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 (req → F 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;
  Question 2: 100% coverage = complete design?
  (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 et al. 1999]

- 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 specification.

Bisimulation is very strict!
The reduced tableau can be huge!
Defining Coverage

(1) “Simulation Approach” cont.:

- Four criteria in comparing an implementation \( I \) with the reduced tableau \( S \) of the specification.
  
  (a) **UnImplementedStartState**, which contains the set of states \( w_0' \) in \( W_0' \) for which all \( w \in W_0 \) have \( w \sim w_0' \).

  (b) **UnImplementedState**, which contains the set of states \( w' \in W' \) for which all \( w \in W \) have \( w' \not\in \text{sim}(w) \).

  (c) **UnImplementedTransitions**, which contains the set of transitions \( \langle w', u' \rangle \in R' \) for which \( S \) simulates \( I \) 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 $I_{w,q}'$ (mutant implementation by flipping the value of $q$ in $w$) does not satisfy $S$. 
How are these 2 approaches different?

**Specification:**  \( AGq \lor AG\neg q \)

**System \( I_1 \):**

\[ q \quad \neg q \]

**Reduced tableau \( S_1 \):**

\[ q \quad \neg q \]

**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 $l_2$:**

- $u_0$
- $u_1$

**Reduced tableau $S_2$:**

- $t_0$

**Simulation Approach:**

Criteria 4 is not empty: both states of $l_2$ are simulated by the state $t_0$.

**Mutant-based Approach:**

$l_2$ 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?
I want an example!

(1) $G (grant_1 \rightarrow X grant_2)$?
(2) Number of $grant_2 = 2$?
(3) Redundancy, i.e. $w_2$ can be omitted?

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

Given a CFG called $G$, for an input sequence $t \in (2^l)^*$ such that the execution of $G$ on $t$, projected on the sequence of locations, is $l_0, ..., l_m$, 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 $\xi$ a specification satisfied in $G$, we say a statement $s$ of $G$ is covered by $\xi$ if omitting $s$ from $G$ causes vacuous satisfaction of $\xi$ in the mutant CFG. Similarly, a branch $<l,l'>$ of $G$ is covered if omitting it causes vacuous satisfaction of $\xi$.

• Why *vacuous satisfaction* only?
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
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 \( l \) is covered if the specification is vacuously satisfied in the circuit obtained by fixing the value of \( l \) 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 of the FSM is covered if it is visited during the execution of 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 $G \ (\text{grant}_1 \rightarrow F \ \text{grant}_2)$;
I want an example again!

omitting path $w_0$, $w_2$, $w_3$. ≠ removing transitions $<w_0, w_2>$ and $<w_2, w_3>$

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 \( \xi \) in a FSM \( F \) if the mutant FSM \( F' \) obtained from \( F \) by omitting all behaviors that satisfy \( a \) satisfies \( \xi \) 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.
- Coverage = number of covered states / number of reachable states
- **Recursive** algorithm for a given ACTL formula and a given observed signal.
Outline of the algorithm

- Say, we want to compute $C(S_0, A(f_1 \cup f_2))$.

\[
C(S_0, A(f_1 \cup f_2)) = C(\text{traverse}(S_0, f_1, f_2) \cup C(\text{firstreached}(S_0, f_2), f_2))
\]

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

\[
\text{traverse}(S_0, f_1, f_2) = S_0 \cup \text{traverse}(\text{forward}(S_0'), f_1, f_2)
\]

where, \(S_0' = S_0' \cap T(f_1) \cap T(\neg f_2)\)

- \(\text{forward}(S_0)\) gives states reachable in exactly one step from the start states in \(S_0\).

\[
\text{firstreached}(S_0, f_2) = (S_0 \cap T(f_2)) \cup \text{firstreached}(\text{forward}(S_0 \cap T(\neg f_2)), f_2)
\]
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(o_1,o_2,o_3);
   reg o_1,o_2,o_3;
   initial begin
      o_1=o_2=o_3=0;
   end
   always @ (posedge clk)
      begin
         assign o_1=o_1;
         assign o_2=o_2|o_3;
         assign o_3=~o_3;
      end
endmodule

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

**Circuit Coverage:**
Latch o_1 is uncovered – fixing o_1 to 0 for the whole execution does not affect the satisfaction of S.
Computing Coverage (2)

[H. Chockler et al. 2006]

\[ \text{S: assert } G(o_2 \rightarrow F o_3) : \]

\textbf{FSM Coverage:}
All states in which \( o_1 = 1 \) are unreachable, and thus uncovered w.r.t. all specifications.

\[ \text{000, 010, 011, 100, 110, 111} \]
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 $\xi$.

(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 $A_{\neg \xi}$ for the negation of $\xi$.
- Add a new variable $x$ that encodes a subformula in $\xi$ that is being replaced by true/false. The value 0 for $x$ stands for “no replacement”. Then we check the satisfaction of $\xi$ 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

<table>
<thead>
<tr>
<th>Metric</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutation Coverage</td>
<td>$3n + 2m$</td>
</tr>
<tr>
<td>Vacuity Coverage</td>
<td>$3n + 3m$</td>
</tr>
<tr>
<td>Code Coverage</td>
<td>$2n + 2m + \log k$</td>
</tr>
<tr>
<td>Circuit Coverage</td>
<td>$2n + 3m + \log l$</td>
</tr>
<tr>
<td>FSM Path Coverage</td>
<td>$3n + 4m + \log p$</td>
</tr>
<tr>
<td>Assertion Coverage</td>
<td>$3n + 4m + \log k$</td>
</tr>
<tr>
<td>Hit Count</td>
<td>$3n + 2m + \log l$</td>
</tr>
</tbody>
</table>

**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

Reference List