Testing Distributed Systems

R. M. Hierons
Brunel University, UK
rob.hierons@brunel.ac.uk
http://people.brunel.ac.uk/~csstrmh

TAROT 2009
Work With

- Jessica Chen
- Mercedes Merayo
- Manuel Nunez
- Hasan Ural
Model Based Testing

• We only observe interactions between the system under test (SUT) and its environment.

• To reason about test effectiveness we assume:

  • The SUT can be expressed in same language as the specification.

• We might make additional assumptions.
Multi-port systems

- Physically distributed interfaces/ports
- We place a tester at each port
Context/assumptions

- Testing is **black-box**: we do not have access to anything ‘inside’ the system.
- Communications is **synchronous** or there is a ‘slow environment’.
- Specification is **complete**.
- The system under test behaves like an unknown model that can be represented using the same formalism as the specification.
Motivation

• Initially just testing – and testing in the distributed test architecture.
• The discussion will be around both
  • testing and
  • implementation/conformance relations.
Structure

- Background

- Testing from:
  - deterministic finite state machines
  - nondeterministic finite state machines
  - input output transition systems
Background
Global Traces

- A global trace is an input/output sequence
- We assume there are m ports and let:
  - $X$ be the input alphabet/set divided into $X_1, \ldots, X_m$: $x_i$ will denote an input from $X_i$
  - $Y$ be the output alphabet, each element being in $(Y_1 \cup \{-\}) \times \ldots \times (Y_m \cup \{-\})$: $y_i$ will denote an element of $Y_i$
- A global trace is an element of $(X \times Y)^*$
- The set of global traces of a system can be seen as its behaviour.
The distributed test architecture

- We have a tester at each port
- These testers cannot communicate with one another
- There is no global clock

Advantages:
- Simple and cheap to implement
- Might represent expected usage
Consequences

- Each tester observes only the interactions at its port – we cannot observe global traces.

- The tester at port 1 observes $x_1y_1x_1y_1$ and the tester at 2 observes $y_2$ only.
What the testers observe

- Each tester observes a local trace.
Local Traces

- At a port $p$ we observe a local trace: an element of $(X_p \cup Y_p)^*$

- Given global trace $z \in (X \times Y)^*$ and port $p$ we let $\pi_p(z)$ be the projection of $z$ onto $p$.

- We are interested in the case where either:
  - *Testing* can only observe local traces; or
  - *Users* only observe local traces.
Two possibilities

We might have:

• Agents at ports are entirely ‘independent’:
  • No external agent can receive information regarding observations at more than one port
• Or the local traces observed at the ports can be ‘brought together’ later.
Differences

➢ Specification

 Tester 1

 x

 y

 z

 Tester 2

➢ SUT

 Tester 1 SUT Tester 2

 x

 y'

 z

 TAROT 2009
In more detail

- **Specification:**
  - Response to x can either be \((y,z)\) or \((y',z')\) [at ports 1,2]

- **Implementation**
  - Response to x is \((y',z)\)

- Mostly we will assume that local traces can be brought together later.
Controllability problems

- The following test has controllability problems: introduces nondeterminism into testing.
Observability problems

- The following look the same
  tester Spec tester
  \[
  \begin{align*}
  x_1 & \quad x_1 \\
  y_1 & \quad y_1 \\
  y_2 & \\
  \end{align*}
  \]

- Testers/users cannot ‘map’ output to input
  tester SUT tester
  \[
  \begin{align*}
  x_1 & \\
  y_1 & \\
  y_2 & \\
  \end{align*}
  \]
Equivalent global traces

- Since we only observe local traces:
  - Global traces $z$ and $z'$ are indistinguishable if their projections are identical: the local traces are the same.
  - We denote this: $z \sim z'$

- The following are equivalent under $\sim$
  - $x_1/(y_1, y_2)x_1/(y_1, -)$
  - $x_1/(y_1, -)x_1/(y_1, y_2)$

- Both have $x_1y_1x_1y_1$ at port 1 and $y_2$ at 2.
Local verdicts

- Normally a test case ends in a state that denotes the verdict of the test run.
- We have a tester at each port – we could include ‘local verdicts’ in each tester.
- The overall verdict is pass if and only if all ‘local verdicts’ are pass.
- Here:
  - local traces cannot be brought together
- Normally this is not sufficient.
Observation: Test effectiveness is not monotonic.

Example: $x_1$ detects a fault but $x_1 x_1$ does not.
Using an external network

If we connect the testers using an external network, *sometimes* we can overcome controllability and observability problems.
But

- If a system has physically distributed interfaces then the implementation relation should reflect this:
  - Even if we can connect the testers, we should be careful that we do not give the verdict fail when the behaviour is acceptable in use.
  - *The users will only observe local traces.*
Past research

Almost all has focussed on testing from a deterministic finite state machine (DFSM).

Two main topics of interest:

- Generating test sequences that do not suffer from controllability and/or observability problems
- Adding coordination messages (possibly adding a minimum number).
Problems/issues

- A DFSM can have transitions that cannot be executed without causing controllability problems.
- ‘Complete’ test generation algorithms place conditions on the DFSM – they are not general.
- The methods test against the ‘traditional’ implementation relation – aiming to do too much?
The solution

- We need a good understanding of what it means to distinguish two models with distributed ports.

- We have seen that this gives us new implementation relations.

- Then we want to test against these.
Deterministic Finite State Machines
Finite State Machines

- The behaviour of M in state $s_i$ is defined by the set of input/output sequences (traces) from $s_i$. 

```
s_2
a/0
b/1

s_1
b/1
a/0

s_5
a/0
b/1

s_3
b/0
a/1

s_4
a/0
b/0
```
Implementation relations

- For single-port deterministic finite state machines (DFSMs) say that:
  - N conforms to M if and only if N and M are equivalent (they define the same sets of global traces).

- We can express this in terms of their initial states being equivalent.

- If M is nondeterministic we require that N is a reduction of M:
  - Every trace of N is a trace of M.
Properties

- Classical DFSM conformance is just finite state machine (and so finite automaton) equivalence.
- It is an equivalence relation.
- We have an associated notion of minimality:
  - A DFSM $M$ is minimal if there is no smaller DFSM that is equivalent to $M$
- For a DFSM $M$ there is a unique equivalent minimal DFSM.
An implementation relation for distributed systems

- We say that DFSM N conforms to DFSM M if:
  - Every global trace of N is indistinguishable from a global trace of M.

- Equivalently:
  - For every global trace z of N there is a global trace z’ of M such that $z \sim z’$. 
Conformance is weaker than equivalence

- This also shows that it is not an equivalence relation (second has behaviours inconsistent with first).

- Is the first an acceptable design for second?
Observation

- There are many notions of equivalence in the LTS/IOTS literature.
- Have we reinvented one of these?
- Previous notions of equivalence reduce to isomorphism when considering minimal, completely-specified DFSMs.
- Local equivalence does not.
Refinement and testing

- It appears that we must produce new refinement rules and new test techniques.
- Alternatively (if we can observe global traces in testing):
Here we transform to a non-deterministic FSM.
Not a general solution

The language of words that are equivalent under $\sim$ is not regular (intersect with $X^*$)
Controllable testing
Controllability problems

- The following test has a controllability problem.
Controllability for DFSM

An input sequence $x=x_1...x_k$ is controllable if the corresponding trace $z=x_1/y_1...x_k/y_k$ has the following property:

- Every $x_i$, $i>1$, is applied at a port $p_i$ such that $x_{i-1}/y_{i-1}$ has either input or output at $p_i$.

This means:
- The tester at $p_i$ knows when to apply $x_i$.

Ideally we wish to restrict testing to controllable sequences

Initial work focussed on this case.
Distinguishing states

- If we restrict ourselves to controllable testing we need:
  - x causes *no controllability problems* from s and s’
  - x leads to different sequences of interactions, for s and s’, at some port.

- We say that x *locally s-distinguishes* s and s’.

- If no input sequence locally distinguishes s and s’ they are *locally s-equivalent*. 
Testing is weaker

- We cannot locally s-distinguish $s_1$ and $s_4$ but $x_1x_2$ locally distinguishes them.
Distinguishing two states

- Given port \( p \) and states \( s_1 \) and \( s_2 \) of a \( m \)-port FSM \( M \) with \( n \) states:
  - \( s_1 \) and \( s_2 \) are locally \( s \)-distinguishable by an input sequence starting at \( p \) if and only if they are locally \( s \)-distinguished by some such input sequence of length at most \( m(n-1) \).
- This bound is ‘tight’.
- The sequences can be found in low-order polynomial time.
Distinguishing all states

- A ‘complete’ set of input sequences that locally s-distinguish the locally s-distinguishable states of M can be found in $O(pn^2)$, where $p$ is the size of the input alphabet.

- This has at most $n-1$ sequences, each of length at most $m(n-1)$
Minimality

Two possible definitions:

- Def 1: A DFSM is locally s-minimal if it has no locally s-equivalent states.
- Def 2: A DFSM M is locally s-minimal if no DFSM with fewer states is locally s-equivalent to M.

For initially-connected, completely specified, single-port DFSMs, these are the same.
Minimal DFSMs are not always locally s-minimal

- We have seen that $s_1$ and $s_4$ are locally s-equivalent
Merging s-equivalent states

A smaller acceptable design?

\[ x_2/(\cdot, y_2) \quad x_1/(y_1,\cdot) \]

\[ x_2/(\cdot, y_2) \quad x_1/(y_1,\cdot) \]

\[ x_2/(\cdot, y_2) \quad x_1/(y_1,\cdot) \]

\[ x_2/(\cdot, y_2) \quad x_1/(y_1,\cdot) \]
Minimising: smallest FSM

Even smaller:

\[
\begin{align*}
x_2/(-, y_2) & \quad x_1/(y_1,-) \\
S_1 & \quad S_2 \\
x_2/(-, y_2) & \quad x_1/(y_1,-) \\
S_4 & \quad S_3 \\
x_2/(-, y_2) & \quad x_1/(y_1,-) \\
S_1 & \quad S_1
\end{align*}
\]

TAROT 2009
Consequences

- We had two alternative definitions.
  - Def 1: A DFSM is locally s-minimal if it has no locally s-equivalent states.
  - Def 2: A DFSM M is locally s-minimal if no DFSM with fewer states is locally s-equivalent to M.

- For multi-port DFSMs these differ.
- Def 2 is ‘better’?
Canonical FSMs

- Given DFSM M, we can find:
  - Maximal $M_{\text{max}}$ that is locally s-equivalent to M
  - Minimal $M_{\text{min}}$ that is locally s-equivalent to M

- We can find them efficiently.
Results

- DFSM N is locally s-equivalent to DFSM M if and only if N is a reduction of $M_{\text{max}}$.

- The set of DFSMs that are s-equivalent to a DFSM M forms a bounded lattice.
Refinement and testing

We now know that:

- FSM $M$ 
- FSM $M_{\text{max}}$
- Implementation $N$

s-equivalence
reduction
Overcoming observability problems in controllable testing

➢ It is sufficient to do following:

• For each test sequence $x$ also use every prefix of $x$ (from the initial state).

➢ So, if we can repeat tests and we have no controllability problems then we can avoid observability problems.
Open questions

- How do we ‘target’ local equivalence or local s-equivalence in test generation?

- When can we produce equivalent of $M_{\text{max}}$ for local equivalence?

- Generating locally minimal/s-minimal DFSSMs? Second problem is NP-hard but is polynomial if we have two ports.
Nondeterministic FSMs (NFSMs)
An additional problem

- We might have pairs of allowed traces with prefixes like the following:
Choice

- A tester makes a choice based on its observations.
- This is the notion of ‘local choice’.
- Already studied in the context of Message Sequence Charts (e.g. non-local choice pathologies).
- Difference in problems considered and our problem has additional ‘structure’
Controllability for NFSMs

There may be several global traces that can occur in response to an input sequence x:

- We require that none of them have controllability problems
- Each tester makes local choice:
  - there should not be two prefixes z and z’ of traces in response to x that look the same to a tester at port p and yet this tester should behave differently after them

The first of these conditions is guaranteed if we have the second (local choice).
Test generation

- How do we generate input sequences that have no controllability problems?

- One possibility:
  - Start with the empty sequence and at each stage either terminate or extend with an input that does not lead to a controllability problem

- Question: how to direct this?
Observability problems

- If observe global traces, controllable $x_1x_1$ shows: implementation is not a reduction of the specification.

- We cannot detect this with local traces.

- Contrast with DFSSMs where we can use prefixes.
Open Questions

- Test generation algorithms (guided).
- Locally s-distinguishing states.
- How can we add coordination messages in order to overcome controllability and observability problem?
- Can we add a minimum number?
The Oracle Problem

- For DFSMs this:
  - Can be solved in polynomial time for controllable test sequences
  - Otherwise is NP-hard

- For NFSMs:
  - NP-hard even for controllable testing
  - Polynomial if we restrict further
New notions of conformance. Option:

- We can restrict to controllable test sequences

For controllable testing:

- Oracle problem can be solved in polynomial time
- Have unique ‘min’ and ‘max’ machines
- Can test against ‘max’ model for reduction using traditional methods
- Could develop from ‘max’ model?
Input Output Transition Systems (IOTSSs)
IOTS models

- IOTS models are more general than NFSMs:
  - They can be infinite state models
  - Input and output need not alternate.

- We assume:
  - IOTs are input enabled
  - We can observe quiescence

- It is normal to precede the names of inputs by ? and outputs by !
Example

- A process that can receive input $?i_1$, then output $!o_2$ and then output $!o_1$.

- After $!o_1$ it is quiescent.
Implementation relations

- There is a standard implementation relation called ioco
- Can we adapt ioco to multi-port systems?
- Can we produce test cases for this new implementation relation?
Two equivalent processes

- We cannot distinguish the following:

```
  ?i₁
   ↓
  !o₂
   ↓
  !o₁

  ?i₁
   ↓
  !o₁
   ↓
  !o₂
```

- The reason: we cannot ‘stop’ after \(?i₁!o₂\).
When do we make observations?

- For an FSM we observe the projections of input/output sequences - we can ‘stop’ after an input/output sequence.
- When can we ‘stop’ when considering IOTSSs? Possibly:
  - Whenever we have quiescence.
- We can then ‘bring together local traces’
An implementation relation dioco

We say that i dioco s if:

- For every trace z of i that can take i to a quiescent state, there is some trace z’ of s such that $z' \sim z$.

This means:

- If i has a ‘run’ z then s has a specified behaviour that is ‘equivalent’ to z.
dioco does not imply ioco

Example:

\[
\begin{align*}
\text{?i}_1 & \quad \text{!o}_2 \\
\text{!o}_1 & \quad \text{?i}_1 \\
\text{!o}_1 & \quad \text{!o}_2
\end{align*}
\]
Result

- If $s$ and $i$ are input enabled then:
  - $i \text{ ioco } s$ implies that $i \text{ dioco } s$

- Normally IOTS implementations are required to be input enabled.

- So:
  - For input enabled specifications we have that $i \text{ dioco}$ is weaker than $i \text{ ioco}$.
Controllability

- Similar to NFSMs:
  - A test case is controllable if each tester can make ‘local choices’

- Result:
  - We can decide in polynomial time whether a test case is controllable.
Additional implementation relations?

- In dioco we assume traces can be brought together at the end of testing.
- We have allowed the use of test case with controllability problems.
- So, there are alternative implementation relations.
An example

- We can require that local traces are not brought together.
- Makes sense if this corresponds to expected usage.
- We require:
  
  For every trace $z$ of the implementation and port $p$ there is a trace $z'$ of the specification such that $\pi_p(z) = \pi_p(z')$
Can be weaker

- Specification and implementation

- Looks ok if we cannot bring together local traces.
Can be stronger

- No quiescence:

- Suggests: only allowing traces ending in quiescence is problematic.
Additional alternatives

- Instead of only considering quiescent traces we could:
  
  - Combine (conjoin) the previous two implementation relations.
  
  - Consider infinite traces.
Current and future work

Work in progress
- Canonical FSMs.
- Testing from NFSMs.
- Implementation relations for IOTSSs.
- Test generation for IOTSSs.
- Considering infinite traces.

Future work
- Generating complete test suites.
- Generating test cases to satisfy a test criterion.
- Minimising an FSM.
- Timed models.
Papers

These include:

Conclusions

- If a system has distributed interfaces/ports then we have different implementation relations.
- This can affect testing but also development.
- We get new notions of e.g. a design being minimal.
- The effect is even greater for nondeterministic models/systems.
Questions?