

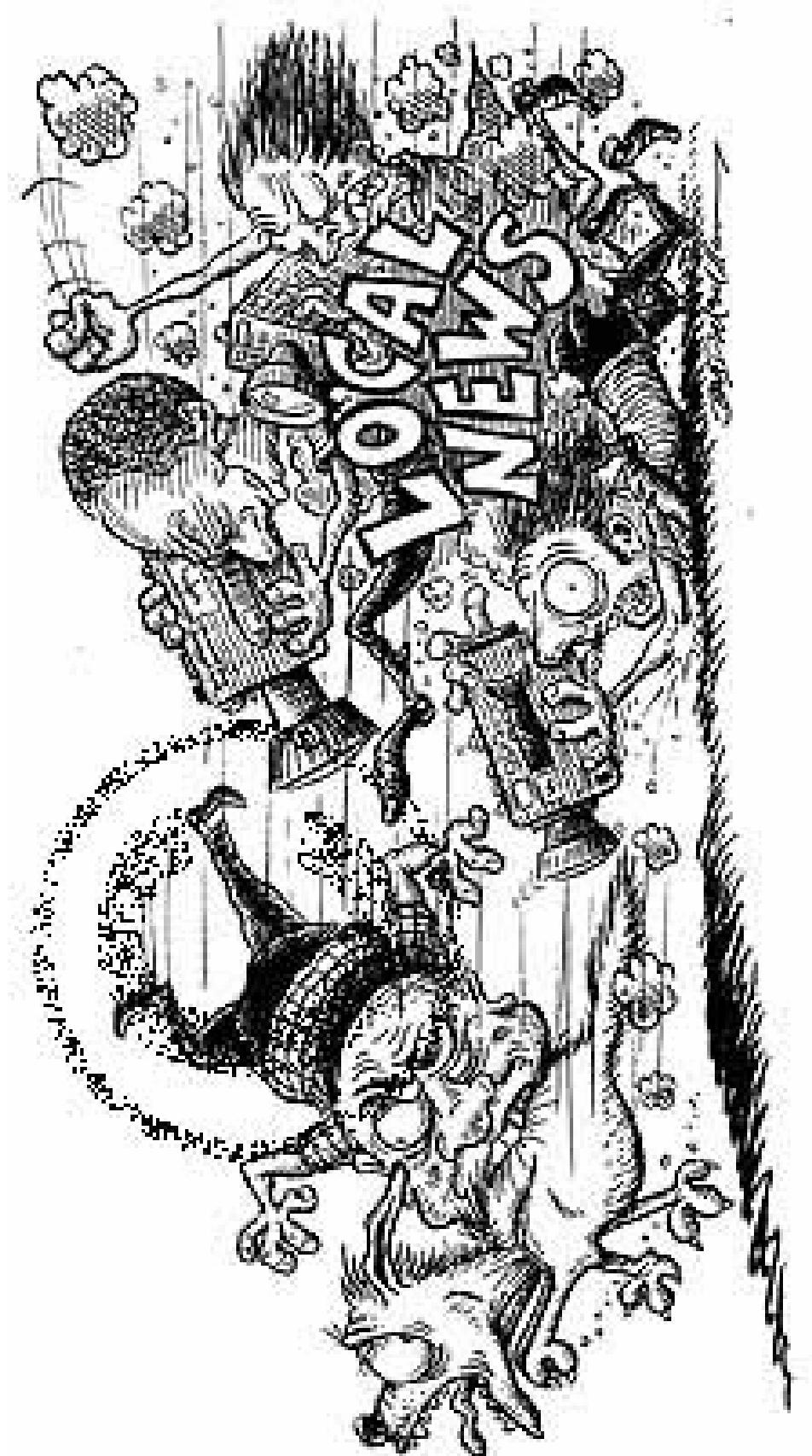
Coverage Metrics

Wenchao Li
EECS 219C
UC Berkeley

Outline of the lecture

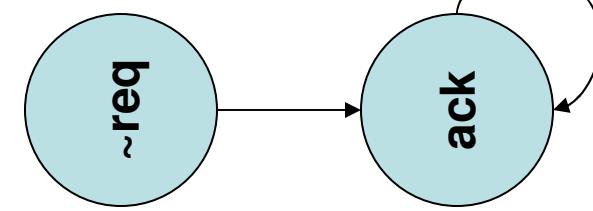
- Why do we need coverage metrics?
- Criteria for a good coverage metric.
- Different approaches to define coverage metrics.
- Different types of coverage metrics.

A different kind of coverage



Why do we need Coverage Metrics?

A simple Example:



LTL specification:

Assert G ($\text{req} \rightarrow F \text{ ack}$);



Extreme Case:

The other extreme?

0 specification!

1 spec for every state transition

Ok, so suppose now we know
we need more specifications,
but do we know what
specifications to write?

Why do we need Coverage Metrics?

In general:

- specs are not necessarily complete;
- need to prompt/assist hardware/software testers;
- tradeoff between cost of providing coverage and performance/reliability;
- should we have a single coverage metrics or many application-dependent coverage metrics?
- coverage metrics in simulation-based verification → coverage metrics in formal verification.

What are the criteria for a good coverage metrics?

- **Direct correspondence with bugs.**
Question 1: checking incompleteness of specifications ≠ finding redundancies in the system?
- **Reasonable computational and human effort to:**
 - (a) compute the metrics;
 - (b) interpret coverage data and generate stimuli to exercise uncovered aspects;
 - (c) achieve high coverage;
 - (d) minimal modification to validation framework.
- **Knowledge of the design required? – Coverage Metrics for Blackbox Testing.** [M. W. Whalen et. al. 2006] [Ajitha Rajan 2006]
- **“Observability” – complete specifications vs. abstract specifications.**

Defining Coverage

(1) “**Simulation Approach**”: [S. Katz

The reduced tableau can be huge!

- ACTL Safety properties.
- A well-covered implementation should closely resemble the *reduced tableau* of its specification.
- Hence, a fully covered implementation is **bisimilar** to the *reduced tableau* of its specification, i.e. has the same set of behaviors as the spec!

Bisimulation is very strict!

Defining Coverage

(1) “**Simulation Approach**” cont.:

- Four criteria in comparing an implementation / with the reduced tableau S of the specification.
 - (a) **UnImplementedStartState**, which contains the set of states w_0' in W_0' for which all $w \notin W_0$ have $w \not\in \text{sim}(w_0)$.
 - (b) **UnImplementedState**, which contains the set of states $w' \in W$ for which all $w \in W$ have $w' \notin \text{sim}(w)$.
 - (c) **UnImplementedTransitions**, which contains the set of transitions $\langle w', u' \rangle \in R'$ for which S simulates / even without the transition $\langle w', u' \rangle$.
 - (d) **ManyToOne**, which contains the set of states $w' \in W'$ for which $\text{sim}^{-1}(w')$ is not a singleton.

Four criteria are empty *iff* the implementation and the *reduced tableau* of the specification are **bisimilar**.

Defining Coverage

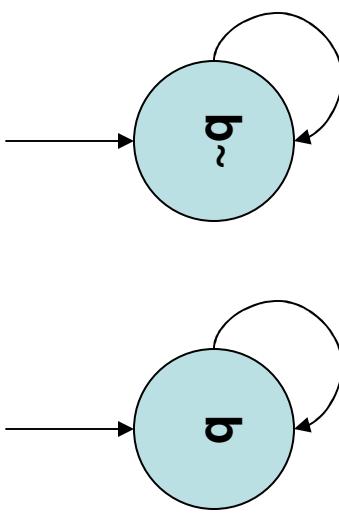
(2) “**Mutant-based Approach**”: [Y. Hoskote et. al. 1999]

- Inspired by mutation coverage in simulation-based verification. [D. L. Dill 1998]
- Formally, for an implementation I (modeled as a labeled state-transition graph), a state w in I , and an observable signal q , we say that w is q -covered by a specification S if $I_{w,q}^S$ (mutant implementation by flipping the value of q in w) does not satisfy S .

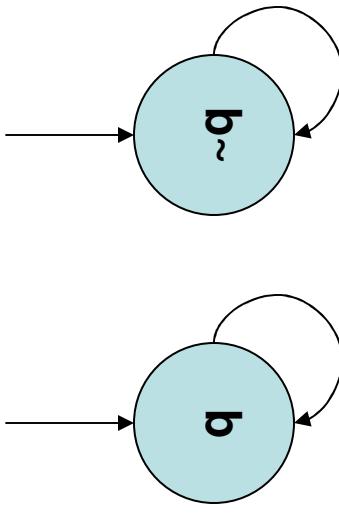
How are these 2 approaches different?

Specification: $AGq \vee AG\neg q$

System I_1 :



Reduced tableau S_1 :



Simulation approach: all 4 criteria are empty \rightarrow full coverage.

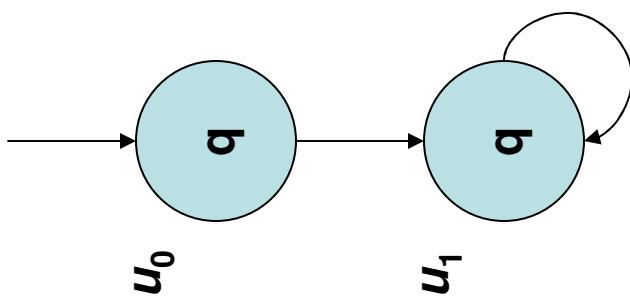
Mutant-based approach: both states of I_1 are not q -covered.

How are these 2 approaches different?

Specification: AGq

System I_2 :

Reduced tableau S_2 :



Simulation Approach:

Criteria 4 is not empty:
both states of I_2 are
simulated by the state t_0 .

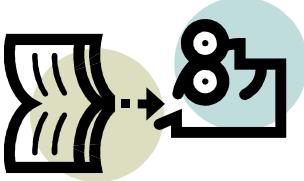
Mutant-based Approach:

I_2 is q-covered by S_2 .

Mutant-based Approach

Two coverage checks:

- (1) **Falsity coverage**: does the mutant FSM still satisfy the specification?
- (2) **Vacuity coverage**: if the mutant FSM still satisfies the specification, does it satisfy it vacuously?



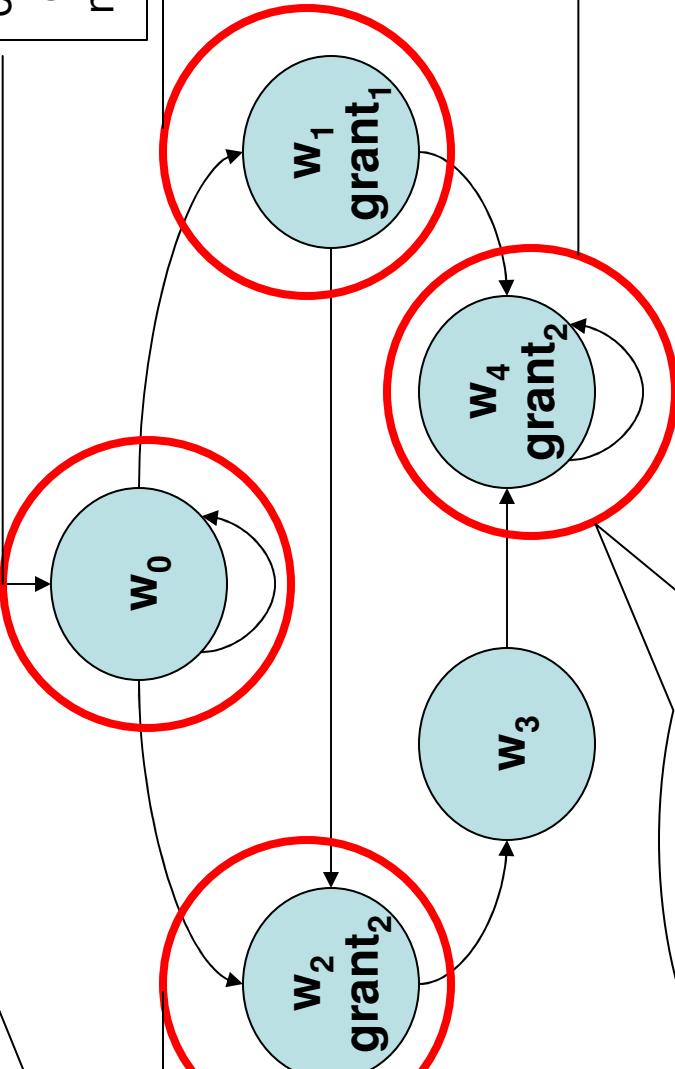
- (1) $G(\text{grant}_1 \rightarrow X \text{ grant}_2)$?
 (2) Number of $\text{grant}_2 = 2$?
 (3) Redundancy, i.e. w_2 can be omitted?

mples!

w_0 is vacuity-covered by S w.r.t. mutation on grant_2

w_1 is vacuity-covered by S w.r.t. omission of w_1 or mutation on grant_1

w_4 is falsity-covered by S w.r.t. mutation on grant_2



LTL
structure-covered
(flipped always) vs.
node-covered
(flipped only once)

Question: Is w_4 structure-covered or node-covered by S w.r.t. the mutation on grant_2 ?

Types of coverage metrics

A. **Syntactic-coverage** metrics

- Code-based coverage
- Circuit coverage
- Hit count

B. **Semantic-coverage** metrics

- FSM coverage
- Assertion coverage
- Mutation coverage

Types of coverage metrics

A. Syntactic-coverage metrics

- **Code-based coverage**

- Circuit coverage

- Hit count

B. Semantic-coverage metrics

- FSM coverage

- Assertion coverage

- Mutation coverage

Code Coverage (simulation)

- statement coverage
- branch coverage
- expression coverage

What if there is
concurrency?

- Given a CFG called G , for an input sequence $t \in (2^I)^*$ such that the execution of G on t , projected on the sequence of locations, is l_0, \dots, l_m , we say that a statement s is covered by t if there is $0 \leq j \leq m$ s.t. l_j corresponds to s ; a branch $< l, l' >$ is covered by t if there is $0 \leq j \leq m-1$ such that $l_j = l$ and $l_{j+1} = l'$.

Code Coverage (F.V.)

- Given a CFG called G and ξ a specification satisfied in G , we say a statement s of G is covered by ξ if omitting s from G causes vacuous satisfaction of ξ in the mutant CFG. Similarly, a branch $\langle l, l' \rangle$ of G is covered if omitting it causes vacuous satisfaction of ξ .
- Why **vacuous satisfaction** only?

Types of coverage metrics

A. **Syntactic-coverage** metrics

- Code-based coverage

Circuit coverage

B. **Semantic-coverage** metrics

- FSM coverage
- Assertion coverage
- Mutation coverage

Circuit Coverage (simulation)

- **latch coverage**
- **toggle coverage**
- A latch is covered if it changes its value at least once during the execution of the input sequence.
- An output variable is covered if its value has been toggled (requires the value to be changed at least twice).

Circuit Coverage (F.V.)

- Replace the question by the question of whether disabling the change causes the specification to be satisfied vacuously.
- A latch / is covered if the specification is vacuously satisfied in the circuit obtained by fixing the value of / to its initial value.
- An output o is covered if the specification is vacuously satisfied in the circuit obtained by allowing o to change its value only once.

Types of coverage metrics

A. **Syntactic-coverage** metrics

- Code-based coverage
- Circuit coverage

• **Hit count**

B. **Semantic-coverage** metrics

- FSM coverage
- Assertion coverage
- Mutation coverage

Hit Count (simulation)

- Replace binary coverage queries with quantitative measurements – the number of times an object has been visited.
- Visited often → functionality better covered.

Hit Count (F.V.)

- The **minimal** number of visits in which we have to perform the mutation (or omission of the element) in order to falsify the specification in the design or to make it vacuously satisfied.

Types of coverage metrics

A. **Syntactic-coverage** metrics

- Code-based coverage
- Circuit coverage
- Hit count

B. **Semantic-coverage** metrics

• **FSM coverage**

- Assertion coverage
- Mutation coverage

FSM Coverage (simulation)

- A state or a transition covered if it is visited during the input sequence
- Transition coverage can be extended to **path coverage**.
- Problem?

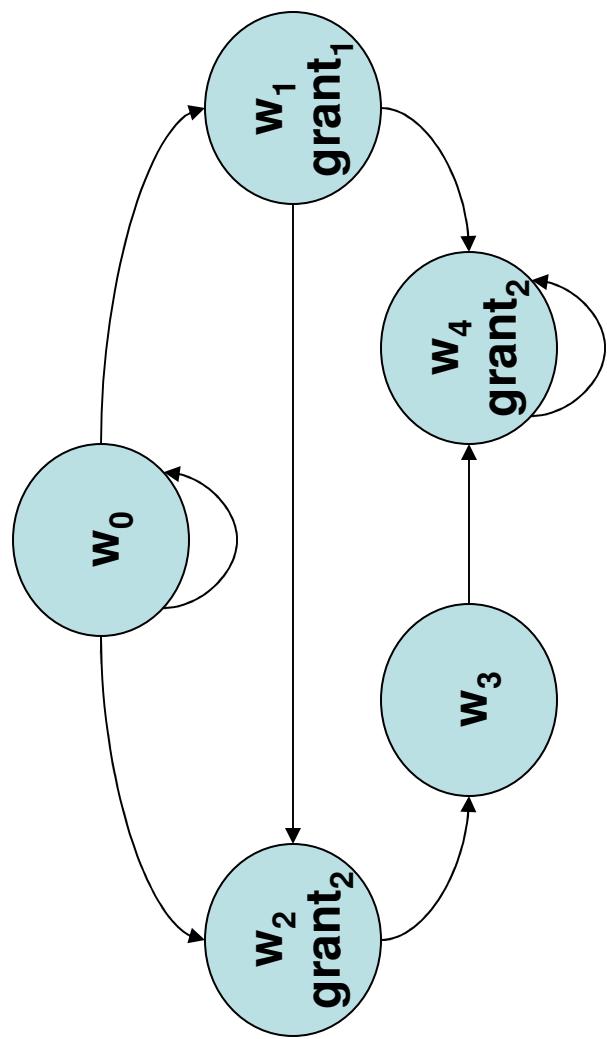
computing the path coverage is expensive!

linking the uncovered parts of the FSM to uncovered parts of the HDL program is not trivial!

FSM coverage (F.V.)

- In **state coverage**, we check the influence of omission of a state w or changing the values of the output variables in w on the (nonvacuous) satisfaction of the specification.
- In **path coverage**, we check the influence of omitting or mutating a finite path on the (nonvacuous) satisfaction of the specification.

I want an example again!

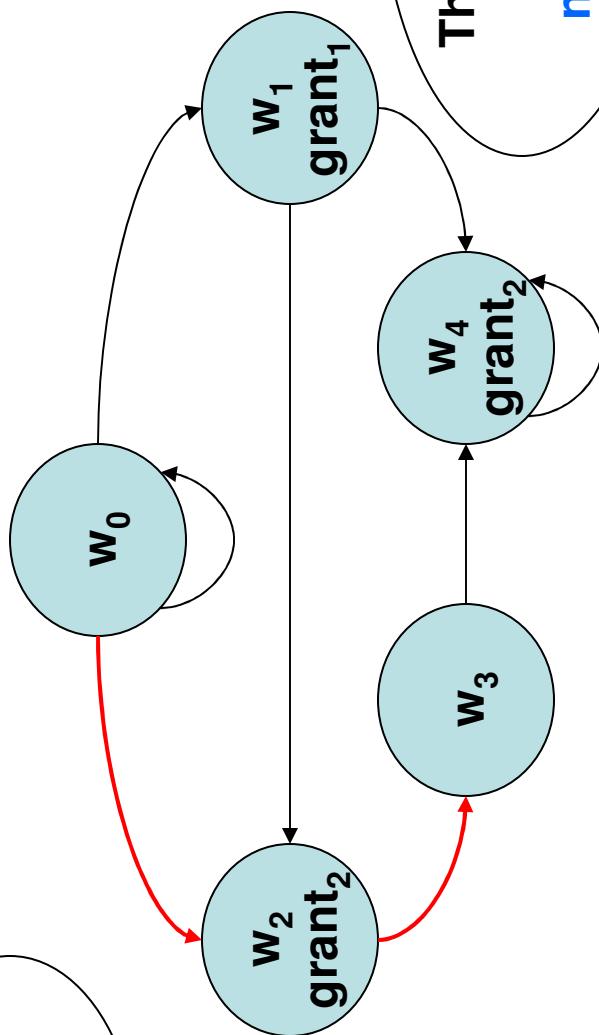


LTL specification S: assert G (grant₁ → F grant₂);

| war sample again!

removing transitions
 $\langle w_0, w_2 \rangle$ and
 $\langle w_2, w_3 \rangle$

omitting path w_0, w_2, w_3 .



The specification
is satisfied
nonvacuously.

LTL specification S: assert G ($grant_1 \rightarrow F grant_2$);

Types of coverage metrics

A. Syntactic-coverage metrics

- Code-based coverage
- Circuit coverage
- Hit count

B. Semantic-coverage metrics

- FSM coverage
- **Assertion coverage**
- Mutation coverage

Assertion Coverage (simulation)

- Also called “functional coverage”.
- Assertions can be **propositional** or **temporal**.
- A test t covers an assertion a if the execution of the design on t satisfies a .
- Measures % assertions covered for a given set of input sequences.

Assertion Coverage (F.V.)

- An assertion a is covered by a specification ξ in a FSM F if the mutant FSM F' obtained from F by omitting all behaviors that satisfy a satisfies ξ nonvacuously.
- What coverage metric that we have talked about is similar to this one?
FSM Path Coverage!

Types of coverage metrics

A. **Syntactic-coverage** metrics

- Code-based coverage
- Circuit coverage
- Hit count

B. **Semantic-coverage** metrics

- FSM coverage
- Assertion coverage
- **Mutation coverage**

Mutation Coverage (simulation)

- User introduces a small change to the design and check for erroneous behavior.
- The coverage of a test t is measured as the % mutant designs that fail on t .
- The goal is to find a set of input sequences s.t. for each mutant design there exists at least one test that fails it.

Mutation Coverage (F.V.)

- By mutation, we actually mean local mutation.
- We have talked about this.

Mutant-based Approach

Two coverage checks: [H. Chockler et. al. 2006]

- (1) **Falsity coverage**: does the mutant FSM still satisfy the specification?
- (2) **Vacuity coverage**: if the mutant FSM still satisfies the specification, does it satisfy it vacuously?

Computing Coverage (1)

- **Mutant-based** Approach: [Y. Hoskote et. al. 1999]
- The goal is to find the set of covered states.
- $\text{Coverage} = \frac{\text{number of covered states}}{\text{number of reachable states}}$
- **Recursive** algorithm for a given ACTL formula and a given observed signal.

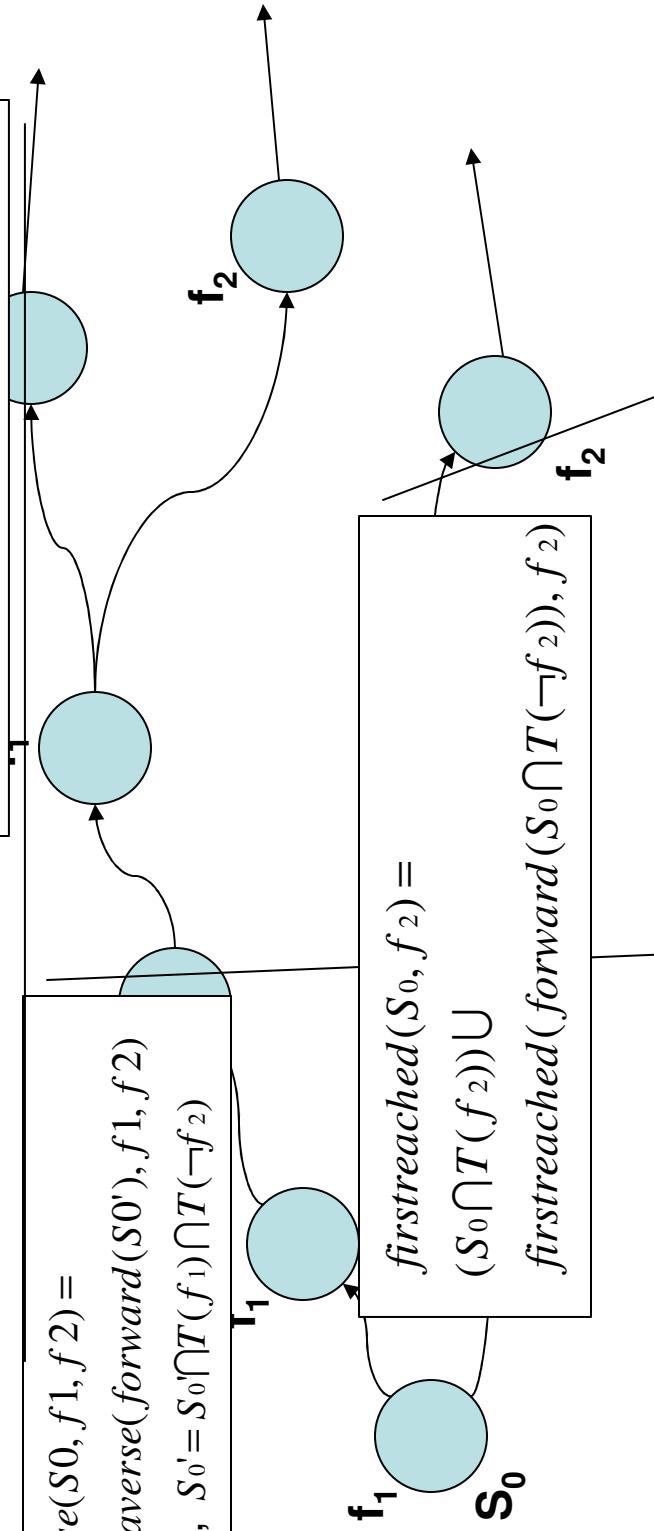
$T(b)$ represent the set of states which satisfy b .

- Say, we want to comp

Of the algorithm

forward(S_0) gives states reachable in exactly one step from the start states in S_0 .

$traverse(S_0, f_1, f_2) = S_0 \cup traverse(forward(S_0'), f_1, f_2)$
where, $S_0' = S_0 \cap T(f_1) \cap T(\neg f_2)$



$$C(S_0, A(f_1 \cup f_2)) = C(traverse(S_0, f_1, f_2) \cup C(firstreached(S_0, f_2), f_2))$$

Computing Coverage (2)

[H. Chockler et. al. 2006]

```
module example(O1,O2,O3);
reg O1,O2,O3;
initial begin
    O1=O2=O3=0;
end
always @ (posedge clk)
begin
    assign O1=O1;
    assign O2=O2|O3;
    assign O3=~O3;
end
endmodule
```

S: assert G(O₂ → F O₃)

Code Coverage:

This statement is uncovered by the specification w.r.t. to both omission and mutations.

Computing Coverage (2)

[H. Chockler et. al. 2006]

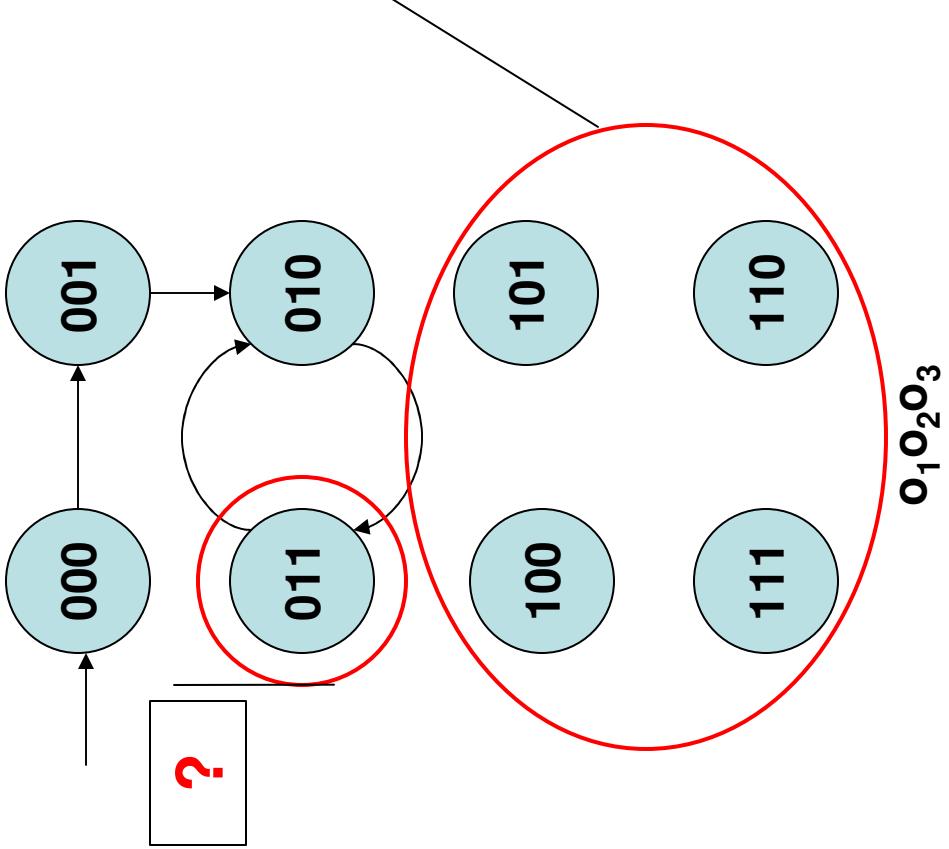
```
module example(O1,O2,O3);
reg O1,O2,O3;
initial begin
    O1=O2=O3=0;
end
always @ (posedge clk)
begin
    assign O1=O1;
    assign O2=O2|O3;
    assign O3=~O3;
end
endmodule
```

Circuit Coverage:

Latch O₁ is uncovered
– fixing O₁ to 0 for the
whole execution does
not affect the
satisfaction of S.

Computing Coverage (2)

[H. Chockler et. al. 2006]



S: assert $G(o_2 \rightarrow F o_3)$;

FSM Coverage:

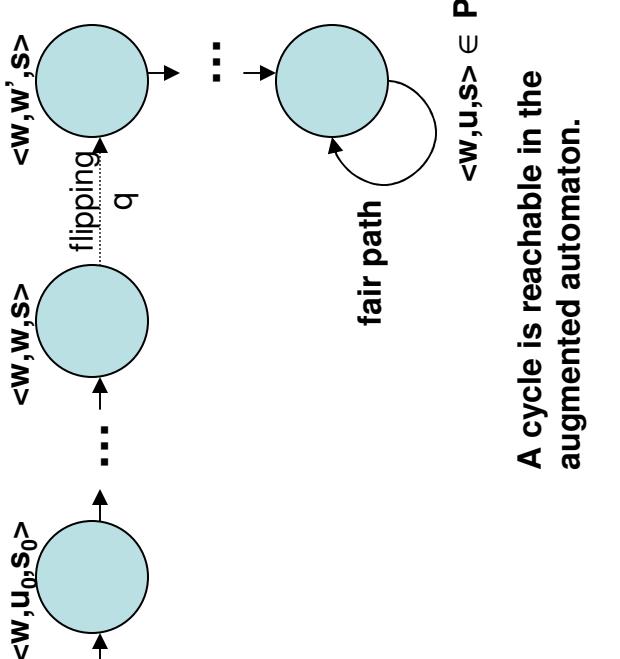
All states in which $o_1 = 1$ are unreachable, and thus uncovered w.r.t. all specifications.

Computing Coverage (2)

[H. Chockler et. al. 2006]

- (1) Naive way: **enumerate** through all mutant FSM to check both falsity and vacuity coverage for a specification ξ .

- (2) A Better way: compute coverage **symbolically**.



- The idea is to look for a fair path in the product of the mutant FSM F' and an automaton $A_{\sim\xi}$ for the negation of ξ .

- Add a new variable x that encodes a subformula in ξ that is being replaced by true/false. The value 0 for x stands for “no replacement”. Then we check the satisfaction of ξ in the system.

- Consider an augmented product with state space $2^x \times 2^x \times S$.

A cycle is reachable in the augmented automaton.

Complexity of Computing Coverage

Metric	Complexity
Mutation Coverage	$3n + 2m$
Vacuity Coverage	$3n + 3m$
Code Coverage	$2n + 2m + \log k$
Circuit Coverage	$2n + 3m + \log l$
FSM Path Coverage	$3n + 4m + \log p$
Assertion Coverage	$3n + 4m + \log k$
Hit Count	$3n + 2m + \log t$

Complexity in terms of ROBDD variables required.

n and m are the number of variables required for encoding the state space of F and $A_{\sim\xi}$; l denotes the number of latches in the circuit; k denotes the number of assertions/number of lines of nodes; p denotes the length of the path; t denotes the threshold of hit count.

Conclusion

- Two problems inherent with any coverage metrics.
- Two approaches to define coverage metrics for formal verification.
- Different semantic and syntactic coverage metrics – how are they similar to/different from the ones used in simulation.
- Still an **open problem**.

Reference List

- Michael W. Whalen, Ajitha Rajan, Mats P.E. Heimdahl, Steven P. Miller. [Coverage metrics for requirements-based testing](#). Proc. of International Symposium on Software Testing and Analysis, page 25-36, 2006.
- Ajitha Rajan. [Coverage Metrics to Measure Adequacy of Black-Box Test Suites](#). Proc. of International Conference on Automated Software Engineering, page 335-338, 2006.
- S. Katz, O. Grumberg, D. Geist. “[Have I written enough properties? - A method of comparison between specification and implementation](#)”. Proc. of 10th ACM Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARM), page 280-297, 1999.
- D.L. Dill. [What's between simulation and formal verification?](#) Proc. of 35th Design Automation Conference, page 328–329, 1998.
- Y. Hoskote, T. Kam, P.-H Ho, X. Zhao. [Coverage estimation for symbolic model checking](#). Proc. of 36th Design Automation Conference, page 300–305, 1999.
- Hana Chockler, Orna Kupferman, Moshe Y. Vardi. [Coverage metrics for formal verification](#). Internal Journal on Software Tools for Technology Transfer (STTT), Vol. 8, Issue 4, Page 373-386, August 2006.

Reference List

- Hana Chockler, Orna Kupferman, Moshe Y. Vardi. [Coverage metrics for formal verification](#). Proc. of 12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME), 2003.
- Hana Chockler, Orna Kupferman. [Coverage of Implementations by Simulating Specifications](#). Proc. Of 2nd IFIP International Conference on Theoretical Computer Science: Foundations of Information Technology in the Era of Networking and Mobile Computing, page 409-421, 2002.
- Hana Chockler, Orna Kupferman, Robert P. Kurshan, Moshe Y. Vardi. A [Practical Approach to Coverage in Model Checking](#). In Proc. of 13th International Conference on Computer Aided Verification, page 66-78, 2001.
- Hana Chockler, Orna Kupferman, Moshe Y. Vardi. [Coverage Metrics for Temporal Logic Model Checking](#). In Proc. of 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, page 528-542, 2001.
- Orna Kupferman, Moshe Y. Vardi. [Vacuity Detection in Temporal Model Checking](#). In Proc. of 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods, page 82-96, 1999.