Formal Testing of Distributed Systems

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

Work With

- Mercedes Merayo
- Manuel Nunez

- Jessica Chen
- Hasan Ural

Challenges in Testing

- These include:
  - Scale
  - Concurrency
  - Distribution
  - The oracle problem.

- Potential solution, model-based testing:
  - Automate testing on the basis of a formal model or specification.

Model Based Testing

- We only observe interactions between the system under test (SUT) and its environment.
- To reason about test effectiveness we assume:
  - The behaviour of the SUT can be expressed in the same language as the model.

Models for distributed and networked systems

- Such systems typically:
  - Have states and actions
  - Are concurrent

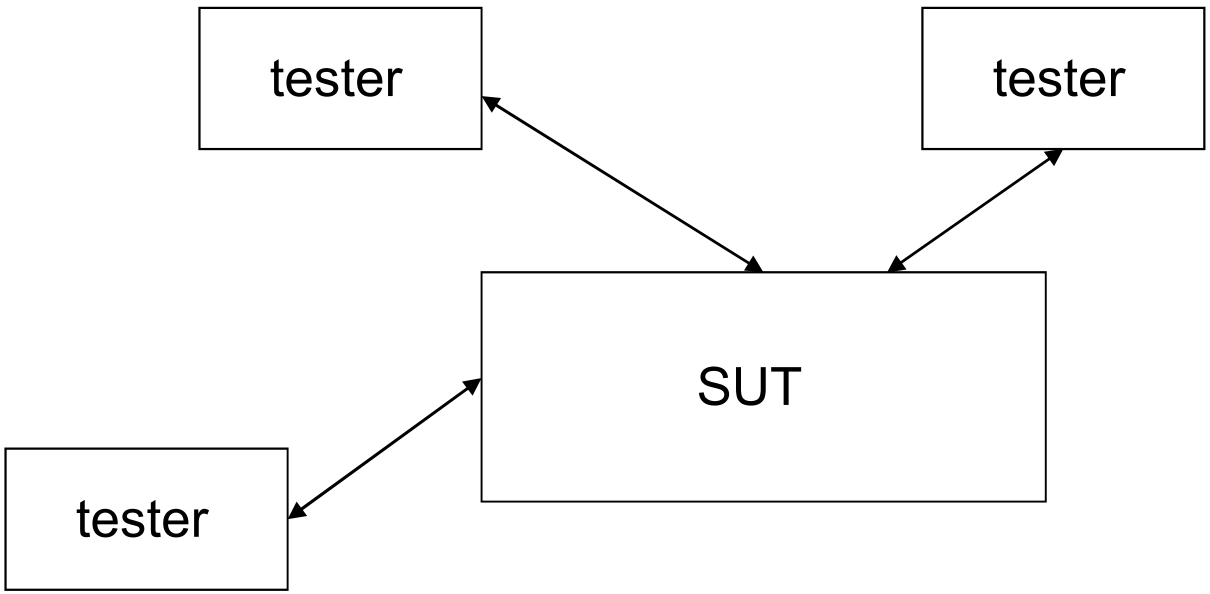
- If we take a black-box view, the last issue is less important

Formal languages

- Typically use states and transitions between states triggered by actions.
- Many can be seen as one of:
  - Finite state machines
  - Labelled transition systems (and input output transition systems)
- Former less general but the models are easier to analyse.

Multi-port systems

- Physically distributed interfaces/ports.
- A tester at each port.



Distributed testing

- Mainly focus on the simplest approach:
  - The testers cannot communicate with one another
  - There is no global clock
  - Observations are ‘local’

Motivation

- Initially just testing/test generation.
- The discussion will be around both
  - *testing* and
  - *implementation/conformance relations*.
- Testing from:
  - input output transition systems and possibly
    - deterministic finite state machines
    - nondeterministic finite state machines

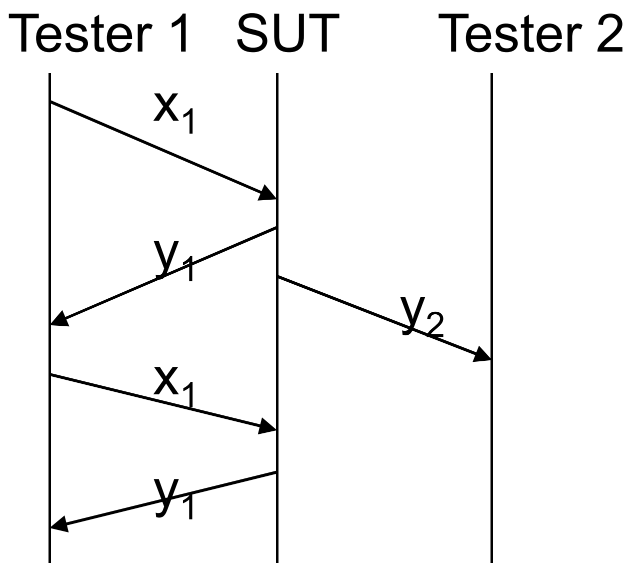
Testing and Observations

Global Traces

- A global trace is a sequence of inputs and outputs.
- We assume there are m ports and:
  - x_p will denote an input at port p (from X_p)
  - $(y_1,\dots,y_m) \in Y$, $Y=(Y_1 \cup \{-\}) \times \dots \times (Y_m \cup \{-\})$, will be an output
- A global trace is an element of $(X \times Y)^*$

Consequences

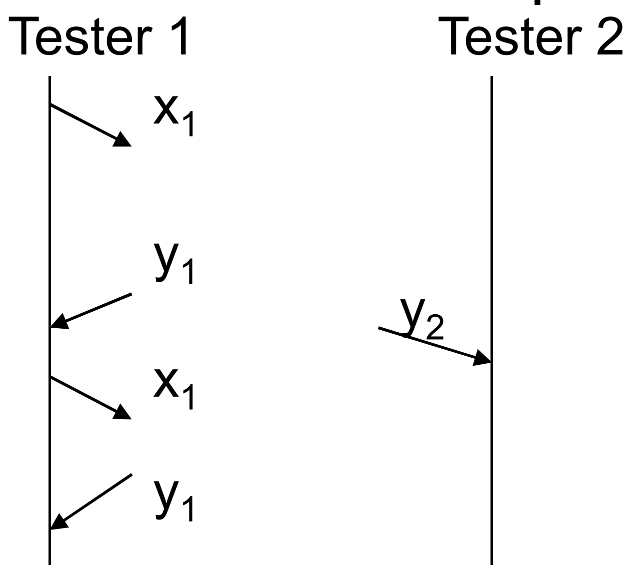
- Each tester observes only the interactions (*local trace*) at its port



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

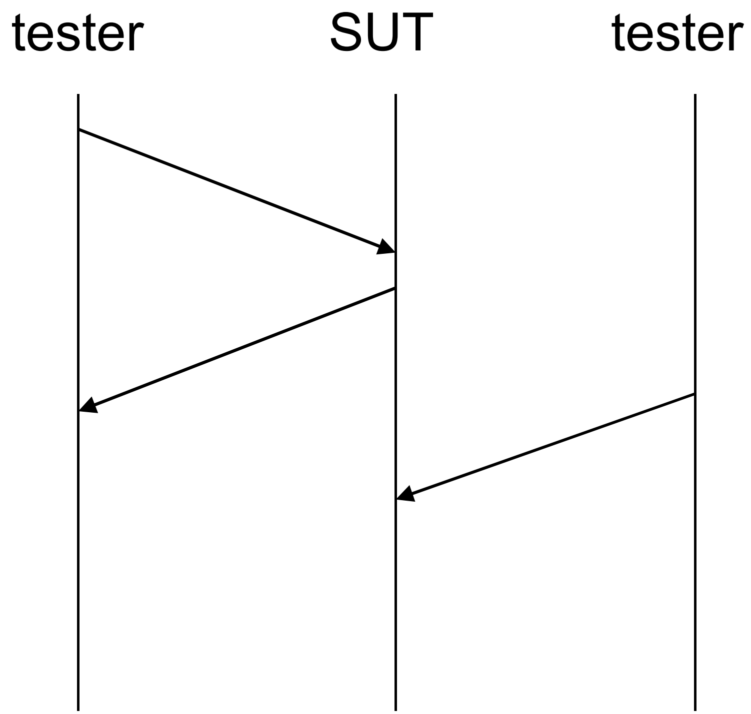
What the testers observe

- Given global trace z, the tester at p observes a local trace $\pi_p(z)$.



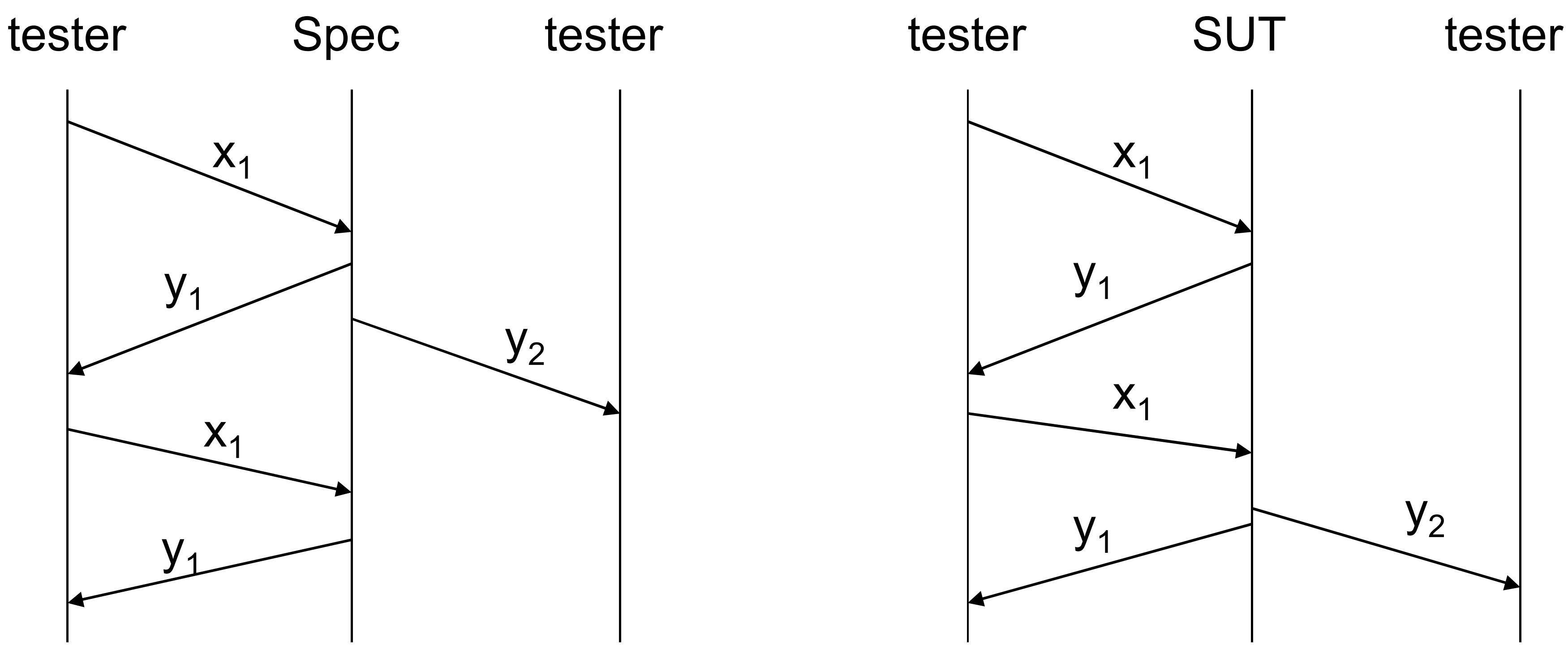
Controllability problems

➢ The following test has a controllability problem: introduces nondeterminism into testing.



Observability problems

- The following look the same



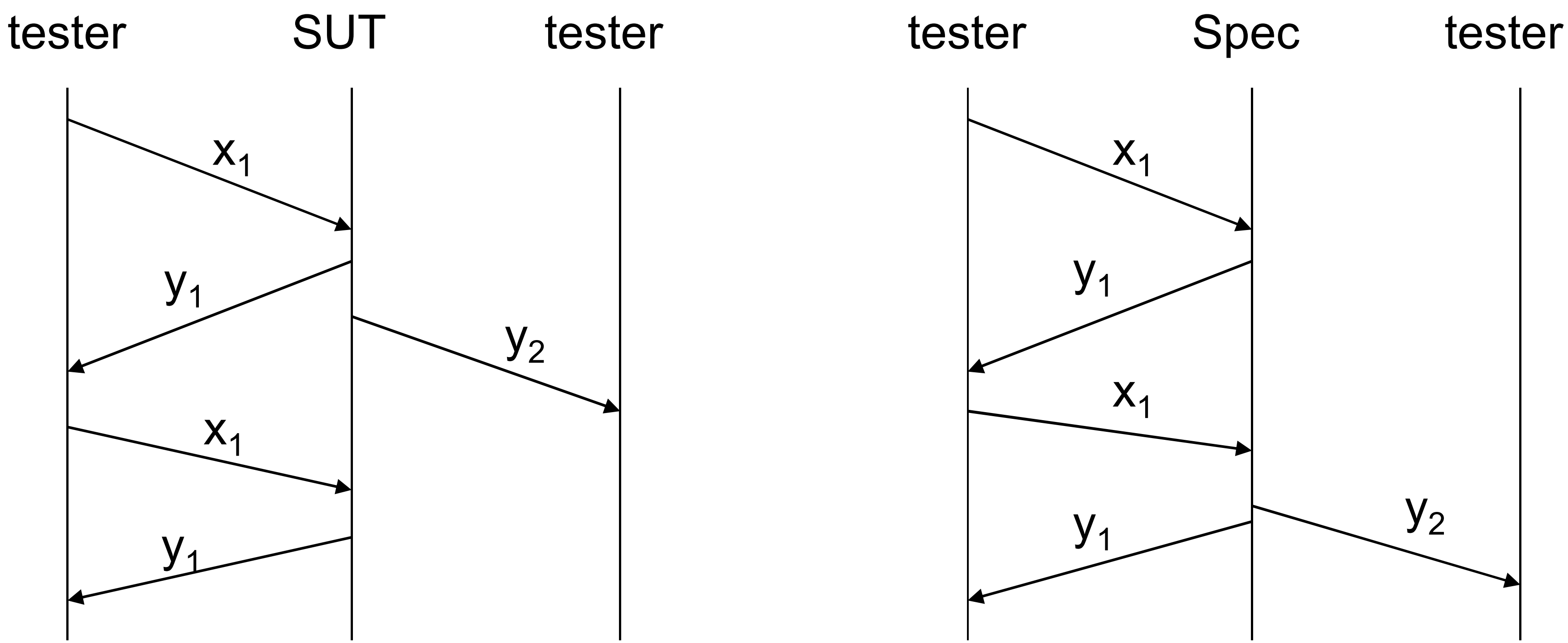
- Testers/users cannot 'map' output to input

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.

Problem: Test effectiveness is not monotonic

- Example: x_1 detects a fault but x_1x_1 does not.



Two approaches to defining implementation relations

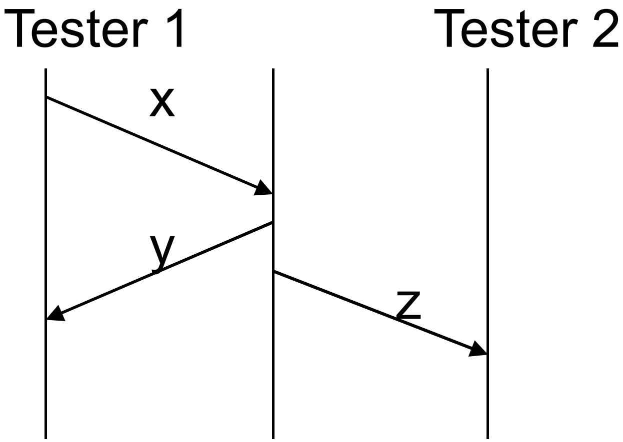
- We might have:

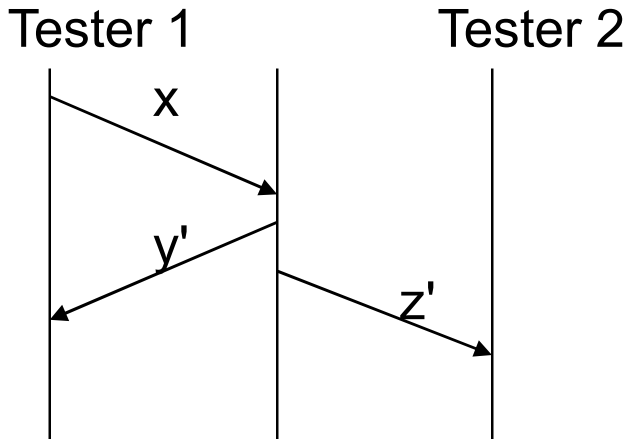
  - Agents at ports are entirely ‘independent’:
    - No external agent can receive information regarding observations at more than one port

  - Or the local traces observed at the ports can be ‘brought together’ later.

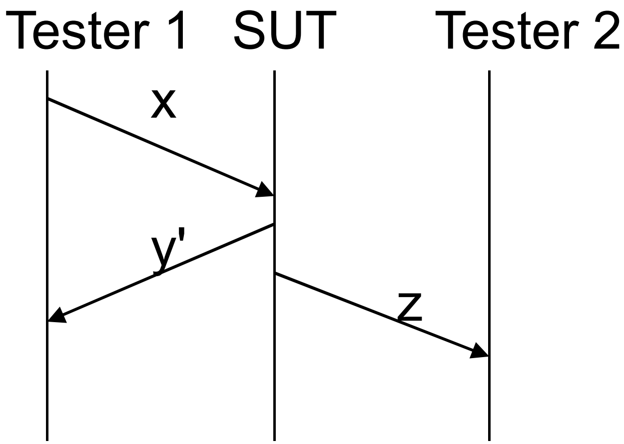
Differences

- Specification



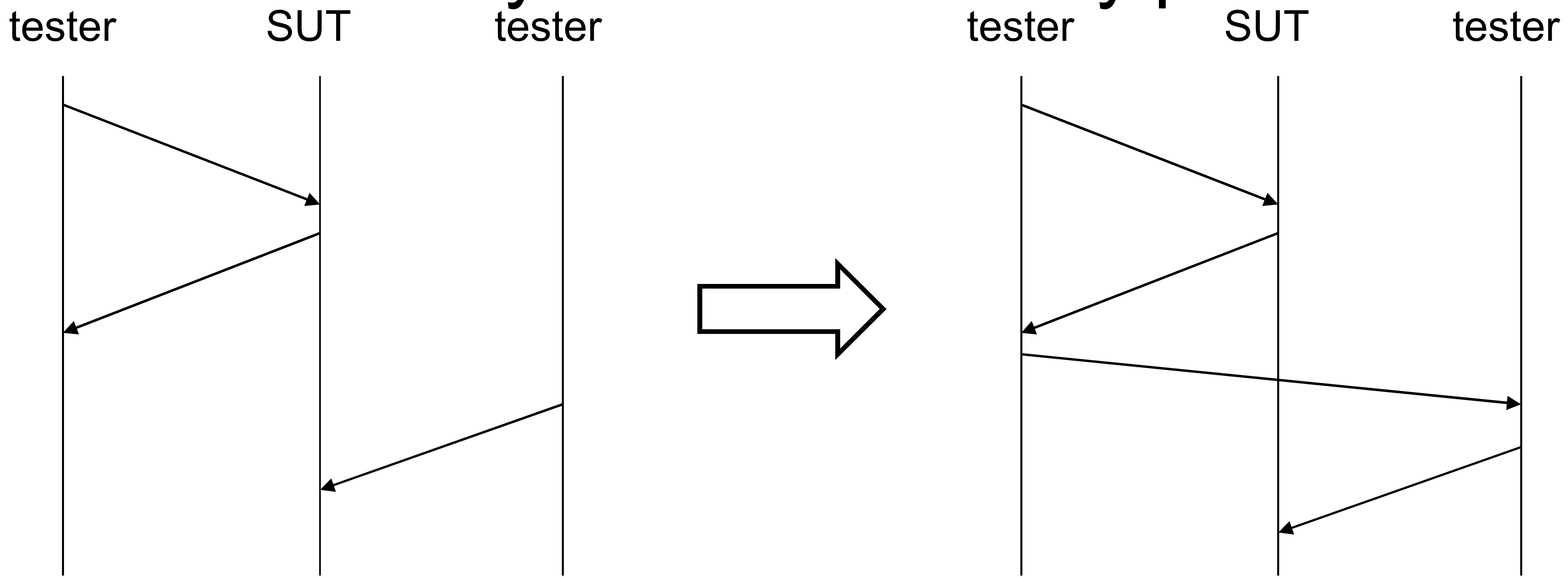


- SUT



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

- Mainly on testing from a deterministic finite state machine (DFSM):
  - 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 can’t be executed without controllability problems.
- 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?
- Using DFSMs is restrictive.

The solution

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

- This gives us new implementation relations.

- We want to test against these.

Input Output Transition Systems (IOTSs)

The models

- These are labelled transition systems in which we distinguish between input and output.
- We have states and transitions between the states.
- Notation:
  - Normally we precede the name of an input by ? and the name of an output by !.

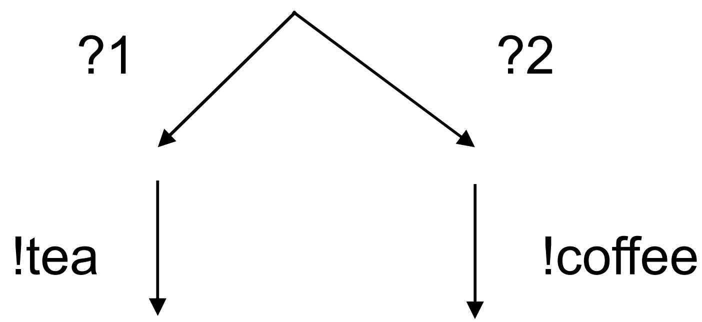
Internal events and quiescence

- We have two special types of events:
  - Internal events (τ) are state transitions that do not require input and do not produce output.

  - A state s is quiescent if from s output cannot be produced without first providing input.
  - If s is quiescent then we add a self-loop transition from s with label δ.

A simple example

- A (very) simple coffee machine



- We have not shown the self-loops for quiescence.

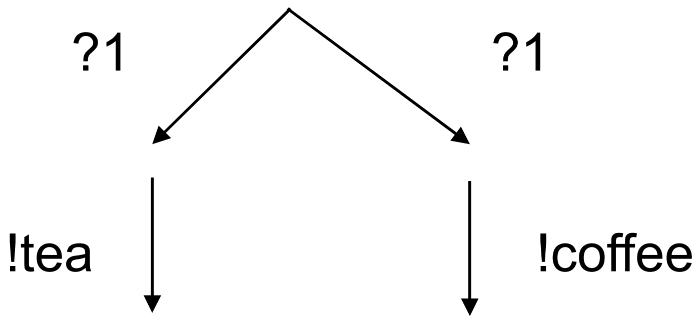
IOTS models

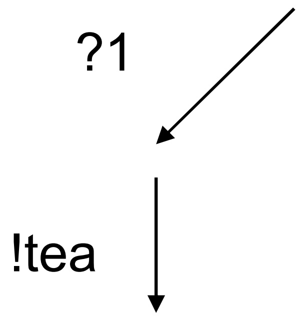
- IOTS models are more general than FSMs:
  - They can be infinite state models
  - Input and output need not alternate
  - There can be internal (unobservable) actions.
- We assume:
  - IOTSs are input enabled
  - We can observe quiescence

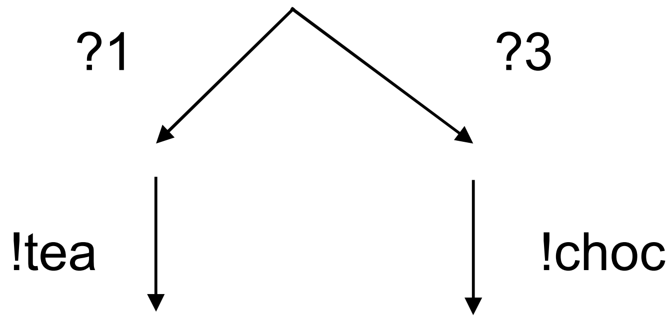
Implementation relations

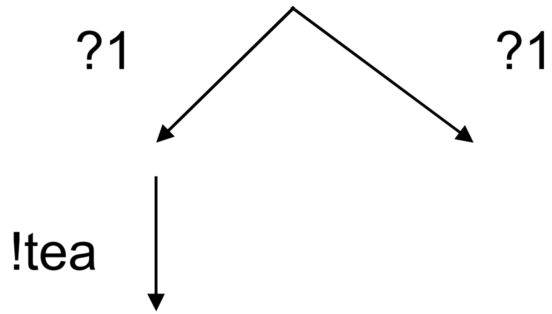
- There is a standard implementation relation (for testing) called ioco
- It requires:
  - If σ is a (suspension) trace of the specification s and the implementation can produce output !o after σ then s must be able to produce output !o after σ

Correct implementations?









Two equivalent processes

- We cannot distinguish the following:

- Note: assume processes completed to make them input-enabled.

Issue

- When can we ‘bring together’ local observations’?

- In this example not after $?i_1!o_1$ or $?i_1!o_2$

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 IOTSs? 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' ~ z.
- This means:
  - If i has a 'run' z that ends in quiescence then s has a specified behaviour that is 'equivalent' to z.

dioco does not imply ioco

- Example:

Result

- If s and i are input enabled then:
  - i ioco s implies that i dioco s
- Normally IOTS implementations are required to be input enabled.
- So:
  - For input enabled specifications we have that dioco is weaker than ioco.

Test cases

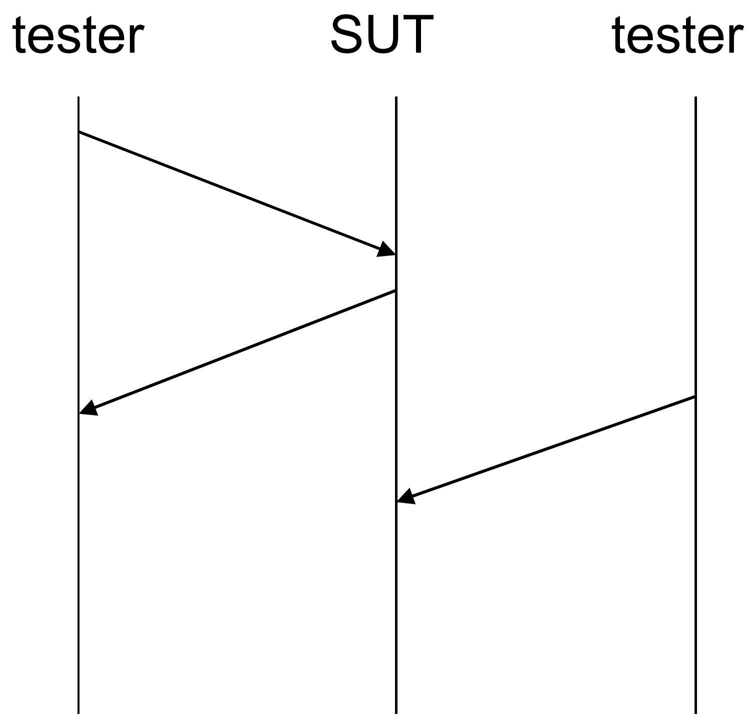
- These can be defined as processes that can interact with the SUT.
- We can have:
  - A global tester that interacts with every port
  - One local tester for each port.
- In our context, we cannot implement a global tester (but we can map it to a set of local testers).

Controllability

- A local tester observes only the events at its port.
- As a result, if it has to supply an input then it can only know when to do this on the basis of its observations.

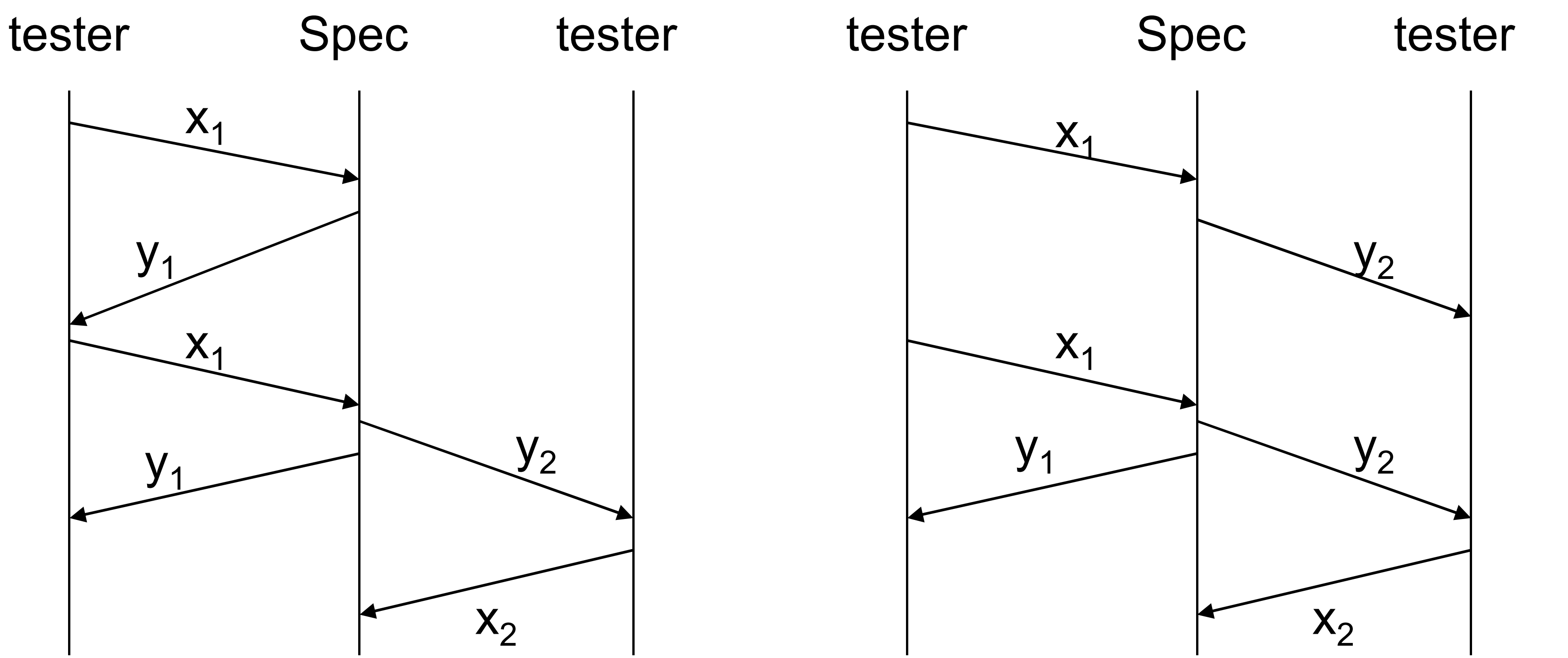
A controllability problem

- The tester at port 2 does not know when to send its input.



The effect of nondeterminism

- 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’.
- Also studied in the context of Message Sequence Charts (e.g. non-local choice pathologies).
- Difference in problems considered and our problem has additional ‘structure’

Defining controllability

- A test case t is controllable if each tester can make 'local choices'
  - there should not be two prefixes z and z' of traces that can be produced using t that look the same to a tester at port p and yet this tester should behave differently after these.
- 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.

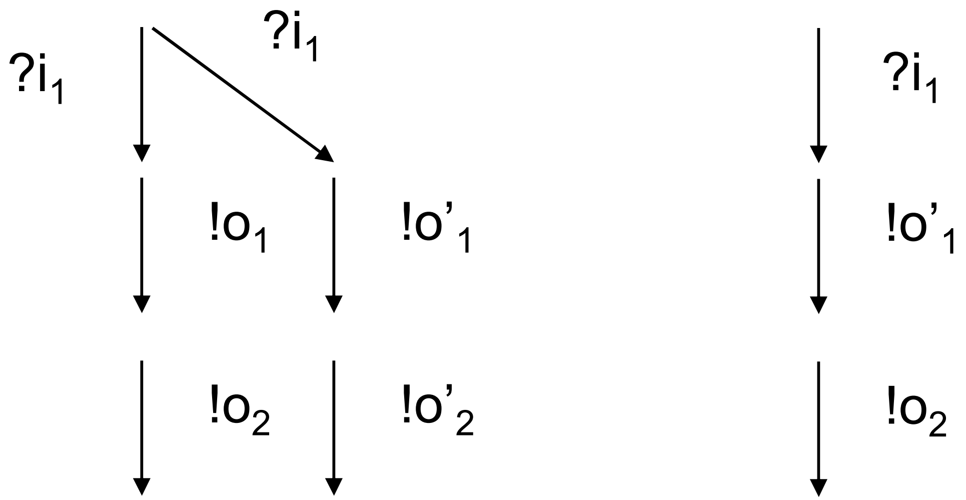
An example

- 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

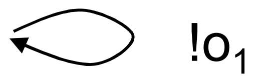
- Specification and implementation

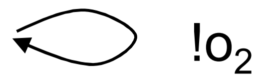


- Looks ok if we cannot bring together local traces.

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.

Using infinite traces

- We can compare the infinite traces of the implementation with those of the specification.
- This is an answer to ‘when do we bring together local traces’.
- In practice we will have to define conservative decision procedures for oracles.

Other Types of Models

The following are equivalent

- $!o_1!o_2$, $!o_2!o_1$
- $!o_1!o_1!o_2$, $!o_2!o_1!o_2$
- ….
- $(!o_1)^{1000}!o_2$, $!o_2(!o_1)^{1000}$
- ….

- When does this stop being reasonable?

One possible approach

- We could include time in our model.
- Problem:
  - Local clocks need not synchronise.
- We might have e.g.:
  - bounds in drift,
  - information about time taken by messages,
  - messages between testers
- This is future work.

Using scenarios

- An alternative:
  - Allow the users and testers to effectively synchronise at certain points.
- We can
  - consider *scenarios* and;
  - add explicit synchronisation points in a specification.

Adding probabilities

- Some systems have probabilistic requirements.
- We can add probabilities to transitions.

- It is straightforward to extend IOTSs to probabilistic IOTSs.

A Generative Approach

- In a state s the sum of probabilities of transitions leaving s add up to 1.
- The implementation relations are similar to dioco – we just add requirements regarding probabilities.

- However, if we have inputs and outputs this approach requires us to have probabilistic information regarding the environment.

A reactive/generative approach

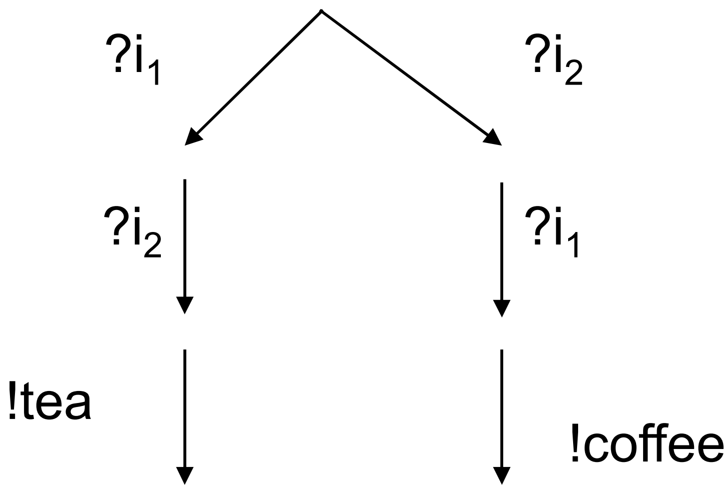
- Instead we can assume that:

  - There is no probabilistic information regarding inputs from the environment (a reactive approach).

  - In state s, the sum of the probabilities of outputs from the SUT (including δ) is 1: outputs are generative.

Probabilities of observations

- Consider the following



- What is the probability of observing !coffee after $?i_1?i_2$

The problem

- We can have races between events at different ports.
- We have no probabilistic information regarding the outcome of these races.

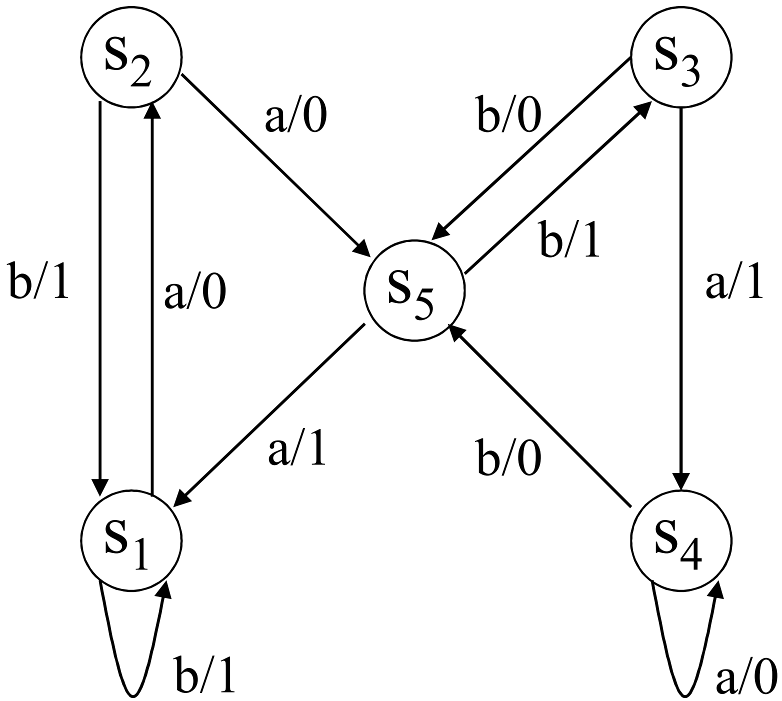
Possible solutions

- Two alternatives:
  - Outlaw such situations (effectively say that we know nothing about the probabilities).
  - Assume that the (unknown) environment has such probabilities and define corresponding implementation relations.

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



An implementation relation for distributed systems

- We say that DFSM N conforms to DFSM M if:

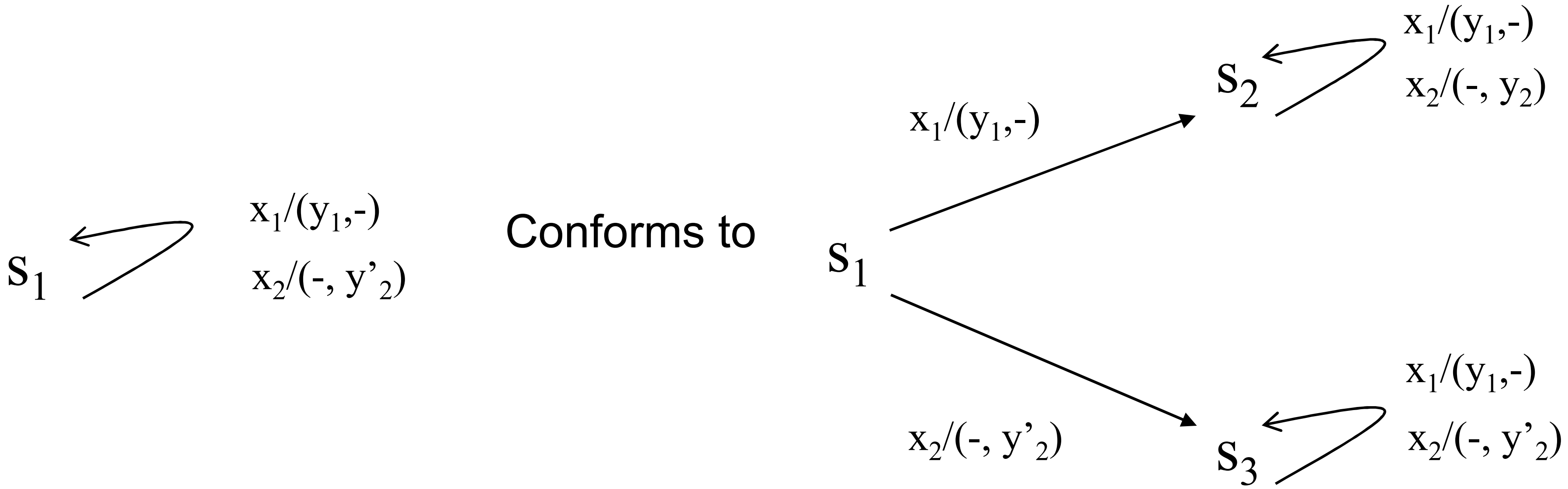
  - Every global trace of N is indistinguishable from a global trace of M.

- 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 can have output y_2).



- Is the first an acceptable design for second?

Key components of testing

- When testing from an FSM we want to be able to:
  - Reach states
  - Distinguish states (and machines)
  - Check output against the specification (oracle problem).

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

Reaching and distinguishing states

- Problem
  - Is there a strategy for each tester that leads to testing taking the FSM to a particular state (or distinguishes two states)?
- This problem is undecidable.
- Decidable for controllable testing from a DFSM (result does not hold for NFSMs).

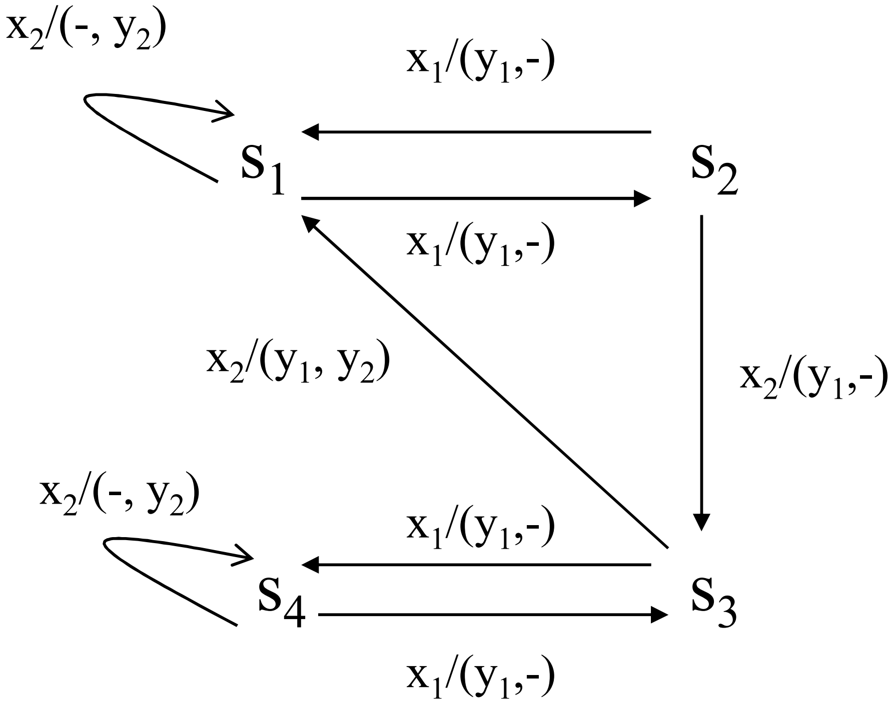
Controllable testing

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

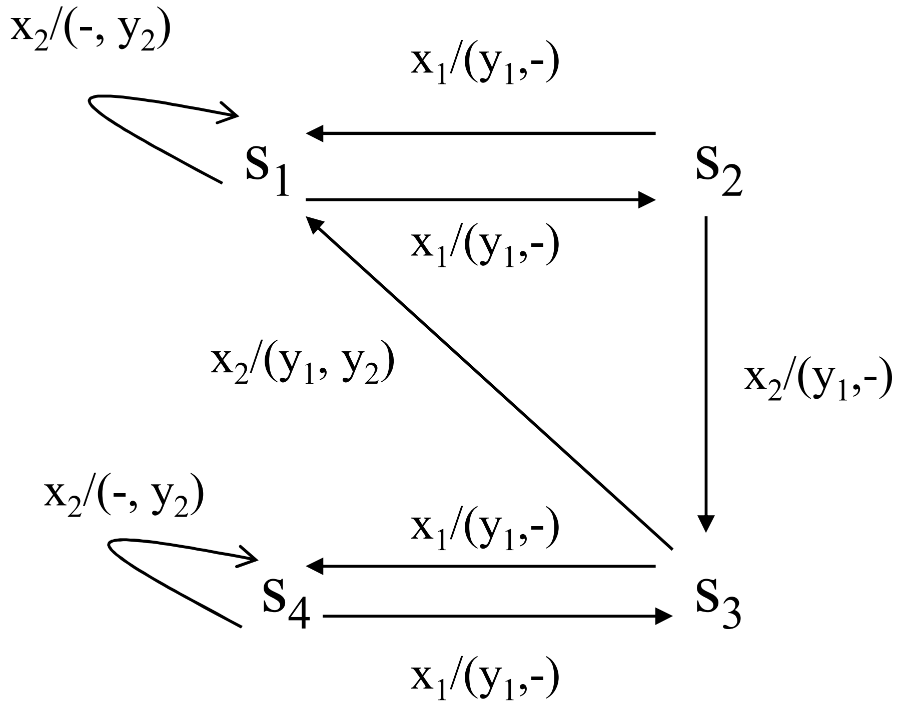
- Given port p and states s_1 and s_2 of a m-port FSM M with n states:
  - s_1 and s_2 are locally s-distinguishable by an input sequence starting at p if and only if they are locally s-distinguished by some such input sequence of length at most m(n-1).
- This bound is 'tight'.
- The sequences can be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

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.

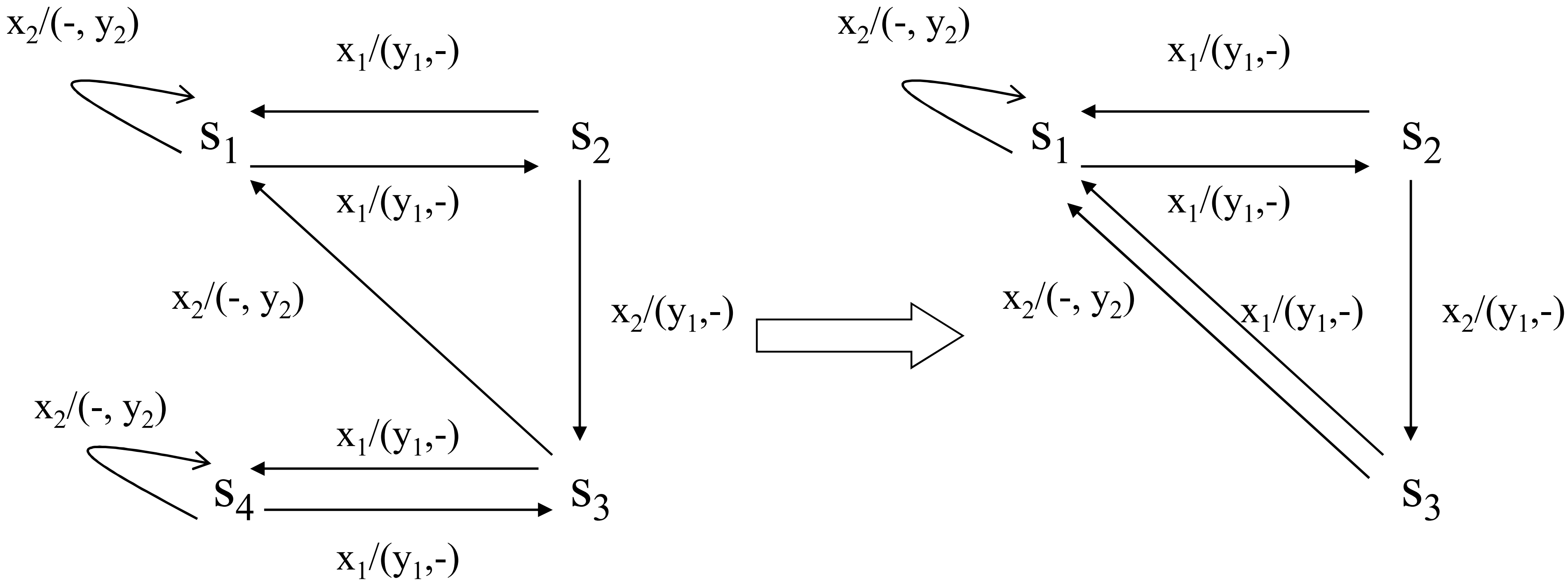
Minimal DFSMs are not always locally s-minimal

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



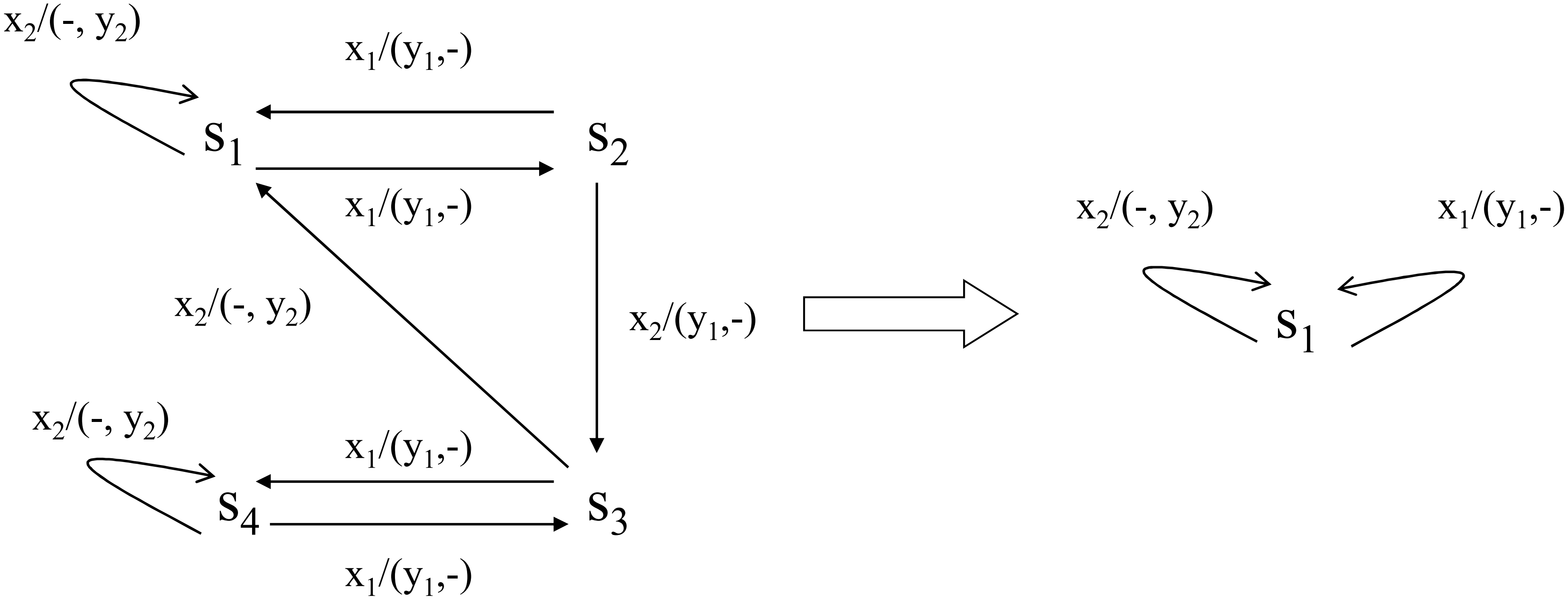
Merging s-equivalent states

- A smaller acceptable design?



Minimising: smallest FSM

- Even smaller:



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

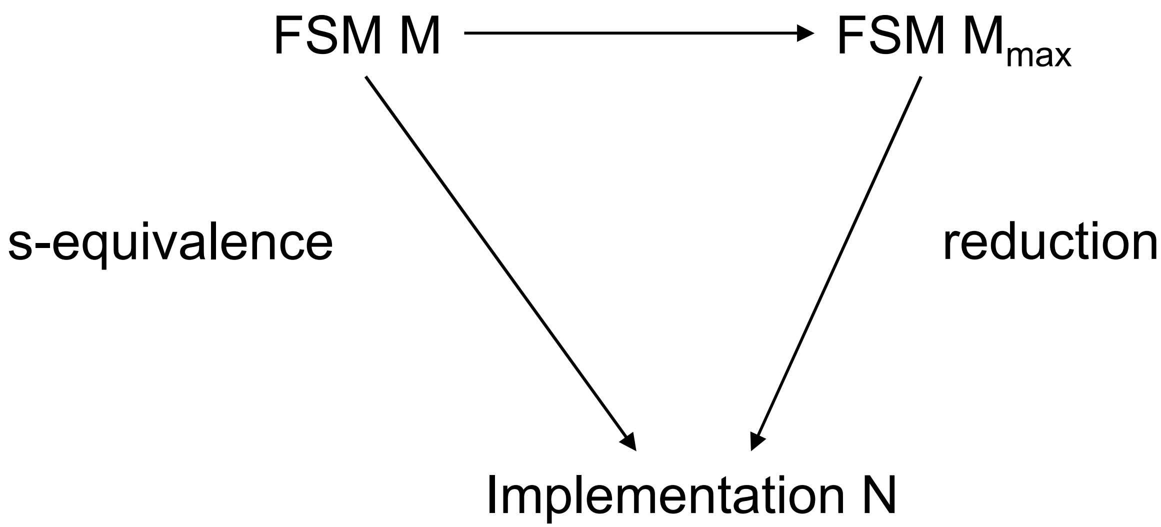
- Given DFSM M, we can find:
  - Maximal M_{max} that is locally s-equivalent to M
  - Minimal M_{min} that is locally s-equivalent to M
- We can find them efficiently.

Results

- DFSM N is locally s-equivalent to DFSM M if and only if N is a reduction of M_{max}.
- The set of DFSMs that are s-equivalent to a DFSM M forms a bounded lattice.

Refinement and testing

- We now know that:



Summary: controllable testing

- Benefits of restricting to controllable test sequences for DFSMs
  - 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?
- However: limits testing

Future work

- Generating test cases to satisfy a test criterion.
- Generating complete test suites.
- Minimising an FSM.
- Testing using coordination messages but the ‘new’ implementation relations
- Timed models.
- Enriching models with data, stochastic time, ...

Papers (FSMs)

- B. Sarikara and G. Von Bochmann, Synthesis and Specification Issues in Protocol Testing, *IEEE Transactions on Communications*, 3**2** 4, pp. 389-395: 1984.
- R. Dssouli and G. von Bochmann. Error detection with multiple observers, *Protocol Specification, Testing and Verification V*, pp. 483-494: 1985.
- R. Dssouli and G. von Bochmann,. Conformance testing with multiple observers, *Protocol Specification, Testing and Verification VI*, pp. 217-229: 1986.
- J. Chen, R. M. Hierons, and H. Ural. Overcoming observability problems in distributed test architectures*, Information Processing Letters*, **98**, pp. 177-182: 2006.
- R. M. Hierons and H. Ural. The effect of the distributed test architecture on the power of testing, *The Computer Journal*, **51** 4, pp. 497-510: 2008.
- R. M. Hierons: Canonical Finite State Machines for Distributed Systems, *Theoretical Computer Science*, **411** 2, pp. 566-580: 2010.
- R.M. Hierons: Reaching and Distinguishing States of Distributed Systems, *SIAM Journal of Computing* (to appear)

Papers (IOTSs)

- R. M. Hierons, M. G. Merayo, and M. Nunuez. Implementation relations for the distributed test architecture, *20th FIP International Conference on Testing Communicating Systems (TestCom 2008)*, LNCS 5074, pp. 200-215: 2008.
- R. M. Hierons, M. G. Merayo, and M. Nunez. Controllable test cases for the distributed test architecture, *6th International Symposium on Automated Technology for Verification and Analysis (ATVA 2008)*, LNCS volume 5311, pp. 201-215: 2008.
- R. M. Hierons and M. Núñez: Scenarios-based Testing of Systems with distributed Ports, *The 10th International Conference on Quality Software (QSIC 2010)*, 2010.
- R. M. Hierons and M. Núñez: Testing probabilistic distributed systems, *30th IFIP Formal Techniques for Networked and Distributed Systems (FORTE 2010)*, LNCS, 2010.

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?