Fault Model-Based Testing from State-Oriented Models

Alexandre Petrenko
Computer Research Institute of Montreal (CRIM), Canada

HSST 2016, Halmstadt, Sweden
To Model or Not to Model, That is the Question

Testers become test modellers

“Palais Idéal” is still built by many without any modelling
Given Test Model, What To Cover By Tests?

- Usual model/specification coverage
  - Executions
  - Structural elements
- Testing can be used to show the presence of bugs, but never to show their absence [Dijkstra]
- Yet can testing at least guarantee that no bugs of a predefined type are left?
- To achieve this fault models are needed
- Model coverage is not fault coverage
General Fault Model for Testing

(*Specification, Conformance Relation, Fault Domain*)

- Specification is a state-oriented model: Finite State Machine (FSM) or Input Output Transition System (IOTS)
- Fault domain is a set of Implementations Under Test (IUT) treated as black box and modelled by FSM or IOTS, it formalizes test assumptions
- Conformance relation is defined as a relation on FSM or IOTS
Complete Test Suite for Fault Model

• Test suite is *sound* if no conforming IUT from a given fault domain fails it
• Test suite is *exhaustive* if each nonconforming IUT from a given fault domain fails it
• Test suite is *complete* for a given fault model if it is both, sound and exhaustive

• Since life is short, complete test suite must be finite

• The first complete test suites called checking experiments have been studied since 1960s (*model-based testing has started!*
Test Completeness and Fault Coverage

Complete test suite provides a full fault coverage within a given fault domain

- Conformance relation
- Fault domain
- Nonconforming implementations
- Conforming implementations

A smaller fault domain usually requires a shorter complete test.
The Holy Grail for Fault Model-based Testing

A method which given an instance of a fault model generates a minimal complete test suite
Tutorial Aims at Explaining

• Diversity of FSM and IOTS models used for fault model-based testing
• Variety of fault models
• Basic ideas of constructing complete test suites for some fault models
Finite State Machines
Typical Finite State Machine

**Inputs:** coin, button, hit

**Outputs:** lamp, espresso, idle

**States:** 1 (wait for coin), 2 (wait for button)
Specification FSM \( A = (S, s_0, X, Y, \delta, \lambda) \)

- \( S \) - finite set of states, \( s_0 \) is initial state
- \( X \) - finite set of inputs
- \( Y \) - finite set of outputs
- \( \delta : S \times X \rightarrow S \) - transition function
- \( \lambda : S \times X \rightarrow Y \) - output function

Mealy machine

FSM can be viewed as a Finite Automaton
- each state is final (accepting)
- input-output pair is a symbol
Equivalence Relation

• Aka trace equivalence
• Let $A = (S, s_0, X, Y, \delta, \lambda)$
• States $s$ and $s'$ are equivalent if for each input sequence $\alpha \in X^*$ it holds that $\lambda(s, \alpha) = \lambda(s', \alpha)$
• States $s$ and $s'$ are distinguishable if there exists input sequence $\alpha \in X^*$ such that $\lambda(s, \alpha) \neq \lambda(s', \alpha)$; $\alpha$ is distinguishing sequence for $s$ and $s'$
• The length of a distinguishing sequence for two states in (completely specified) FSM does not need to exceed the number of states
• $A$ is reduced (minimal) if it has no equivalent states
• Each completely specified FSM has a unique reduced form
Conformance Relation for Mealy Machines

- Conformance relation is equivalence
- Conforming IUT is modelled by a machine equivalent to the specification machine
- For each nonconforming IUT there exists an input sequence distinguishing it from the specification machine; this is a test on which it fails
Faults in FSM Implementations

- Faults are modelled as mutants of a specification machine forming a fault domain
- Mutations mimic the implementation process
- Initialization fault – wrong initial state
- Output fault - wrong output of transition
- Transfer fault - wrong end state of transition
  - Creating additional states
  - Reducing or maintaining the number of states
- Transition fault - wrong output or end state of transition
- Mutants can have multiple faults (high order mutants)
Mutation Testing with FSM

- Mutation testing is suggested first in 1964 for FSM; each mutant models specific type of hardware faults (stuck-at 0, 1)
- It predates software mutation testing
- Tests are obtained by mutant killing
Mutant Killing

Test hit.button kills the mutant
FSM Mutant Killing

Spec and mutant IUT

Spec × IUT

Spec

IUT

equal outputs?

This is FSM product used, e.g., for verification purposes

no
Mutant Killing with FSM Product Equipped with Fail State

All tests distinguishing Mutant and Spec are given by the paths to the fail-state $bba$, $bbbba$ and so on. $b\{bb\}^*ba$

If the product has no fail-state, the mutant conforms to Spec
General Fault Domain

• A set of mutants each modelling a particular test assumption
• A general fault domain is the universe of all machines with a predefined maximal number of states $m$
• The number of all Mealy machines with $m$ states, $p$ inputs, $o$ outputs is $$(m \times o)^{m \times p}$$
• Test assumption is that anything may go wrong, i.e., all transition faults, but the number of states in IUT does not exceed a chosen limit $m$
• Complete test suites for this fault domain are traditionally called checking experiments, aka m-complete test suites
Deriving Complete Test Suite for Enumerated Mutants

Fault domain is a list of arbitrary mutants

Step 1. For each mutant in a fault domain
  • Construct the product
  • Determine one or several tests killing the mutant

Step 2. Determine a minimal number of tests killing all mutants by solving a set cover problem
  • More mutants need more tests
  • Mutant enumeration can be avoided following the state identification approach (checking experiment)
State Identification
(Complete Tests for Initialization Faults)

• Fault domain is a set of mutants, each mutant is an instance of the specification machine $A = (S, s_0, X, Y, \delta, \lambda)$ initialized in a different state, i.e., $M_i = (S, s_i, X, Y, \delta, \lambda), s_i \neq s_0$

• The number of mutants is $n - 1$, where $n$ is the number of states in the specification machine

• A complete test suite $W(s_0)$ is a set of input sequences distinguishing $s_0$ from all the other states

• $W(s_0)$ is a state identification experiment for state $s_0$ or simply state identifier for $s_0$
Variety of State Identifiers

• Let $W(s_0), ..., W(s_{n-1})$ be state identifiers for $A = (S, s_0, X, Y, \delta, \lambda)$
• If $W(s_0) = ... = W(s_{n-1}) = W$ then $W$ is a characterization set of $A$
• If $W(s_i)$ is a singleton then it is called a UIO sequence of $s_i$
  it provides an output signature of the state
• If a sequence is an UIO of all the states then it is a distinguishing sequence of $A$
• Family of Harmonized State Identifiers (HSI) \{W(s_0), ..., W(s_{n-1})\} such that in each pair of state identifiers $W(s)$ and $W(s')$ there exist two sequences with a common prefix $\alpha$, such that $\lambda(s, \alpha) \neq \lambda(s', \alpha)$
Example

\[ W(s_1) = b \] is UIO
\[ W(s_3) = a \] is UIO
\[ W(s_2) = \{a, b\}, \] \( s_2 \) has no UIO sequence
\{a, b\} is a characterization set of A
A has no distinguishing sequence:

If we take \( a \) then states \( s_1 \) and \( s_2 \) are no longer distinguishable

If we take \( b \) then states \( s_2 \) and \( s_3 \) are no longer distinguishable
Distinguishing Sequence for FSM

\[ a \backslash b \]

\[ a/0 \quad b/1 \]

\[ a/0 \quad b/1 \]

\[ a/1 \quad b/0 \]

\[ a/1 \quad b/0 \]

\[ 1 \quad 0 \quad 1 \quad 0 \]

\[ 2 \quad 0 \quad 1 \quad 1 \]

\[ 3 \quad 1 \quad 0 \quad 1 \]

\[ 4 \quad 0 \quad 0 \quad 1 \]

\[ 5 \quad 1 \quad 1 \quad 0 \]

\[ aba \] is a distinguishing sequence for \( A \)
Example of Harmonized State Identifiers

Family of Harmonized State Identifiers \( \{W(1), W(2), W(3), W(4), W(5)\} \)

Output sequences

\[
W = \{aba\}
\]
\[
W(1) = \{aba\}
\]
\[
W(2) = \{aba\}
\]
\[
W(3) = \{ab\}
\]
\[
W(4) = \{ab\}
\]
\[
W(5) = \{ab\}
\]
What Are State Identifiers For?

• Used by various methods which generate complete test suites for Mealy machines

• Characterization sets in the W-method (Vasilevskiy, 1973; Chow 1978, improved by Bochmann et al 1991, Wp-method)

• Distinguishing sequences in the DS-method (Hennie 1964; continuously being improved)

• UIO sequences in the UIOv-method (Vuong, 1989)

• Harmonized State Identifiers in the HSI-method (Petrenko & Yevtushenko, 1990; later H and SPY methods)
M-Complete Test Suite

• Fault model is (Specification, Conformance Relation, Fault Domain)
• Specification is a reduced Mealy machine $A = (S, s_0, X, Y, \delta, \lambda)$, $|S| = n$
• Conformance relation is equivalence
• Fault domain is the universe of all machines with a predefined maximal number of states $m$
  • Fault do not increase the number of states, $m = n = |S|$
  • Faults may increase the number of states, $m \geq n = |S|$
• Test suite complete for such a fault model is called $m$-complete test suite
The Basic Idea of Constructing M-Complete Test Suites For FSM

• Reach every state in any IUT from the fault domain
• Check each reached state using some state identifiers built from the specification FSM
• Execute all transitions from each reached state
• Check the end state of each transition using the same state identifiers
• Corresponding test fragments are
  • transfer sequences to reach states in the specification FSM
  • sets of sequences extending the transfer sequences to execute all transitions from IUT states
  • state identifiers
W-method for N-complete Test Suite

• $A = (S, s_0, X, Y, \delta, \lambda)$ with $n$ states

• Determine a single transfer sequence for each state, obtaining a state cover $V = \{\alpha_0, \ldots, \alpha_{n-1}\}$, where $\alpha_0 = \varepsilon$, empty sequence;

• Determine a transition cover by adding each input to every transfer sequence in the state cover $VX = \{\alpha_i x | \alpha_i \in V, x \in X\}$

• Construct a characterization set $W$

• Concatenate sets $V \cup VX$ and $W$ to obtain an $n$-complete test suite

$$(V \cup VX)W = \{\alpha \beta | \alpha \in (V \cup VX), \beta \in W\}$$
Example with W-method, I

**button** identifies the states, it is a DS so is **coin**, but it is more expensive 😊
Example with W-method, II

State Cover $V = \{\varepsilon, a, aa, aab, aaba\}$

Transition Cover $VX = \{\varepsilon, a, aa, aab, aaba\}\{a, b\} = \{a, aa, aaa, aaba, aabaa, b, ab, aab, aabb, aabab\}$

Choose, e.g., $W = \{aba\}$

$$(V \cup VX)W = \{\varepsilon, a, aa, aab, aaba, aaa, aabaa, b, ab, aabb, aabab\}\{aba\} = \{aba, aaba, aaaba, aababa, aabaaba, aaaaba, aabaaaba, baba, ababa, aabababa, aababababa\}$$
**W-method for M-Complete Test Suite**

- Fault domain is the universe of all machines with a predefined maximal number of states \( m \), fault may increase the number of states, \( m \geq n = |S| \)
- The number of additional states in IUT can reach \( m - n \); any additional state can be reached by an state cover extended with all input sequences of length at most \( m - n \) in the set \( X^{\leq (m - n)} \)
- Transition cover then becomes \( VX^{\leq (m - n)}X = VX^{\leq (m - n + 1)} \)
- M-complete test suite is then \( VX^{\leq (m - n + 1)}W \)
- N-complete test suite is a special case of m-complete test suite \( VX^{\leq 1}W \)
HSI-method

- It uses a family of harmonized state identifiers $F = \{W(s_0), \ldots, W(s_{n-1})\}$ instead of a single characterization set $VX^{\leq (m-n+1)} \subseteq \mathbb{N}^{m-n+1}$.

$$VX^{\leq (m-n+1)} \subseteq \mathbb{N}^{m-n+1} \subseteq \mathbb{N}^{m-n+1}$$

- This method allows to define state identifiers for partially specified FSMs which may not have any characterization set.
Example with HSI-method

State Cover $V = \{ \varepsilon, a, aa, aab, aaba \}$

Transition Cover $V_X = \{ \varepsilon, a, aa, aab, aaba \} \{a, b\} = \{a, aa, aaa, aaba, aabaa, b, ab, aab, aabb, aabab\}$

HSI: $W(1) = W(2) = \{aba\}$
$W(3) = W(4) = W(5) = \{ab\}$

The resulting n-complete test suite is shorter than the one of W-method, since the latter always concatenates $aba$, while now its prefixes are used.
How Complex are Complete Tests for CFSM?

- Given FSM with $n$ states, any two non-equivalent states can already be distinguished by at most $n-1$ inputs
- We need no more than $n-1$ input sequences to obtain a characterization set/HSI
- The length of $n$-complete test is at most $pn^3$, where $p$ is the number of inputs
- The length of $m$-complete test is exponential $p^{m-n+1}n^3$, $m \geq n$
Mealy Machine is Completely Specified

- \( A = (S, s_0, X, Y, \delta, \lambda) \)
  - \( S \) - finite set of states, \( s_0 \) is initial state
  - \( X \) - finite set of inputs
  - \( Y \) - finite set of outputs
  - \( \delta : S \times X \rightarrow S \) - transition function
  - \( \lambda : S \times X \rightarrow Y \) - output function

- There are partially specified (partial) machines
FSM Redefined

- FSM $A = (S, s_0, X, Y, D, \delta, \lambda)$
  - $S$ - finite set of states, $s_0$ is initial state
  - $X$ - finite set of inputs
  - $Y$ - finite set of outputs
  - $D \subseteq (S \times X)$ - specification domain
  - $\delta: D \rightarrow S$ - transition function
  - $\lambda: D \rightarrow Y$ - output function

Complete FSM if $D = S \times X$; partial FSM (PFSM) if $D \subseteq (S \times X)$

Acceptable input sequences in $X^*$ traverse defined transitions

$\Omega_A(s)$ denote the set of all acceptable input sequences for state $s$
Example

Acceptable sequences – (coin.button)*
It is a partial test model, but FSM implementations are completely specified
How Missing Transitions Can Be Treated?

The behavior is not specified for some states and inputs, i.e., some transitions are missing

- **Implicitly defined** transitions
  - self-loops with the output “null/idle”, inputs are ignored (angelic completion)
  - Transitions lead to a sink state with the output “error”

- **Undefined by default** transitions, i.e., don’t care transitions lead to a chaos sink state with all outputs (demonic completion)

- **Forbidden** transitions
  - The environment cannot provide certain input sequences, e.g., testing via context (see slides later)
  - Certain tests are not allowed (Chernobyl)
  - Input variables can change only one at a time
Example

Undefined by default transitions

Implicitly defined transitions
Testing from Partial FSM

• Input enableness is usually assumed for implementations
• Following a chosen completeness assumption implicitly defined and undefined by default transitions become explicitly defined and a partial FSM is sometimes replaced for testing by a complete FSM
• Do we actually know how PFSM was completed in IUT?
• Partial FSM with forbidden transitions cannot be replaced by any complete FSM
• The equivalence conformance relation needs to be weakened and fault models adapted
PFSM Relations

• $A = (S, s_0, X, Y, D, \delta, \lambda)$

• States $s$ and $s'$ are *compatible* if each input sequence acceptable in both states produces the same output sequence
  \[ \Omega_A(s) \cap \Omega_A(t) = \emptyset \text{ or } \lambda(s, \alpha) = \lambda(s', \alpha) \text{ for all } \alpha \in \Omega_A(s) \cap \Omega_A(s') \]

• $s'$ is *quasi-equivalent* to $s$ if they are compatible and sequences acceptable in $s$ are also acceptable in $s'$
  \[ \lambda(s, \alpha) = \lambda(s', \alpha) \text{ for all } \alpha \in \Omega_A(s) \text{ and } \Omega_A(s') \supseteq \Omega_A(s) \]

• States $s$ and $s'$ are *distinguishable* if there exists $\alpha \in \Omega_A(s) \cap \Omega_A(s')$ such that
  \[ \lambda(s, \alpha) \neq \lambda(s', \alpha) \]

• The length of a distinguishing sequence for two states does not need to exceed “$n \text{ choose } 2$” = $n(n-1)/2$; cf. $n$ for complete FSM

• $A$ is *reduced* if it has no compatible states

• Compatibility is not transitive so partial FSM can have several reduced forms as opposed to complete FSM
Upper Bound for Distinguishing Sequences in PFSM

Determining a sequence distinguishing states 1 and 2
The bound “$n$ choose 2” is tight
M-Complete Test Suite for PFSM

• Fault model is (Specification, Conformance Relation, Fault Domain)
• Specification is a partial not necessarily reduced FSM
  \[ A = (S, s_0, X, Y, D, \delta, \lambda) \]
• Conformance relation is quasi-equivalence,
  since all IUTs are complete machines
• Fault domain is the universe of all complete machines
  with a predefined maximal number of states \( m \)
• The bound \( m \) cannot be lower than the number of pairwise
distinguishable states in \( A \) which can be smaller than \( n = |S| \)
Building m-Complete Test Suite for PFSM

- State identification approach fails because of compatible states
- State-Counting approach (Petrenko & Yevtushenko, 1989, 2000) generalizes the state identification approach
- The idea is based on the above example: determine all acceptable input sequences from each state such that each state of the specification is traversed at least $m$ times
- The value of $m$ can be reduced if the specification has
  - quasi-equivalent or
  - distinguishable states, so state distinguishing sequences can be employed
Mealy Machine is Deterministic

• $A = (S, s_0, X, Y, \delta, \lambda)$
  
  $S$ - finite set of states, $s_0$ is initial state
  $X$ - finite set of inputs
  $Y$ - finite set of outputs
  $\delta: S \times X \rightarrow S$ - transition function
  $\lambda: S \times X \rightarrow Y$ - output function

• There are nondeterministic machines
FSM Redefined Again

\[ A = (S, s_0, X, Y, T) \]

- \( S \) - finite set of states, \( s_0 \) is the initial state
- \( X \) - finite set of inputs
- \( Y \) - finite set of outputs
- \( T \) - transition relation \( T \subseteq S \times X \times Y \times S \)

A quadruple \( (s, x, y, s') \in T \) is a transition

\( A \) is complete if for each \( (s, x) \in S \times X \) it has at least one transition

\( A \) is deterministic if for each \( (s, x) \in S \times X \) it has at most one transition

\( A \) is nondeterministic if for some \( (s, x) \in S \times X \) it has several transitions

\( A \) is observable if for each \( (s, x, y) \in S \times X \times Y \) it has at most one transition; non-observable FSM can be made observable by determinization
Is Nondeterminism Really a Problem in Testing?

• Nondeterminism contradicts the general feeling that computer systems are deterministic, so we might wonder whether there is any real meaning to use the term nondeterminism, or whether it is a useful, but essentially meaningless, formal trick (A. J. Dix, 1990)

• It is a good practice to design systems for testability, and one important aspect of testability is that you can reproduce test results many times, and this implies that the system should be as deterministic as possible. The issue of testing nondeterministic systems is therefore, albeit real, relatively minor at least in the current practice (A. Huima, 2011)
Various Sources of Nondeterminism

• Concurrency modelled by interleavings
• Abstractions, under-specification (undefined by default transitions), options, uncertainty
• Asynchronous communication, unknown delays
• Partial controllability
• Partial observability
• Distributed interfaces: the actual order of events is unknown
• Random choice implemented in some applications
NFSM Example

in Spec!

in Implementations!
Testing NFSM is a Challenge

• In NFSM input sequence can produce several output sequences and bring the machine into different states

• Two main test scenarios
  • Specification is nondeterministic, but IUT is deterministic
  • Both, specification and IUT, are nondeterministic

• Testing nondeterministic IUT requires some fairness/all weather assumption:
  there exists $k$ such that, if input sequence is applied to IUT $k$ times, the tester concludes that all possible output sequences are observed

• Testing becomes adaptive, so tests are also NFSM as opposed to DFSM tests which are input sequences
NFSM Relations

• FSM $A = (S, s_0, X, Y, T)$
• States $s$ and $s'$ are (trace) equivalent if their sets of traces coincide, i.e., $Tr(s) = Tr(s')$
• $s$ and $s'$ are distinguishable if $Tr(s) \neq Tr(s')$, there exists input sequence $\alpha \in X^*$ such that $out(s, \alpha) \neq out(s', \alpha)$
  $\alpha$ is distinguishing sequence for $s$ and $s'$
• $s$ is a reduction of $s'$ if $Tr(s) \subseteq Tr(s')$, trace inclusion
• $s$ and $s'$ are $r$-compatible, if they have a common reduction, i.e., there exists a state of a complete FSM that is a reduction of both states
• $s$ and $s'$ are $r$-distinguishable, if they have no common reduction, i.e., no state of any complete FSM can be a reduction of both states
  • several input sequences may be needed
  • the bound is $n^2$
Example of R-distinguishability

Specification NFSM

Fragment of the product

Adaptive test r-distinguishing

for states 1 and 3

Adaptive tests r-distinguishing states are called *separators*
Fault Models for NFSM

- Fault model is \((\text{Specification}, \text{Conformance Relation}, \text{Fault Domain})\)
- Specification is NFSM \(A = (S, s_0, X, Y, T)\)
- Case 1
  - Fault domain is the universe of all NFSMs with a predefined maximal number of states \(m\)
  - Conformance relation is trace equivalence
- Case 2
  - Fault domain is the universe of all NFSMs with a predefined maximal number of states \(m\)
  - Conformance relation is trace inclusion
- Case 3
  - Fault domain is the universe of all DFMSs with a predefined maximal number of states \(m\)
  - Conformance relation is trace inclusion
M-Complete Test Suite for NFSM and Equivalence Relation

• Fault model is (Specification, Conformance Relation, Fault Domain)
• Specification is a reduced NFSM $A = (S, s_0, X, Y, T)$
• Conformance relation is trace equivalence
• Fault domain is the universe of all NFSMs with a predefined maximal number of states $m$
• M-complete test suite can be obtained by mimicking steps of the W-method for DFSM

$$\forall X \leq (m-n+1) W$$
Fault Model for NFSM and Reduction Relation

• Fault model is (Specification, Conformance Relation, Fault Domain)
• Specification is an NFSM $A = (S, s_0, X, Y, T)$
• Conformance relation is reduction
• Fault domain is the universe of all NFSMs and DFSMs with a predefined maximal number of states $m$, not less than the number of pairwise $r$-distinguishable states
• Multiple test execution only if the fault domain includes NFSMs
• Characterization set $W$ for the equivalence relation cannot be used, separators play its role
Recall the Basic Idea of Constructing m-Complete Test Suites

• Reach every state in any IUT from the fault domain
• Check the reached state using some state identifiers
• Execute all transitions from each reached state
• Check the end state of each transition using the same state identifiers
• Fragments of complete test suite
  • transfer sequences to reach states in the specification FSM
  • sets of sequences extending the transfer sequences to execute all transitions from IUT states
  • state identifiers
Challenges of Constructing Transfer Sequences

• Transfer sequences to reach states in the specification FSM if it is nondeterministic?

• State reachability for NFSM
  • Reachable by preset transfer input sequence (deterministically reachable)
  • Reachable by adaptive tests, i.e., adaptive preambles (definitely reachable)
  • No adaptive homing test exists

• Some states of NFSM $A = (S, s_0, X, Y, T)$ may not even be implemented in an IUT conforming wrt reduction relation, since it is allowed not to have all the traces of $A$
State Preamble Example

How state 2 can be reached?

Here is a preamble
Building m-Complete Test Suite For NFSM

• Preambles are used to reach definitely reachable states
• Traversal sequences are determined as in case of partial DFSM, such that each state of the specification is traversed no more than $m$ times
• The parameter $m$ can be reduced if the specification has
  • r-compatible states or
  • r-distinguishable states
• State separators used for state identifiers

The SC-method by Petrenko & Yevtushenko (recent versions in 2011 and 2014)
M-Complete Test Suites Come at a Price

• Scalability and expressiveness
  • M-complete tests may explode
  • All the ingredients of the FSM model are finite, as a result such a model without extensions may become to be too big to construct and execute tests
  • FSM model does not support the use of variables/parameters

• How we can deal with this
  • Generate tests incrementally
  • Use communicating FSMs
  • Use mutation machines to specify fault domains as subsets of the universe of all FSMs
  • Consider extended FSM models
Incremental Generation of Complete Test Suite

• The idea is to find sufficient conditions for $p$-completeness
• Given a test suite determine additional tests which satisfy these conditions
• Increment $p$ and iterate until the desired value is reached or the size of the obtained test suite reaches a predefined limit

• See the P-method, see Petrenko & Simao, “Fault Coverage-Driven Incremental Test Generation”, 2009
M-Complete Test Suites Come at a Price

How to deal with this

• Generate tests incrementally
• Use communicating FSMs
• Use mutation machines to specify fault domains as subsets of the universe of all FSMs
• Consider extended FSM models
Testing Communicating FSMs

• Given a system of ComFSM, derive a (complete) test suite using external inputs and outputs
• Variety of testing scenarios
  • faults in individual FSMs or in channels
  • some FSMs are fault-free
  • internal actions are observable or not
  • distributed testing with synchronized testers
Example of Communicating FSMs
M-Complete Test Suites Come at a Price

How to deal with this

• Generate tests incrementally
• Use communicating FSMs
• Use mutation machines to specify fault domains as subsets of the universe of all FSMs
• Consider extended FSM models
Using Mutation Machines to Specify Fault Domains

$I/O = \{x/y \mid x \in X, \ y \in Y\}$

Chaos FSM, which represents all possible mutants in the fault domain with at most 4 states
It is called a \textit{mutation machine}, each deterministic submachine is a mutant
Their number is \((|S| \times |O|)^{|S| \times |I|} = (4 \times 2)^4 \times 2 = 16,777,216\)
4-complete test suite for this fault model kills all the mutants
Example of Restricted Fault Domain

The mutation machine has 6 suspicious transitions in red
3 mutated transitions in dash lines (output fault and two transition faults
The fault domain includes 8 deterministic submachines, so 7 mutants
they can be enumerated as in mutation testing
Testing Based on Mutation Machines

• Fault model is (Specification, Conformance Relation, Fault Domain)
• Specification is a DFSA $A = (S, s_0, X, Y, \delta, \lambda)$
• Conformance relation is equivalence
• Fault domain is the set of all deterministic submachines of $A$
• Test generation methods using state identification facilities can be adopted to deal with non-chaotic mutation machines
• See our papers co-authored with Koufareva, Yevtushenko and Grunskiy

• Test completeness can be checked for this fault model
• The idea is to use again an FSM product
• See our paper “Multiple Mutation Testing from FSM”, FORTE 2016
Distinguishing Automaton

Specification FSM

Mutation FSM
Mutation Coverage Analysis

The idea is to determine constraints on transitions in all the mutants killed by a test case.

Example: \textit{baaba}

two revealing executions

Any mutant is killed by the test \textit{baaba} if it includes suspicious transitions \((3, a, 1, 3)\) OR \((3, a, 0, 3)\) AND \((3, b, 0, 3)\) AND \((3, a, 0, 3)\)

The negated constraint characterizes the survived mutants.
Input: a test suite $TS$ for a specification and mutation machine

1. For each $\alpha \in TS$, build the disjunction of constraints excluding the $\alpha$-revealing executions
2. Build the conjunction of the obtained disjunctions and add the constraint that excludes the solution defining the specification machine
3. Check the satisfiability of the constraint by calling a solver
4. If it is not satisfiable terminate with the message “$TS$ is complete”, otherwise check whether the mutant defined by a solution is conforming
5. If it is nonconforming terminate with the message “$TS$ is incomplete”, otherwise add the constraint that excludes the conforming mutant and go to Step 3
Experimental Results for Randomly Generated Mutation Machines

A specification for an automotive controller with 336 transitions

<table>
<thead>
<tr>
<th>Mut. Trans.</th>
<th>$M_{hvac}$</th>
<th>$M_{+20}$</th>
<th>$M_{+100}$</th>
<th>$M_{+428}$</th>
<th>$M_{+764}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>66</td>
<td>146</td>
<td>474</td>
<td>810</td>
<td></td>
</tr>
<tr>
<td>6.9×10^{10}</td>
<td>5.5×10^{16}</td>
<td>3.1×10^{38}</td>
<td>2.2×10^{108}</td>
<td>3.9×10^{163}</td>
<td></td>
</tr>
<tr>
<td>Mutants</td>
<td>36</td>
<td>55</td>
<td>140</td>
<td>369</td>
<td>1111</td>
</tr>
<tr>
<td>Tests</td>
<td>3</td>
<td>7</td>
<td>24</td>
<td>86</td>
<td>1013</td>
</tr>
<tr>
<td>Sec.</td>
<td>152</td>
<td>99</td>
<td>140</td>
<td>369</td>
<td>1111</td>
</tr>
</tbody>
</table>
M-Complete Test Suites Come at a Price

How to deal with this

• Generate tests incrementally
• Use communicating FSMs
• Use mutation machines to specify fault domains as subsets of the universe of all FSMs
• Consider extended FSM models
Extended FSM Models

• FSM can be extended with variables, parameters, guards and operations on them (time can also be added)

• Once internal variables are used the number of complete states may become unbounded, so FSM methods are not applicable

• Fault model-based testing of extended FSMs (EFSMs) has not yet reached the level of maturity of that of the classical FSMs

• Mutation approach with mutation operations is applicable

• EFSM equivalence is in general undecidable

• This motivates investigation of special classes of EFSMs for which complete test suites could be derived (FSM with Symbolic Inputs, see our paper in ICTSS 2015)
Complete Test Generation for FSM Needs More Research

• M-complete test suite for partial non-observable NFSM?
• Complete test suites using mutation machines?
• Complete test suites for an arbitrary fault model?
• General types of Extended FSM?
Input/Output Transition Systems
Outline

• Informal discussion of IOTS

• For formal definitions and ioco based testing theory, see the HSST 2014 tutorial, Tretmans, “Model-Based Testing - There is Nothing More Practical than a Good Theory”

• The ioco conformance relation and problems of using it

• Asynchronous testing

• Special IOTS classes for which complete test suites are suggested
  • Mealy IOTS
  • Input-eager IOTS
Labeled Transition Systems

• Difference between inputs and outputs is abstracted away
• LTS is a finite automaton, every state is accepting
• LTS communicate by rendezvous
• Actions are controllable, “refusal” is observed

\[
\begin{align*}
\text{s} & \xrightarrow{a} \text{s'} \\
\text{s} & \xrightarrow{a/a} \text{s'}
\end{align*}
\]

\{b,c,d\}/refusal

• Several refusal-based conformance relations and test methods exist, see Glabbeek, “The linear time-branching time spectrum”, 1990
Example of Input/Output Transition System

State is quiescent if no output is enabled. Its observation needs a timer.

Inputs may occur at states that are not quiescent.

Rich choice of conformance relations which use traces, quiescent traces, suspension traces.
Input Enableness of IOTS

- IOTS is LTS with Input-Output and always enabled inputs
- I/O automaton, interface automaton
- Interpretation of unspecified transitions similar to partial FSM
  - Implicitly defined transitions: inputs lead to a sink state with the output “error”
  - Undefined by default transitions: don’t care transitions lead to a chaos sink state with all outputs (demonic completion)
  - Forbidden transitions: the environment cannot provide some input sequences
IOCO Relation and Test Generation

• i ioco-conforms to s iff
  • if i produces output x after trace σ, then s can produce x after σ
  • if i cannot produce any output after trace σ, then s cannot produce any output after σ (quiescence δ)

• Test generation algorithm: apply the following steps recursively, non-deterministically:
  • Terminate with pass
  • Give a next input to the IUT
  • Check the next output of the IUT

• If it is executed infinitely long, a complete test suite is obtained, but it is not finite
• If it yields a finite test suite its fault coverage is unknown
• Most of the existing work is on test purpose-based test generation and not on fault coverage
• Variety of ioco-like relations retaining its main features
The Use of IOCO for Testing is Problematic

For general type of IOTS tests for this relation need to block outputs, may not be sound and controllable

\[ !cf = \text{coffee} \]
\[ !es = \text{espresso} \]
\[ ?cn = \text{coin} \]
\[ \delta = \text{quiescence} \]

IOCO-based test blocks outputs according to version 1996 and TGV
To Block or Not to Block, That is the Question

We define input-output transition systems to model systems with inputs and outputs, in which outputs are initiated by the system and never refused by the environment, and inputs are initiated by the system’s environment and never refused by the system, see Tretmans, "Model based testing with labelled transition systems", 2008

• TGV, TorX, SpecEx, UPPAAL/TRON rely on blocking?
Example of Uncontrollable Tests

\( \delta \)

\( !cf \rightarrow s_0 \rightarrow s_1 \)

\( ?cn \rightarrow s_0 \rightarrow s_1 \)

\( !es \rightarrow s_1 \rightarrow s_2 \)

\( ?cn \rightarrow s_2 \rightarrow s_0 \)

\( !cf \rightarrow s_2 \rightarrow s_1 \)

\( ?cn \rightarrow s_1 \rightarrow s_2 \)

\( !es \rightarrow s_1 \rightarrow s_2 \)

\( ?cn \rightarrow s_2 \rightarrow s_1 \)

IOCO-based test according to version 2008 is input-enabled (SpecEx, but not TGV)
Input Enabledness of Test Cases?

• Contradicts to
  • We will not allow test cases with a choice between input and output nor a choice between inputs [1996]
  • A test case is controllable if no choice is allowed between input and output or between inputs [TGV, FSM]
• If outputs cannot be blocked/preempted they could be stored in queues
• It is queued or asynchronous testing
Example of Unsound Tests

Unsound Asynchronous Test

Unsoundness is due to distortion in observing outputs via queue
iocof, ioconf, mioco, rtioco, iocor, sioco, hioco, ...
..., altsim
Asynchronous Testing for Output Faults

Mealy IOTS

Without I/O conflicts, both models have the same behaviour

IOTS is then a Mealy IOTS

All the FSM methods generating complete tests are fully applicable
Input-eager IOTS

• Assume that each implementation IOTS when it is state with the input/output conflict, it does not produce any output if its input queue contains an input. We call such IOTS *input-eager*

• Fault model is (Specification, Conformance Relation, Fault Domain)
  • IOTS with $n$ input states
  • IOCO
  • The set of all input-eager IOTSs with at most $n$ input states

• $n$-complete test suite can be generated by constructing test fragments similar to those for NFSM

• Simao & Petrenko, “Generating Complete and Finite Test Suite for ioco: Is It Possible?”, MBT 2014
Example of Sound Tests for Input-eager IOTS

\[ \delta \quad \text{!} \text{cf} \]

\[ \text{!} \text{cf} = \text{coffee} \]
\[ \text{!} \text{es} = \text{espresso} \]
\[ ? \text{cn} = \text{coin} \]
\[ \delta = \text{quiescence} \]
Complete Test Generation for IOTS Needs More Research

• Asynchronous testing for faults other than output faults?
• IOTS needs to be minimal to adopt state identification approaches, how to minimize it?
• M-complete test generation for general type of IOTS?
• Mutation machine for IOTS?
• Any way of dealing with I/O conflicts other than assuming input-eagerness or using queues?
• IOTS with extensions?
The Quest for the Holy Grail Goes On