FSM-based test derivation methods: From TAROT-1 to TAROT-12

Nina Yevtushenko, Tomsk State University, Russia
nyevtush@gmail.com
TAROT (Training And Research On Testing) is a Marie Curie Research Training Network (MCRTN).

It focuses on the protocols, services and systems testing, that is an essential but empirical and neglected domain of validation and Quality of Service (QoS).

Then the TAROT network aims to strengthen and develop the collaboration among major European testing communities.

Moreover TAROT will promote testing in education, research, software engineering and industry.

In order to achieve this objective, the participants will provide training courses, including Ph.D. programs and summer schools.

In addition, workshops will be organized, thanks to which the TAROT network will communicate its results, and maybe find other partners.

Ana Cavalli, coordinator of TAROT
TAROT 2005

TAROT -1 has been held in Paris in 2005

Was an event of big success

Participants agreed to have the annual Summer TAROT School

It is the 12\textsuperscript{th} Summer TAROT School now

At each Summer school a lot of attention has been paid to test derivation based on transition models and this School inherits this tradition
Outline

- *FSM based test derivation: Why FSMs?*
- *Test models for FSMs*
  - White box
  - Black box: W-methods and its derivatives
  - Grey box
- *Deriving tests for complete deterministic FSMs*
  - Initialized FSMs: W-method and its derivatives
  - Non-initialized FSMs: Checking sequences
- *Partial and nondeterministic FSMs: Reducing the complexity of test derivation*
  - Adaptive testing
  - Using appropriate projections
- *Extended and Timed FSMs*
- *Conclusions*
A fragment of C code

```c
...  
{    
  unsigned char n1, n2, v;  
  //initialize n1, n2  
  v = n1 + n2;  
  return v;  
}
```

Is this code safe?

How to check that $v = n1 + n2$ is not bigger than 255?

Otherwise, the result will be wrong

$150 + 150 = 300 \pmod{256} = 44$
Conformance testing

```c
int f(int *a, int size_a)
{
    int i, m;
    i = 0;
    m = a[0];
    while(i < size_a)
    {
        if(m < a[i]) m = a[i];
        i++;
    }
    return m;
}
```

The function returns the maximal integer in the array \( a \) where \( size_a \) is the dimension of \( a \)

How to check that the function is correctly implemented?

How many arrays should be checked?

Is it enough to check all the arrays of dimension 3?
Hardware testing (shift register)

There is no link

How to check?

It is not enough to apply all input sequences of length 3

An input sequence 1*** of length > 3 has to be used

How to check this fact?

Starts at 0000
Model based test derivation

• Solution: to use transition systems as formal models for deriving tests

**Question**: What can be applied and what can be observed

We assume that

• Inputs can be applied
• Output actions can be observed
• A system moves from state to state under inputs and produces outputs
• States cannot be observed
Conformance Testing

Conformance Relation

Spec

Test Derivation

Test Cases (Test Suite)

IUT (Imp)

Apply to

Observed Output

Expected Output

Expected = Observed

FAIL

Yes

Pass

No
Finite automata and FSMs: why

FSMs

I/O automata

Advantages
• Can have infinite number of states, inputs and outputs
• Each transition corresponds to an input or an output or to a non-observable action, i.e., an output can be produced to a sequence of inputs
• A complete test suite is derived from a complete successor tree

Disadvantages
• Complete tests are infinite while testing time is finite
• Still there is a problem with distinguishing sequences when Imps are explicitly enumerated
• Races between inputs and outputs

FSMs

Advantages
• Finite number of states, inputs and outputs
• Each transition corresponds to a pair ‘input/output’
• No non-observable actions
• A complete test is derived with respect to a given fault model

Disadvantages
• Finite tests with the guaranteed fault coverage
• Good background for deriving distinguishing sequences
• No races between inputs and outputs: next input is applied after receiving the output to the previous input

In both cases, IUT is input enabled

12th TAROT Summer School
Limiting the number of Imp states

! All faulty Imps within and possibly much more are detected

Will be detected with a complete test suite

All possible implementations
FSM based test derivation

Extract:
- A Formal FSM Specification $Spec$ (requirements) of the System
- Formally describe a set of faulty implementations

Derive a finite set of finite input sequences ($Test Suite$) such that after applying them to IUT we can guarantee that $Imp$ conforms to $Spec$

- **Conforms:** has many definitions depending on the Formal Specification
Fault model in Conformance Testing

\[ <\text{Spec}, \mathcal{R}, \text{FD}> \]

Formal Specification

Conformance relation

Fault Domain, i.e.

All Faulty Implementations (explicitly or implicitly described)

Guaranteed Fault Coverage:

A complete test suite w.r.t. \( <\text{Spec}, \mathcal{R}, \text{FD}> \) has to detect each \( \text{Imp} \in \text{FD} \) such that \( \text{Imp} \) does not conform (i.e., not equivalent, not reduction, etc) to \( \text{Spec} \)
FSM Model in Conformance Testing

< Spec, $\mathcal{R}$, FD >

FSM Specification

Fault Domain, i.g.,

Conformance relation

e.g., Equivalence ($\simeq$), Reduction ($\leq$), etc

FSMs which describe all possible Imp

Guaranteed Fault Coverage:

A complete test suite w.r.t. <Spec, $\mathcal{R}$, FD> has to detect each FSM Imp $\in$ FD such that Imp does not conform (i.e., not equivalent, not reduction, etc) to Spec
FSMs (Finite State Machines)

Fault models for initialized complete deterministic FSMs

Complete test suites

Fault models for non-initialized complete deterministic FSMs

Checking sequences
Finite State Machine (FSM)

\[ S = (S, I, O, h_S) \] is an FSM

- \( S \) is a finite nonempty set of states with the initial state \( s_0 \)
- \( I \) and \( O \) are finite input and output alphabets
- \( h_S \subseteq S \times I \times O \times S \) is a behavior relation
FSM $\mathcal{S} = (S, I, O, h_\mathcal{S})$ can be

- **deterministic** if for each pair $(s, i) \in S \times I$ there exists at most one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_\mathcal{S}$
  otherwise, $\mathcal{S}$ is **nondeterministic**

- **complete** if for each pair $(s, i) \in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_\mathcal{S}$
  otherwise, $\mathcal{S}$ is **partial**

- **initialized** if there is the initial state $s_1$ otherwise, otherwise, $\mathcal{S}$ is **non-initialized**

This one is non-initialized, complete and deterministic
One of FSMs for PAP (Password Authentication Protocol)

RAR\(^+\) - «good» login
RAR\(^-\) - «bad» login
SAA - Ack
SAN – Nack

![Flowchart diagram](attachment:image.png)
Complete deterministic FSMs

Deterministic complete FSM is a 5-tuple \((S, I, O, \delta_S, \lambda_S)\)

- \(S\) is a finite set of states with the initial state \(s_1\)
- \(I\) is a finite non-empty set of inputs
- \(O\) is a finite non-empty set of outputs

- Transition function \(\delta_S(s, i)\)
- Output function \(\lambda_S(s, i)\)

\((s, i, o, s')\) is a transition from state \(s\) under input \(i\) to state \(s'\) with the output \(o\) if \(\delta_S(s, i) = s'\) and \(\lambda_S(s, i) = o\)

\(! At each state for each input sequence there is a single output sequence!\)
Equivalence relation between initialized complete deterministic FSMs

FSMs \textit{Imp} and \textit{Spec} are \textit{equivalent} if their output responses to each input sequence coincide.

Caution: Number of input sequences is infinite, while we can apply only finite number of input sequences when testing the conformance.

Equivalent FSMs have the same set of traces:

\begin{align*}
\text{Spec} & : s_1 \ldots s_n \\
\text{Imp} & : t_1 \ldots t_m
\end{align*}
Reduced FSM

A complete deterministic FSM is *reduced* if every two different states are not equivalent.

 FSM is reduced
 Separating sequences:
 \( \gamma(s1, s2) = x \)
 \( \gamma(s2, s3) = y \)
 \( \gamma(s1, s3) = z \)

For each deterministic complete FSM there exists a reduced FSM with the same Input/Output behavior, i.e., a reduced FSM with the same set of traces.

Conclusion: we can consider only reduced specification FSMs.
Test derivation for initialized FSMs

Fault model - \(<\textit{Spec}, \equiv, FD>\)

\textit{Spec} is a complete deterministic reduced FSM

\textit{FD} – fault domain that contains complete deterministic FSMs, possibly with more states

\[\downarrow\]

- \textit{Output} faults
- \textit{Transfer} faults
- Implementation has \textit{more} states and transitions

! Reliable reset is assumed
Fault model

\(< Spec, \cong , FD >\)

\(Spec\) – the initialized specification FSM with \(n\) states

\(!\text{ Usually } Spec\) is a complete deterministic reduced FSM

\(FD\) is the fault domain that contains each FSM that describes each possible IUT that is complete and deterministic

Equivalent FSMs have the same set of traces
A test case is a finite input sequence of the specification FSM Spec. A test suite is a finite set of test cases.

We assume that each implementation FSM Imp has a reliable reset \( r \) that takes the Imp from each state to the initial state.

Each test case in the test suite is headed by \( r \), i.e. is applied to Imp at the initial state.
Complete test suite

Fault domain $FD$ - the set of FSMs that describe all possible faults when implementing the specification:

$$FD = \{Imp_1, \ldots, Imp_n, \ldots\}$$

A test suite $TS$ is complete w.r.t. $FD$ if $TS$ detects each FSM $Imp \in FD$ that is not equivalent to Spec

If the fault domain contains each FSM over alphabets $I$ and $O$ and $Spec$ is complete and deterministic then there is no complete test suite w.r.t. such fault domain
**Example**

**Inverter**

FSM Spec with a single state

- Complete test when $Imp$ has a single state
  - $\{01\}$ or $\{10\}$

- Complete test when $Imp$ has at most two states
  - $\{01, 10, 00, 11\}$

  ! Nothing can be deleted

**Conclusion:** a complete test significantly depends on the number of states of $Imp$
Test architecture

Conformance relation – the *equivalence*
Deriving FSM based tests

*Test assumptions*

• We can ‘build’ a complete deterministic FSM that simulates a faulty implementation
• There can be faults of three types:
  - Transition faults
  - Output faults
  - New faulty transitions can be added
• When testing we can only apply input sequences and observe output sequences

! Sometimes states also can be observed but we do not discuss such testing
FSM based test models

- White box (explicit enumeration)
- Black box (the IUT structure is unknown: possibly the upper bound on the number of the IUT states is available)
- Grey box (the IUT structure is partly available)
Explicit enumeration (white box testing)

Explicit enumeration can be used when the number of mutants of $Spec$ is not big.

Faults are explicitly enumerated.

**Advantage**: Easy to implement

**Disadvantage**: Cannot be applied when the number of faults (the number of mutants) is huge.

Check whether $Spec$ and $Imp$ are equivalent.

$$Spec \cap Imp$$

If $Spec \cap Imp$ is not complete then derive a distinguishing sequence (a test case that kills a faulty implementation $Imp$).

Methods for deriving distinguishing sequences for two deterministic FSMs are well elaborated.
Distinguishing sequences for two FSMs

If $Spec \cap Imp$ is not complete
    then derive an input sequence $\alpha$ to reach a state with an undefined input $i$

The sequence $\alpha$ is a *distinguishing* sequence

If $Spec$ has $n$ states while $Imp$ has $m$ states then
    the length of $\alpha$ is at most $m + n - 1$ (despite the fact that the product $Spec \cap Imp$ can have up to $mn$ states)

! Other methods for deriving a distinguishing sequence can be used
Black box testing

- An implementation FSM under test is not known

- Tests are derived based on the specification FSM

**Question:** What can be guaranteed in this case?

**Reply:** If nothing is known about the FD then a complete test suite cannot be derived (Moore, 1956, Gill, 1964)

The set FD should be finite and the weakest assumption is that the upper bound on the number of states of an implementation FSM is known.
Most popular test derivation methods for black box testing

• Transition tour (guaranteed killing output faults)

**Transition tour** is a set of input sequences that traverse each transition of the specification FSM

• W-method and its derivatives (guaranteed killing output and transfer faults)
One of FSMs for PAP

RAR^+ - «good» login
RAR^- - «bad» login
SAA - Ack
SAN – Nack
Transition tour for the PAP model

Test suite:
RAR⁺
RAR⁻RAR⁻RAR⁻

Expected output reactions:
SAA
SAN SAN SAN SAN
Detecting an output fault

Test suite:
RAR⁺
RAR⁻RAR⁻RAR⁻

Expected:
SAA
SAN SAN SAN

Observed:
SAA
SAN SAA SAN
Trying to detect a transfer fault

Test suite:
RAR⁺
RAR⁻RAR⁻RAR⁻RAR⁻

Expected:
SAA
SAN SAN SAN SAN

Observed:
SAA
SAN SAN SAN SAN

A transition fault is not necessary detected by a transition tour!!!
Black box testing (guaranteed killing transfer faults)

- Most methods for detecting transfer faults in initialized complete deterministic FSMs are based on W-method
- Spec is a complete deterministic reduced FSM with $n$ states
- The upper bound $m$ on the number of states of an implementation FSM is known
- The fault models $<S, \cong, S_n>$ or $<S, \cong, S_m>$, $m \geq n$
Time-line for W-method and its derivatives

- **1973**: The idea behind the W-method
- **1978**: W-method
- **1990**: UIO-method
- **1991**: HIS-method
- **2004**: Wp-method
- **2005**: H-method
- **2009**: SPY-method
Isomorphic FSMs

Two FSMs $Spec$ and $Imp$ are isomorphic iff

1. There exists one-to-one $\Psi: T \rightarrow S$ between states, $\Psi(t_1) = s_1$

2. The same $\Psi$ is kept between transitions $\lambda_{Imp}(t, i) = \lambda_{Spec}(\Psi(t), i)$ and $\Psi(\delta_{Imp}(t, i)) = \delta_{Spec}(\Psi(t), i)$

$Spec$ and $Imp$ have the same number of states.
Test suite derivation for detecting transfer faults \( (m = n) \)

Two states \( s_j \) and \( s_k \) of the specification FSM are equivalent if the FSM has the same output response at states \( s_j \) and \( s_k \) to each input sequence.

**Proposition.** Given complete deterministic reduced specification FSM \( Spec \) and a complete deterministic implementation FSMs with the same number of states, \( Spec \) and \( Imp \) are equivalent iff \( Imp \) is isomorphic to \( Spec \).
How to check if an implementation is isomorphic to $Spec$

1. To assure that a given implementation $Imp$ has $n$ states

2. To assure that for each transition of $Spec$ there exists a corresponding transition in the FSM $Imp$

! We forget about the infinite set of input sequences and check finite number of transitions
Reduced FSM

Given a complete deterministic reduced FSM, for every two different states there exists a sequence that distinguishes these states (separating sequence)

 FSM is reduced
 Separating sequences:
\( \gamma(s_1, s_2) = x \)
\( \gamma(s_2, s_3) = y \)
\( \gamma(s_1, s_3) = z \)

For each deterministic complete FSM there exists a reduced FSM with the same Input/Output behavior, i.e. a reduced FSM with the same set of traces
Conclusion: we can consider only reduced specification FSMs
Separating sequences

As we do not directly observe states of Imp, we use separating sequences to draw some conclusions.

States $s_j$ and $s_k$ of Spec are separated by input sequence $\alpha$ if Spec has different output responses at $s_j$ and $s_k$ to $\alpha$.

If Imp produces different outputs to $\alpha$ then Imp is at two different states $t_j$ and $t_k$ when is applied

$... t_j\alpha/\beta_1 \; ... \; tk\alpha/\beta_2 \; ...$
When testing against FSMs ...

1) Reaching each FSM state \( s \)

2) Distinguishing \( s \) from any other FSM state

3) Traversing each transition to check the output and final state

- 1) can be solved via an application of a transfer sequence
- 2) can be solved via an application of a separating sequence
W-method \((m = n)\)

1. For each two states \(s_j\) and \(s_k\) of the specification FSM \(Spec\) derive a distinguishing sequence \(\gamma_{jk}\) Gather all the sequences into a set \(W\) that is called a \textit{distinguishability set}

2. For each state \(s_j\) of the FSM \(Spec\) derive an input sequence that takes the FSM \(Spec\) to state \(s_j\) from the initial state Gather all the sequences into a set \(CS\) that is called a \textit{state cover set}
W-method (2)

3. Concatenate each sequence of the state cover set $V$ with the distinguishability set $W$: $TS_1 = V.W$

4. Concatenate each sequence of the state cover set $V$ with the set $iW$ for each input $i$: $TS_2 = V.I.W$

! The shortest test suites are derived when FSM has a distinguishing sequence

W-method (3)

4. Concatenate each sequence of the state cover set \( V \) with the set \( iW \) for each input \( i \): \( TS_2 = V.i.W \)

Proposition. If an implementation FSM \( Imp \) that passed \( TS_1 \) passes also \( TS_2 \) then one-to-one mapping \( \Psi \) satisfies the property:

\[
\lambda_{Imp}(t, i) = \lambda_{Spec}(\Psi(t), i) \land \Psi(\delta_{Imp}(t, i)) = \delta_{Spec}(\Psi(t), i)
\]

i.e., FSM \( Imp \) is isomorphic, and thus, is equivalent to \( Spec \)
W-method (4)

Test suite returned by W-method

All the sequences that are prefixes of other sequences can be deleted from a complete test suite without loss of its completeness
W-method (5)

When a state cover $V$ is prefix closed, while the distinguishability set $W$ is suffix closed, the set $V.I.W$

is a complete test suite for the case when the IUT has not more states than the specification
Example

FSM with three states

State identification

Output to $i_1i_1$

1: 00
2: 01
3: 10
Example (2)

Spec

Complete test suite
Experimental results for W-method

<table>
<thead>
<tr>
<th>State num.</th>
<th>Input num.</th>
<th>Output num.</th>
<th>Trans. num.</th>
<th>Average length</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>6</td>
<td>6</td>
<td>180</td>
<td>2545</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>10</td>
<td>300</td>
<td>3393</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>6</td>
<td>300</td>
<td>5203</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>10</td>
<td>500</td>
<td>6773</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>10</td>
<td>1000</td>
<td>17204</td>
</tr>
</tbody>
</table>
Experimental results (conclusion)

Theoretically:
Length is $O(kn^3)$ where
$k$ – number of inputs
$n$ - number of states

Experiments show:
- tests are much shorter than corresponding theoretical upper bounds
- test suites are fast generated (compared with explicit enumeration)

STILL LONG ENOUGH
Conclusions:
1. The set $V/I$ is presented in each complete test suite (each transition at each state must be traversed)
2. The length of a complete test suite significantly depends how states are identified, i.e., on the choice of state identifiers.
Modifications of W-method

1. DS-method
2. UIO-method
3. Wp-method
4. UIOv-method
5. HSI-method

Depending how a set of separating sequences is defined

! H-method allows to identify states with separating sequences derived on-the-fly
! SPY method allows to check transitions after different transfer sequences of a state cover set
H- and SPY-methods

- **H-method**
  Allows to use different state identifiers when checking different transitions

**Conclusion:** State identifiers can be derived on the fly

- **SPY-method**
  Allows to use different input sequence when reaching a state where a transition is checked

**Conclusion:** Transfer sequences can be derived on the fly

Still there are no necessary and sufficient conditions for a test suite to be complete
Using different state identifiers in $H$-method

$W_2 = \{y\}$, $W_3 = \{x\}$ but $H_2 = \{x, y\}$, $H_3 = \{x, y\}$
H-method (illustration)

Spec

HIS-method

H-method

$L = 41$

$L = 25$
SPY-method (illustration)

HSI-method

SPY-method

$L = 41$

$L = 26$
## Experimental results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>6</td>
<td>6</td>
<td>180</td>
<td>1626</td>
<td>1105</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>10</td>
<td>300</td>
<td>2175</td>
<td>1568</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>6</td>
<td>300</td>
<td>3261</td>
<td>2142</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
<td>10</td>
<td>500</td>
<td>4305</td>
<td>2852</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>10</td>
<td>1000</td>
<td>10503</td>
<td>6880</td>
</tr>
</tbody>
</table>
Conclusions

1. As it is known, the DS-method returns shortest test suites
   But: less than 10% of specifications possess a DS

2. H- and SPY- methods return tests that are comparable with those returned by DS-method
   And: can be applied to any reduced (partial or complete) specification

3. The test quality is very good

4. Test suites returned by all above methods are still too long for real systems: the abstraction level should be carefully chosen
A number of protocols have been considered

- SCP
- POP3
- Time
- TCP
- ...

Java implementation of each protocol has been developed and the μjava tool has been used for the mutant derivation.

All the tests returned by HIS method detect 100% of implementation faults injected by the μjava tool.

The ratio between test suite length returned by different methods is almost the same as for randomly generated FSMs.
Faults can increase the number of states of an implementation FSM

Faulty implementation can have more states than the specification

\[ m \text{ – number of states of } \text{Imp} \]

\[ n \text{ – number of states of } \text{Spec} \]

\[ m > n \]

• Fault model \(<S, \equiv, \mathcal{F}_m>\)

A single transfer fault in the specification EFSM of a Simple Connection Protocol (SCP) can transform the corresponding FSM into an FSM with more states
W - method and its modifications

1. State cover set $V$ is augmented with all input sequences of length $m - n$

2. State identifiers are applied according to a given method

! The length of a test suite becomes exponential w.r.t. the number of Spec inputs

!! Experiments show almost the same relationship between length of test suites returned by different modifications of W - method
Publications


Minimizing FSM-based tests for conformance testing

The test quality is very good

BUT

Test suites returned by all above methods are too long

**Question:** how to shorten test suites, preserve some fault coverage without explicit enumeration of faulty FSMs

**Solution:** to consider user-driven faults
How to reduce the length of a test suite

**Solution:** To partition the set of transitions of the specification FSM into clusters and check only transitions of one cluster at each step

↓

* Incremental testing or testing *user-driven faults*

Experimental results are very promising especially for the case when faults can increase the number of states of the specification
Incremental testing or user-driven faults

Only some transitions should be checked

An implementation is assumed to be known up to the transitions that should be checked

Other transitions are not changed
Fault model for incremental testing

Fault model - \(<\text{Spec}, \cong, \text{Sub}(\text{MM})>\)

\text{Spec} is a complete deterministic specification FSM

\text{MM} is a mutation (nondeterministic FSM) where unmodified transitions are as in the specification while modified transitions are chaos transitions

! A bit more tricky when \(m > n\) but this is enough for today lecture
Fault domain for incremental testing

(2)

Initial $Spec_{in}$

$s1 \rightarrow s2$

$x/1$

$x/0$

Initial $Imp_{in}$

$t1 \rightarrow t2$

$x/1$

$x/0$

Modified $Spec$

$s1 \leftrightarrow s2$

$x/1$

$x/0$

Possible implementations

$t1 \rightarrow t2$

$x/1$

$t1 \rightarrow t2$

$x/1$

$t1 \leftarrow t2$

$x$

$t1 \rightarrow t2$

$x/0$

$t1 \rightarrow t2$

$x/1$

$t1 \rightarrow t2$

$x/1$

$t1 \rightarrow t2$

$x/1$

$t1 \rightarrow t2$

$x/1$
**Complete test suite**

*Incremental complete test suite* has to detect each nonconforming implementation where all unmodified specification transitions are known

The fault domain has the finite number of FSMs

\[ FD = \{Imp_1, \ldots, Imp_k\} \]

Number of mutant FSMs = \((n \cdot p)^t\)

\(n\) – number of states, \(p\) – number of outputs, \(t\) – number of modified transitions
When is it enough to check only modified transitions?

1. When the final state of each modified transition has a state identifier in the unmodified part of the modified Spec

2. When each modified transition is reachable through unmodified transitions in the modified Spec

*Solution: to derive partitions in order to satisfy the above properties*
Final state of each modified transition has a state identifier in the unmodified part

Example: add two new transitions

Only modified transitions are tested

\[ TS = \{ r.x.x.yy, r.xx.x.yy \} \]

\[ \text{Compare: } HSI\_\text{length} = 25 \]

If the whole \textit{Imp} is tested
All states are reachable through unmodified transitions

Example

State $s_3$ has no state identifier in the unmodified part but each state is reachable through unmodified transitions. $yy$ is a DS.

Only modified transitions are tested

Compare: length = 15
HSI_length = 25
General procedure

1. For each state that is reachable via unmodified transitions identify the state and check only modified transitions from this state
2. For each state that has a state identifier in the unmodified part identify the state (if reachable via modified transitions) and check modified transitions
3. For all other states, identify the state and check each outgoing transition
4. Delete sequences that do not traverse modified transitions

Step 3 can be improved
## Experimental results

<table>
<thead>
<tr>
<th>s</th>
<th>i</th>
<th>HSI length</th>
<th>0-5% modif</th>
<th>5-10% modif</th>
<th>10-15% modif</th>
<th>15-20% modif</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10</td>
<td>2992</td>
<td>93</td>
<td>337</td>
<td>490</td>
<td>785</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>5818</td>
<td>148</td>
<td>477</td>
<td>999</td>
<td>1513</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>5333</td>
<td>135</td>
<td>518</td>
<td>957</td>
<td>1450</td>
</tr>
<tr>
<td>35</td>
<td>10</td>
<td>6588</td>
<td>148</td>
<td>539</td>
<td>1013</td>
<td>1537</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>3737</td>
<td>89</td>
<td>345</td>
<td>636</td>
<td>887</td>
</tr>
</tbody>
</table>
Experimental results (2)

Ratio $H = \frac{\text{HSI\_length}}{\text{IncrTest\_length}}$

<table>
<thead>
<tr>
<th></th>
<th>0-5 % modif</th>
<th>5-10 % modif</th>
<th>10-15 % modif</th>
<th>15-20 % modif</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>36.0</td>
<td>11.3</td>
<td>6.1</td>
<td>4.0</td>
</tr>
</tbody>
</table>

The ratio slightly increases when the number of transitions increases
Implementation can have more states than the specification

A faulty implementation can have more states than the specification

\[ m > n \]
**State cover of Imp**

**Question:** As a modified Imp inherits some transitions from the Spec, possibly there exists a shorter set than $V.\text{Pref}(I^{m-n})$ that is a state cover set of each possible Imp?

**Reply:** Yes, a state cover set $V.\text{Pref}(I^{m-n})$ can be reduced
## Experimental results

<table>
<thead>
<tr>
<th>$n$ (Spec)</th>
<th>$m$ (Imp)</th>
<th>Input_num</th>
<th>Modif %</th>
<th>Incr_length</th>
<th>HSI_length</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>21</td>
<td>4</td>
<td>30</td>
<td>343</td>
<td>3773</td>
</tr>
<tr>
<td>20</td>
<td>22</td>
<td>4</td>
<td>20</td>
<td>339</td>
<td>17238</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>8</td>
<td>30</td>
<td>1014</td>
<td>?</td>
</tr>
<tr>
<td>40</td>
<td>42</td>
<td>8</td>
<td>30</td>
<td>1060</td>
<td>?</td>
</tr>
</tbody>
</table>
Conclusions

Incremental test derivation methods return much shorter test suites

Future work (for example):
Based on incremental testing methods to derive a test suite that detects single and double output/transition faults of Spec
Publications


Testing non-initialized FSMs

No reliable reset
or
The reset is very expensive
Finite State Machine (FSM)

$S = (S, I, O, h_S)$ is an FSM

- $S$ is a finite nonempty set of states with the initial state $s_0$
- $I$ and $O$ are finite input and output alphabets
- $h_S \subseteq S \times I \times O \times S$ is a behavior relation

Two complete non-initialized FSMs are equivalent if for each state of one machine there is an equivalent state in another machine.
Checking sequences [Hennie64]

• Non-initialized FSMs
• The fault model $<\text{Spec}, \cong, \subseteq_n>$ where Spec is a reduced strongly connected complete deterministic FSM that has a distinguishing sequence

An input sequence $\alpha$ is a checking sequence if for each FSM Imp with at most $n$ states that is not equivalent to Spec, Spec and Imp have different output responses to $\alpha$

$\alpha$ separates (distinguishes) Spec from any non-equivalent FSM with at most $n$ states
The method for deriving a checking sequence is the same: to reach each state and to traverse each transition; states are identified using a distinguishing sequence.

It is much harder to reach a state without a reliable reset.

The length of a distinguishing (separating) sequence (if it exists) is exponential w.r.t the number of states of the specification FSM.
How to decrease the complexity?

Switching from preset to adaptive test derivation strategy
Research groups of M. Yannakakis, R. Hierons, H. Yenigün, A. Simão, A. Petrenko, N. Yevtushenko,

Providing effective heuristics
Research groups of A. Zakrevskiy, H. Yenigün, R. Brayton, A. Cavalli
Adaptive testing for FSMs

Next input depends on the responses to previous inputs

Next input depends on the output to previous inputs
The length of adaptive checking sequence is less than the length of preset sequences

**Conclusion:** adaptive checking sequences are shorter than preset


Conclusions

- FSMs are useful for deriving high quality test suites; however, as FSM specifications have many states, tests are too long.
- The problem is how to extract FSM from an informal specification.
- Usually an extracted FSM is partial and non-deterministic.
Non-classical FSMs

Unfortunately, FSMs extracted from real systems are not complete and deterministic

• Partial deterministic
• Complete non-deterministic
• Partial non-deterministic
• Non-observable

How to derive tests?
Partial specification

1. *Spec* can be **partially specified**;

*Imp* is a **complete** FSM

2. To complete *Spec* adding loops for undefined transitions with output ‘IGNORE’.

3. *Imp* conforms to *Spec* iff *Imp* is quasi-equivalent to *Spec*, i.e., has the same behavior for defined input sequences
A complete FSM Imp is \textit{quasi-equivalent} to Spec if their output responses coincide for each input sequence that is defined in the Spec.
W-, Wp-, UIOv-methods cannot be used

W-, Wp, UIOv- methods cannot be generally used as not each partial FSM has the distinguishability set W

Distinguishability set does not necessary exist

HIS, H, SPY still can be applied,
Moreover, Spec is not required to be reduced
Non-deterministic FSMs (NFSMs)

Tabular Representation of a NFSM

- **States**: \{a, b\}
- **Inputs**: \{x, y\}
- **Outputs**: \{0, 1, 2, 3\}

<table>
<thead>
<tr>
<th>Input/state</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>a / 0,1,2,3</td>
<td>a / 1,2</td>
</tr>
<tr>
<td>y</td>
<td>b / 1,2</td>
<td>a / 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b / 3</td>
</tr>
</tbody>
</table>

At state **a** under the input **x**, we have four transitions:

(a, x, 0, a), (a, x, 1, a), (a, x, 2, a), (a, x, 3, a)
Why non-determinism?

• For example, when we have limited Controllability or Observability as in Remote Testing

• Due to the optionality

• Due to the abstraction level

• ...

12th TAROT Summer School
At state $a$, for input trace $x$, output traces:
\[ \text{out}(a, x) = \{0, 1, 2, 3\} \]

At state $a$, for input trace $x.y$, output traces are:
\[ \text{out}(a, x.y) = \{0.1, 0.1, 1.1, 1.2, 2.1, 2.2, 3.1, 3.2\} \]

(I/O)Traces of an FSM: all I/O sequences that can be derived from the initial state of the FSM
More Coformance Relations Between nondeterministic FSMs

• FSMs $P$ and $S$ are **indistinguishable** if
  \[
  \forall \alpha \in I^* \ (out_P(p_1, \alpha) = out_S(s_1, \alpha))
  \]

• FSMs $P$ and $S$ are **non-separable** if
  \[
  \forall \alpha \in I^* \ (out_P(p_1, \alpha) \cap out_S(s_1, \alpha) \neq \emptyset)
  \]

• FSMs $P$ and $S$ are **r-compatible** if there exists
  a complete FSM is a reduction of both FSMs, $P$ and $S$

! There are methods for deriving complete test suites w.r.t. various conformance relations for NFSMs

!! Sometimes all-weather-conditions have to be used
IRC protocol

[RF2812]
Inconsistencies detected

- Wrong code reply to the command *NICK* with the empty parameter (without nickname)

- Wrong server processing when using already occupied nickname

- Command MODE is wrongly processed

PASS(2)/NULL NICK(1)/{431}
PASS(2)/NULL NICK(3)/NULL USER(3,0,5)/001 NICK(3)/{433}
PASS(2)/NULL NICK(3)/NULL USER(3,0,5)/001 MODE(1,7)/{461}


Complexity problems for nondeterministic FSMs
Some primitive complexity into...

Time

...This is what it counts for an algorithm $A$...

$n$ is the size of the input of a problem $\mathcal{P}$

1) **Time** – can be considered as the number of primitive operations, in the worst case, to solve the problem

// number of transitions of the corresponding Turing machine

2) **Space** – can be considered as the size of memory to be used, in the worst case, to solve the problem

// the length of a tape in use of the corresponding Turing machine
What is good and what is bad?

When the time is polynomial
• There exists an algorithm that solves the problem in a polynomial time
• The problem is in P then

When the time is not polynomial
• Maybe, there exists an algorithm that verifies the solution in a polynomial time? Then the problem is in NP
• Or maybe there exists an algorithm that solves the problem using a polynomial space?
Then the problem is in PSPACE

! P is good, for small degrees of the polynomials 😊
NP and PSPACE – not really
Bad... very bad ‘news’

*Most of the problems in Model based testing are PSPACE-complete*

In particular...

The problem of checking the existence of a distinguishing sequence for complete deterministic FSMs

The problem of checking the existence of a distinguishing sequence for complete nondeterministic FSMs

The problem of checking the existence of a homing / synchronizing sequence for complete non-reduced (non-)deterministic FSMs

*Test sequences and checking sequences are somewhat hard to derive…*
How to decrease the complexity?

Utilizing scalable representations allows to ‘hide’ the complexity
Research groups of R. Brayton, R. Jiang, A. Mischenko, T. Villa, J. Tretmans, V. Kunz, H. Yenigün

Considering specific types of bugs in the software, i.e., specific fault models
Research groups of J. Offut, F. Wotawa, N. Yevtushenko

Switching from preset to adaptive test derivation strategy
Research groups of M. Yannakakis, N. Yevtushenko, A. Petrenko, A. Simão, R. Hierons

Providing effective heuristics
Research groups of A. Zakrevskiy, H. Yenigün, R. Brayton, A. Cavalli, A. Simão
How to decrease the complexity (2)?

Simplifying a derivation of test sequences

1) **Using scalable representations**
   Logic circuits, for example?

2) **Considering proper FSM classes**
   1-distinguishing, merging free,…

3) **Developing effective heuristics**
   Check if a given FSM has a submachine with ‘good’ transfer and distinguishing properties

4) **Switching from preset to adaptive test derivation strategy**
   Already saw that this can help when deriving checking sequences even for deterministic FSMs

…

*Each of the above is good for appropriate FSM classes*
Conclusions

• **Theoretically:** almost all the problems in software testing that provide the guaranteed fault coverage have terrible (exponential or more!!!) complexity

• **Practically:** methods and tools for decreasing the complexity seem to be promising

\[ \Downarrow \]

*New models (or new heuristics) need to appear and new methods and tools need to be provided to decrease the complexity*

\[ \Downarrow \]

*We do have something for the future work 😊*
Working together with

Original results presented here were obtained in collaboration with research groups lead by

Prof. Ana Cavalli (and scientific group under her supervision)
Prof. Khaled El-Fakih
Prof. A. Petrenko (Canada and Russia 😊)
Prof. Ades Simão
Prof. H. Yenigün
PhD Natalia Kushik
Scientific group of Tomsk State University
Thank you!