How Does Nondeterminism Occur in Test Models and What Do We Do with It?

Alexandre Petrenko
Centre de Recherche Informatique de Montréal
CRIM, Canada
alexandre.petrenko@crim.ca

MBT 2014, Grenoble

Acknowledgements
Apologies & Disclaimer

- A. Petrenko thanks A. Petrenko
- This sounds improperly reflexive
- Apologies for the violation of the naming convention
  - Alexander K. Petrenko, aka
    - Alexander Petrenko, A. K. Petrenko
    - Александр Константинович Петренко
  - Alexandre Petrenko, aka
    - Alex Petrenko, A. F. Petrenko
    - Александр федорович Петренко
- If one finds a wrong A. Petrenko, we apologize for any inconvenience it may cause
- Disclaimer: we try to make sure that our citation indexes grow independently

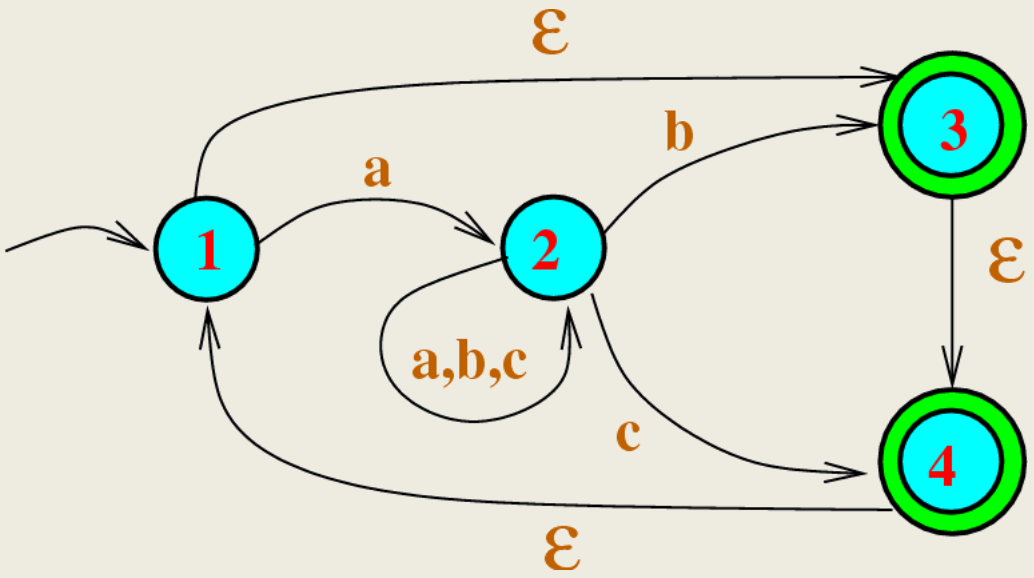
Scope

- I will focus on nondeterminism of test models in MBT (*nondeterminism is abbreviated to ND*)
  - What?
  - Where?
  - Why?
  - How?
- I will not discuss many other things related to ND, e.g., probabilistic models
- Strong statements if made are used to help some of you fight jetlag and/or open discussions in workshop

Basic Models

- NFA
- LTS, IOTS
- NFSM
- EFSM
- Sequence diagrams, MSC

NFA, Nondeterministic Finite Automaton



NFA with instantaneous transitions

NFA has an equivalent DFA

LTS, IOTS

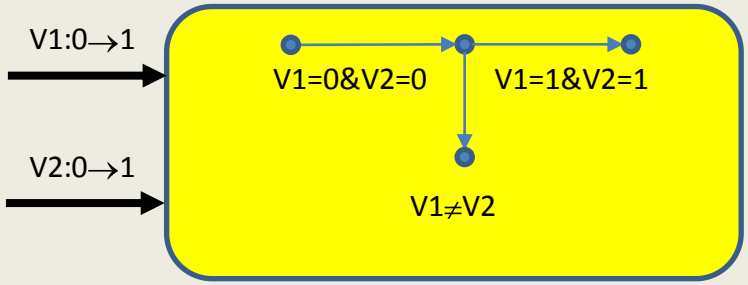
- ND LTS is NFA with all states accepting
- IOTS, aka IOLTS refines LTS by partitioning the alphabet into input and output actions
- ND IOTS
  - Multiple initial states
  - Underlying automaton is NFA, tau transitions
  - Output transitions emanating from state (even in case of DFA) differently from input transitions
  - I/O conflicts, input and output transitions emanating from state
  - Non-catastrophic divergence
    - Output divergence
    - Cycle of tau transitions, livelock
- Extended also with clocks, guards and assignments on variables, which may also induce or be used to specify ND, though some ND in untimed models is resolved in timed models

ND FSM

- Transitions are labeled by Input/Output pair
- No tau transitions
- ND occurs when
  - FSM has several initial states
  - Underlying automaton is NFA, FSM is non-observable, which can be transformed into observable, by NFA2DFA
  - Transitions emanating from state have same input, but different outputs
- ND FSM can be unfolded into ND IOTS
- ND IOTS can be folded into ND FSM if it has no tau transitions, divergence, I/O conflicts

EFSM

- Extensions with guards is a source of ND, when guards are not disjoint
- Extensions with timers is a source of ND, when timeout occurs with input event
- Deployed in an asynchronous platform, input variables changes may result in ND, since intermediate values can trigger a wrong transition (input races in asynchronous hardware)



ND in Sequence Diagrams

- Parallel fragments and co-regions
- The alternative fragments represent choices that the implementation may choose between in order to conform to the specification
- Race conditions and weak sequencing
- Implied scenarios - inconsistency or ND?
- Process divergence and non-local choice

Process Divergence
Non-local Choice

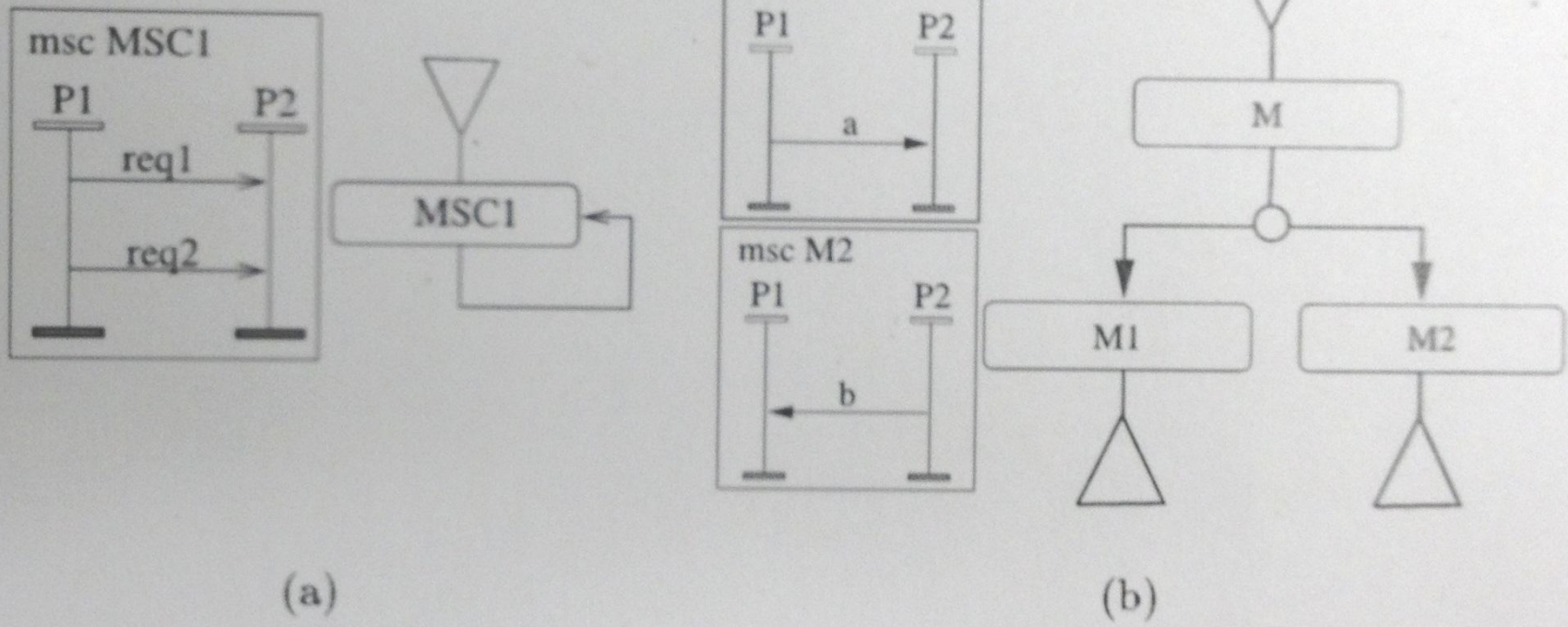


Fig. 1. MSC specification with: (a) process divergence; (b) non-local branching choice

Design and Test Models

- Test models differ from design models
  - Behavior testable with a given tester
  - Assumptions of a test engineer/modeller
- Design models have often semantics defined by a commercial tool
- Tools supporting testing developed by other companies need to use test models with a clear semantics
- Open source tool development is gaining the momentum

How Does ND Occur?

- Sources intrinsic to IUT, as an atomic unit for testing
  - Inherent ND; discussions on “can a commercial software product be ND?” are out of scope
  - IUT is composed from deterministic modules
- Sources extrinsic to IUT
  - SUT where IUT is embedded (integration testing)
  - Test environment
  - Test modeller

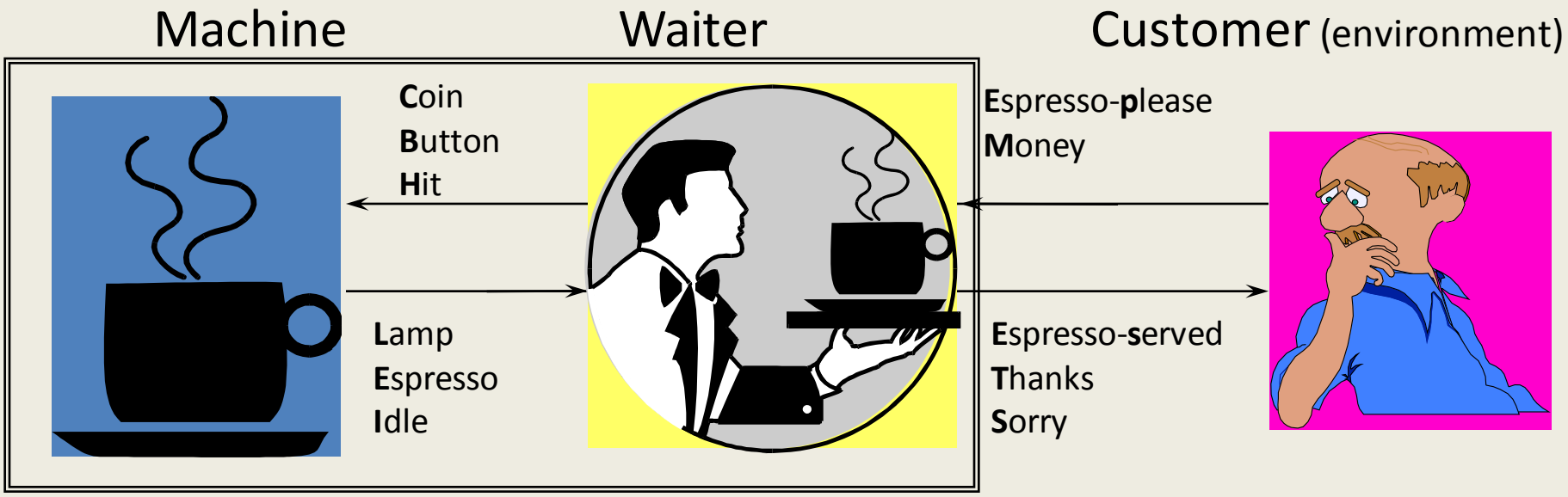
ND in Modular IUT

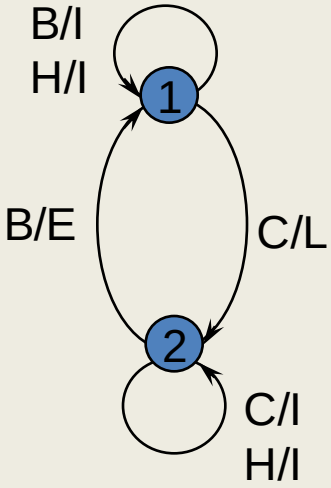
- Message-based communication
  - Asynchronous interactions via queues
  - Races lead to ND
  - The environment providing inputs in transient (non quiescent) states trigger ND
- Shared variables communication
  - Intermediate output valuations if available to the environment may appear ND

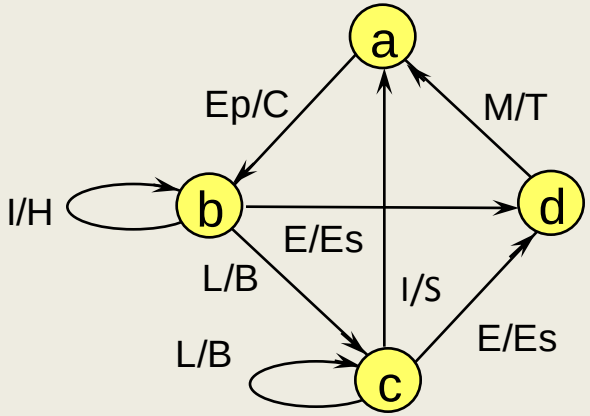
Extrinsic: IUT is in SUT

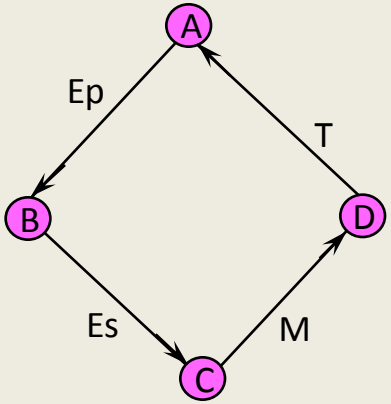
- SUT is modelled by communicating modules and IUT is only one of them to be tested, the other are assumed to be fault-free, they constitute the context for IUT
- Embedded testing, aka testing in context
- Context creates controllability and observability problems
- With the context equally processing different IUT behaviors a test model of IUT is ND (example provided)

Coffee Shop SUT

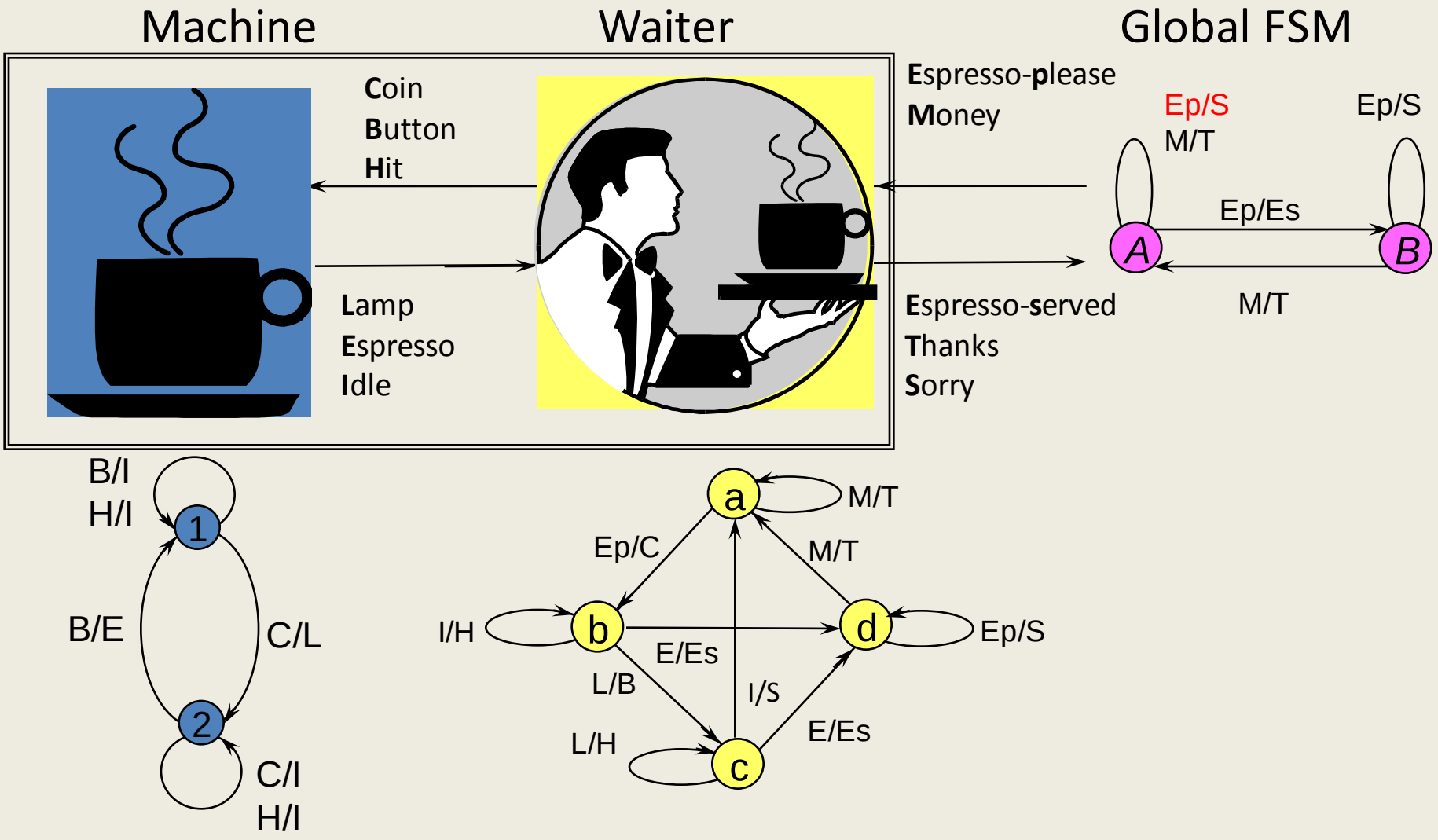






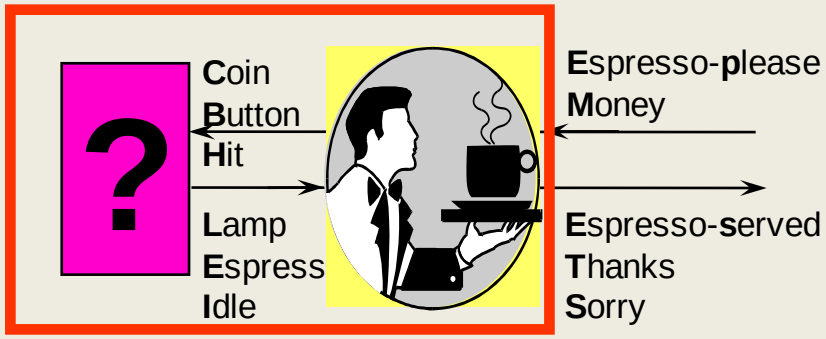


Composing FSMs

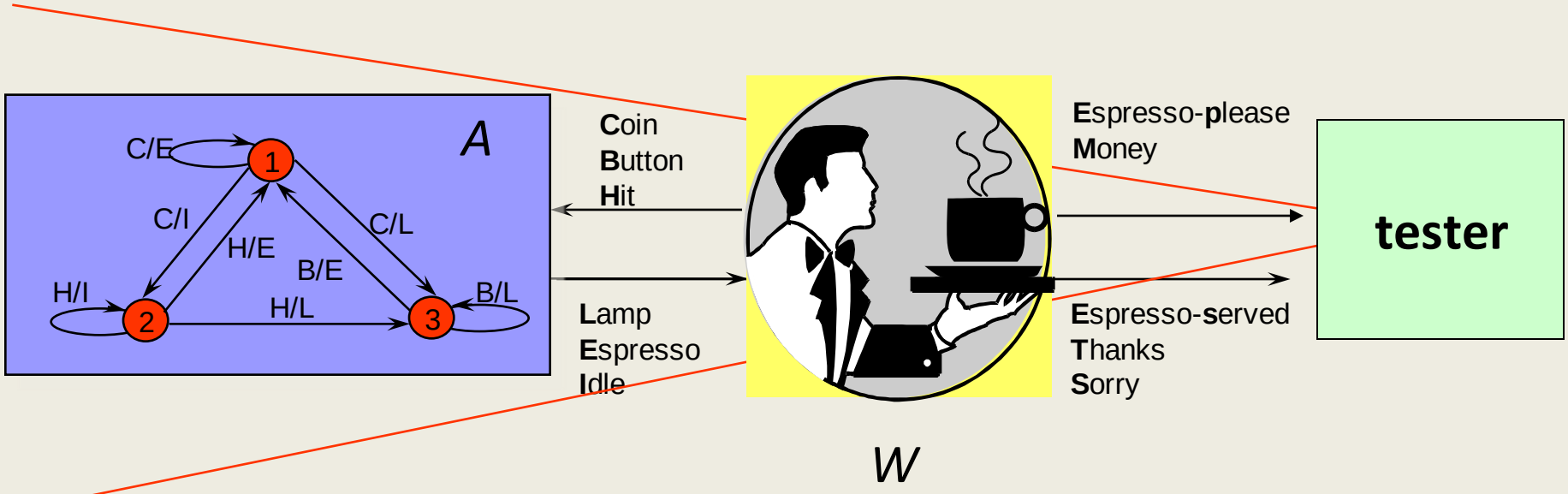


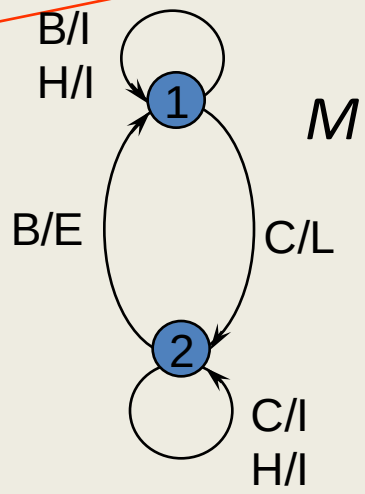
Embedded Testing

- Assume we can access the coffee machine only via the waiter
  - it is not directly controllable
  - it is not directly observable
- What is the test model to check the conformance?
- It should allow IUT to have fault tolerated by the context



ND Test Model of Coffee Machine



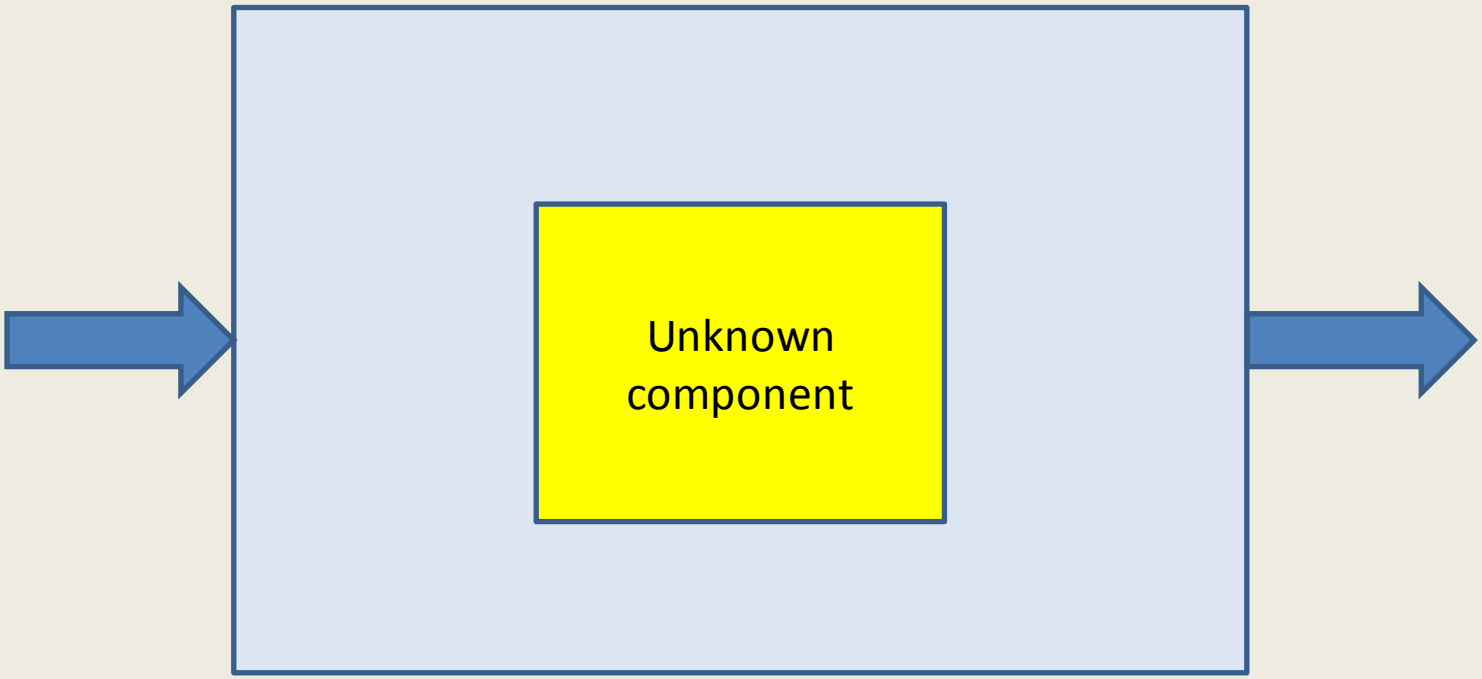


Tester cannot distinguish *A* from *M* since *M* composed with *W* gives the same global FSM

IUT appears ND to the tester

The Problem of Unknown Component

Combined with a known part of a system
it must satisfy a given system's specification



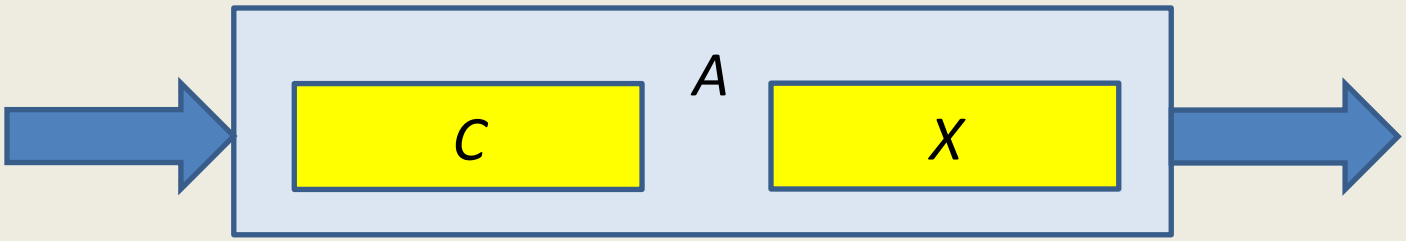
Unifying Framework

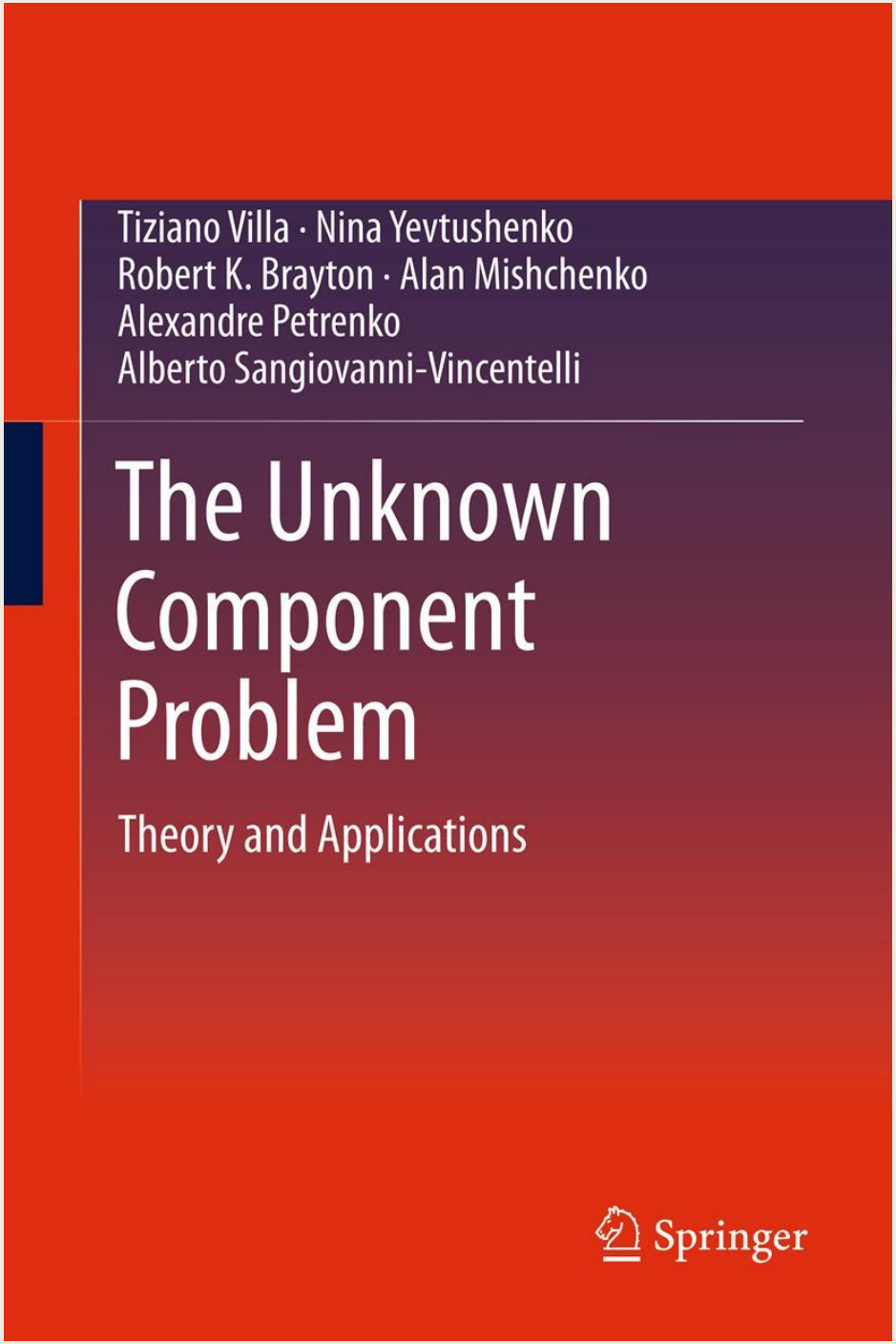
- Abstract equations over languages

$$C \; comp \; X \subseteq A$$

- Solution S

$$S \subseteq C \; comp \; A$$




Tiziano Villa · Nina Yevtushenko
Robert K. Brayton · Alan Mishchenko
Alexandre Petrenko
Alberto Sangiovanni-Vincentelli
The Unknown Component Problem
Theory and Applications
Springer

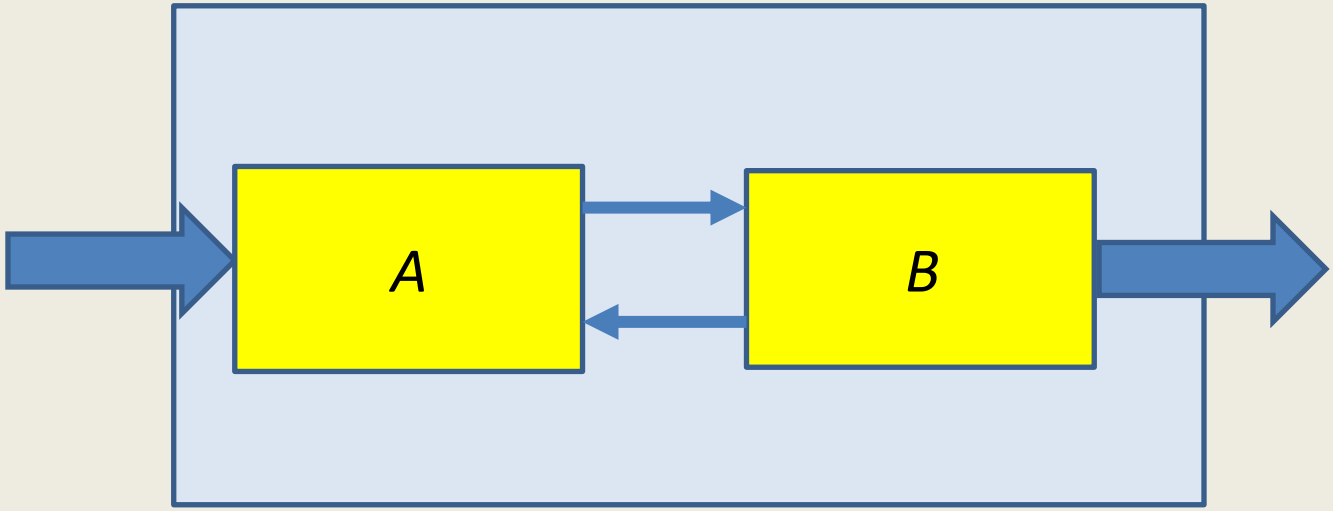
How Does ND Occur?

- Sources intrinsic to IUT, as an atomic unit for testing
  - Inherent ND; discussions on “can a commercial software product be ND?” are out of scope
  - IUT is composed from deterministic modules
- Sources extrinsic to IUT
  - SUT where IUT is embedded (integration testing)
  - Test environment
  - Test modeller

Extrinsic: Test Environment

- Distributed interfaces
- Fast vs. slow tester (quiescence of SUT)
- Queued testing
- Impaired controllability (e.g., some inputs are controlled by other running components)
- Limited observability, e.g., imprecise timestamping in observers
- Distributed testers

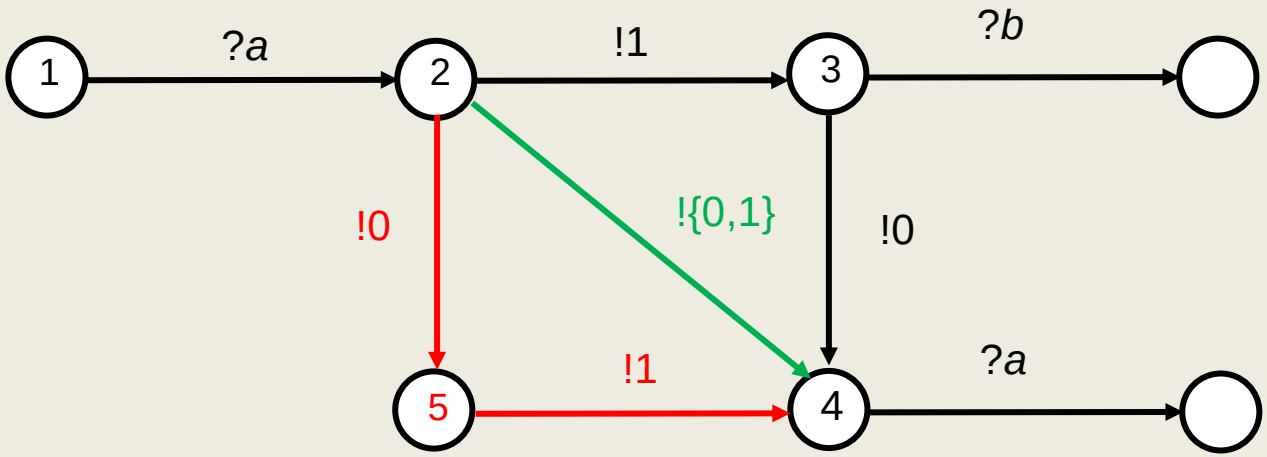
Fast vs. Slow Tester



Fast tester can submit input only when *A* is in quiescent state, but the whole SUT may not necessarily be in a global quiescent state

How to detect global quiescence in a distributed system?

Limited Observability

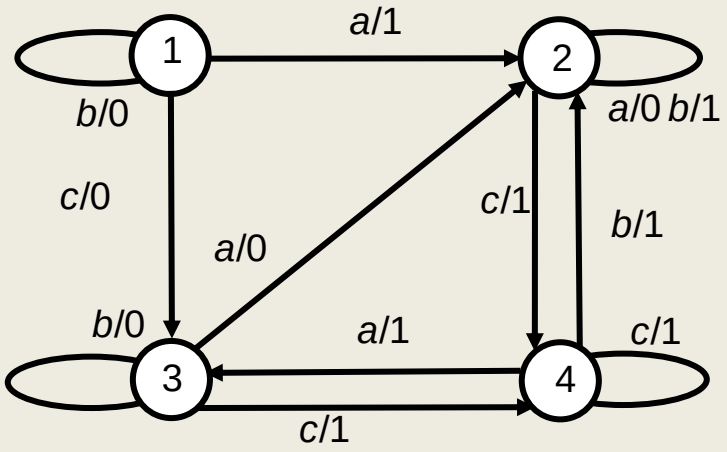


And what if the tester wants to submit b only after 1?

Extrinsic: Test Modeller

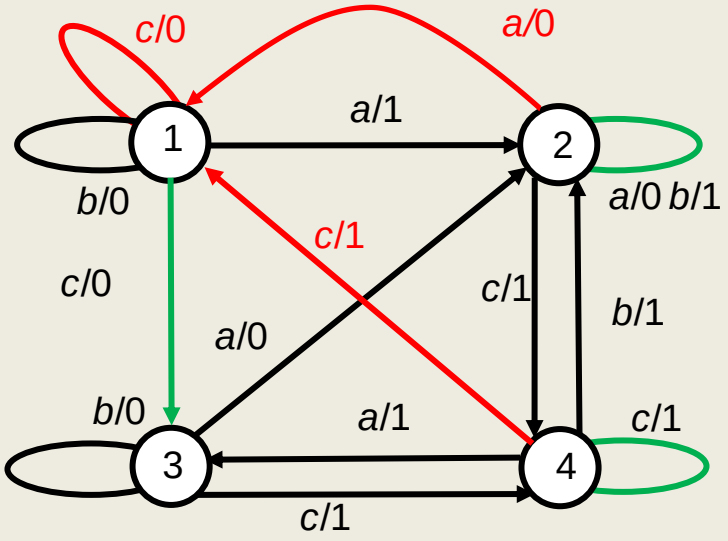
- Uses ND to define a test model for a given tester
- Adds ND expressing
  - Uncertainty and partial knowledge
  - Don’t cares and underspecification
  - Abstractions, quotients, slices, e.g., of EFSM
- Models faults obtaining ND test models (example provided)
- Uses model transformation that yields ND
- Uses fragmented test models , e.g., HMSC, a set of SD, if combined, exhibit ND (implied scenarios)
- Superinduces ND by using inadequate formalism

ND Test Models to Formulate Fault Assumptions



Assume we want to build tests detecting any combination of the following faults (red) in (green) transitions

We build ND test model, which compactly represents n! mutants, where n is the number of mutated transitions, called
Mutation machine (fault function)



Dealing with ND Test Models

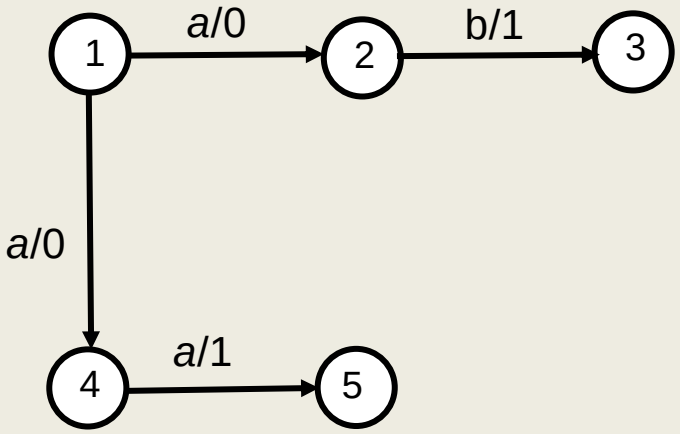
- What does ND reflect?
  - ND inherent to IUT
  - IUT enclosed in SUT
  - Limited power of a tester
  - Variability from the test modeller
- Do we use ND test model for an ND IUT or not?
- How to force ND IUT to exhibit all the behavior with a test case (complete testing assumption)?
- Should a conforming IUT show all the behavior as the model, i.e., trace containment or trace equivalence?
- Can chosen conformance relation be tested by a given tester?
- How do we actually test for it?

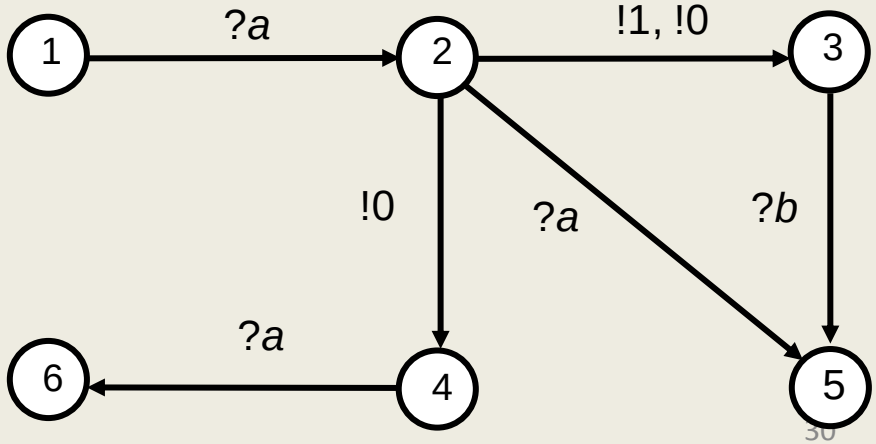
Testing ND IUT

- On-the-fly testing
- Pre-compute test cases
  - A *preset* test case uses a single input sequence which should repeatedly applied to IUT to ensure that the complete testing assumption is satisfied (IUT is assumed to be fair to the tester)
  - An *adaptive* test case is an acyclic IOTS or FSM, input-complete (from IUT) and at most one output in each state

On-the-fly Testing

- Termination conditions
  - Time limit
  - Length limit
  - Reaching target states
- ND models esp. not input-enabled with I/O conflicts are problematic
- No fault coverage?

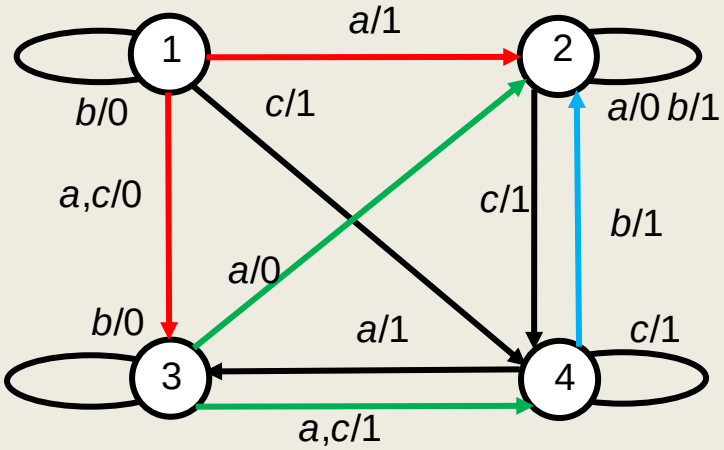


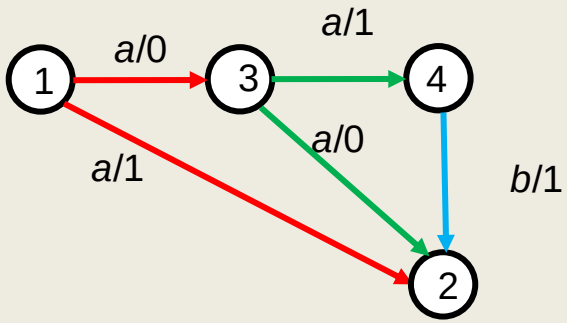


Key Problems in Test Generation from ND Models

- Test cases
  - How state can be reached?
  - How transition can be covered?
  - How state can be identified?
- Test suites
  - What are the test generation termination criteria, test coverage or adequacy criteria
  - How to generate a minimal one for a given criterion?

State Reachability

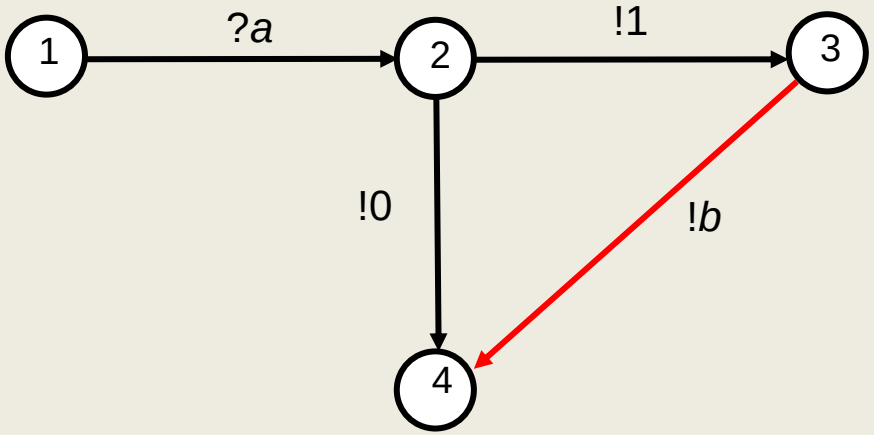




- If we want to take IUT into a state that correspond to a given state of ND test model it is not always possible, since a conforming IUT may not have any corresponding state
- If it is has the state of the test model that is “definitely reachable”, as it must have a corresponding state in any conforming IUT
- MC may not help solve this problem if test models are ND

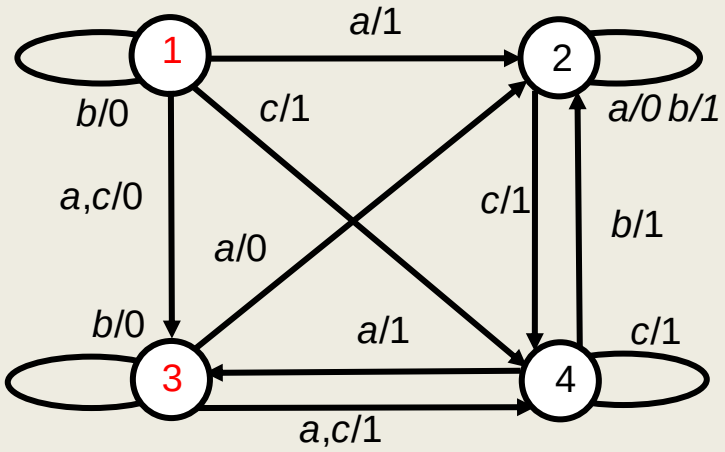
Covering Transitions

Transitions emanating from a “non-definitely reachable” state of the test model (red) transitions may not “implemented” in a conforming IUT

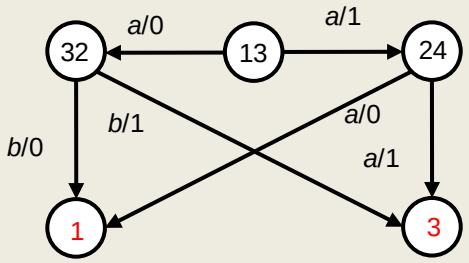


MC may not help solve this problem if test models are ND

State Identification via Distinguishability



Distinguishing states 1 and 3



- In case of FSM, we often consider that test models are minimized, and do not have indistinguishable states
- At the same time, the existing work on IOTS does not care about minimality of test models
- See our paper with Simao

State Distinguishability and Mutant Killing

- Mutant killing approaches in case of ND test models rely on state distinguishability which in turn depends on the conformance relation
- Strong and weak distinguishing sequences
- As we concluded in Boroday, S., Petrenko, A., and Groz, R., *"Can a Model Checker Generate Tests for Non-Deterministic Systems?"* Electronic Notes in Theoretical Computer Science, 190 (2) August 2007 model checking in case of ND will often generate only a fragment of test case

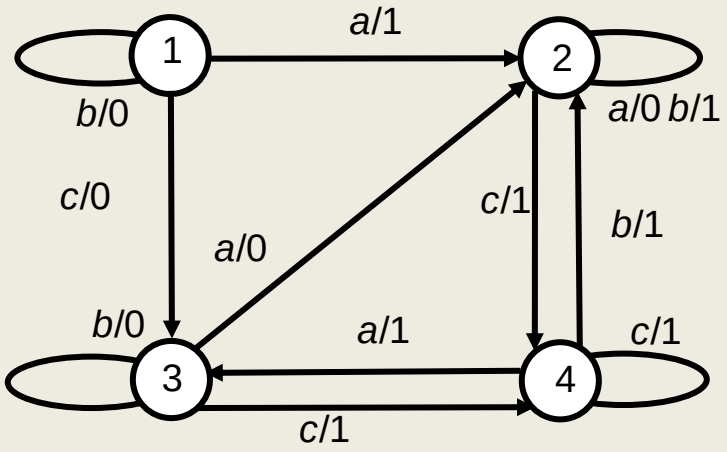
Test Coverage Criteria

- Test purposes
- ND test model coverage
  - Transition coverage for queued testing of IOTS with I/O conflicts (see Huo & Petrenko)
- Fault coverage
  - ND model (faults in ND IUT)
  - ND mutation machine obtained from D model (faults in D IUT)

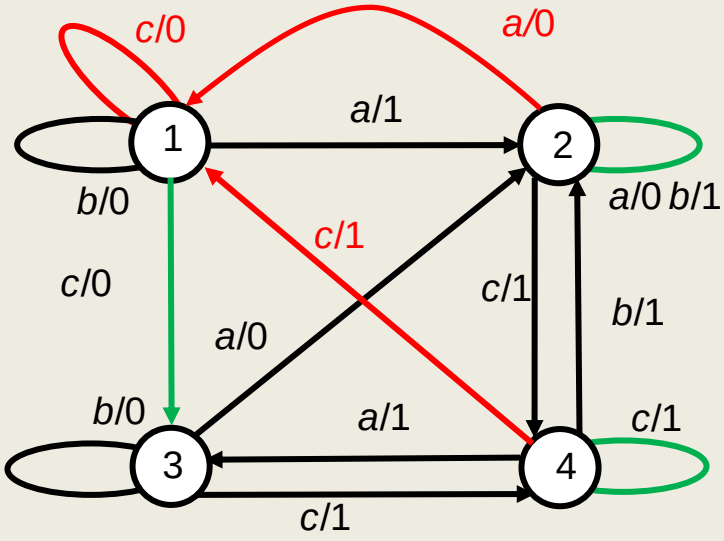
Fault Coverage is Based on Generic Fault Model

- (Specification model, Conformance relation, Fault domain)
- Specification model is FSM, EFSM, IOTS
- Conformance relation is trace equivalence, trace inclusion, x-ioco, where x is present or not
- Fault domain is a set of IUT models, mutants of the specification model, aka failure models
  - Universe of FSMs, IOTS with certain properties, e.g., state number
  - NFSM that has a specification model as a submachine

Example of Fault Domain



Assume we want to build tests detecting any combination of the following faults (red) in (green) transitions



Mutation machine defines a fault domain, as a set of its deterministic submachines

Note that the mutation machine with all possible 4 states mutants is a chaos machine

Conclusions

- Explaining ND in test models by just a simple under-specification does not help us advancing in testing with such models
- Better understanding of the nature of various flavors and uses of ND
- Construction of test models from available artefacts, including model transformation approaches and tools
- Test generation theories are not mature enough to offer solutions to industry dealing with ND

Thank you very much