Model-based testing

Marco Di Natale
Scuola Superiore S. Anna- Pisa, Italy

Taken from
Chapter 4: Conformance Testing by Angelo Gargantini
Chapter 1: Homing and Synchronizing Sequences, by Sven Sandberg
Model-based testing

Purpose of this Lesson
  – Learn methods for checking correctness in the implementation of an FSM
Model-based testing

**Conformance testing** between FSMs
   - typically a model and its implementation

Given a FSM specification $M_S$, for which we know the transition diagram, and another FSM $M_I$, which is the implementation and for which we can only observe the behavior, we want to know if $M_I$ correctly implements $M_S$.

Also called *fault detection* or *machine verification*
Conformance testing

$M_I$ conforms to $M_S$ if and only if their initial states are equivalent and they will produce the same output sequence for any possible input sequence.

To prove this we need to find a set of input sequences that we can apply to $M_I$ to prove its equivalence. Note that applying all input sequences is equivalent to applying the concatenation of all the input sequences. This concatenation is called checking sequence

A checking sequence for $M_S$ is an input sequence that distinguishes the class of machines equivalent to $M_S$ from other machines.
Conformance testing

Checking sequences differ for the cost to be produced, the size of the test suite (their total length) and their fault detection capability.

- They should be rather short to be practically applicable
- They should cover the implementation as much as possible and detect as many faults as possible
Model-based testing

Assumptions (requirements)

- $M_S$ is reduced or minimal
  
  Q1: How to compute a minimal FSM given a specification?

- $M_S$ is deterministic and completely specified: the state transition and the output function are defined for every state and every input symbol

- $M_S$ is strongly connected. Every state is reachable from every other state via one or more transitions
  
  • At least all states must be reachable from the initial one, if a reset transition is available, allowing machines with deadlocks

- $M_I$ does not change during testing and it has the same set of inputs and outputs as $M_S$.
  
  • Implications here (data inconsistency for incorrect concurrent implementations)
Notation

\( \lambda(s,x) = \) output function: \( s \) is the state, \( x \) the input
\( \sigma(s,x) = \) state function: \( s \) is the state, \( x \) the input
Model-based testing

- How to compute a minimal FSM given a specification

Two states $s$ and $t$ are equivalent iff $\lambda(s,x) = \lambda(t,x)$ for each possible input sequence $x \in I^*$. That is, for each input sequence, the machine starting in $s$ will produce the same output as the machine starting in $t$.

- Possibly checked using the simulation relation

- But there is a better way ...

If states $s$ and $t$ are equivalent, then the machine obtained by merging the two states is equivalent to the original one. For each machine there is an equivalent one with a minimum number of states, called reduced or minimized machine.
Model-based testing

Given a machine $M$, the minimized machine equivalent to $M$ can be obtained by a partition refinement procedure. A partition of $S$ is a set $\{B_1, B_2, ..., B_n\}$ of subsets of $S$ (also called blocks), such that $\bigcup B_i = S$ and $B_i \cap B_j = \emptyset$.

Given a mealy machine, the states of the equivalent minimized machine are the coarsest (with minimum number of elements) partition of $S$ such that, whenever $s$ and $t$ are in the same block, then

- $\lambda(s,a) = \lambda(t,a)$ for each input $a$ and
- $\sigma(s,a)$ and $\sigma(t,a)$ are in the same block for each $a$

The coarsest partition can be found starting from an initial partition of $S$ where $s$ and $t$ are in the same block iff $\lambda(s,a) = \lambda(t,a)$.

Then, the initial partition is iteratively refined:

- Take a block $B_i$
- Examine $\sigma(s,a)$ for each $s \in B_i$ and $a \in I$. Partition $B_i$ so that $s$ and $t$ stay in the same block iff $\sigma(s,a)$ and $\sigma(t,a)$ are in the same block of the current partition. (repeated until refinements are possible)
Initial partition (based on output)

$B_1 = \{2, 5\} \quad B_2 = \{0, 1, 3, 4\}$

Consider $B_1$

$\sigma(2, 1) = 3 \quad \sigma(5, 1) = 0$  

0 and 3 are in the same partition

Consider $B_2$

$\sigma(0, 1) = 1 \quad \sigma(1, 1) = 2 \quad \sigma(3, 1) = 4 \quad \sigma(4, 1) = 5$

Refined in $B_2 = \{1, 4\}$  

$B_2 = \{0, 3\}$
Model-based testing

Assumptions (not essential)

- $M_S$ and $M_I$ have an initial state and $M_I$ is in its initial state before we conduct a conformance test.
  - If not, we can apply a homing sequence to $M_I$. The initial state is $s_1$.
- $M_I$ has the same number of state as $M_S$.
  - Faults do not increase the number of states
    - Not included faults that create inconsistent states, such as, race conditions
  - Possible faults then can only be of two types
    - **Output faults**: the transition produces the wrong output
    - **Transfer faults**: the implementation goes to a wrong state.
Example

Faulty implementations

Output fault

Transition faults
Model-based testing

Assumptions (not essential) continues …

- $M_S$ and $M_I$ have a special input \textit{reset} that brings them back to the initial state without producing any output.
  
  • This assumption will be relaxed

- $M_S$ and $M_I$ have a special input \textit{status} to which they respond with an output that uniquely identifies the state in which they are. The state is not changed
  
  • (If in $s_i$, the output is $s_i$)
  
  • This assumption will be relaxed

- $M_S$ and $M_I$ have a special set of inputs set($s_j$), such that when set($s_j$) is received in the initial state, the machines move to $s_j$ without producing any output.
  
  • This assumption will be relaxed
Algorithm for conformance test

Under these assumptions, this is a conformance test

For all $s \in S$, $a \in I$:

1. Apply a reset message to bring $M_1$ to the initial state
2. Apply set(s) message to transfer $M_1$ to state $s$
3. Apply the input value $a$
4. Verify that the output received conforms to $\lambda_S(s,a)$
5. Apply the status message to verify that the final state conforms to $\delta_S(s,a)$
Algorithm for conformance test

• The algorithm should also test the behavior for set, reset and status
• To test status, simply apply it twice in every state $s_i$ after set($s_i$) (first to test that it returns $s_i$, then to check that it does not change the state)
• Once status is tested, we can test set and reset by applying them in every state and verifying the result with status.

• The algorithm is the concatenation of reset, set($s$), a and status $\forall s \in S$ and $\forall a \in I$.
• The length of the sequence is $4pn$ where $p = |I|$ and $n = |S|$
Algorithm for conformance test

- The main problem of the algorithm is the need for the set(s) input, which is typically not available.
- There is a sequence that avoids the need for set and possibly shortens the test run.
- We need a sequence that traverses every state and every transition, without restarting from the initial state after each test (and without using a set). Such a sequence is called Transition Tour (TT).
- A **Transition Tour** is an input sequence $a_1, a_2, a_3, \ldots, a_n$ that takes the machine to a sequence of states $z_1, z_2, z_3, \ldots, z_n$ such that,
  - for all $s \in S$, there exists $z_j = s$ and, (every state is visited)
  - for all $i \in I$ and $s \in S$, there exists $j$ such that $z_j = s$ and $a_j = i$ (every transition out of every state is taken)
Algorithm for conformance test

• If a **Transition Tour** is available, simply perform the input sequence \( a_1, status, a_2, status, a_3, \ldots status, a_n \) to test conformance.
• The length of the TT sequence is at least \( 2*p*n \).
• The shortest path that traverses each transition exactly once is called Euler Tour
• For connected FSMs an Euler Tour exists if they are also symmetric (every state is the source and destination of the same number of transitions)
  – And can be found in time linear in the number of transitions
• For non-symmetric FSMs, finding the shortest tour is another graph theory well-known problem (Chinese postman problem) that can be solved in polynomial time
Example

<table>
<thead>
<tr>
<th>Checking sequence</th>
<th>b</th>
<th>s</th>
<th>a</th>
<th>s</th>
<th>b</th>
<th>s</th>
<th>a</th>
<th>s</th>
<th>b</th>
<th>s</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start state</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Output</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>End state</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example

![State Transition Diagram](image)

<table>
<thead>
<tr>
<th>Checking sequence</th>
<th>b</th>
<th>s</th>
<th>a</th>
<th>s</th>
<th>b</th>
<th>s</th>
<th>a</th>
<th>s</th>
<th>b</th>
<th>s</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start state</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Output</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1 (_{0})</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>End state</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The Transition Tour method

- The TT method without the status message achieves only **transition coverage** (not status coverage)

- A test that visits all the states but not all transitions is a **state tour** and obtains **state coverage**
  - Simple transition coverage is not enough to test correctness!
Example

Check. seq.  |  a  |  b  |  a  |  b  |  a  |  b  
Start state  |  1  |  2  |  2  |  3  |  3  |  1  
Output       |  0  |  1  |  1  |  1  |  0  |  0  
End state    |  2  |  2  |  3  |  3  |  1  |  1  

Check. seq.  |  b  |  a  |  b  |  a  |  b  |  a  
Start state  |  1  |  1(2)|  2  |  2(3)|  3  |  3(1) 
Output       |  0(1)|  0(1)|  1  |  1(0)|  1(0)|  0  
End state    |  1(2)|  2  |  2(3)|  3  |  3(1)|  1  

Transition faults
Using Separating Sequences instead of status

- The status message is replaced by a (set of) sequences called separating sequences.
- Since $M_s$ is minimal, for every pair of states $s_i, s_j$, there exists an input sequence $x$ that distinguishes between them by creating different outputs: $\lambda(s_i, x) \neq \lambda(s_j, x)$
- That is, we need a “signature” that characterizes each state. This “signature” is a behavior starting from the state.
- Let’s reason about those “signatures” … how long should they be?
Separating sequences

- Define a sequence \( \rho_0, \rho_1, \) of partitions, so that two states are in the same class of \( \rho_i \) if and only if they do not have any separating sequence of length \( i \)
  - \( \rho_0 = \{S\} \)
  - \( \rho_{i+1} \) is a refinement of \( \rho_i \)
    - Lemma: if \( \rho_{i+1} = \rho_i \) for some \( i \), then the rest of the sequence of partitions is constant, \( \rho_j = \rho_i \) for all \( j > i \).
- Since partitions can be refined at most \( n \) times, the sequence is constant after at most \( n \) steps.
- Since the machine is minimized, at this point each partition is a singleton

\[ \rho_i \]

\( S_k, S_l \) have the same output for any sequence of length \( i \)
Separating sequences: example

• Counter modulo 4

• Define a sequence \( \rho_0, \rho_1, \) of partitions, so that two states are in the same class of \( \rho_i \) if and only if they do not have any separating sequence of length \( i \)
  • \( \rho_0 = \{S\} \)
  • \( \rho_{i+1} \) is a refinement of \( \rho_i \)

\[
\begin{array}{cccc}
S0 & S2 & S1 & S3 \\
S3 & S2 & S1 & S0 \\
S3 & S1 & S0 & S2 \\
S3 & S1 & S0 & S2
\end{array}
\]
Separating sequences

- Step 1: build the partitions $\rho$
  
  Start from $\rho_1$
  
  - Two states $s$ and $t$ belong to different partitions of $\rho_1$ iff $\exists a \in I$ such that $\lambda(s,a) \neq \lambda(t,a)$
  
  - $\rho_1$ can be computed according to the definition
    - Try all possible input symbols
  
  Iteratively ....
  
  - Two states $s$ and $t$ belong to different partitions of $\rho_i$ with $i>1$ iff $\exists a \in I$ such that $\sigma(s,a)$ and $\sigma(t,a)$ belong to different sets of $\rho_{i-1}$ and to the same set of $\rho_{i-2}$
Separating sequences

- Step 1: build the partitions $\rho$
  
  Start from $\rho_1$
  
  - Two states $s$ and $t$ belong to different partitions of $\rho_1$ iff $\exists a \in I$ such that $\lambda(s, a) \neq \lambda(t, a)$
  
  - $\rho_1$ can be computed according to the definition
    
    - Try all possible input symbols

![Diagram of partitions]$\rho_0$ and $\rho_1$
Separating sequences

Iteratively ….

- Two states $s$ and $t$ belong to different partitions of $\rho_i$ with $i>1$ iff
  $\exists a \in \mathcal{I}$ such that $\sigma(s,a)$ and $\sigma(t,a)$ belong to different sets of $\rho_{i-1}$ and to the same set of $\rho_{i-2}$
Separating sequences

• Step 2: find the separating sequence for \( s, t \in S \)
  – Find the smallest index \( j \) such that \( s \) and \( t \) belong to different sets of \( \rho_j \)
  – Recursively, the separating sequence has the form \( ax \), where \( x \) is the shortest separating sequence for the pair \( \sigma(s,a) \) and \( \sigma(t,a) \)
  – Thus, we need to find the input \( a \) that takes \( s \) and \( t \) to different sets of \( \rho_{j-1} \) and repeat the process until we reach \( \rho_0 \)
  – The concatenation of all the inputs is the separating sequence

  – (the algorithm needs \( O(n) \) memory)
Separating sequences: example

- Step 2: find the separating sequence for $S_1, S_2 \in S$
  - Find the smallest index $j$ such that $s$ and $t$ belong to different sets of $\rho_j$ (2)
  - Thus, we need to find the input $a$ that takes $s$ and $t$ to different sets of $\rho_{j-1}$ and repeat the process until we reach $\rho_0$ ($a=1$)
  - Recursively, the separating sequence has the form $ax$, where $x$ is the shortest separating sequence for the pair $\sigma(s,a)$ and $\sigma(t,a)$ ($x=1$)
  - $ax = 1,1$
Transition cover set

- The **transition cover set** of $M_s$ is a set $P$ of input sequences such that, for every $s \in S$ and $a \in I$ there exists a sequence $x \in P$ ending with the transition that applies $a$ to $s$.
- $P$ is a set closed under prefix selection
  - If $x \in P$ then prefix$(x)$ in $P$ (the empty sequence $\varepsilon$ is assumed to be part of any $P$).
- One way of constructing $P$:
  - Build a testing tree $T$ of $M_s$ (next algorithm) and then take the input sequences of all the partial paths of $T$.

Building a test tree

1. The initial state of $M_s$ is the root (level 1) of $T$.
2. Suppose the tree is built up to level $k$: to build level $k+1$
   1. For all nodes $t$ at level $k$.
   2. If the node $t$ is equal to another node in $T$ at level $j$ with $j \leq k$, then $t$ is a leaf of $T$.
   3. Otherwise, let $s_i$ be the label of $t$. For every input $x$, if $M_s$ goes from $s_i$ to $s_j$, attach a branch to $t$ with label $x$ and a successor node $s_j$. 
Example of transition cover set
Characterizing set

- The **characterizing set** of $M_s$ is a set $W$ of input sequences such that, for every pair $(s_i, s_j) \in S$ there exists a sequence $x \in W$ such that $\lambda(s_i, x) \neq \lambda(s_j, x)$
  - $W$ is also called separating set
  - $x \in W$ are called separating sequences
- The choice of $W$ is not unique, the fewer are the elements in $W$, the longer are the sequences.

Building a $W$ set
1. Partition the states $S$ into blocks $B_i$, $i=1..r$
2. $W \leftarrow \{\}$, $r=1$, $B_1=S$
3. Repeat until every $B_i$ is a singleton (and $r=n$)
   1. Take two states $s, t \in B_i$ and build their separating sequence $x$
      (algorithm shown in previous slides)
   2. Add $x$ to $W$
   3. Partition the states $s_{ik}$ in every $B_j$ into smaller blocks based on their outputs $\lambda(s_{ik}, x)$

Note: there are no more than $n-1$ partitions, and no more than $n-1$ sequences in $W$
Using P sets and W sets (the W method)

• The method consists in using the set W in place of the status message

• Use the set of P sequences to test all transitions.
• At the end of each sequence \( x_P \), apply all the sequences of W \( x_W \).
• Apply a reset after each pair \( x_{Pi}, x_{Wj} \)
• The total number of sequences is given by the cardinality of \( PxW \)
Example

- The set $W$ is simply \{a, b\} (a distinguishes $s_2$ from both $s_1$ and $s_3$, b distinguishes $s_1$ from $s_3$)
- The set $P$ is \{ε, a, b, ba, bb, bba, bbb\}

<table>
<thead>
<tr>
<th>$P$</th>
<th>ε</th>
<th>a</th>
<th>b</th>
<th>ba</th>
<th>bb</th>
<th>bba</th>
<th>bbb</th>
</tr>
</thead>
<tbody>
<tr>
<td>r.P.W</td>
<td>ra</td>
<td>rb</td>
<td>ra</td>
<td>rb</td>
<td>ra</td>
<td>rb</td>
<td>rb</td>
</tr>
<tr>
<td>trans test</td>
<td>$s_1\to(\varepsilon)$</td>
<td>$s_1\to(a/0) s_1$</td>
<td>$s_1\to(b/1) s_2$</td>
<td>$s_2\to(a/1) s_2$</td>
<td>$s_2\to(b/1) s_3$</td>
<td>$s_3\to(a/0) s_3$</td>
<td>$s_3\to(b/0) s_1$</td>
</tr>
<tr>
<td>output</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>$P$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$b$</th>
<th>$ba$</th>
<th>$bb$</th>
<th>$bba$</th>
<th>$bbb$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r.P.W$</td>
<td>ra</td>
<td>rb</td>
<td>raa</td>
<td>rab</td>
<td>rba</td>
<td>rbb</td>
<td>rbaa</td>
</tr>
<tr>
<td>trans test</td>
<td>$s_1 \rightarrow (\varepsilon)$</td>
<td>$s_1 \rightarrow (a/0) s_1$</td>
<td>$s_1 \rightarrow (b/1) s_2$</td>
<td>$s_2 \rightarrow (a/1) s_2$</td>
<td>$s_2 \rightarrow (b/1) s_3$</td>
<td>$s_3 \rightarrow (a/0) s_3$</td>
<td>$s_3 \rightarrow (b/0) s_1$</td>
</tr>
<tr>
<td>output</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>01</td>
<td>11</td>
<td>11</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$P$</th>
<th>$\varepsilon$</th>
<th>$a$</th>
<th>$b$</th>
<th>$ba$</th>
<th>$bb$</th>
<th>$bba$</th>
<th>$bbb$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r.P.W$</td>
<td>ra</td>
<td>rb</td>
<td>raa</td>
<td>rab</td>
<td>rba</td>
<td>rbb</td>
<td>rbaa</td>
</tr>
<tr>
<td>trans test</td>
<td>$s_1 \rightarrow (\varepsilon)$</td>
<td>$s_1 \rightarrow (a/0) s_1$</td>
<td>$s_1 \rightarrow (b/1) s_2$</td>
<td>$s_2 \rightarrow (a/1) s_2$</td>
<td>$s_2 \rightarrow (b/1) s_3$</td>
<td>$s_3 \rightarrow (a/0) s_3$</td>
<td>$s_3 \rightarrow (b/0) s_1$</td>
</tr>
<tr>
<td>output</td>
<td>0</td>
<td>0</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>001</td>
</tr>
</tbody>
</table>
The Wp method

- The partial W or Wp method has the advantage of reducing the length of the test suite wrt the W method.
- The conformance test is split in two phases
  - During the first phase we test that every state that exists in $M_S$ also exists in $M_I$.
  - During the second phase we check that all transitions (not already checked during the first phase) are correctly implemented.
The Wp method

- Phase 1: test that every state that exists in $M_S$ also exists in $M_I$
- The Wp method uses a State cover set (as opposed to a transition cover set) or Q set
- The state cover set is a set Q of input sequences such that for each $s$ in S there exists an input sequence $x$ in Q that takes the machine to $s$, that is $\sigma(s_1, x) = s$
- A Q set can be easily built by performing a breadth-first visit of the transition graph of MS
The Wp method

- Phase 2: check that all transitions (not already checked during the first phase) are correctly implemented.
- For the second phase, the Wp method uses an identification set $W_i$ specific of each state $s_i$ instead of a generic characterizing set $W$ for all states ($W_i \subset W$).
- An identification set of state $s_i$ is a set $W_i$ of input sequences such that, for each state $s_j \in S$, there exists an input sequence $x \in W_i$ such that $\lambda(s_i, x) \neq \lambda(s_j, x)$ and no subset of $W_i$ has this property.
  - $\bigcup W_i = W$
The Wp method

• Phase 1: The input sequences for phase 1 consist in the concatenation of every \( q \in Q \) with every \( w \in W \) after a reset
  – Every state is checked with a \( W \) set

• If the input sequences do not uncover any fault during this phase, we can conclude that every state in \( M_S \) has a similar state in the implementation (produces the same output for all the sequences in \( W \))

• This is not sufficient to prove that it is equivalent
  – (we need to check all transitions in the next stage)
The Wp method

- Phase 2: To test all transitions, Wp uses the identification sets.
- For every transition from $s_j$ to $s_i$ on input $a$, we apply a sequence $x$ (after reset) that takes the machine to $s_j$ along transitions already verified in phase 1.
- Then we apply the input $a$ that takes the machine to $s_i$ we verify the correctness of the output, and we apply one identification sequence of $W_i$.
- We repeat the previous for all the sequences in $W_i$ and, if these tests do not uncover faults, by applying them to every transition, we can verify that $M_I$ conforms to the specification.
Bibliography

Taken from


Chapter 4: Conformance Testing by Angelo Gargantini
Chapter 1: Homing and Synchronizing Sequences, by Sven Sandberg