Directions in Model Based Testing

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

What is Model Based Testing?

- At its simplest:
  - We build a model
  - We use this model to help in testing
- A model represents what is to be tested.
- It need not model the entire system and can leave out many details (abstraction).
- Models are often much simpler than requirements.

What sorts of languages?

- Almost anything. Languages used include:
  - Diagrammatic notations (e.g. statecharts, SDL, sequence diagrams)
  - Formal specification languages (Z, B, CASL, LOTOS)
  - High level code (e.g. python)
  - Finite state machines

Why bother?

- Benefits include:
  - Automating test generation
  - Automating checking of results
  - Validating requirements/design through building model
  - Regression testing – change the model not the tests
- But
  - There is an initial cost of building the model
  - Building a model requires particular skills/training
  - The model may be wrong

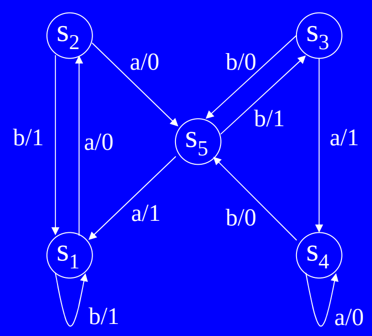
Topics

- I will say a little about:
  - Coverage and automated generation from state machines
  - Testability and transformations for state machines
  - Regression testing and ordering adaptive/TTCN-3 test cases
- Context: black-box testing

Coverage and Test automation

Finite State Machines

- The behaviour of M in state s_i is defined by the set of input/output sequences from s_i

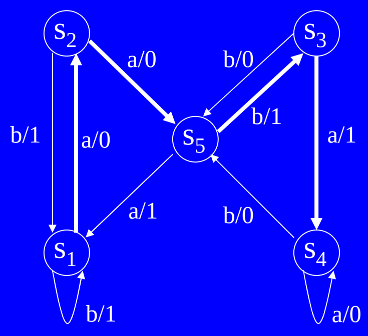


Test coverage

- There are many popular notions of code coverage such as: Statement, Branch, MC/DC, LCSAJ, …
- It is natural to define measures of model coverage.
- For FSMs we have:
  – State coverage
  – Transition coverage

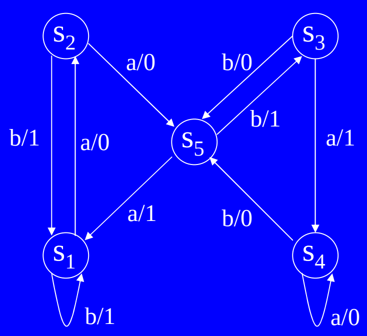
Example (state)

- We could use input sequence aaba
- Gives us no confidence in the transitions not covered



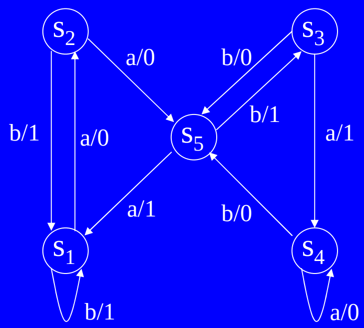
Example (transition)

- Here we can use babaabbbaaba
- We may not observe an incorrect final state of a transition.
- Example: last transition in above.
- Instead, we can check the final states of transitions.



Distinguishing Sequences

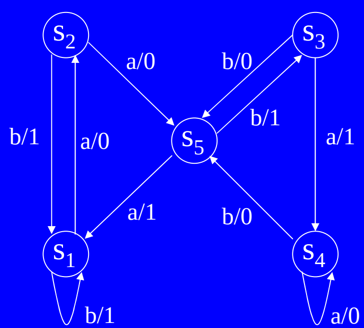
- A distinguishing sequence is an input sequence that leads to a different output sequence for every state.



- Here e.g. aba

Unique Input/Output Sequences

- A UIO for state s is defined by an input sequence x such that the output from s in response to x is different from the output from any other state s'.



- UIO for s_2: a/0 a/1

Characterizing sets

- A set W of input sequences such that:
  - for every pair s, s’ of distinct states there is an input sequence in W that leads to different output sequences from s and s’.
- Note:
  - we can easily extend this to non-deterministic models.

Relative merits

- If we have a distinguishing sequence then we can use this for every state
- Every (minimal) FSM has a characterization set but we may need to run multiple tests to check a transition
- Practitioners report that many real FSMs have (short) UIOs.

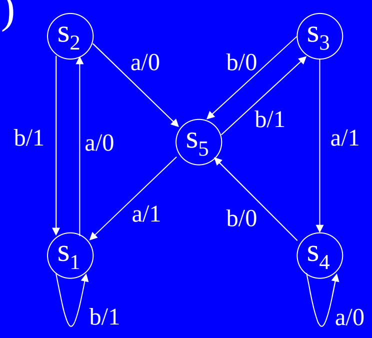
Test generation based on coverage

- In order to test a transition t it is sufficient to:
  - Use a preamble to reach the start state of t
  - Apply the input of t
  - Check the final state of t (if required)
  - Return to the initial state using a postamble/reset
- We can do this for every transition and automate the process.

TTCN-3 User Conference

Example

- To test transition $(s_2,s_3,a/0)$ we could:
  - Apply a to reach s_2
  - Apply input a from the transition
  - Apply the distinguishing sequence aba
  - Then reset



TTCN-3 User Conference

Efficient test generation

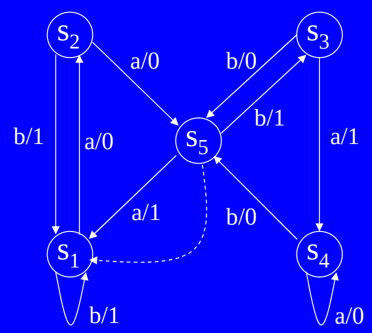
- We could follow a transition test by another transition test.
- We might produce one sequence to test all of the transitions, benefits including:
  - Fewer test inputs
  - Longer test sequence so more likely to find faults due to extra states.

A simple approach

- The following produces a single sequence:
  - Start with the preamble and test for a transition t_1.
  - Now choose another transition t_2 and move to its start state and then add a test for t_2.
  - Repeat until we have included tests for every transition.
- How do we choose a best order in which to do this?

Representing a transition test

- For transition (s_5,s_3,b/0) using distinguishing sequence aba we can add an extra edge:
  - From s_5
  - Input baba
  - To s_1



Solving the optimisation problem

- Our problem can be seen as:
  - find a shortest sequence that contains every ‘extra’ edge.
- This is an instance of the (NP-hard) Rural Postman Problem.
- There is an algorithm that is optimal if:
  - There is a reset to be tested; or
  - Every state has a self-loop
- This approach has been implemented in tools.

Overlap

- The Rural Postman approach produces separate tests for the transitions and connects these.
- However, the transition tests might overlap.
- There are algorithms that utilize this.

Resets

- We may have to include resets in a test sequence.
- It has been found that resets:
  - Can be difficult to implement, possibly requiring human involvement and reconfiguration of a system.
  - Can make it less likely that faults due to additional states will be found.
- However, we can find a test sequence that has fewest resets – and can do so in polynomial time.

A problem with coverage

- No guarantees:
  - Even if we have checked the final state of every transition we may fail to detect faulty implementations.
- This is because:
  - The methods to check states work in the model but might not work in the implementation.
- The (limited) empirical evidence suggests that:
  - These approaches are more effective than transition coverage
  - They often do not provide full fault coverage even if there are no additional states.

Fault Models

- A fault model is a set F of models such that:
  - The tester believes that the implementation behaves like some (unknown) element of F.
- Fault models allow us to reason about test effectiveness:
  - If the system under test passes a test suite T then it must be equivalent to one of the members of F that passes T.
- Similar to *Test Hypotheses* and *mutation testing*.

Test generation using fault models

- The aim is:
  - Produce a test suite T such that no faulty member of F passes T.
- If our assumption is correct then:
  - If the implementation passes T then it must be correct
- So, testing can show the absence of bugs (relative to a fault model).

Fault models for FSMs

- The standard fault model is:
  - The set F_m of FSMs with the same input and output alphabets as the specification/model M and no more than m states, some predefined m.
- A test suite is a *checking experiment* if it determines correctness relative to F_m.
- A checking experiment is a *checking sequence* if it contains only one sequence.

Generating a checking experiment

- There are algorithms for producing a checking experiment using a characterization set:
  - Given fault model F_m and FSM M with n states, these are exponential in n-m.
- There are polynomial time algorithms for producing a checking sequence if:
  - our FSM M has a known distinguishing sequence and m=n.
- However:
  - No known efficient algorithm for producing a shortest checking sequence
  - There is a polynomial algorithm for minimizing the number of resets.

Papers

- These include:

  - A. V. Aho, A.T. Dahbura, D. Lee, and M. U. Uyar, 1991, An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours, *IEEE Trans. on Communications*, 39, 11, pp. 1604-1615.
  - T. S. Chow, 1978, Testing Software Design Modeled by Finite-State Machines. *IEEE Trans. Software Engm* 4 3, pp. 178-187.
  - R. M. Hierons and H. Ural, 2006, Optimizing the Length of Checking Sequences, *IEEE Trans. on Computers*, 55 5, pp. 618-629.
  - R. M. Hierons, 2004, Using a minimal number of resets when testing from a finite state machine, *Information Processing Letters*, 90 6, pp. 287-292.
  - M. Kapus-Kolar, 2007, Test as Collection of Evidence: An Integrated Approach to Test Generation for Finite State Machines, *The Computer Journal*, 50 3, pp. 315-331.

Future work

- Many potential areas:
  - Domain specific fault models.
  - Verifying fault models.
  - Concurrent communicating FSMs.
  - Adding time, …

Testability Transformations for Extended Finite State Machines

Extended finite state machines

- FSMs with:
  - Memory (variables)
  - Inputs with parameters
  - Outputs with parameters
  - Guards on transitions
- Languages such as SDL and Statecharts have more features.

Testing from EFSMs

- One approach is:
  - Choose a test criterion
  - Find a set of paths through EFSM that satisfy the criterion
  - Generate an input sequence for each path.
- Note:
  - FSM techniques produce sequences that test control structure, we can add sequences for dataflow.
- There is a problem: we might choose infeasible paths.

Testability transformations

- We could rewrite the EFSM so that:
  - all paths are feasible; or
  - there is a known set of feasible sufficient paths.
- Note: in general, this problem is uncomputable.

Special case

- The problem can be solved when:
  - All assignments and guards are linear
- Approach has been applied to real protocols (Uyar and co-authors).

General case

- We can split states on the basis of:
  - Transition guards (preconditions)
  - Transition postconditions
- However:
  - Analysis requires us to reason about predicates
  - May lead to exponential increase in number of states.
- Framework has been described but little more.

Estimating feasibility

- A transition can make a sequence infeasible through its guard.
- We might estimate how ‘difficult’ it is to satisfy a guard.
- Use the score for each transition to estimate the ‘feasibility’ of a sequence.
- This can direct us towards ‘better’ sequences.

Initial results

- Experiments with:
  - a simple function that estimates ‘feasibility’
  - two EFSMs
- we get:
  - a correlation between estimate of feasibility and actual feasibility.

Papers

- These include:
  - M. A. Fecko, M. U. Uyar, A. Y. Duale, P. D. Amer, 2003, A Technique to Generate Feasible Tests for Communications Systems with Multiple Timers, *IEEE/ACM Trans. on Networking*, 11 5, pp. 796-809.
  - A. Y. Duale and M. U. Uyar, 2004, A Method Enabling Feasible Conformance Test Sequence Generation for EFSM Models. *IEEE Trans. on Computers*, 53 5, pp. 614-627.
  - R. M. Hierons, T.-H. Kim, and H. Ural, 2004, On The Testability of SDL Specifications, *Computer Networks*, 44 5, pp. 681-700.

Future Work

- Many problems to be solved:
  - Transformations for non-linear arithmetic
  - Domain specific transformations
  - Estimating feasibility using ‘more refined’ information
  - Larger case studies regarding estimating feasibility

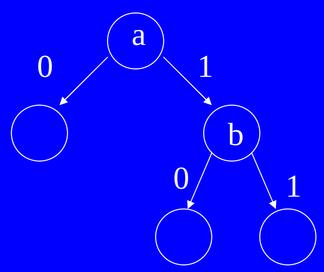
Ordering to reduce the cost of test application

Motivation

- There is a cost associated with running our tests.
- This is a repeated cost due to regression testing.
- If we can reduce this cost/time we can speed up development and the fix/retest cycle.

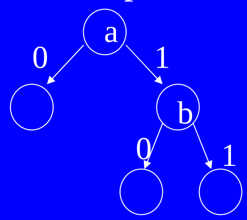
Finite Adaptive Test cases

- Can be thought of as a decision tree.



Formally defining (finite) adaptive test cases

- We will initially consider adaptive test cases that represented by *finite* trees.
- Such an adaptive test case is one of:
  - *null* (apply no input)
  - *(x,f)* for an input *x* and function *f* from outputs to adaptive test cases.



TTCN-3 User Conference

Context

- We will:
  - Assume that the adaptive test cases are already given
  - Assume that we reset between adaptive test cases
  - Focus on minimising the cost of executing our adaptive test cases
- Note: sometimes this is important, but not always!
- We wish to minimise the cost of testing *without reducing its inherent effectiveness*.

TTCN-3 User Conference

Selective regression testing

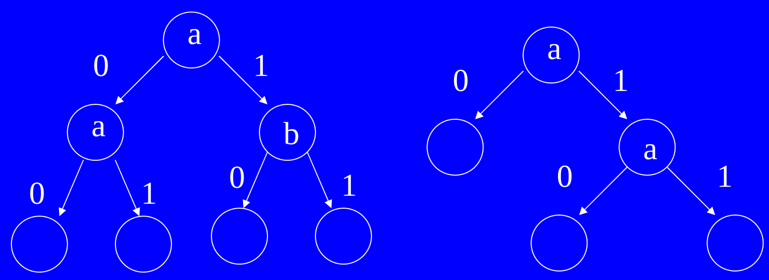
- There are methods that choose a subset of a regression test suite.
- Most based on maintaining coverage.
- Can lead to significant reduction in test suite size but … also a reduction in test suite effectiveness.

Testing deterministic systems

An initial observation

- Suppose we apply adaptive test case γ, observe trace x/y, reset and apply γ again.
- Since the system under test is deterministic we will again observe x/y.
  - There is no need to apply an adaptive test case more than once.
  - If we have already observed trace x/y then we do not have to apply γ.

Test cases can relate



- Here a/0,a/0 for the first adaptive test case tells us that we will get response a/0 to the second (applied after a reset).

So

- We might have adaptive test cases γ_1 and γ_2 such that:
  - There is some *possible* response to γ_1 that would determine the response of the system under test to γ_2.
- We denote this $\gamma_2 \leq \gamma_1$.
- Clearly $\leq$ is not symmetric.
- Note: we (usually) can't just eliminate γ_2 in advance.

Consequence

- The expected cost of testing depends upon the *order* in which the adaptive test cases are to be applied.
- Question: how can we find an order that minimises the expected cost of testing?

Deciding $\leq$

- $\gamma_2 \leq \gamma_1$ if and only if sav(γ_1 ,γ_2) where:

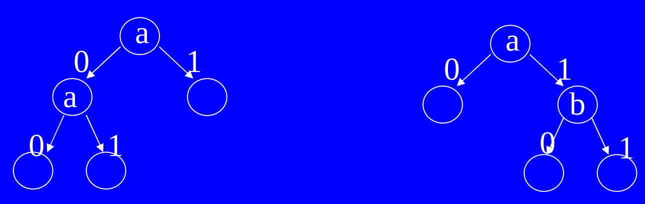
  $sav(\gamma,null) := true$

  $sav(null,(x,f)) := false$

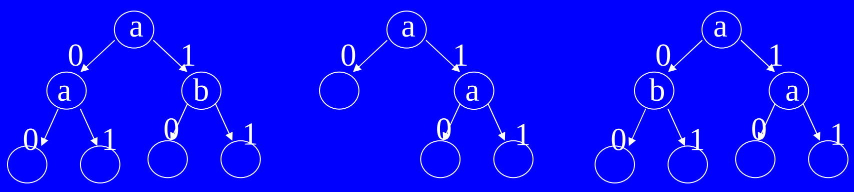
  $sav((x_1,f_1),(x_2,f_2)) := (x_1=x_2) \wedge \; \exists y.sav(f_1(y),f_2(y))$

- Good news: this requires time that is **linear** in the size of the adaptive test cases.

The relation $\leq$ is not antisymmetric



The relation $\leq$ is not transitive

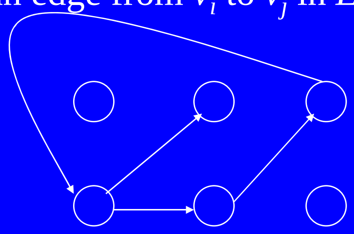


The optimisation problem

- Given our set of adaptive test cases we want to:
  – Find an ordering that minimises the expected cost of testing.
- We can rephrase this as:
  – Find an ordering that maximises the expected saving through not having to apply some of the adaptive test cases.

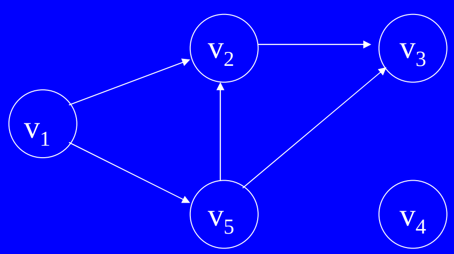
The dependence digraph

- Given set $\Gamma=\{\gamma_1,\ldots,\gamma_n\}$ of adaptive test cases the dependence digraph $G=(V,E)$ is:
  - $V=\{v_1,\ldots,v_n\}$
  - There is an edge from v_i to v_j in E if and only if $\gamma_j \leq \gamma_i$.

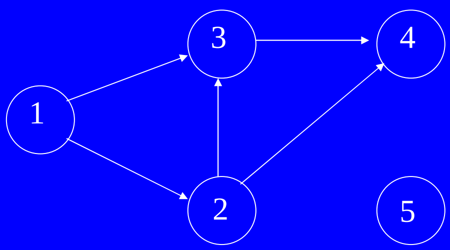


Example

- What is the best ordering given the following dependence digraph?

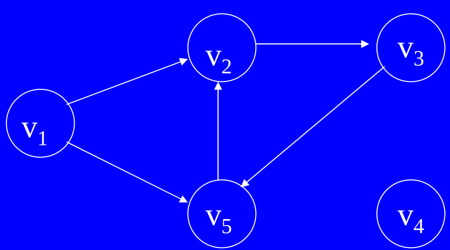


This order?



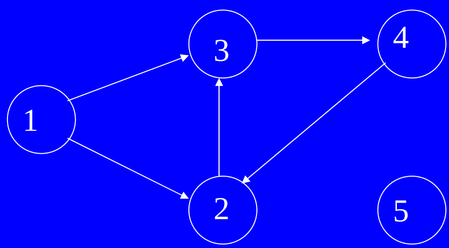
TTCN-3 User Conference

And this one?



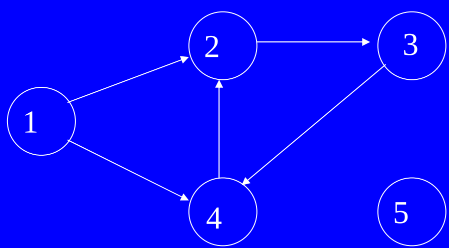
TTCN-3 User Conference

One order



TTCN-3 User Conference

An alternative

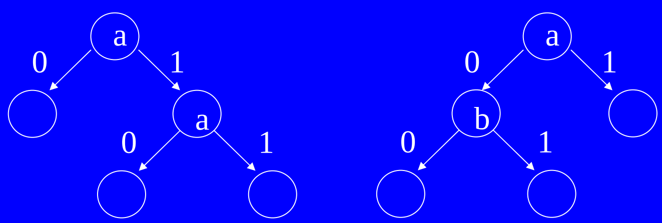


TTCN-3 User Conference

Solving in terms of the dependence digraph

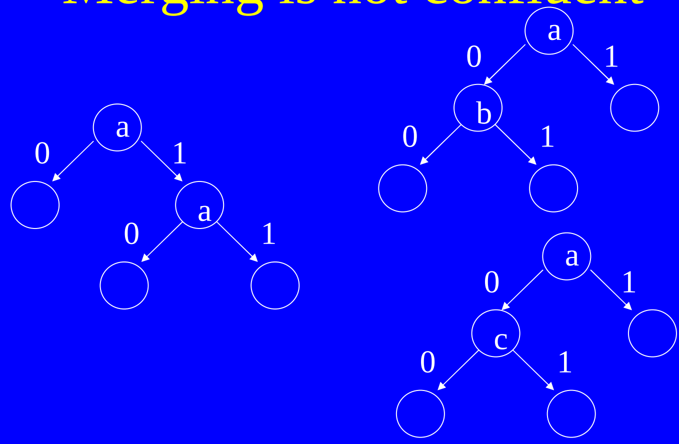
- If we consider *only G* then the optimal order is:
  - – The order that minimises the number of edges that ‘point backwards’
- Finding this is an instance of the Feedback Arc Set (FAS) problem.
- Our problem is NP-hard.

Merging adaptive test cases



- These may be merged

Merging is not confluent



- How can we find a ‘best’ way of merging?

Reducing the size of the problem

- We can:
  - Merge adaptive test cases
  - Separately consider classes of ‘independent’ adaptive test cases.
- Result:
  - these two approaches do not ‘conflict’.

A special case: acyclic dependence digraph

- Here we simply repeat the following until all adaptive test cases have been chosen:
  - Choose a vertex v_i of G with no edge entering it.
  - Add γ_i to the end of the current order and delete v_i and the corresponding edges from G.
- We are finding an ordering based on a DAG.

A simple algorithm

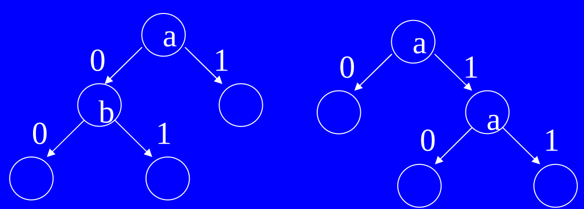
- We can:
  - Solve the FAS problem to find some feedback arc set A.
  - Let $G'=(V,E\backslash A)$
  - Find an order based on G'

Another factor: expected saving

- The potential saving varies.
- So does the likelihood of saving:
  - If $\gamma_2 \leq \gamma_1$, how likely is it that we will have to use γ_2 if we first use γ_1?
- We might estimate the *expected saving* from using γ_1 before γ_2?
- A simple approach: give the dependence digraph weighted edges.

A complication

- There can be a relationship between the edges of the dependence digraph.



- Each is related to the other, but the savings are mutually exclusive.

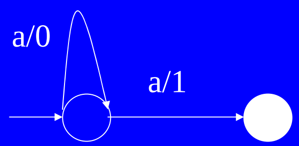
Infinite Adaptive Test Cases

Adaptive test case need not be bounded

- We have assumed that our adaptive test cases are given by finite trees.
- This does not allow us to include tests such as: continue doing ‘x’ until ‘y’ happens and then …
- In order to represent these as trees we need infinite trees.

Tests are FSMs

- An FSM representing an adaptive test case in which we repeatedly apply a, looping until we receive output 1.



- There is one final state.

Languages

- Given an FSM *M* we let *L*(*M*) denote the corresponding regular language.
- If we have FSMs M_1 and M_2 representing adaptive test cases γ_1 and γ_2 respectively then the response to γ_1 can predict the response to γ_2 if and only if:
  - There is a sequence in $L(M_2)$ that is a prefix of a sequence in $L(M_1)$.

A decision procedure

- Given adaptive test cases γ_1 and γ_2 it is sufficient to do the following:
  – Define FSMs M_1 and M_2 representing γ_1 and γ_2 respectively in which every state of M_1 is a final state.
  – The response to γ_1 can predict the response to γ_2 if and only if $L(M_1) \cap L(M_2) \neq \emptyset$.
- This is decidable in polynomial time.

Nondeterministic Implementations

The issue

- We can no longer predict the behaviour of an adaptive test case based on a trace.
- Even if we apply the same adaptive test case again, we can observe a different trace.

Repeating tests

- Suppose we apply an adaptive test case γ 10 times and observe only two traces.
- Is this different from only seeing two traces having applied it 1000 times?

Possible approaches

- We can produce results by either:
  - Making a fairness assumption
  - Assuming that all possible observations have at least a given probability
  - Making no assumptions
- The stronger the assumptions made:
  - the greater the potential for reducing the cost of testing
  - the greater the potential for reducing test effectiveness.

Fairness Assumptions

- We assume that for some predetermined *k*, if we apply an adaptive test case *k* times then we see all possible responses.
- If in testing we apply each adaptive test case *k* times then we can apply analysis similar to that for deterministic implementations.
- We get results similar to the deterministic case.

Bounds on probabilities

- We could assume that:
  - In every state *s* of the SUT and for input *x*, each possible response of the SUT to *x* when in s has probability at least *p* for some fixed/known *p*.
- We can then use results from statistical sampling theory to provide a degree of confidence in having observed all possible traces in response to an adaptive test case.

How it works

- We apply each adaptive test case a sufficient number of times for us to have the required confidence that no other traces can result from its application.
- We can then apply the function defined for the fairness assumption.

Making no assumptions

- We have observed all possible responses of the SUT to γ if we have observed every trace than *any* implementation can produce in response to γ.
- So: the response to γ can be determined by a set of adaptive test cases $\gamma_1,\ldots, \gamma_n$ if and only if:
  - $L(\gamma) \subseteq L(\gamma_1) \cup \ldots \cup L(\gamma_n)$

On-the-fly methods

- There are on-the-fly methods for deterministic implementations
- These will do no worse than the preset methods
- However, they require a more sophisticated environment and additional processing during the application of a test case.

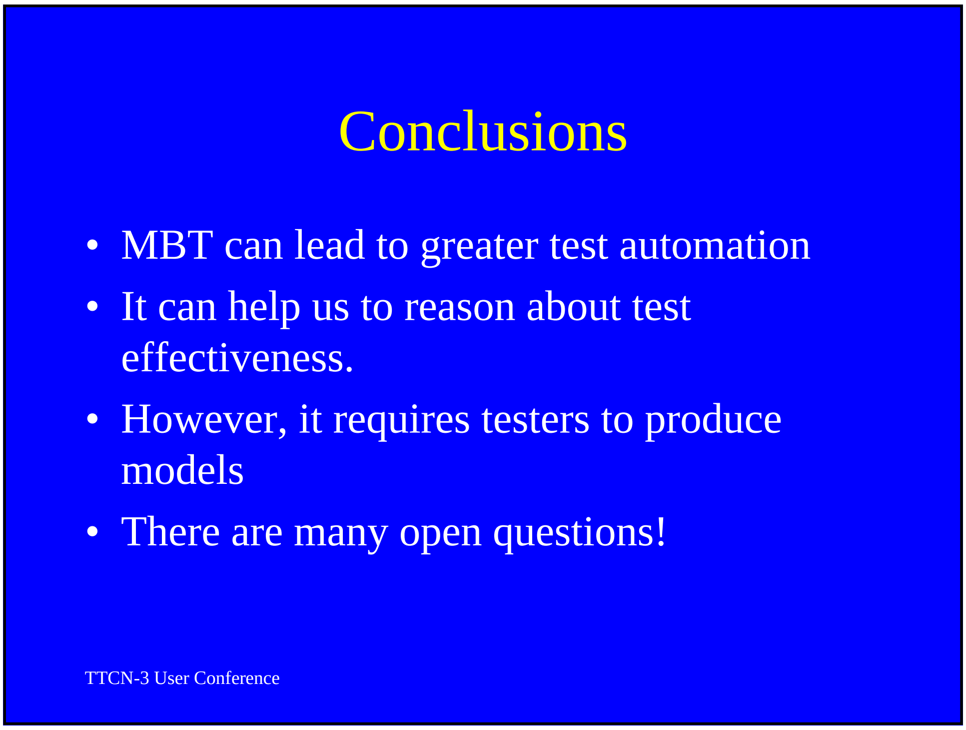
Papers - ordering

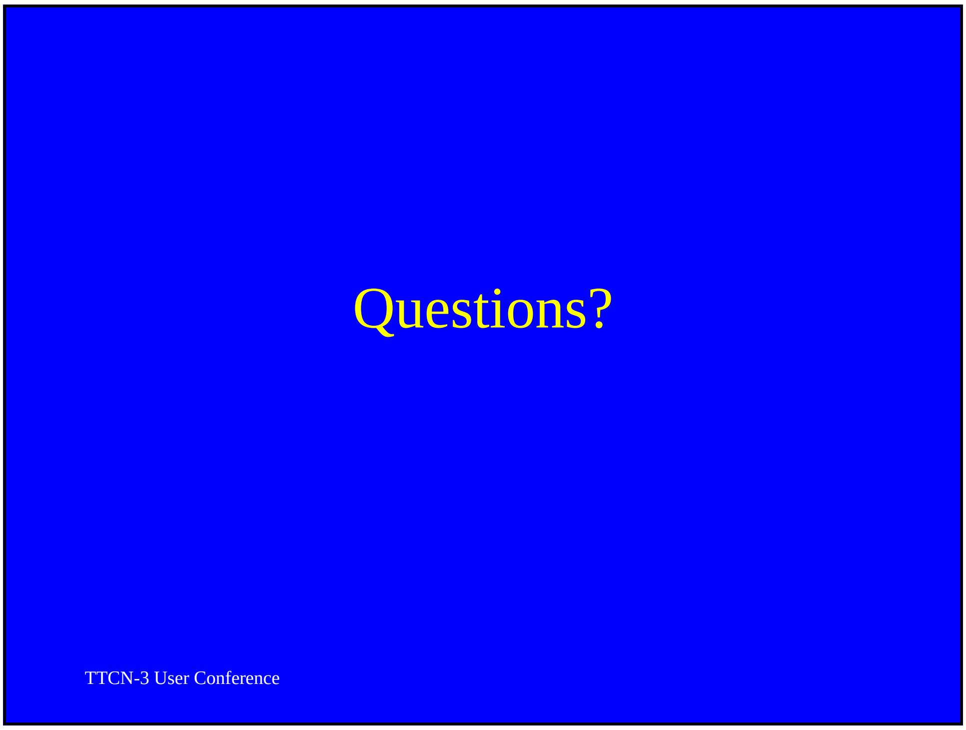
The following are particularly relevant.

- R.M. Hierons and H. Ural, 2003, Concerning the ordering of adaptive test sequences, FORTE.
- R.M. Hierons and H. Ural, 2007, Reducing the cost of applying adaptive test cases, Computer Networks, **51** 1, pp. 224-238.
- R. M. Hierons, 2006, Applying adaptive test cases to nondeterministic implementations, Information Processing Letters, **98** 2, pp. 56-60.
- A. Petrenko and N. Yevtushenko, 2005, Conformance Tests as Checking Experiments for Partial Nondeterministic FSM, FATES.
- Jourdan, G-V, Ural, H., and Zaguia, N., 2005, Minimizing the number of inputs while applying adaptive tests, Information Processing Letters, **94** 4, pp. 165-169.

Future work

- Could include:
  - Optimization using wider range of sources of information.
  - Test cases that are timed, distributed …
  - Empirical studies.
  - On-the-fly with non-deterministic implementations.
  - Combining with selective regression testing?


Conclusions
• MBT can lead to greater test automation
• It can help us to reason about test effectiveness.
• However, it requires testers to produce models
• There are many open questions!
TTCN-3 User Conference


Questions?
TTCN-3 User Conference