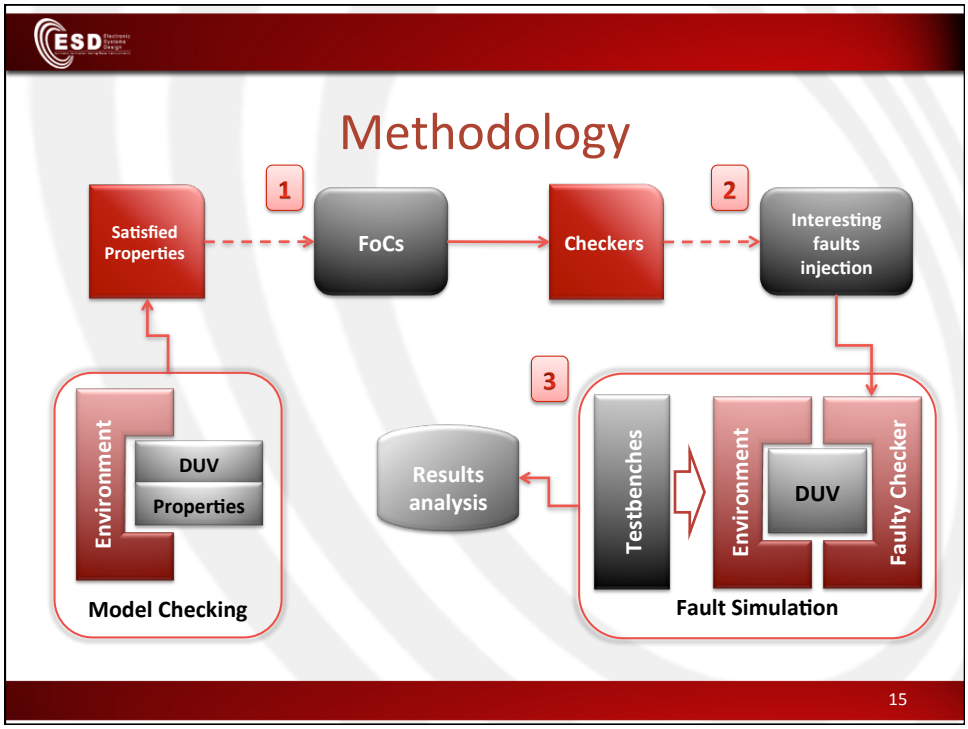
- [Beer97]
  - identification of w-ACTL formulas for which vacuity detection can be done efficiently
    - A w-ACTL formula is an ACTL formula in which for all binary operators at least one of the operand is a propositional formula
- [Vardi99]
  - vacuity detection for CTL* formulas consists of replacing **each sub-formula** ψ of a formula ϕ by true and false
  - vacuity with respect to sub-formulas occurrences
- [Beer01]
  - vacuity detection for CTL* formulas consists of replacing **only minimal sub-formulas** by either true or false according to their polarity
  - vacuity with respect to sub-formulas occurrences

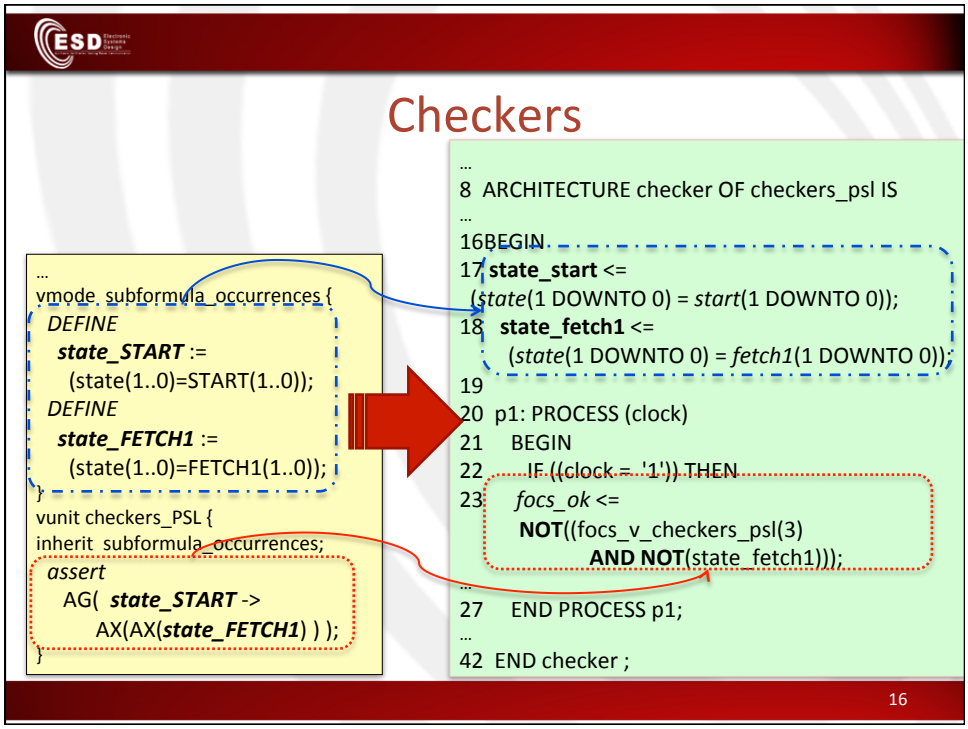
State of the art

- [Armoni03]
  - vacuity detection with respect to both sub-formulas and sub-formulas occurrences but
  - required manual approach for analyzing vacuity alert
- [Gurfinkel07]
  - the method is based on a naive algorithm based on model checking formulas obtained by replacing atomic sub-formulas with unconstrained Boolean variables
  - the approach detects vacuity as defined in [Beer97], [Beer01]

Goals

- New vacuity analysis approach:
  - without model checking
  - without *witness formulas* definition
  - effective as current approaches
  - more efficient
  - based on fault simulation
- It requires:
  - checker generation
  - interesting faults injection
  - results analysis


Methodology
1
2
3
Satisfied Properties
FoCs
Checkers
Interesting faults injection
Environment
DUV
Properties
Model Checking
Results analysis
Testbenches
Environment
DUV
Faulty Checker
Fault Simulation
15


Checkers
...
vmode subformula_occurrences {
DEFINE
state_START :=
(state(1..0)=START(1..0));
DEFINE
state_FETCH1 :=
(state(1..0)=FETCH1(1..0));
}
vunit checkers_PSL {
inherit subformula_occurrences;
assert
AG(state_START ->
AX(AX(state_FETCH1)));
}
...
8 ARCHITECTURE checker OF checkers_psl IS
...
16 BEGIN
17 state_start <=
(state(1 DOWNTO 0) = start(1 DOWNTO 0));
18 state_fetch1 <=
(state(1 DOWNTO 0) = fetch1(1 DOWNTO 0));
19
20 p1: PROCESS (clock)
21 BEGIN
22 IF ((clock = '1')) THEN
23 focs_ok <=
NOT((focs_v_checkers_psl(3)
AND NOT(state_fetch1)));
...
27 END PROCESS p1;
...
42 END checker ;
16

Interesting Faults

- They perturb only one minimal sub-formula in the checker
  - Sub-formulae are grouped by scope of the temporal operators to address tautology
    - *Contemporaneous:* occurrences of the same subformula that are in the scope of the same temporal operators
    - *Opposite:* contemporaneous occurrences of the same subformula whit opposite polarity
- its value is related to sub-formula polarity
- if faults are detected, formula does not pass vacuously

Interesting Faults

- Definition 4
  - Let
    - ϕ a formula in a logic with polarity
    - C the corresponding checker,
    - S the set of minimal sub-formulas of ϕ, and
    - A the set of concurrent assignments of C storing the value of sub-formulas in S during simulation.
  - The set of interesting faults for C is defined as follows:
    - F = { stuck-at X on a | $a \in$ A }
      where X = false if the sub-formula associated to assignment a has positive polarity, X = true otherwise

Interesting Faults

- Example:
  - considering the property
    - AG(state_START→AX(AX(state_FETCH1)));
  - state_START is of negative polarity:
    - corresponding interesting fault is stuck-at-1
    - AG(true →AX(AX(state_FETCH1)));
  - state_FETCH1 is of positive polarity:
    - corresponding interesting fault is stuck-at-0
    - AG(state_START →AX(AX(false)));

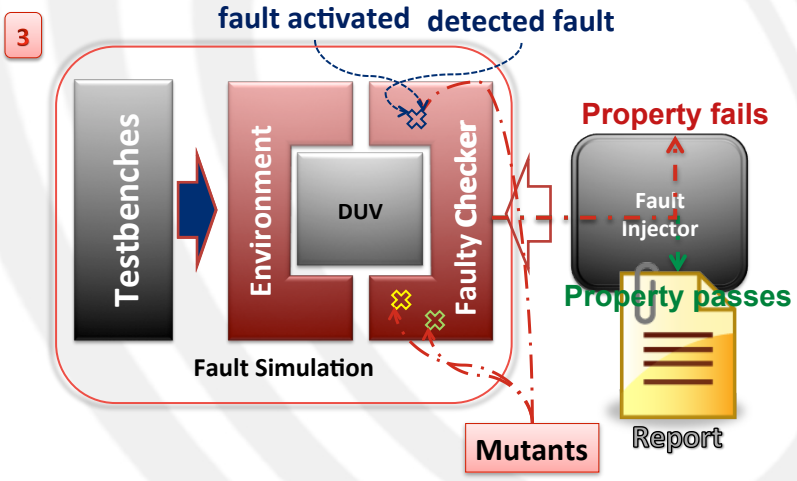
Interesting Faults

- Theorem 2 (Detectable faults vs. vacuity)
  - Let
    - ϕ a formula in a logic with polarity,
    - C the corresponding checker,
    - S the set of minimal sub-formulas of ϕ, and
    - F the set of interesting faults of C associated to S
  - The fault $f \in F$, such that f is associated to sub-formula $\psi \in S$ as defined in Def. 4, **is detectable, iff ϕ is not ψ-vacuous**

Interesting Faults

- Previous theorem allows:
  - To reason about vacuity by fault simulating interesting faults in the checker instead of model checking witness formulae
    - A checker failure due to the effect of an interesting fault f corresponds to prove that the sub-formula ψ perturbed by f affects the truth value of ϕ
- Consequently, the sequence of values generated by the testbench that causes the checker failure (i.e., the test sequence of f) is an *interesting witness* proving that ϕ is not ψ-vacuous

Simulation Environment



Pros and Cons

- Testbenches, drawbacks:
  - inefficiency of testbenches used during fault simulation may prevent the detection of interesting faults
  - from practical point of view:
    - formula does not pass vacuously if interesting faults are detected
    - on the contrary, it is not possible to conclude that a formula is vacuous

Pros and Cons

- Testbenches, advantages:
  - in ABV testbenches must be as effective as possible
  - from practical point of view:
    - it is mandatory refining testbenches that allow a formula to pass vacuously
  - in this context, the proposed approach is satisfactory

Pros and Cons

- Checkers, drawbacks:
  - it is not possible to conservatively generate checkers for all kind of temporal formulas
  - FoCs is able to generate checkers for formulas defined according to the *Simple Subset* of PSL
  - restrictions guarantees that formulas can be simulated

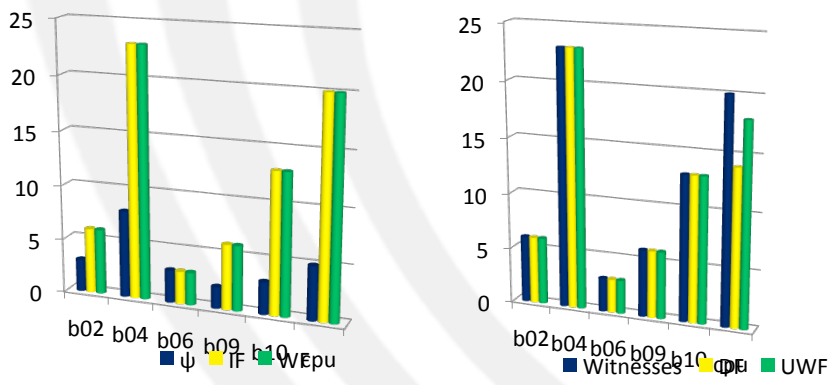
Pros and Cons

- Checkers, advantages:
  - proposed methodology is independent from the tool adopted for checkers generation
  - the idea of storing (and perturbing) the values of sub-formulas in explicit signals driven by concurrent statements can be exploited whatever the structure of the generated checker is

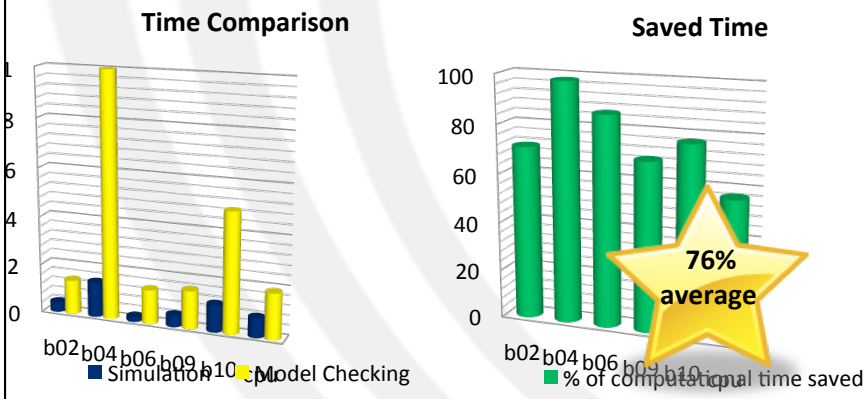
Experimental results

- Benchmarks:
  - ITC-99 benchmarks;
  - control unit of an 8-bit CPU with an instruction set architecture composed of 13 instruction;
- Properties:
  - for ITC-99 benchmarks have been taken from the *vis-verilog-models-1.0* archive;
  - for cpu design have been defined by analyzing its specification;
- Checkers:
  - generated by using FoCs;
- Interesting faults injection and testbench generation:
  - using the features of the Laerte++ automatic test pattern generator;
- Compared with:
  - formal vacuity analysis performed according to the approach proposed in [Beer01] by using VIS to model check witness formulae.

Experimental Results



Verification Time



29

Tautologies: Problem

- Not marked as vacuous formulas when two *separate occurrences* of the same sub-formula are considered to be different sub-formulas
  - e.g., $(a \vee \neg a)$:
    $(false \vee \neg a) = (\neg a)$
    $(a \vee \neg true) = (a)$
- marked as vacuous by considering *sub-formulas instead of occurrences*
  - e.g., $(a \vee \neg a)$:
    $(true \vee \neg \text{true}) = (a \vee \neg a)$
    $(false \vee \neg false) = (a \vee \neg a)$

30

Tautologies: Problem

- But if we use sub-formulae instead of occurrences for
  - $\alpha \wedge (G (\neg \beta \vee \alpha))$
    - false $\wedge$ (G ($\neg \beta \vee$ false)) is false
    - $\alpha \wedge$ (G ($\neg$ true $\vee \alpha$)) is $\alpha \wedge$ (G α) which can be falsified
  - is marked as non-vacuous, even if it is vacuous when β never occurs

Tautologies: Solution

- Modify the formula to remove temporal overlapping among operators
- Opposite (contemporaneous) formulae are substituted simultaneously
- Non opposite (contemporaneous) formulae are substituted individually

Tautologies

- Example
  - $G(\neg \alpha \vee F(\alpha))$
    - Is equivalent to
    - $G(\neg \alpha \vee \alpha \vee X(F(\alpha)))$
  - Sub-formulae $S = \{\neg\alpha_1, \alpha_1, \alpha_2, \alpha_3\}$
  - $Min(S) = \{\alpha_1, \alpha_2, \alpha_3\}$
    - α_1, α_2 are opposite
    - α_3 has a different temporal scope

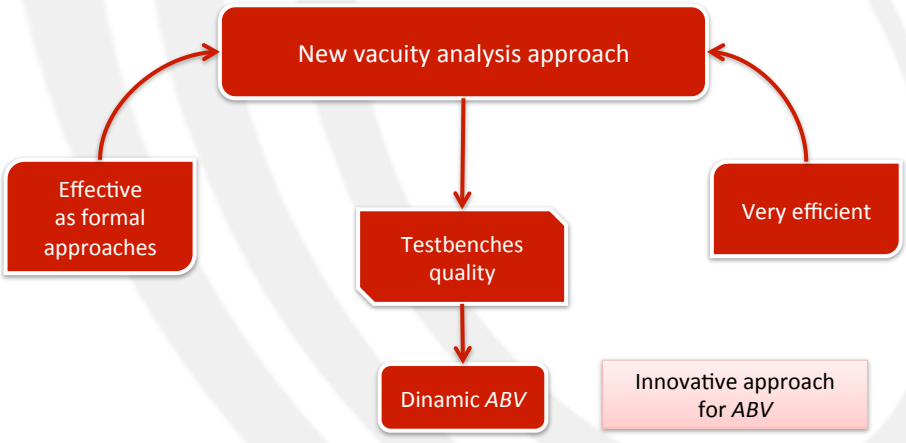
Tautologies

- $\varphi_1 = \alpha \vee \neg\alpha$
  - $Min(S) = \{\alpha_1, \alpha_2\}$
  - α_1, α_2 are opposite, → simultaneous substitution
    - false $\vee$ ¬ false = true → vacuous
    - true $\vee$ ¬ true = true → vacuous

Tautologies

- $\varphi_2 = \alpha \wedge (G(\neg \beta \vee \alpha))$
  - Let us consider that β never occurs
  - Min(S)=$\{\alpha_1, \alpha_2, \beta_1\}$ are not opposite
  - $\varphi_2[\alpha_1 \leftarrow \text{false}] = \text{false}$
  - $\varphi_2[\beta_1 \leftarrow \text{true}] = \alpha \wedge G(\alpha)$
    - Possibly false
  - $\varphi_2[\alpha_2 \leftarrow \text{false}] = \alpha \wedge G(\neg \beta)$
    - Since β never occurs and φ_2 is satisfied by the model, $\varphi_2[\alpha_2 \leftarrow \text{false}] = \alpha = \text{true} \rightarrow$ vacuous

Conclusions



References

- L. Di Guglielmo, F. Fummi and G. Pravadelli *"Vacuity Analysis by Fault Simulation"*, Proc. of ACM/IEEE MEMOCODE 2008
- L. Di Guglielmo, F. Fummi and G. Pravadelli *"Vacuity Analysis for Property Qualification by Mutation of Checkers"*, Proc. of ACM/IEEE DATE 2010