Coverage Based Test-Case Generation using Model Checkers

Sanjai Rayadurgam and Mats P. E. Heimdahl
Goals

- Reduce (high) costs of testing safety-critical software
- Save time
- Ensure certain level of coverage
- Automated test derivation
Main idea

- Use model checker to generate complete test sequences

- Based on:
  Software artifact (e.g. source code, software specifications) that can be represented as **finite state model**

  **Structural coverage (testing) criteria** formalized and expressed as **temporal logic formula**

- Produces counter examples which can be transformed into test cases
Main idea

Example:

Test a transition between states A and B, guarded with condition C.
Formulate a condition that in state A, C must be true, and the next state must be B.

Challenge model checker to find a counter example by negating the created condition.
What is...?

- **Model checker**
  Tools that explore the reachable state space of a model and report if properties of interest are violated in some state
  
  e.g. A state of a program should never be reachable, but was reached at some point. This leads to a counter example

- **Temporal logic**
  Extension of the logical truth operators (¬, ∨, ∧, →) with modal operators

- **Linear temporal logic (LTL)**
  encode formulas about the future of paths
  (e.g. a condition will be true until another fact becomes true)
System Model

- Uniquely determined by the value of $n$ variables $\{x_1, x_2, \ldots, x_n\}$ where $x_i$ takes its value from its domain $D_i$

- The reachable state space of the system is a subset of $D = D_1 \times D_2 \times \ldots \times D_n$

- The system may move from one state to another but only if it follows the rules of its transition relation, which defines the legal moves
The transition relation for the variable $x_i$ consists of three components: a set of pre-state values of $x_i$, a set of post-state values for $x_i$ and the condition which guards when $x_i$ may change from a pre-state value to a post-state value.

**Definition 1.**
A *predicate* is a boolean valued function parameterized by variable references.

**Definition 2.**
A *clause* is a predicate that cannot be broken down into sub-predicates connected by boolean operators.

**Definition 3.**
A *pre-state predicate* for a variable $x_i$ is a predicate whose only parameter is the variable reference $x_i$. We use $\alpha_{i,j}$ to denote the $j^{th}$ pre-state predicate of $x_i$.

**Definition 4.**
A *post-state predicate* for a variable $x_i$ is a predicate whose parameters are from the set $\{x_1, \ldots, x_n; x'_1 \ldots x'_i\}$ in which every clause includes the variable reference, $x'_1$. We use $\beta_{i,j}$ to denote the $j^{th}$ post-state predicate of $x_i$. 

A. Knaus  Coverage Based Test-Case Generation using Model Checkers 7/24
Definition 5.
A **simple transition** for a variable $x_i$ is a conjunction of a pre-state predicate for $x_i$, a post-state predicate for $x_i$, and a predicate called the **guard** whose parameters are from the set $\{x_1, \ldots, x_n; x'_1 \ldots x'_i\}$. We $\gamma_{i,j}$ to denote the $j^{th}$ guard and $\delta_{i,j}$ to denote the $j^{th}$ simple transition of $x_i$. Thus, $\delta_{i,j} = \alpha_{i,j} \land \beta_{i,j} \land \gamma_{i,j}$.

Definition 6.
The **complete transition** for a variable $x_i$, denoted $\delta_i$, is the disjunction of all simple transitions for $x_i$. Thus, $\delta_i = \bigvee_{j=1}^{n_i} \delta_{i,j}$ where $n_i$ is the number of simple transitions for the variable $x_i$.

Definition 7.
The **transition relation** $\Delta$, is the conjunction of the complete transitions of all the variables $x_1, \ldots, x_n$. Thus, $\Delta = \bigwedge_{j=1}^{n_i} \delta_i$
System Model - Transition Relation

- Special case:
  The *initial state* predicate, \( \rho \) is similar to a *transition relation* in which all the guards and the pre-state predicates are absent (i.e., equivalent to the constant predicate *true*).

- We now define a **basic transition system** \( M \) as a tuple, \( M = (D, \Delta, \rho) \), where \( D \) represents the state-space of the system, \( \Delta \), represents the transition relation, and \( \rho \) characterizes the initial system state.
Structural Coverage Criteria

- Simple transition (triple of predicates $\alpha, \beta, \gamma$) used for test criteria

- Predicates are parameterized by variable references and states are assignment of values to variables

- It is meaningful to evaluate predicates ($\rho$) using states ($s_i$) as arguments

  e.g. $\rho(s_1)$ where $s_1$ is the init. state

- A test case is simply a sequence of states
Simple Transition Coverage

- **Definition 8.**
  A test suite is said to **achieve simple transition coverage** for a basic transition system \( M = (D, \Delta, \rho) \), if for any simple transition \((\alpha, \beta, \gamma)\) of any variable \(x\), there exists a test case \(s\) such that for some \(i\),
  \[
  \alpha(s_i) \land \beta(s_i, s_{i+1}) \land \gamma(s_i, s_{i+1}) \quad \text{holds true}
  \]

- In other words, for every simple transition for each variable, there is a test case in the test suite in which the simple transition is taken
Simple Guard Coverage

- Definition 9.
  A test suite achieves simple guard coverage if for any simple transition \((\alpha, \beta, \gamma)\) there exist test cases \(s\) and \(t\) such that for some \(i\) and \(j\):
  \[
  \alpha(s_i) \land \beta(s_i, s_{i+1}) \land \gamma(s_i, s_{i+1})
  \]
  the simple transition is taken in the transition from \(s_i\) to \(s_{i+1}\) in the test case \(s\)

  \[
  \alpha(t_j) \land \neg \beta(t_j, t_{j+1}) \land \neg \gamma(t_j, t_{j+1})
  \]
  the simple transition is \textbf{not} taken in the transition from \(t_j\) to \(t_{j+1}\) in the test case \(t\)

- guard considered as the decision point in the program
Complete Guard Coverage

- Definition 10.
  If the clauses in a guard $\gamma$ of a simple transition $(\alpha, \beta, \gamma)$ are \{${c_1, c_2, \ldots, c_l}$\}, a test suite that achieves complete guard coverage must include a test case $s$ for any given boolean vector $u$ of length $l$, such that for some $i$, 
  \[
  \Delta = \bigwedge_{k=1}^{l}(c_k(s_i, s_{i+1}) = u_k)
  \]

- analogous to the multiple condition coverage in program source code

- For every $c$ clause (atomic units of predicates)
  \[\rightarrow \text{state explosion} \]

A. Knaus
Coverage Based Test-Case Generation using Model Checkers
Clause-wise Guard Coverage

- Based on modified condition/decision coverage (MC/DC)

- MC/DC was developed as a practical and reasonable compromise between decision coverage and multiple condition coverage

- MC/DC is satisfied if a test suite covers:
  - every point of entry and exit in the program has been invoked at least once
  - every condition in a decision in the program has taken on all possible outcomes at least once
  - each condition has been shown to independently affect the decision’s outcome
Clause-wise Guard Coverage

- A condition is shown to independently affect a decision’s outcome by varying only that condition while holding all other conditions at that decision point fixed.

- Thus, a pair of test cases must exist for each condition in the test-suite to satisfy MC/DC. However, test case pairs for different conditions need not necessarily be disjoint.

  Needed size for N conditions \( \rightarrow N+1 \) for one predicate.
Clause-wise Guard Coverage

- **Definition 11.**
  A test suite $S$ **covers** a clause $c$ of the guard $\gamma$ of a simple transition $(\alpha, \beta, \gamma)$, if it contains two test cases $s$ and $t$, such that for some $i$ and $j$:

  1. $\alpha(s_i) \land \beta(s_i, s_{i+1}) \land \gamma(s_i, s_{i+1})$
     the simple transition is taken in the transition from $s_i$ to $s_{i+1}$ in the test case $t$

  2. $\alpha(t_j) \land \neg \beta(t_j, t_{j+1}) \land \neg \gamma(t_j, t_{j+1})$
     the simple transition is **not** taken in the transition from $t_j$ to $t_{j+1}$ in the test case $t$

  3. $c(s_i, s_{i+1}) = \neg c(t_j, t_{j+1})$
     the value of the clause $c$ of the guard $\gamma$ differs for the two transitions

  4. $\forall d \neq c \text{ in } \gamma, d(s_i, s_{i+1}) = d(t_j, t_{j+1})$
     all other clauses in the guard $\gamma$ have the same value in both transitions.
Test Case Generation

- The properties are constructed in such a way that they fail for the given system specification, which in our case is the basic transition system $M = (D, \Delta, \rho)$, leading the model checker to produce a counter-example (trap properties).

- The counter-example shows a valid sequence of states that any conforming implementation should follow. This sequence of states becomes a test case.
Trap Property Generation

- Based on CGC
- Expressed in Linear Temporal Logic (LTL)
- Using concrete values for the outcome of the guards ($\gamma$) for the test cases $s$ and $t$; (Following the rules of CGC 3 & 4)
  
  $$s_i \rightarrow u \quad t_j \rightarrow v$$
Trap Property Generation

- Formulate a pair of trap properties, so that both will lead to a counterexample

\[ G(\alpha \land (\bigwedge_{k=1}^{l} c_k = u_k) \implies \neg \beta) \]

it is **globally** (G) **true** for the basic transition system M that if the pre-state condition \( \alpha \) holds and the clauses in the guard evaluate to truth values indicated by the vector \( \mathbf{u} \), then the post-state condition \( \beta \) will **not** hold.

\[ G(\alpha \land (\bigwedge_{k=1}^{l} c_k = v_k) \implies \beta) \]

it is **globally** (G) **true** for the basic transition system M that if the pre-state condition \( \alpha \) holds and the clauses in the guard evaluate to truth values indicated by the vector \( \mathbf{v} \) then the post-state condition \( \beta \) will hold.
Example

- Cruise Control System

The system has four state variables `Own_velocity`, `Front_Velocity`, `Sensitivity` and `Cruise Control`.

- The sensitivity can be `High` or `Low` depending on user setting or based on the states of the own and front vehicle velocities.

```plaintext
if (Prev_Sensitivity == ST_Low)
    if (SensitivitySetting == TY_Sens_High || (Own_Velocity == High && Front_Velocity == Low))
        Sensitivity = ST_High;
```
Example

- Guard condition for simple transition:
  
  \[
  \text{DV\_SensitivitySetting} = \text{TY\_Sens\_High} \; \text{*/1*/} \\
  | \quad \text{(EQ\_Own\_Velocity} = \text{ST\_High} \; \text{*/2*/} \\
  \quad \text{& EQ\_Front\_Velocity} = \text{ST\_Low}) \; \text{*/3*/}
  \]

- The following three boolean vector are used for \((u, v)\):
  
  \[
  ([T F F], [F F F]), ([F T T], [F F T]), ([F T T], [F T F])
  \]
Example

- Generated trap properties

1. G( (P_Sensitivity = Low)
   & (DV_Sensitivity = TY_Sens_High)
   & !(EQ_Own_Velocity = ST_High)
   & !(EQ_Front_Velocity = ST_Low)
   => !(Sensitivity = High))

2. G( (P_Sensitivity = Low)
   & !(DV_Sensitivity = TY_Sens_High)
   & !(EQ_Own_Velocity = ST_High)
   & !(EQ_Front_Velocity = ST_Low)
   => (Sensitivity = High))

3. G( (P_Sensitivity = Low)
   & !(DV_Sensitivity = TY_Sens_High)
   & (EQ_Own_Velocity = ST_High)
   & (EQ_Front_Velocity = ST_Low)
   => !(Sensitivity = High))

4. G( (P_Sensitivity = Low)
   & !(DV_Sensitivity = TY_Sens_High)
   & !(EQ_Own_Velocity = ST_High)
   & !(EQ_Front_Velocity = ST_Low)
   => !(Sensitivity = High))

5. G( (P_Sensitivity = Low)
   & !(DV_Sensitivity = TY_Sens_High)
   & (EQ_Own_Velocity = ST_High)
   & !(EQ_Front_Velocity = ST_Low)
   => !(Sensitivity = High))
Example

- **Counter example for prop. 1**
  
  \[
  \begin{array}{ll}
  \text{DV\_Front\_Speed} & 0 \\
  \text{DV\_Own\_Speed} & 0 \\
  \text{DV\_Sensitivity} & \text{TY\_Sens\_Low} \\
  \text{EQ\_Front\_Velocity} & \text{ST\_Low} \\
  \text{EQ\_Own\_Velocity} & \text{ST\_Low} \\
  \text{P\_EQ\_Sensitivity} & \text{ST\_None}
  \end{array}
  \quad \begin{array}{ll}
  & 50 \\
  & 0 \\
  & \text{TY\_Sens\_High} \\
  & \text{ST\_High} \\
  & \text{ST\_Low} \\
  & \text{ST\_Low}
  \end{array}
  \]

- **Counter example to prop. 2**
  
  \[
  \begin{array}{ll}
  \text{DV\_Front\_Speed} & 0 \\
  \text{DV\_Own\_Speed} & 0 \\
  \text{DV\_Sensitivity} & \text{TY\_Sens\_Low} \\
  \text{EQ\_Front\_Velocity} & \text{ST\_Low} \\
  \text{EQ\_Own\_Velocity} & \text{ST\_Low} \\
  \text{P\_EQ\_Sensitivity} & \text{ST\_None}
  \end{array}
  \quad \begin{array}{ll}
  & 50 \\
  & 0 \\
  & \text{TY\_Sens\_Low} \\
  & \text{ST\_High} \\
  & \text{ST\_Low} \\
  & \text{ST\_Low}
  \end{array}
  \]

- \( DV \) = input data
- \( EQ \) = observe data
- \( P \) = value of previous state
Conclusion / Problems

- Scale well to larger systems
- Various structural coverage criteria are covert
- Automatic instantiation of the test sequences with actual data for the system
  - State space explosion can affect the search for counter-examples
  - Model may be too conservative -> generated test cases cannot happen