Incompleteness Analysis

Graziano Pravadelli
Dipartimento di Informatica Università di Verona

Agenda

- Motivations
- State of the art
- Main idea
- Property coverage
- Witness coverage
- Coverage accuracy
- Conclusions and references

Motivations

- Lack of exhaustiveness is the biggest problem for dynamic verification
- (Semi-)Formal verification is an alternative to overcome this limitation
  - Model checking (ABV) used to identify design errors

Motivations

- Formalization of properties is a challenging task
  - Properties could be false requiring redefinition of
  - Design
    - Confuted properties may highlight design errors
  - Properties
    - Confuted properties may represent wrong behaviors
  - Environment
    - Confuted properties may highlight a too strict environment

Motivations

- If defined properties are true
  - have I written enough properties?
  - is the design correct?
- Model checking (ABV) can prove the presence of bugs, but not their absence!
- A *coverage metric* is required

State of the art

- Mutation-based approaches
  - Mutated implementations for measuring property coverage
- Implementation-based approaches
  - Coverage is analyzed without mutating the implementation

State of the art

- Hoskote (DAC 99)
  - property coverage by analyzing whether a mutation on an observed signal, at a certain state, changes the truth values of the defined properties
    - Works only for a subset of ACTL
    - Syntax dependent
      - Different syntaxes, same semantics can have different coverages

State of the art

- Somenzi (DAC 03)
  - Similar to Hoskote but extended for a subset of CTL
- Chockler (CAV 01, CHARME 01)
  - Safety LTL and CTL are addressed

State of the art

- For all these approaches
  - Formal methods are applied
  - Worst case complexity is exponential in the size of the properties
  - Consider only state coverage metrics
    - incompleteness of properties related to wrong paths of an implementation cannot be detected

State of the art

- Hsiung (ATVA 04)
  - Mutation model in state graph
  - Does not consider *semantic mutations*
    - Eg. Changing the value of variables in a state
  - But consider *structural mutations*
    - Eg. Inserting or deleting a state or a transition
  - Requires re-checking properties for each mutations
    - Too time consuming

State of the art

- Xu (MEMOCODE 05, ASPDAC 06)
  - Transition-based coverage based on transition perturbation
  - Transition traversal does not consider the values of signals
  - Works only for a subset of ACTL
- Grumberg (CHARME 99)
  - Symbolic approach
  - Works only for safety ACTL
  - More complex than other approaches

State of the art

- Sayyaran (VLSI-SOC 06)
  - restricts to 0 and 1 the value of each variable contained into the Ordered BDD corresponding to DUV and checks if the restrictions change the truth value of properties
  - coverage is computed as the fraction of the restrictions on variables that are covered by properties
  - Requires rechecking for each restriction

State of the art

- Classen (FMCAD 07)
  - It does not required the design model
  - Exploits traces looking for time points where output signals are not constrained by properties
  - For identified forgotten case it is not possible to retrieve information about the circuit behavior
  - Only for safety LTL

State of the art

- Oberkonig (FMCAD 07)
  - Quantitatively describes the degree of determination of the output and internal signals mentioned in a specification
  - does not require an implementation of the design
  - proves that all outputs of the design are determined, but this does not mean that the specification is perfect

State of the art

- Drechsler (DATE 07, ISVLIS 07, TCAD 08)
  - Uses BMC focusing on the identification of scenarios where the value of outputs of the circuit are not determined by the set of properties
  - Looks for input and state assignments for which the set of properties do not specify the value of outputs

15

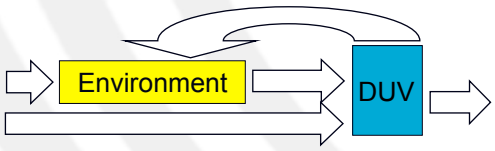
State of the art

- Hoffmann (MWSCAS 09)
  - New CNF coverage metric
  - Exploits SAT heuristics to guide the property definition finding the most important variables to be considered into properties
  - different heuristics prioritize different structural characteristics

16

Background

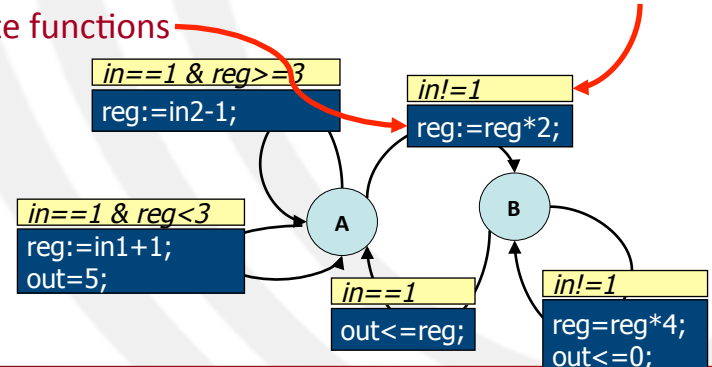
- DUV is modeled by and EFSM
- Environment is modeled by another EFSM



17

EFSM

- More compact than FSM
- Transitions are labeled by enabling functions and update functions



18

Main Idea

(MEMOCODE 2003
TCOMP 2007)

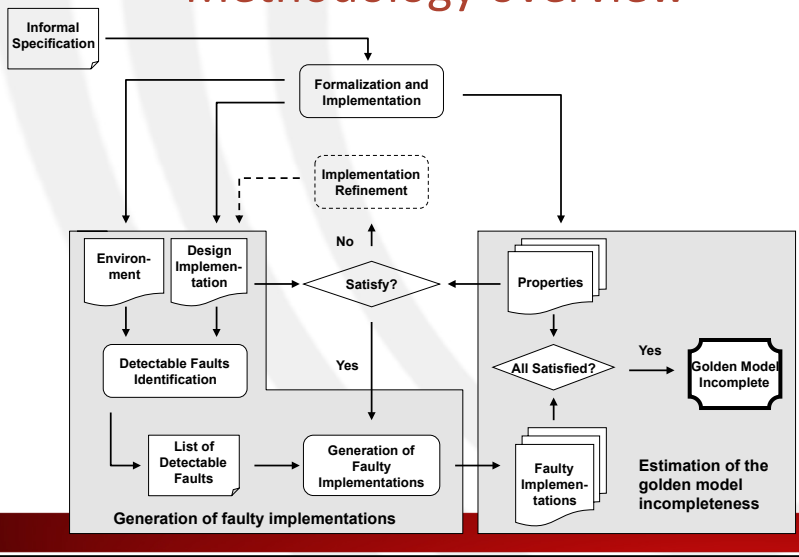
- Is there a relation between fault detection and property completeness?
  - Properties represent the golden model
    - They hold on the design implementation
  - An implementation (EFSM) is perturbed by using mutants

Do all properties hold on the perturbed EFSM?

Properties are not able to distinguish between faulty and fault-free implementations
The set of properties is ***INCOMPLETE***

19

Methodology overview



20

E-detectable faults

- Given a fault-free design I and a faulty design I_f a fault f is
  - detectable if ∃ input sequence such that PO(I) ≠ PO(I_f)
  - ***E-detectable* if ∃ input sequence such that PO(I) ≠ PO(I_f) under the environment *E***
- ***How to identify the set of E-det faults?***
  - ATPG is a possibility

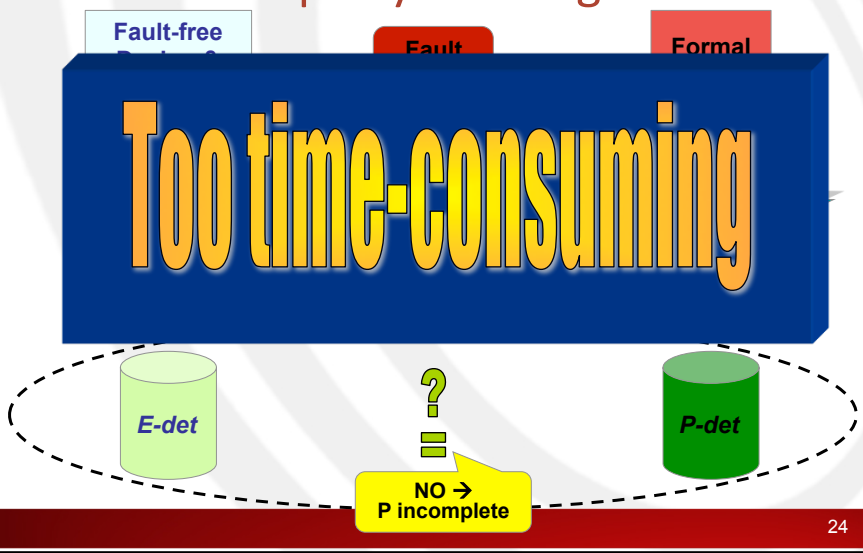
Estimation of property incompleteness

- Anaysis of the capability of properties of distinguishing between fault-free and faulty implementations
  - Property coverage
    - Based on static verification
  - Witness coverage
    - Based on dynamic verification

Property coverage

- Given a fault model, a fault-free implementation *I* and a faulty implementation I_f
  - *p* detects a fault *f* iff *p* holds on *I* and *p* fails on I_f
    - In this case $f \in$ *P-det*
  - **Property Coverage (C_p) = |*P-det*|/|*E-det*|**
  - *P* is complete if C_p=1
- Re-checking is required → too time-consuming

Property Coverage



Witnesses and counterexamples

- For existentially quantified CTL property
  - is a computation path which demonstrates that the property is true
- For universally quantified CTL property
  - is a computation path which demonstrates that the negation of the formula is true
- counterexample for a universally quantified formula is a witness for the dual existentially quantified formula

25

Witnesses and counterexamples

- For LTL properties (universal path quantifier is implicitly assumed)
  - a witness is a computation path π such that π entails p non vacuously
  - a counterexample is a path π such that π does not entails p

26

Witnesses and counterexamples

- A witness for a universally quantified property p is not enough to state that p is true
- A counterexample for p is enough to state that p is false
- A witness for an existentially quantified property p is enough to state that p is true
- A counterexample for p does not show that p is false

Witnesses and counterexamples

- They contain values for input and outputs involved in the corresponding property
- Other signals are undefined
- They can be generated by model checking tools like SMV, NuSMV, RuleBase, …
  - The set of witnesses (counterexamples) generated by the model checker can be efficiently extended by using an ATPG

Input witnesses and counterexamples

- More intuitive definition of witness
  - An input witness for p is a finite sequence of values for the inputs of the DUV such that p is satisfied non vacuously
- More intuitive definition of couterexample
  - An input counterexample for p is a finite sequence of values for the inputs of the DUV such that p is not satisfied

Input witnesses and counterexamples

- input witness (counterexample) is obtained
  - by extracting the input projection of the witness (counterexample) provided by the model checker
  - and by assigning the default value (i.e., by considering VHDL, the lowest value for integers, “U” for standard logics, “0” for bits, etc.) to input signals that are not specified

Input witnesses and counterexamples

- Witness for safety properties?
- Counterexamples for liveness properties?
- *They are infinite path!*
  - However
    - an infinite witness (counterexample) can be found composed of a finite prefix followed by a repeating cycle
  - Thus
    - Input witness (counterexample) is composed of the input projection of the finite prefix followed by the input projection of one occurrence of the repeating cycle

Witness coverage

- Theorem

  If $f \in$ *P-det* then
  - ∃ an input counterexample for p on I_f which is
    - an input witness for p on I
    - a test sequence for f on I
- Definition

  $f \in$ *E-det* is w_p*-det* if
  - ∃ an input w for p such that
    - w is a test sequence for f
    - w is an input counterexample for p on I_f

Witness coverage

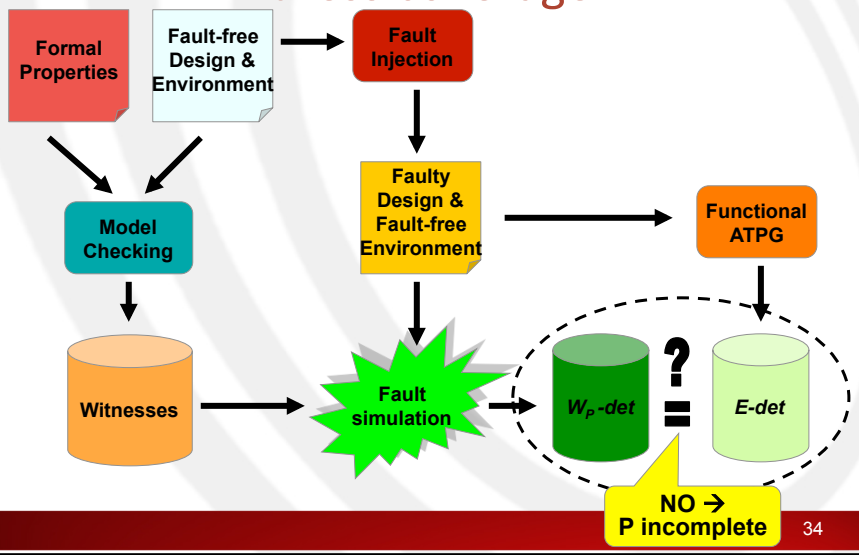
- W_P-*det* = set of faults detected by fault simulating input witnesses
- **Witness coverage (C_W) = $|W_P\text{-}det|/|E\text{-}det|$**

Theorem: $C_P = C_W$

Property re-checking is not required

33

Witness coverage



34

Witness coverage

- How many witnesses?
- What about if the model checker generates only one witness for each property?
  - $C_P = C_W$ iff all witnesses are available
  - Otherwise $C_W \leq C_P$

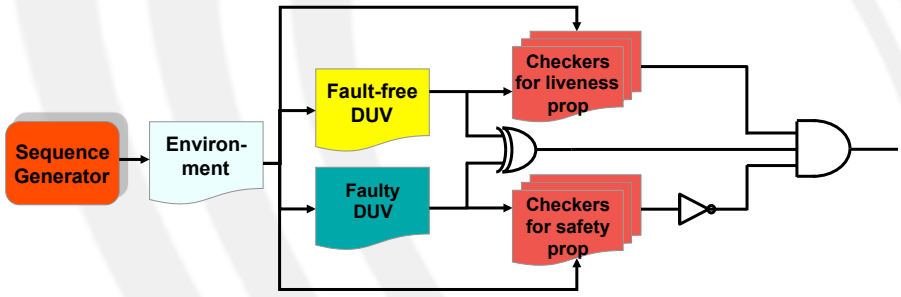
Witness coverage

- Properties are converted into checkers
  - For a safety property
    - Output of the checker is always TRUE until a failure
    - Checkers of safety properties are connected to POs of I_f
  - For a liveness property
    - Output of the checker is always FALSE until it successes
    - Checkers of liveness properties are connected to POs of I

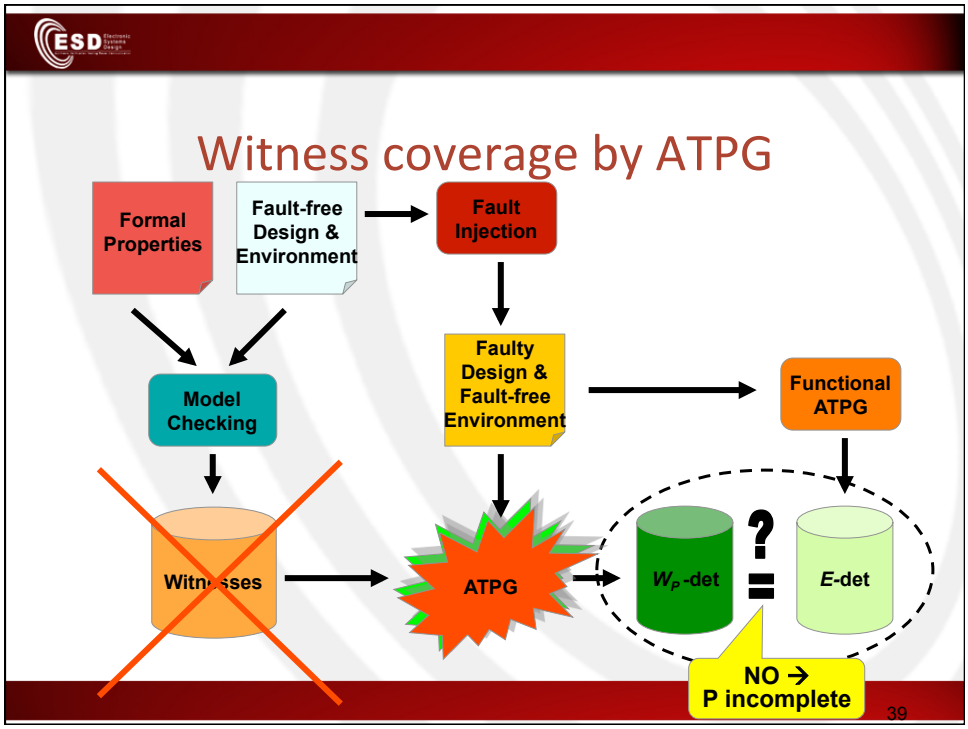
Witness coverage

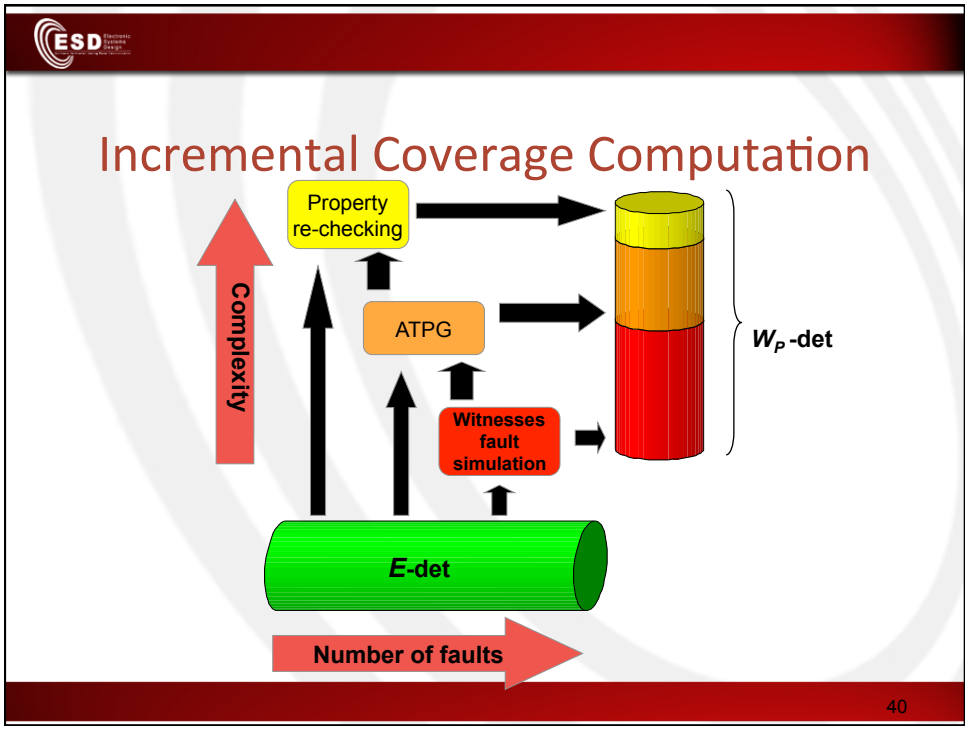
- Given a witness *w* for *p*, $f \in W_p\text{-}det$ if fault simulating *w*
  - *I* and I_f differs on at least one PO involved in *p*
  - and concurrently
    - the checker fails if *p* is a safety property
    - the checker successes if *p* is a liveness property

Witness coverage by ATPG



- A sequences *w* is filtered by the environment
  - *w* is an input witness for a safety property
    - If *f* is detected by *w* on a PO involved in *p* and the checker of *p* fails → $f \in W_p\text{-}det$
  - *w* is a possible input witness for a liveness properties
    - If *f* is detected by *w* on a PO involved in *p* and the checker of *p* successes → $f \in W_p\text{-}det$


Witness coverage by ATPG
Formal Properties
Fault-free Design & Environment
Fault Injection
Model Checking
Faulty Design & Fault-free Environment
Functional ATPG
Witnesses
ATPG
W_P-det
?
=
E-det
NO → P incomplete
39


Incremental Coverage Computation
Property re-checking
ATPG
Witnesses fault simulation
Complexity
W_P-det
E-det
Number of faults
40

Coverage Accuracy

- Approaches based on state coverage
  - 100% state coverage is not enough, since the behavior along a path can be incorrect
  - Our methodology is based on a fault model, thus it includes the notion of path coverage

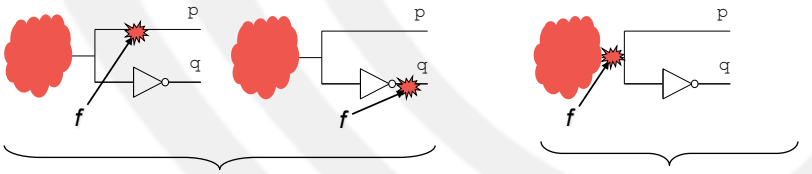
Coverage Accuracy

- Some approaches present problems with tautologies
  - Es.: $\varphi = q \leftrightarrow AF\, q$
    - In Hoskote and Somenzi approaches the coverage of φ is given by the set of states that satisfies q
    - However, it should be 0%, since tautologies cannot state anything about the correctness of a design
    - Our methodology provides 0%, since there are no faults that falsify φ

AG($\wedge_i(f_i)$) is a *linear invariant in q* if
each f_i is a propositional formula and $\exists i$ such that $f_i \mid q \rightarrow 0 = \neg f_i \mid q=1$

Coverage Accuracy

- Hoskote and Somenzi approaches may produce a false sense of security for linear invariant properties
  - Our methodology reflects the real intention of the property
    - Assuring that a combinational relation between signals holds
    - Only faults that induce a violation of such a relation are covered by the invariant
- Es.: $\varphi = AG(p \leftrightarrow \neg q)$ (where p and q are the only outputs)
  - achieves 100% coverage for Hoskote and Somenzi



f is covered by φ **f is not covered by φ**

Experimental Results

Module	Properties	E-det faults	Recheck faults	Coverage	Time (m.)	Recheck time (m.)
ST_BUS	48	1843	14	99.8%	400	36860
ROOT	6	671	0	100%	91	3378
DIV	5	1315	1	99.9%	105	3923
DIST	5	553	0	100%	77	1629

Conclusions

- A methodology to evaluate the quality of properties
  - It is theoretically founded
  - It can be applied to CTL and LTL properties
  - It relies on mutation analysis
  - It exploits fault simulation and ATPG
    - time consuming re-checking is restricted to a small number of faults
  - Compared to other approaches
    - It provides a better accuracy for tautologies and linear invariants
    - It includes the notion of path coverage and not only state coverage
  - It has been completely implemented into the PCC (Property Coverage Checker) tool

References

- A. Fedeli, F. Fummi, G. Pravadelli, U. Rossi, F. Toto, *"On the Use of a High-Level Fault Model to Check Properties Incompleteness"*, Proc. of ACM/IEEE MEMOCODE 2003.
- F. Fummi, G. Pravadelli, F. Toto, "Coverage of Formal Properties based on a High-Level Fault Model and Functional ATPG", Proc. of IEEE ETS 2005.
- A. Fedeli, F. Fummi, G. Pravadelli, *"Properties Incompleteness Evaluation by Functional Verification"*, IEEE Trans. on Computers, vol. 56, n. 4, 2007.