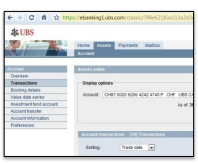
Model-Based Testing

Electronic Funds Transfer (EFT)



EFT Switch

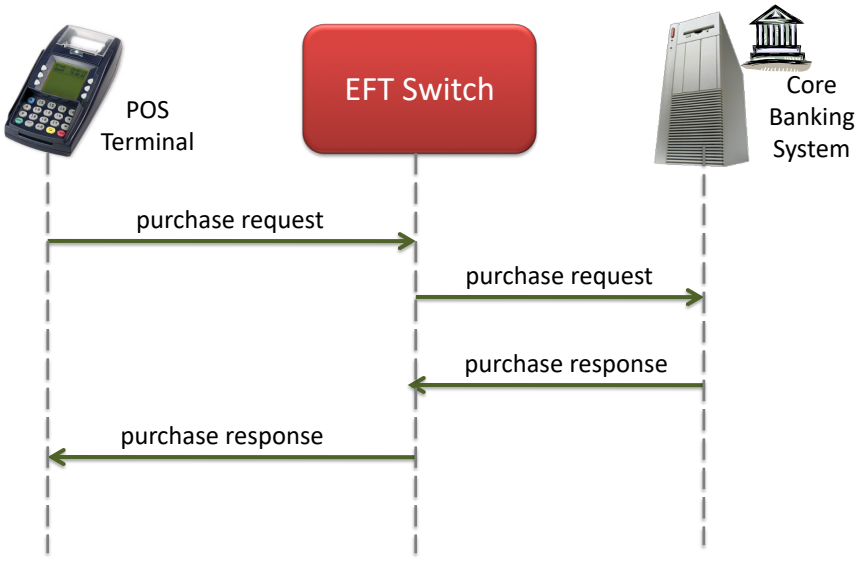
Core Banking System

Interbank Network

EFT Switch

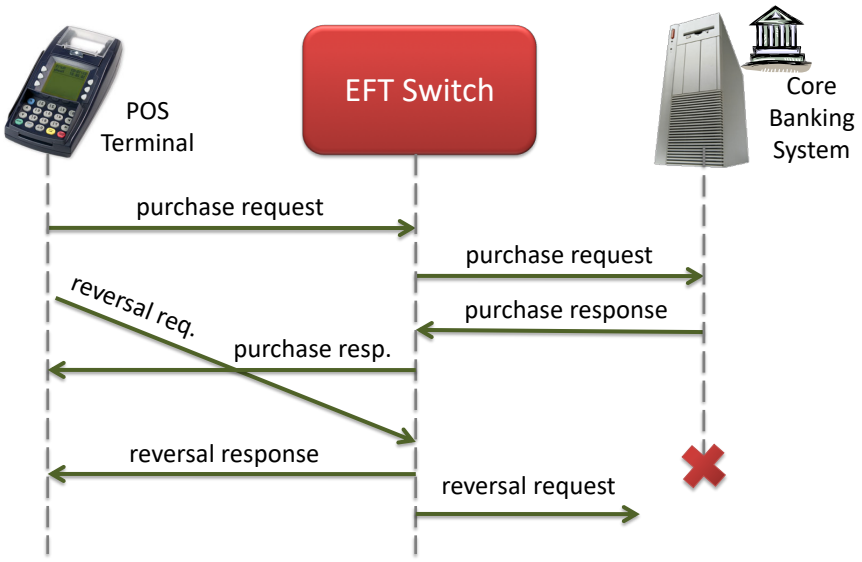
2

Example Scenarios



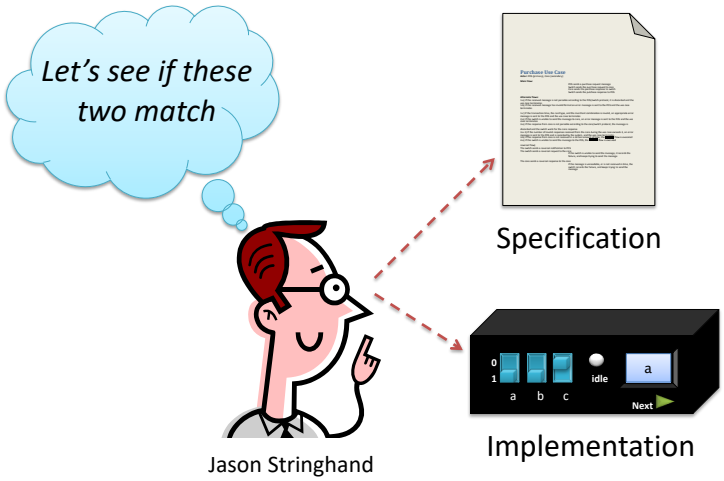
3

Example Scenarios



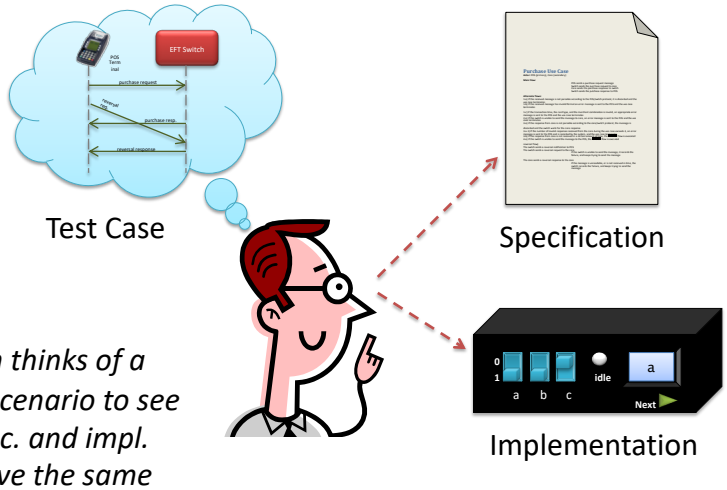
4

Black-Box Testing



5

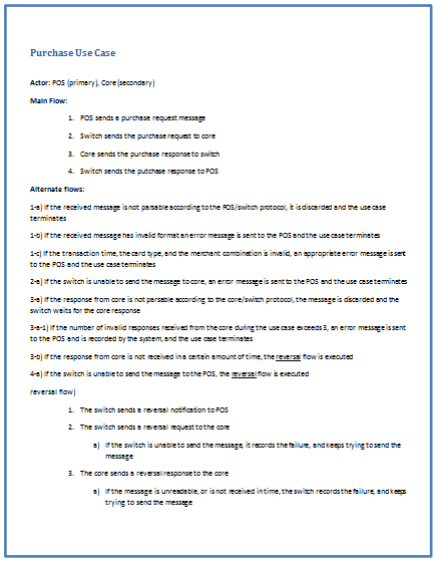
Black-Box Testing



Jason thinks of a test scenario to see if spec. and impl. behave the same

6

Specification by Use Cases



Purchase Use Case

Actor: POS (primery), Core (secondary)

Main Flow:

1. POS sends a purchase request message
2. Switch sends the purchase request to core
3. Core sends the purchase response to switch
4. Switch sends the putchase response to POS

Alternate flows:

1-a) If the received message is not parsable according to the POS/switch protocol, it is discarded and the use case terminates

1-b) If the received message has invalid format an error message is sent to the POS and the use case terminates

1-c) If the transaction time, the card type, and the merchant combination is invalid, an appropriate error message is sent to the POS and the use case terminates

2-a) If the switch is unable to send the message to core, an error message is sent to the POS and the use case terminates

3-a) If the response from core is not parsable according to the core/switch protocol, the message is discarded and the switch waits for the core response

3-a-1) If the number of invalid responses received from the core during the use case exceeds 3, an error message is sent to the POS and is recorded by the system, and the use case terminates

3-b) If the response from core is not received in a certain amount of time, the reversal flow is executed

4-a) If the switch is unable to send the message to the POS, the reversal flow is executed

reversal flow)

1. The switch sends a reversal notification to POS
2. The switch sends a reversal request to the core
   a) If the switch is unable to send the message, it records the failure, and keeps trying to send the message
3. The core sends a reversal response to the core
   a) If the message is unreadable, or is not received in time, the switch records the failure, and keeps trying to send the message

7

Example: Purchase Use Case

Alternate flows:

3-a) If the response from core is not parsable according to the core/switch protocol, the message is discarded and the switch waits for the core response

3-a-1) If the number of invalid responses received from the core during the use case exceeds 3, an error message is sent to the POS and is recorded by the system, and the use case terminates

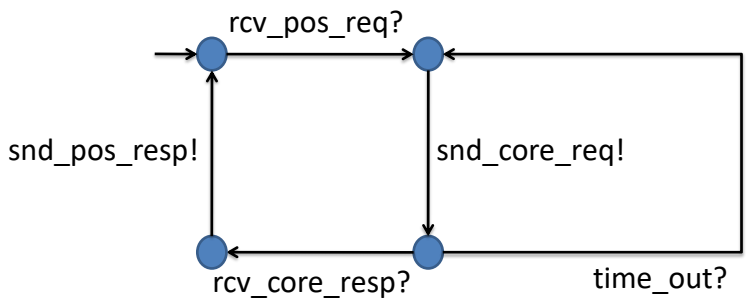
The Problem:

It is not easy to enumerate all possible scenarios of interaction from informal use case descriptions

including concurrency

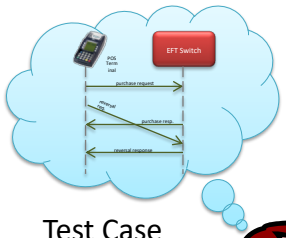
Solution: Model-Based Testing

- Modeling the specification formally
- Deriving test cases (scenarios) from the model



13

Black-Box Testing



Test Case

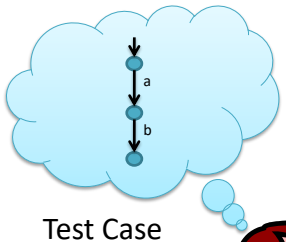
Specification

Jason thinks of a test scenario to see if spec. and impl. behave the same

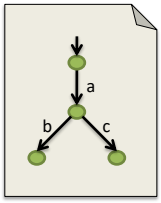
Implementation

14

Model-Based Black-Box Testing



Test Case



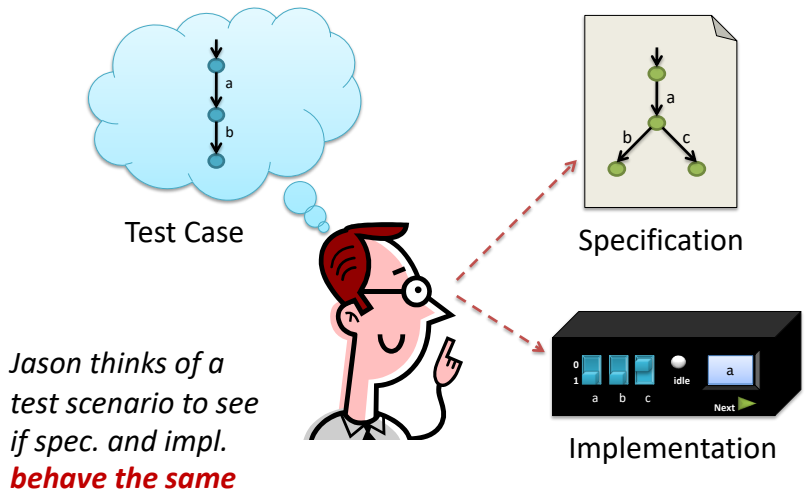
Specification

Jason thinks of a test scenario to see if spec. and impl. behave the same

Implementation

15

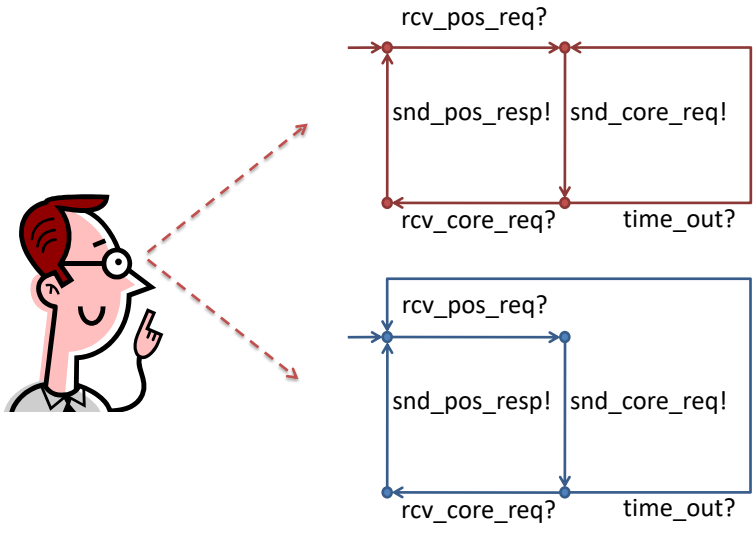
Model-Based Black-Box Testing



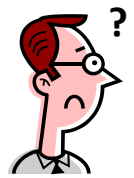
Jason thinks of a test scenario to see if spec. and impl. ***behave the same***

16

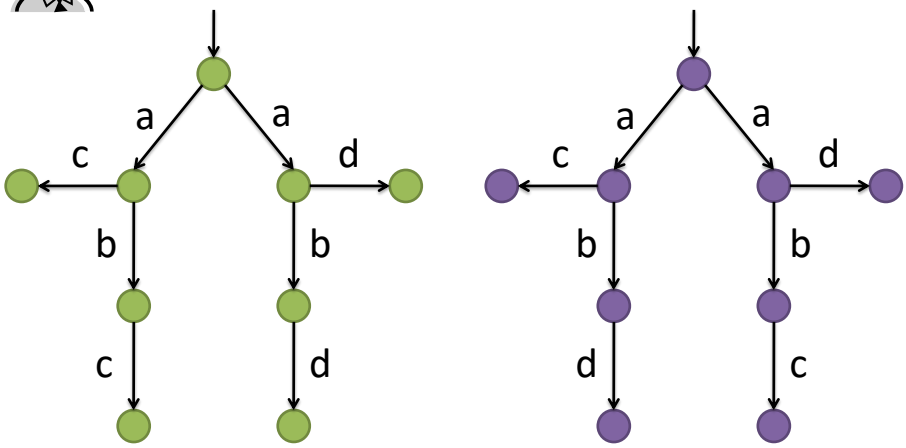
Equivalence By Observation



17



Are these equivalent?



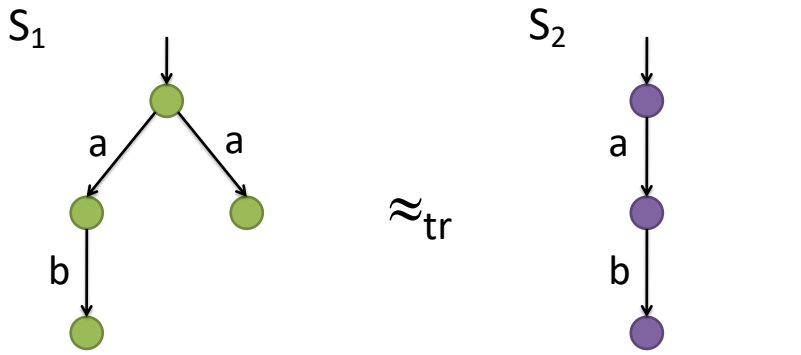
18

THEORY:

I/O CONFORMANCE TESTING (IOCO)

19

Trace Equivalence

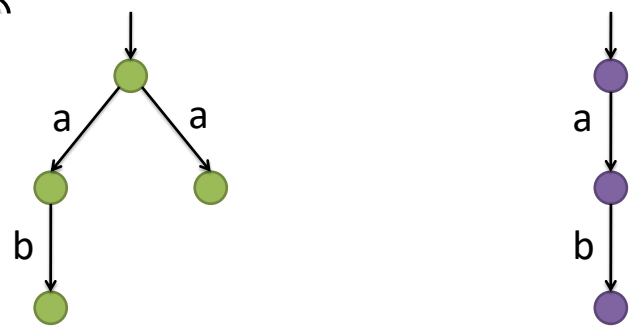


traces(S_1) = {ε, a, ab}

traces(S_2) = {ε, a, ab}

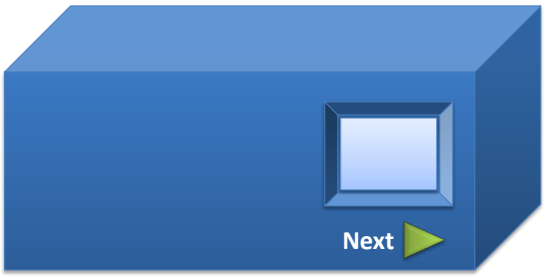
20

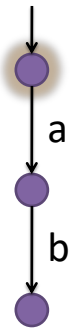
But, these two are not "equivalent"



21

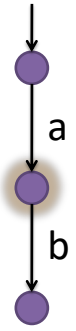
Testing Machine v.1





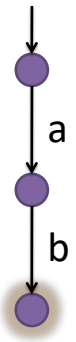
22

Testing Machine v.1



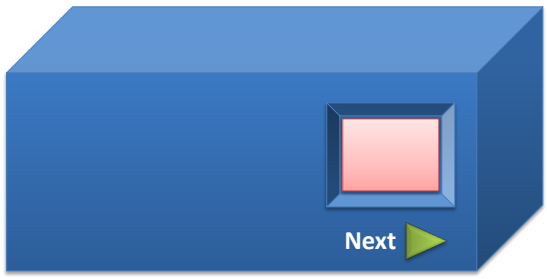
23

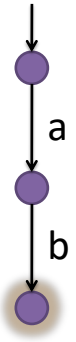
Testing Machine v.1



24

Testing Machine v.1





No more action: Trace Completed!

25

Completed Trace Equivalence

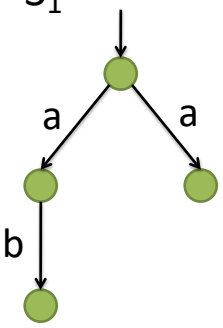
$$I \approx_{ctr} S$$

traces(I) = traces(S)

c_traces(I) = c_traces(S)

26

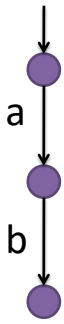
S_1



$\approx_{tr}$

$\not\approx_{ctr}$

S_2



traces(S_1) = {ε, a, ab}

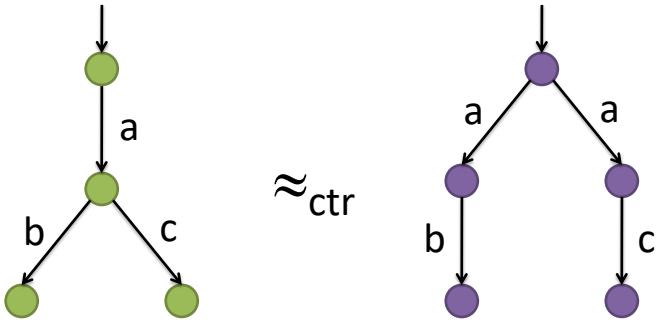
c_traces(S_1) = {a, ab}

traces(S_2) = {ε, a, ab}

c_traces(S_2) = {ab}

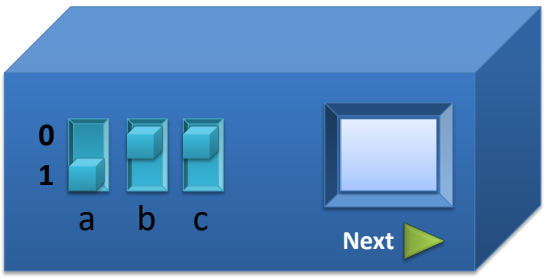
27

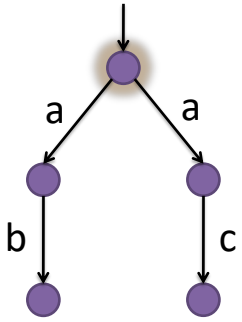
What about these two?



28

Testing Machine v.2

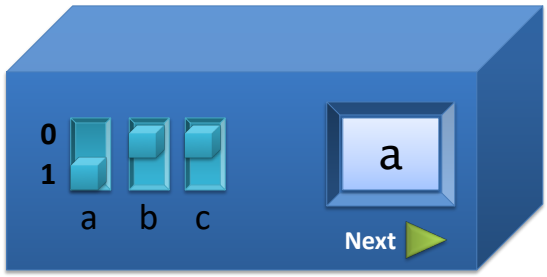


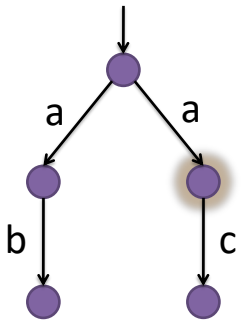


The buttons enable or disable actions that can be performed by the system

29

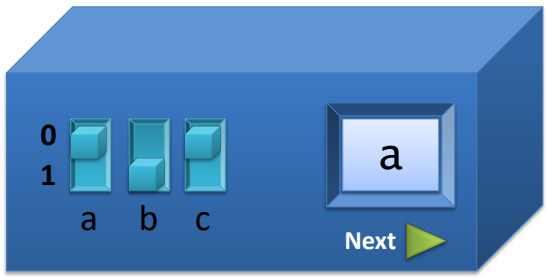
Testing Machine v.2

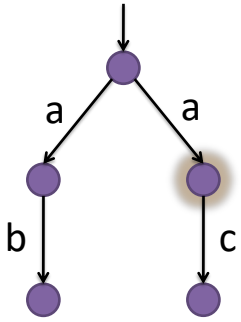




30

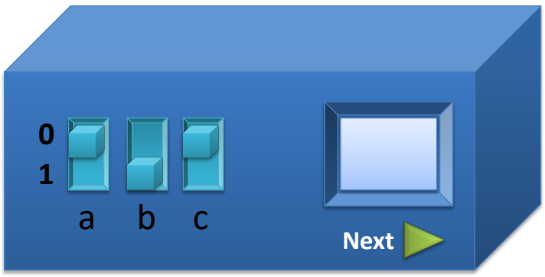
Testing Machine v.2

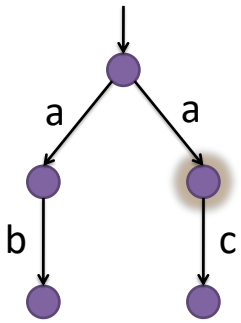




31

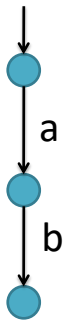
Testing Machine v.2



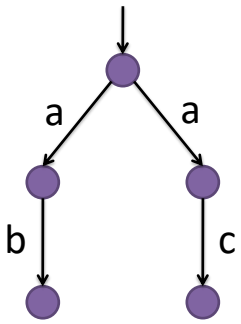


32

What Jason did using switches



Environment



System

33

Testing Equivalence

$$I \approx_{te} S$$

for every environment E:

traces(E || I) = traces(E || S)

c_traces(E || I) = c_traces(E || S)

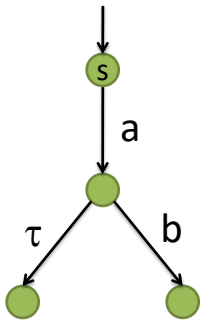
34

Failure Pairs

f-pairs(s) = { $(\sigma, A) \in Act \times 2^{Act}$ | (s after σ) refuses A }

s after σ = { s' | $s \stackrel{\sigma}{\Rightarrow} s'$ }

S refuses A = $\exists s \in S . \forall a \in A . s \stackrel{a}{\not\Rightarrow}$



f-pairs(s) = { (ε, {b}), (a,{a,b}), (ab,{a,b}) }

35

Testing Equivalence

$$I \approx_{te} S$$

iff

f-pairs(I) = f-pairs(S)

There is no sign of "environment" in the definition!

36

$I \approx_{te} S$

for every environment E:

traces(E || I) = traces(E || S)

c_traces(E || I) = c_traces(E || S)

Extensional Characterization

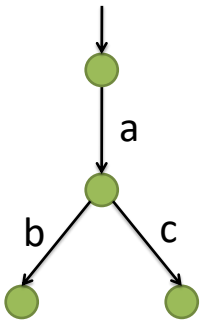
Intentional Characterization

$I \approx_{te} S$

iff

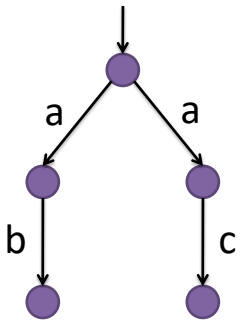
f-pairs(I) = f-pairs(S)

Failure Pairs



$\approx_{ctr}$

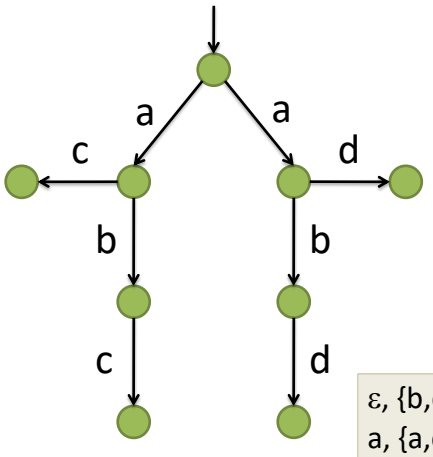
$\not\approx_{te}$

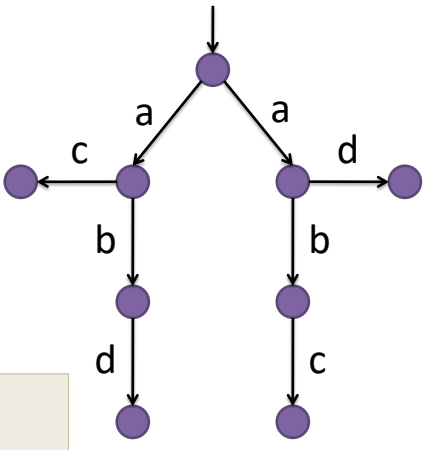


ε, {b,c}
a, {a}
ab, {a,b,c}
ac, {a,b,c}

ε, {b,c}
a, {a,b, c}
ab, {a,b,c}
ac, {a,b,c}

38



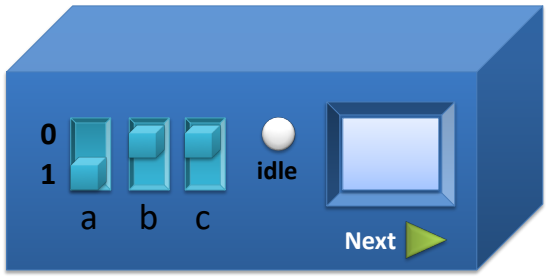


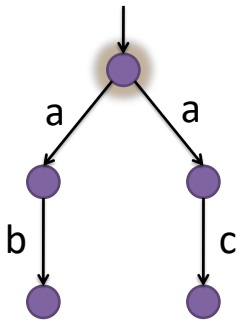
Failure-pairs of both systems:

ε, {b,c,d}
a, {a,c,d}
ac, {a,b,c,d}
ad, {a,b,c,d}
ab, {a,c,d}
abc, {a,b,c,d}
abd, {a,b,c,d}

39

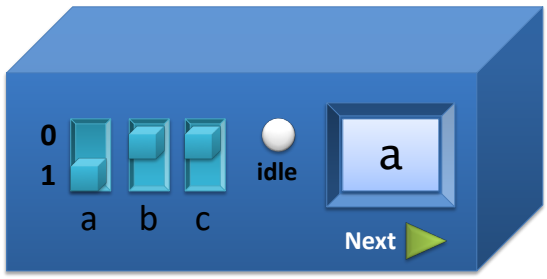
Testing Machine v.3

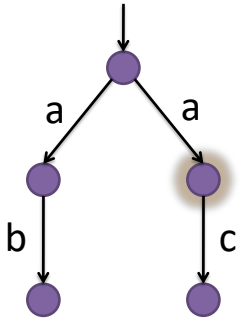




40

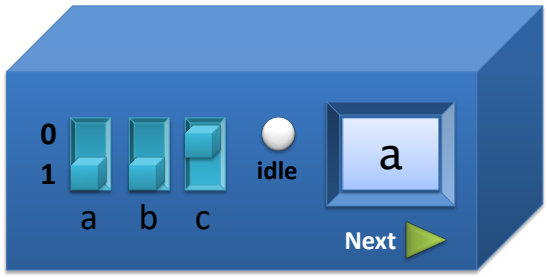
Testing Machine v.3

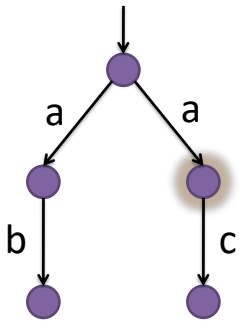




41

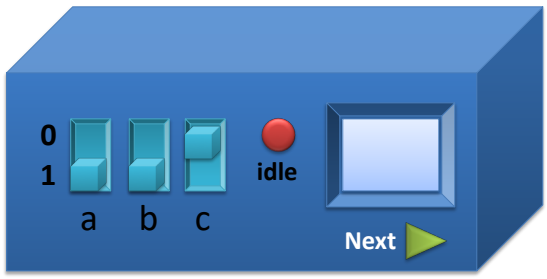
Testing Machine v.3

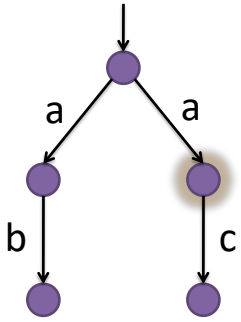




42

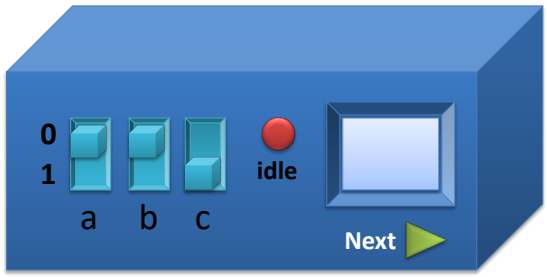
Testing Machine v.3

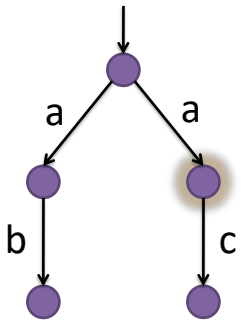




43

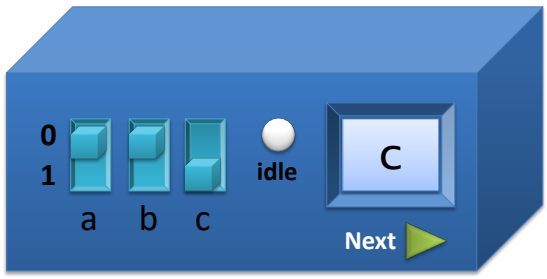
Testing Machine v.3

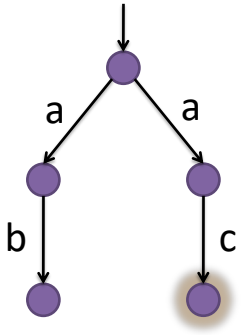




44

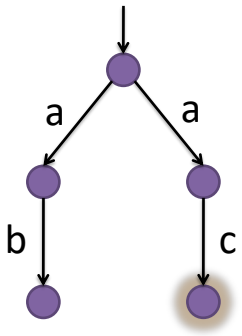
Testing Machine v.3





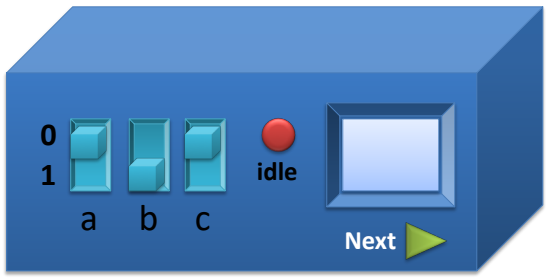
45

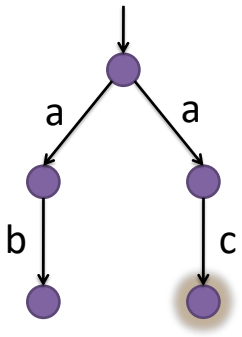
Testing Machine v.3



46

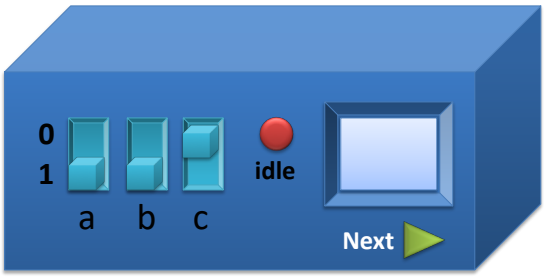
Testing Machine v.3

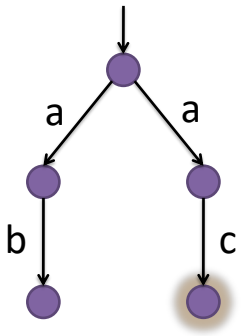




47

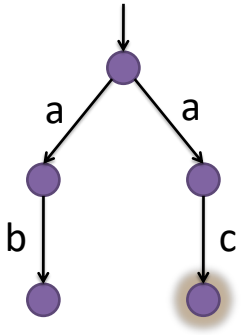
Testing Machine v.3





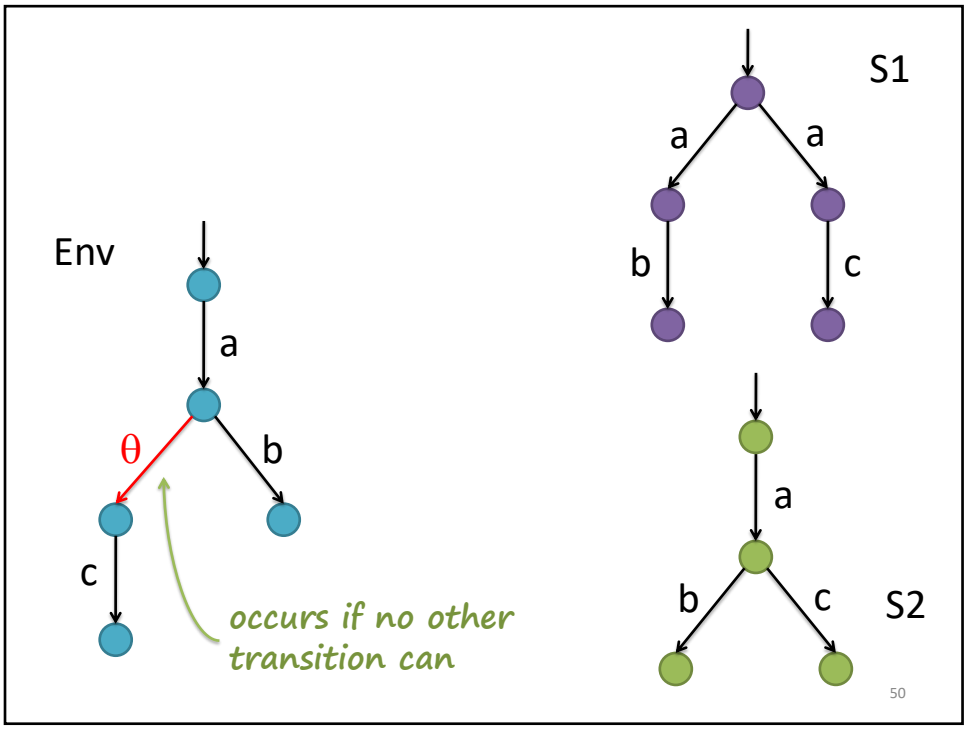
48

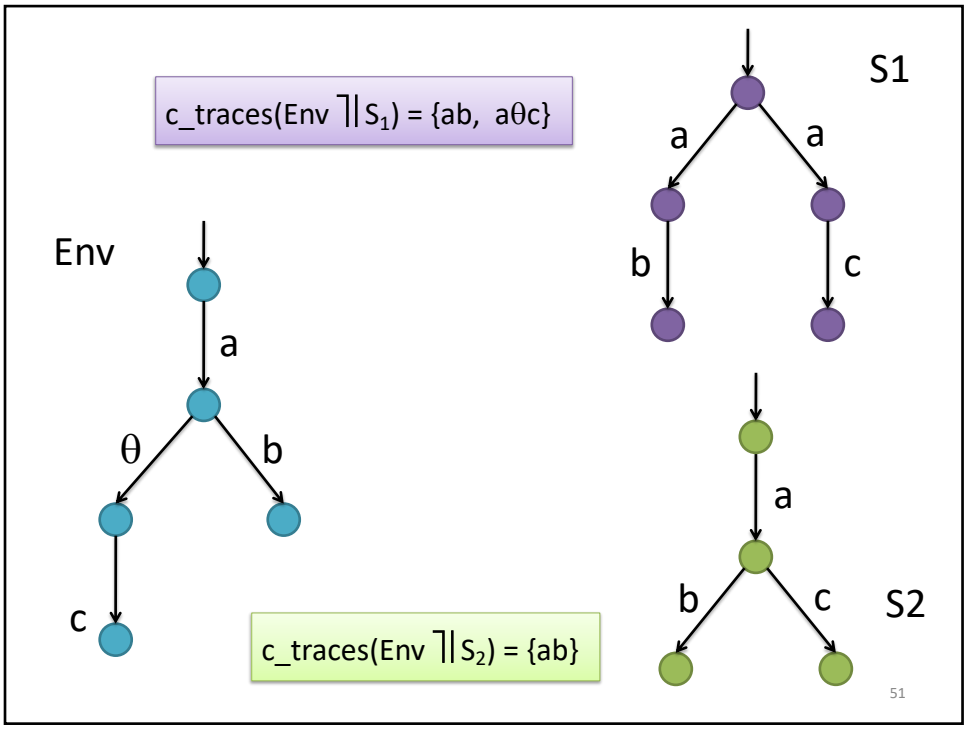
Testing Machine v.3



Now, we know that we have completed a trace.

49


S1
a
a
b
c
Env
a
θ
b
c
occurs if no other transition can
a
b
c
S2
50


c_traces(Env ⫿ S1) = {ab, aθc}
S1
a
a
b
c
Env
a
θ
b
c
c_traces(Env ⫿ S2) = {ab}
a
b
c
S2
51

Refusal Equivalence

$$I \approx_{rf} S$$

for every environment E:

$$traces(E \,⫴\, I) = traces(E \,⫴\, S)$$

$$c\_traces(E \,⫴\, I) = c\_traces(E \,⫴\, S)$$

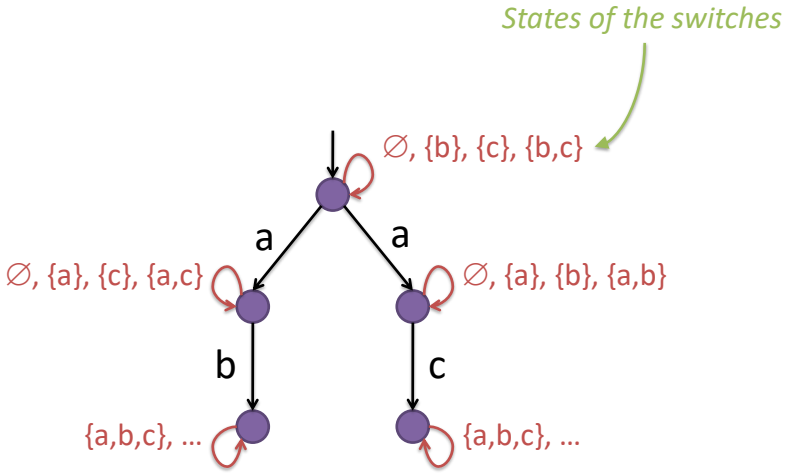
52

Failure Traces

- Sequence over *Act* U 2^{Act}
  - Single actions denote occurrence of an action
  - Action sets denote idle periods with enabled switches in the action set
- Example:
  - {a,b} c d b {b,c} {b,c,d} a *Act*

53

Failure Traces and Refusal Self-Loops



54

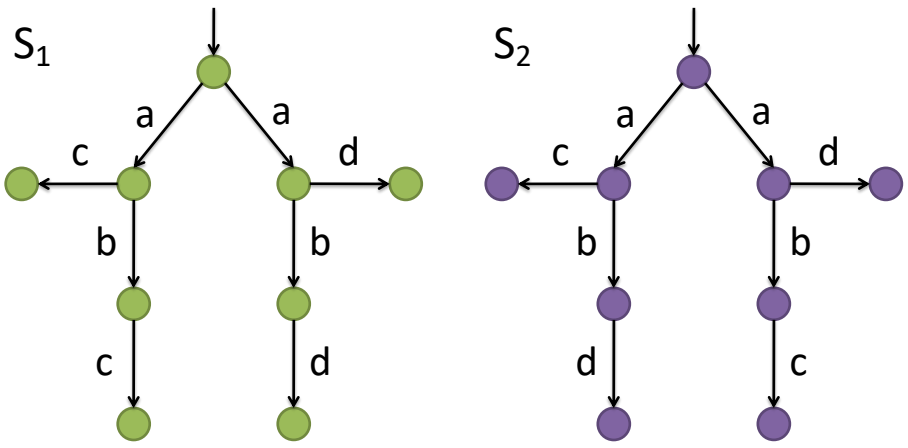
Refusal Equivalence (Intensional Characterization)

$$I \approx_{rf} S$$

iff

f-traces(I) = f-traces(S)

55

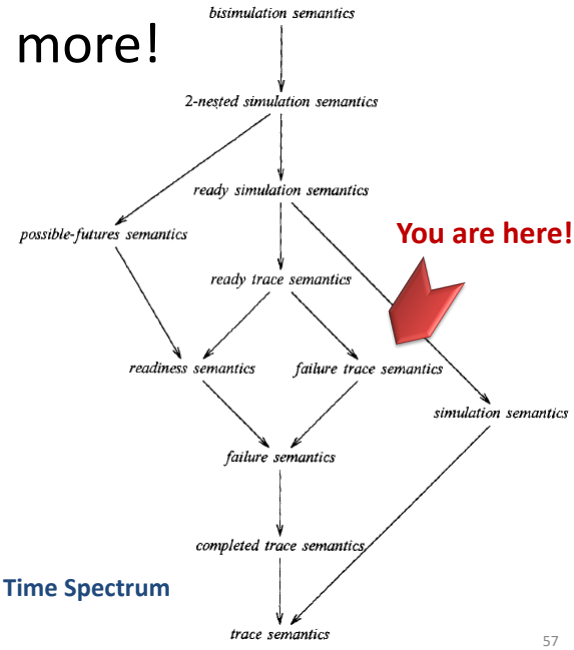


a {d} b c *Act* $\in$ f-traces(S_1)

a {d} b c *Act* $\notin$ f-traces(S_2)

56

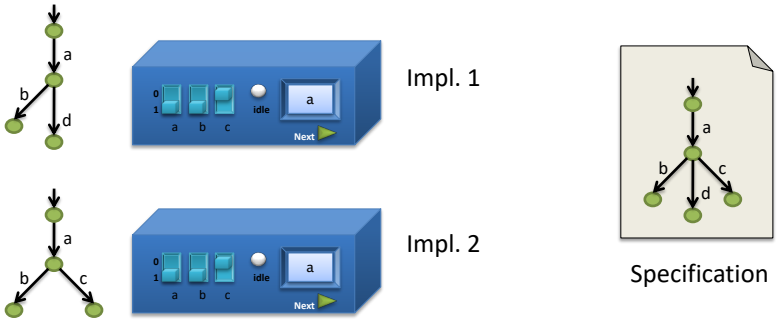
And there is more!



The Linear Time – Branching Time Spectrum
Rob van Glabbeek

57

Defining specifications at a higher-level



Leaving out some decisions into the implementation

58

Testing Pre-order

$$I \sqsubseteq_{te} S$$

for every environment E:

$$traces(E \parallel I) \subseteq traces(E \parallel S)$$

$$c\_traces(E \parallel I) \subseteq c\_traces(E \parallel S)$$

59

Refusal Pre-order

$$I \sqsubseteq_{rf} S$$

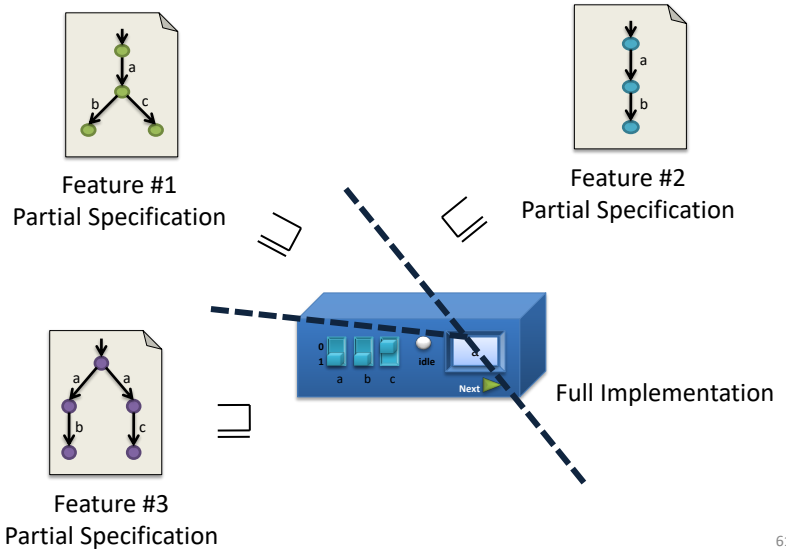
for every environment E:

$$traces(E \parallel I) \subseteq traces(E \parallel S)$$

$$c\_traces(E \parallel I) \subseteq c\_traces(E \parallel S)$$

60

Restriction to Specification



61

Restriction to Specification

$$I \textbf{ conf } S$$

for every environment E:

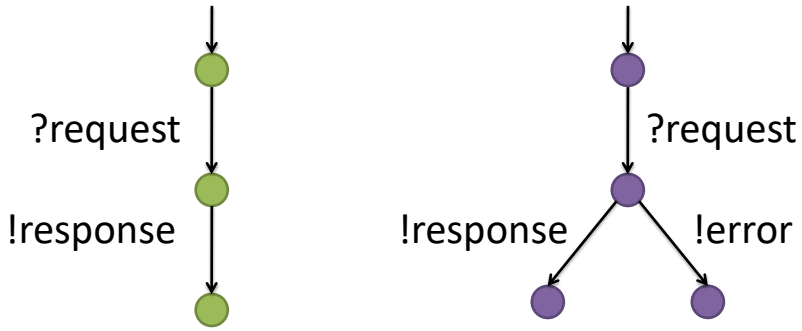
traces(E || I) ∩ traces(S) ⊆ traces(E || S)

c_traces(E || I) ∩ traces(S) ⊆ c_traces(E || S)

62

I/O Transition Systems

Distinguishing between input and output actions



64

Pre-orders on I/O transition systems

- The same notions apply here
  - I/O test pre-order
  - I/O refusal pre-order

$$I \sqsubseteq_{ior} S$$

for every environment E:

$$traces(E \,⫴\, I) \subseteq traces(E \,⫴\, S)$$

$$c\_traces(E \,⫴\, I) \subseteq c\_traces(E \,⫴\, S)$$

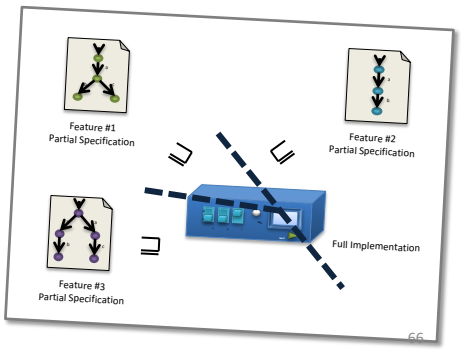
65

I/O Conformance

Informally:

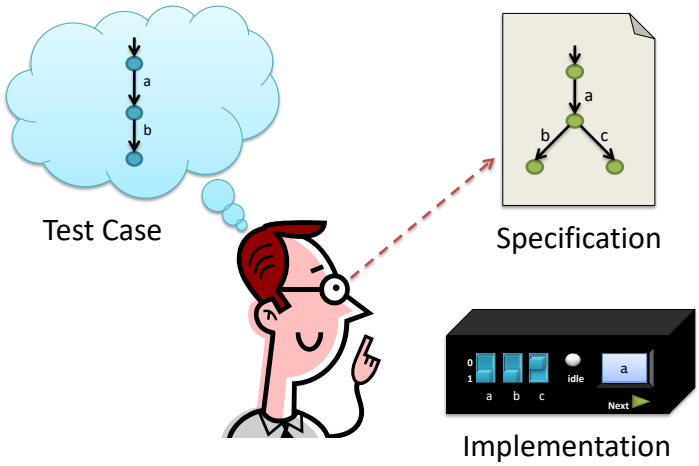
I/O Conformance = I/O Refusal restricted to specification traces

$$I \textbf{ ioco } S$$



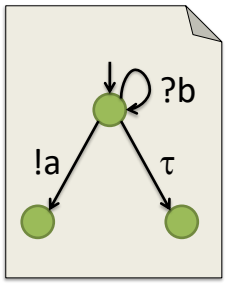
66

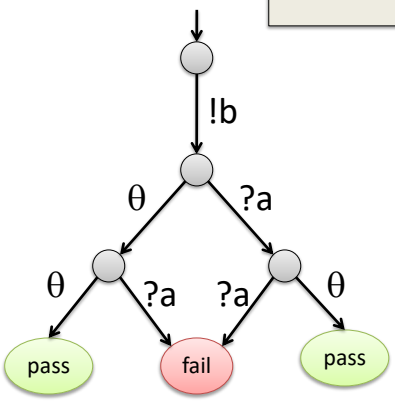
Black-box testing for ioco

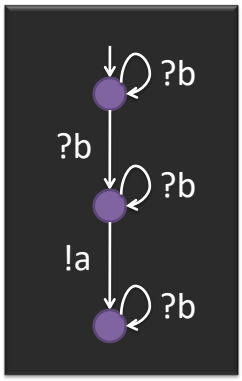


67

Example



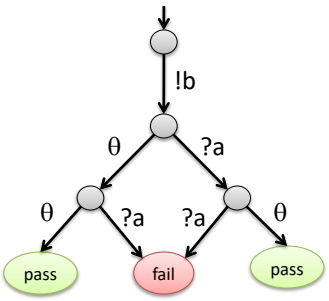




68

ioco Test Cases

- I/O transition systems
- The only terminal states: pass and fail
- Reversed I/O actions
- Special action θ
- Finite and deterministic



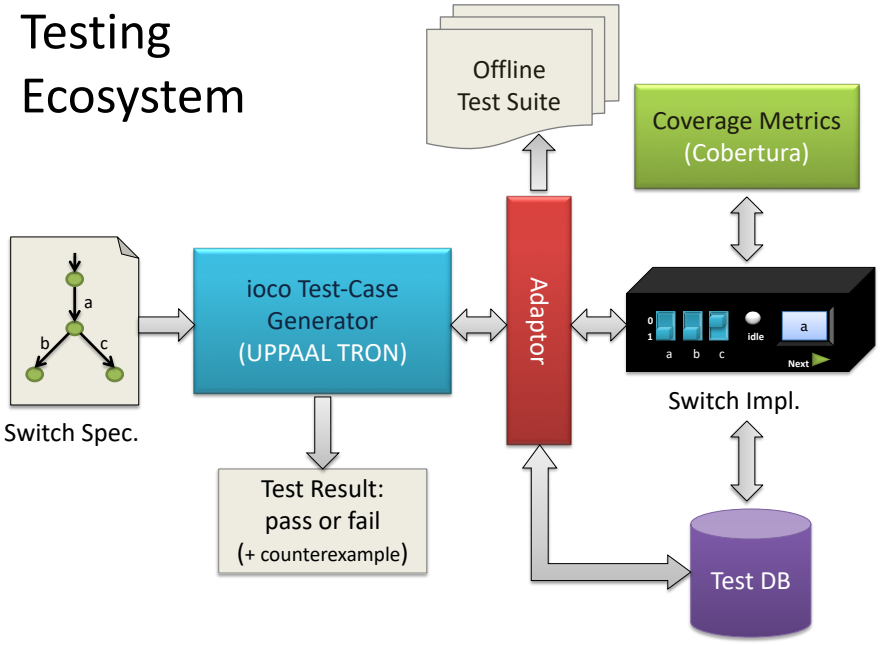
69

Automatic Test Case Generation

- Init:
  - Generate an initial state
- Recursion:
  - At each point in the recursion choose non-deterministically between:
    1. Stopping the recursion
    2. Supplying an input
    3. Observing an output (one transition per output action)

70

Testing Ecosystem

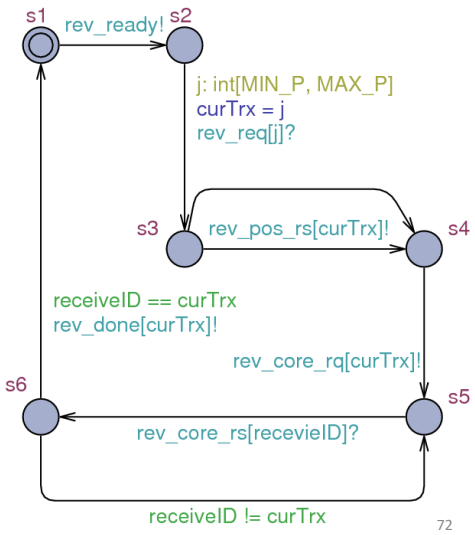


71

Switch Specification

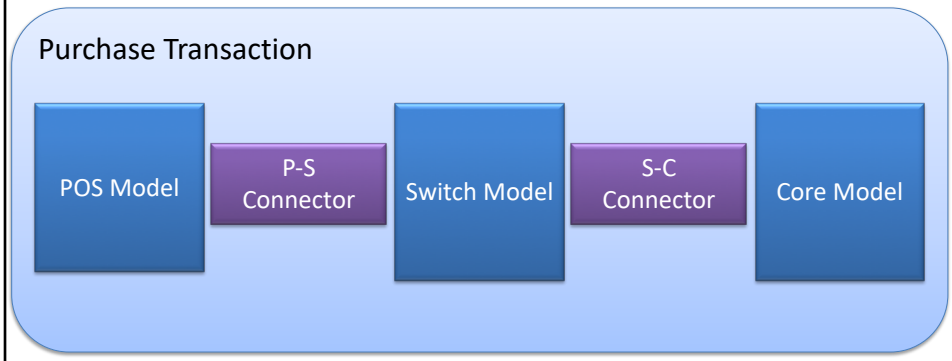
- Modeled in UPPAAL
- ≈ 25 automata

Simplified behavior of switch in a Reversal transaction



72

Specification Structure

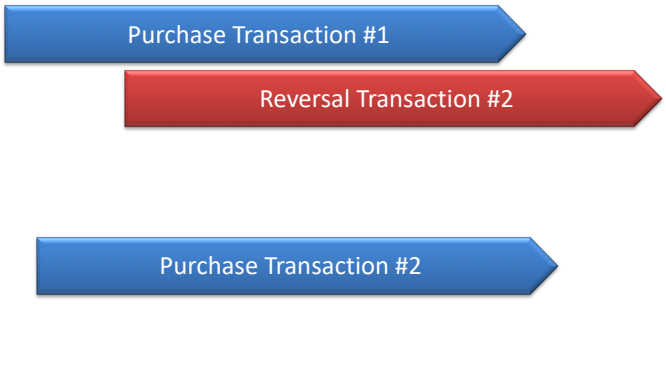


73

Concurrent Transactions

POS #1

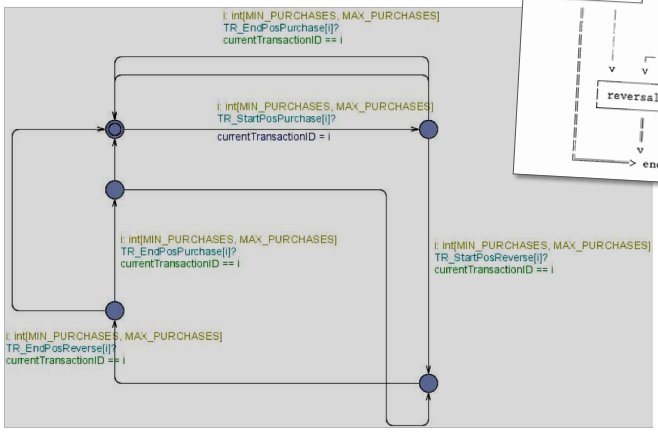
POS #2

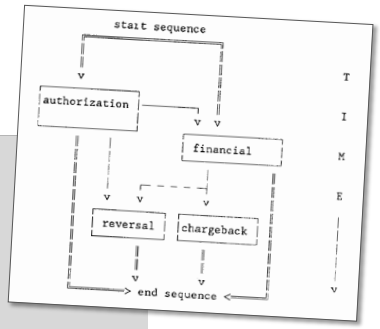


One LTS instance per transaction in the Switch

74

Business Transaction Model

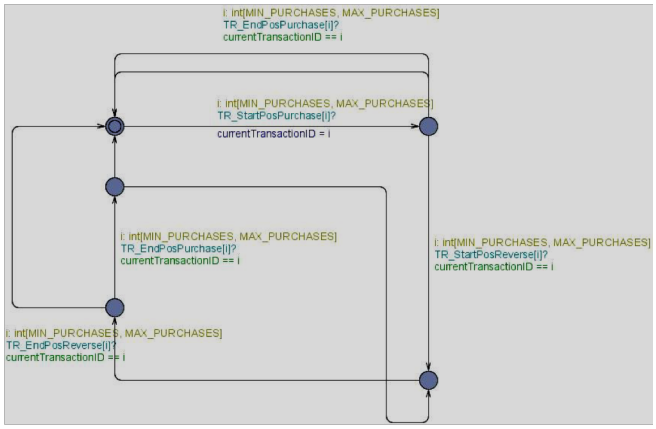




75

Verifying the Specification

A[] forall (i : int[0, MAX_TX]) TransactionFlow[i].start -> TransactionFlow[i].finish



76

Results so far

- Have found a few bugs in the system
  - One traced back to null-pointer dereferencing
  - Poor exception handling
  - Resource pooling problems
- Code coverage of about 40%

77

Final Comments

- Scalability Issues
- Modeling language and tool limitations
  - Modeling data-related business rules
  - Now using UML Statecharts
- Handling time constraints

78