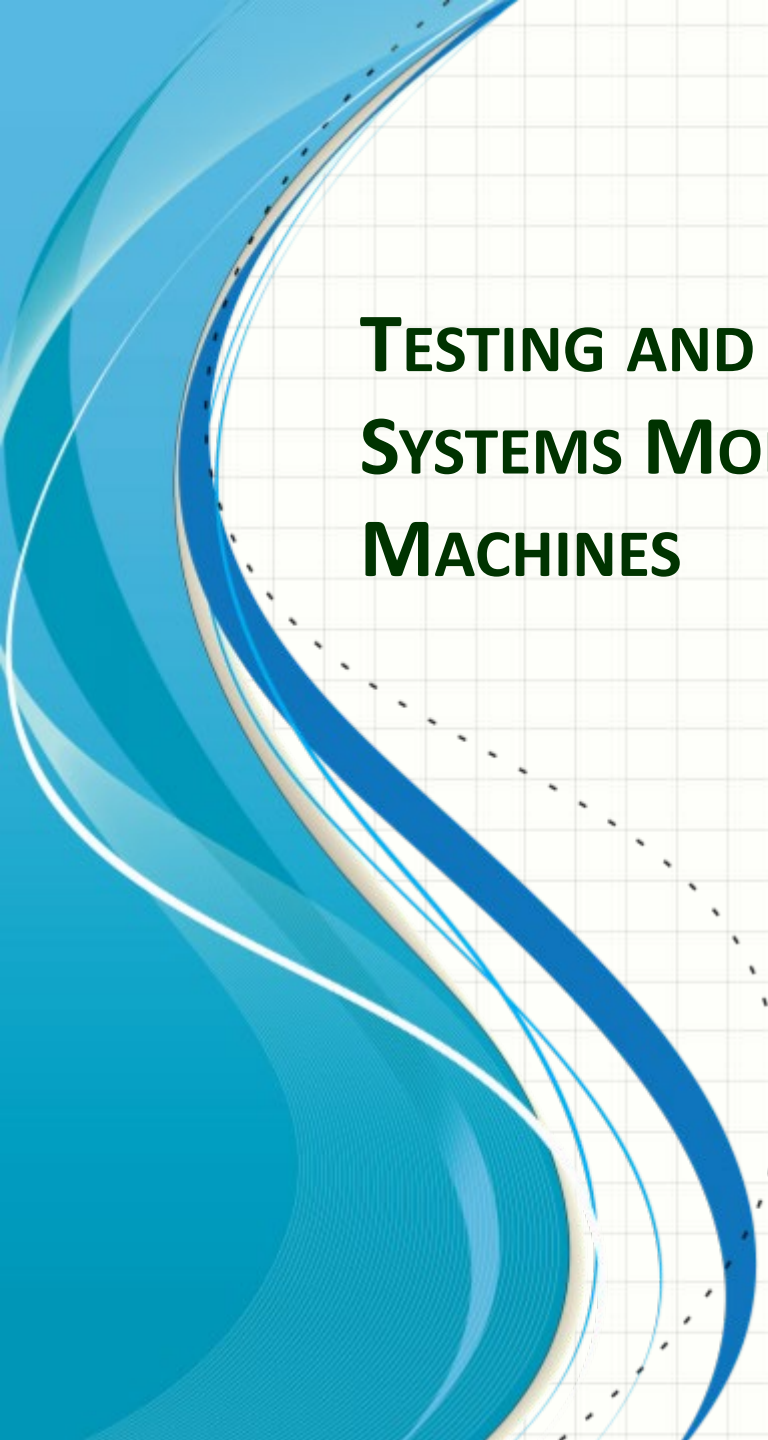


# AUS Repository

## Testing and Assessment of Protocols and Systems Modeled as Extended Finite State Machines

Item Type	Thesis
Authors	Salameh, Tariq Mohammad
Download date	2026-03-07 05:45:59
Link to Item	<a href="http://hdl.handle.net/11073/5906">http://hdl.handle.net/11073/5906</a>



# **TESTING AND ASSESSMENT OF PROTOCOLS AND SYSTEMS MODELED AS EXTENDED FINITE STATE MACHINES**

Tariq M. Salameh

Advisor: Dr. Khaled El Fakih

30<sup>th</sup> of June 2013

# Outlines

- Introduction
- Preliminaries
  - The EFSM Model
  - EFSM Flow Graph
  - Data-Flow, Control-Flow Test Suites
  - EFSM-Based Test Suites
- Mutation Testing
- Coverage Assessment of Mutation Testing
- Assessing Control-Flow, Data-Flow and EFSM Based Test Suites
  - Research Objectives
  - Assessment Methodology
  - Experimental Results
- Testing with Respect to Transfer Faults: A Method and an Assessment
  - Algorithm
  - Experimental Results
- Conclusion
- Future Work



# INTRODUCTION

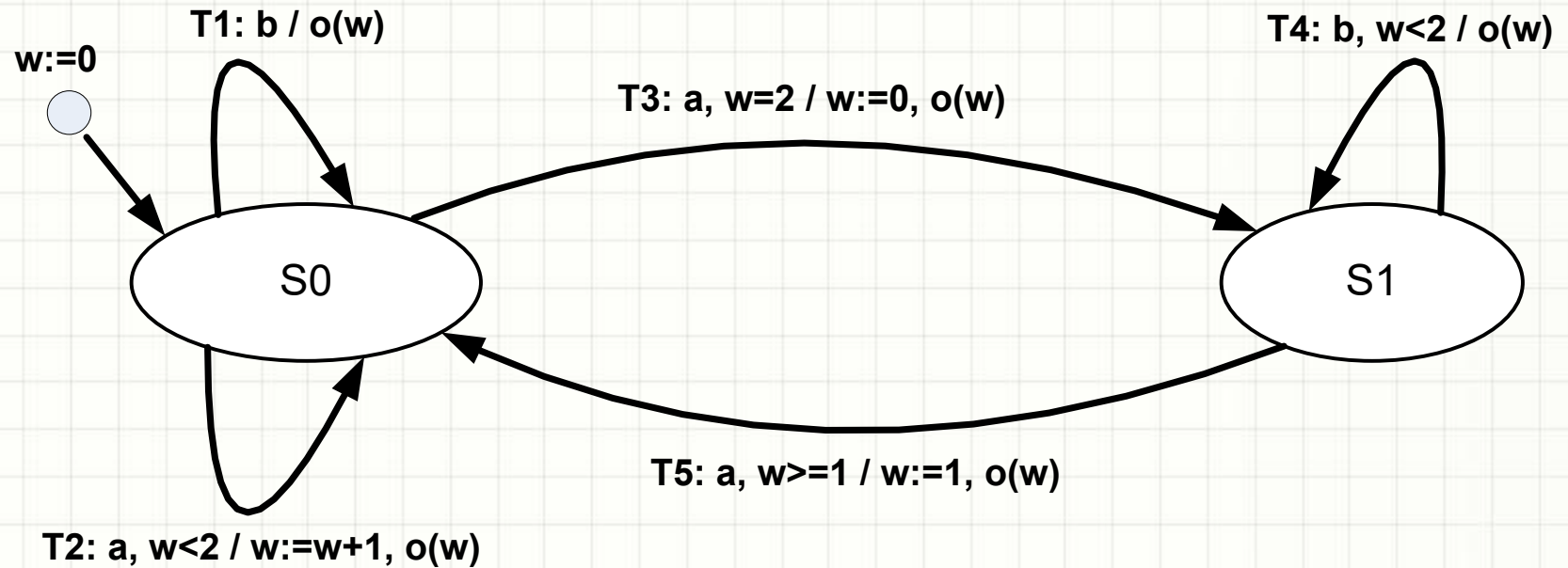
# Introduction

- Test derivation based on **formal models** is now widely used for deriving test suites for different kinds of systems.
- A well-known model that is widely used to represent the specification of a given software system is called the **Extended Finite State Machine (EFSM)** model.



# PRELIMINARIES

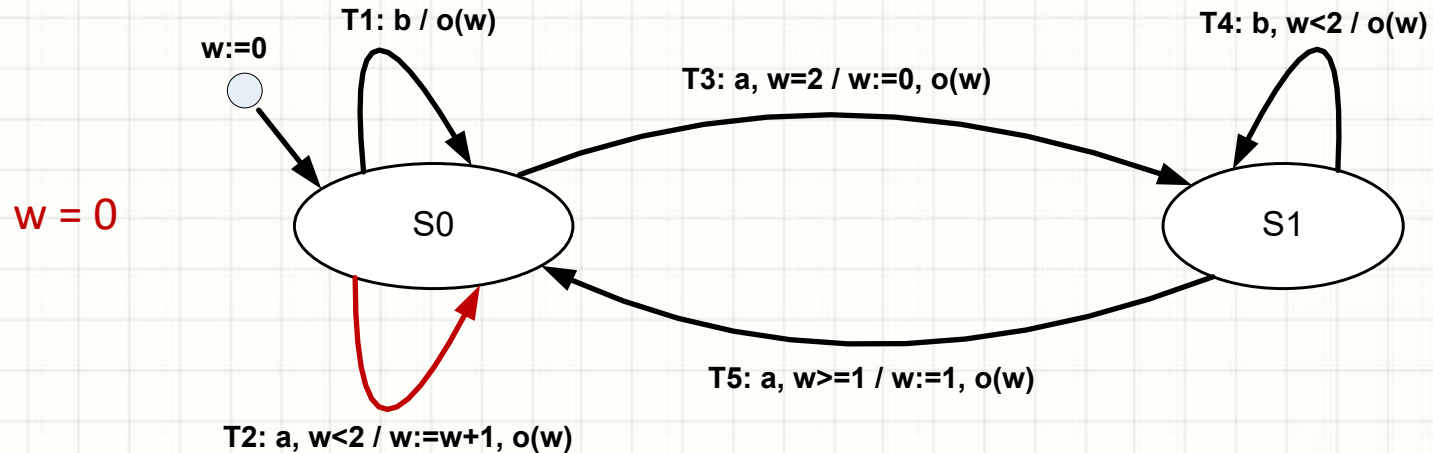
# The EFSM Model



*predicate complete*  
*input complete* } *complete*

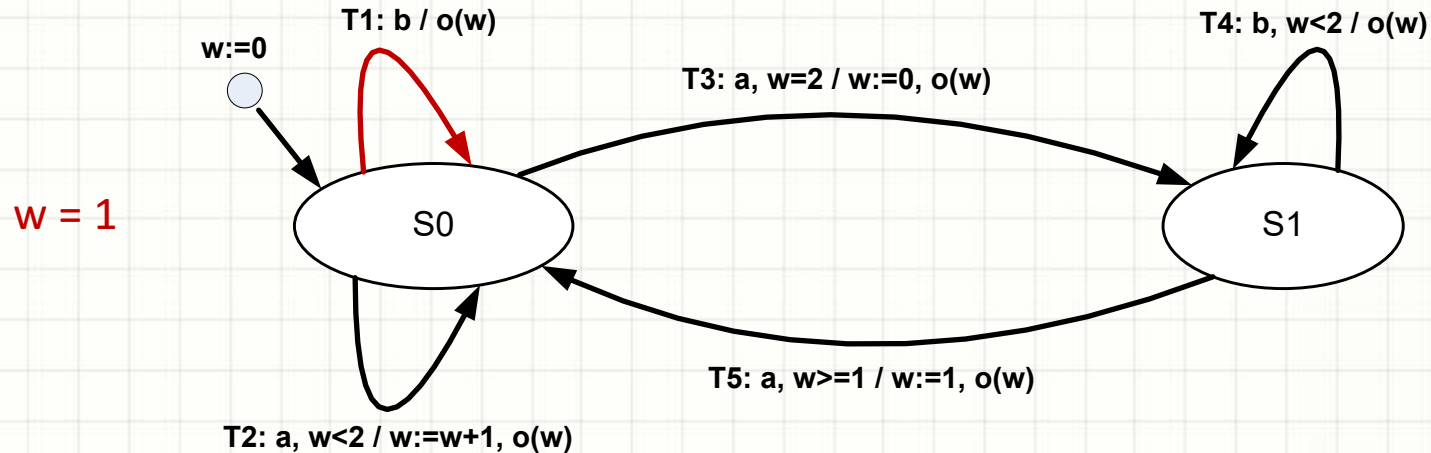
*deterministic*

# Test Cases & Test Suites



- Test Suite
  - Test Case 1
    - $S_0 - ?a/!o(1) - S_0$

# Test Cases & Test Suites (Cont.)

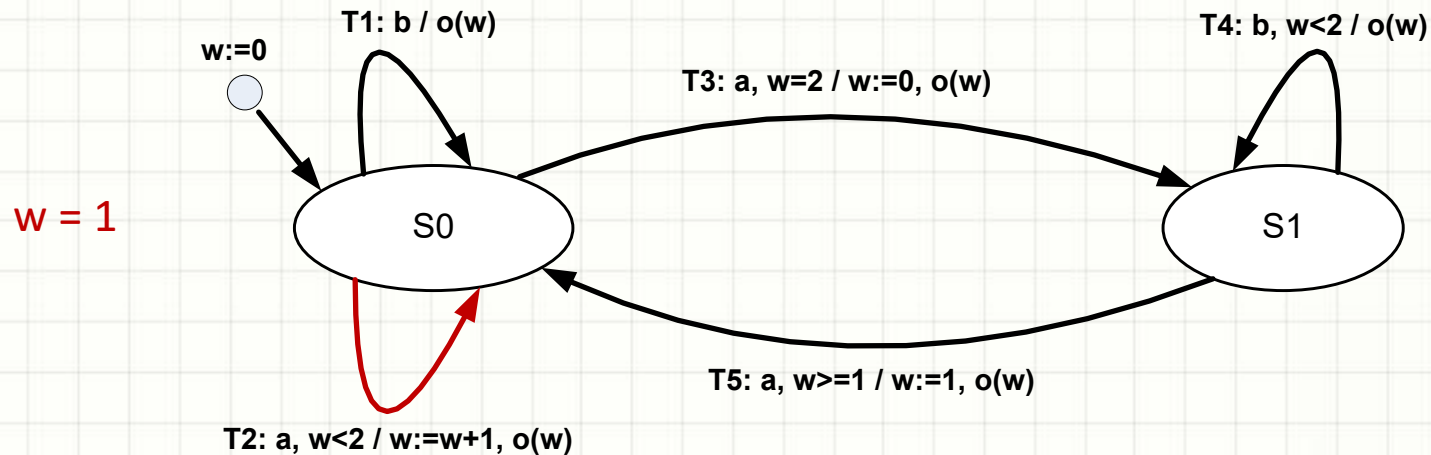


- Test Suite

- Test Case 1

- $S0 - ?a/!o(1) - S0 - ?b/!o(1) - S0$

# Test Cases & Test Suites (Cont.)

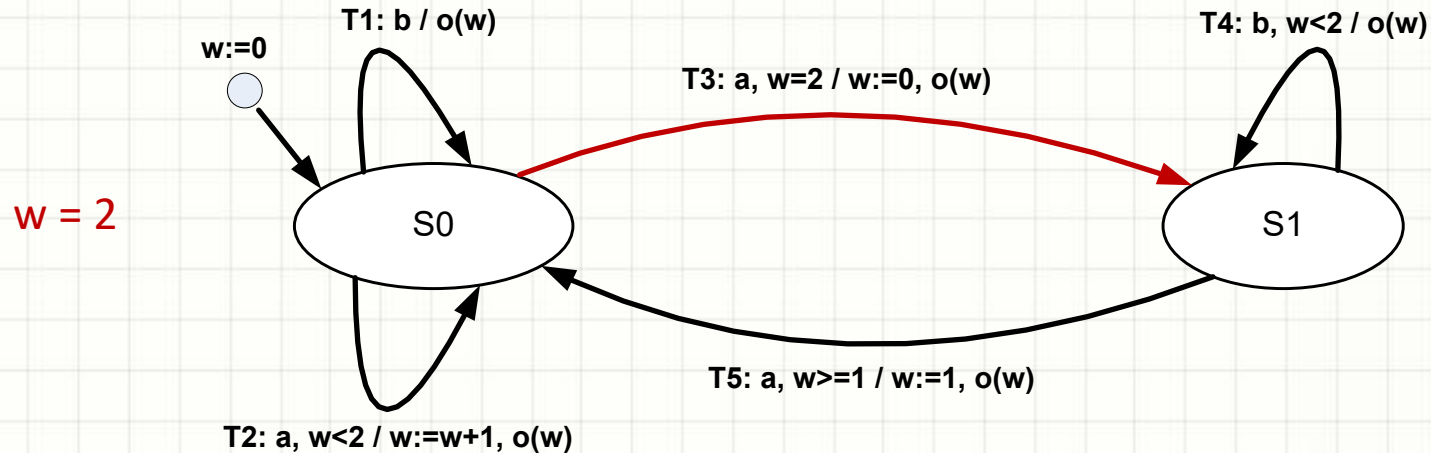


- Test Suite

- Test Case 1

- $S_0 - ?a/!o(1) - S_0 - ?b/!o(1) - S_0 - ?a/!o(2) - S_0$

# Test Cases & Test Suites (Cont.)

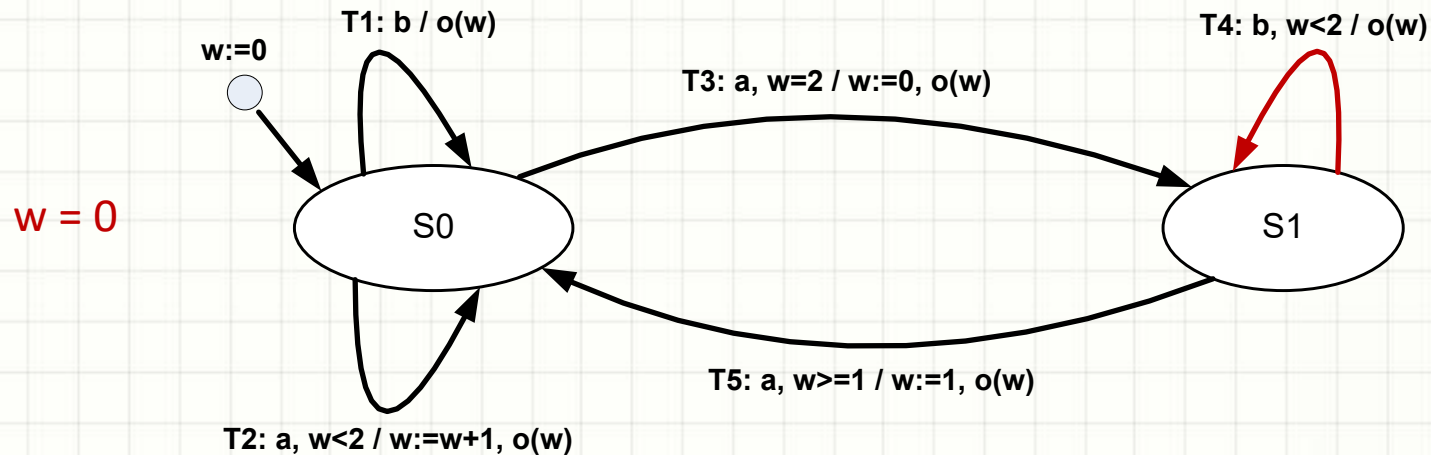


- Test Suite

- Test Case 1

$S0 - ?a / !o(1) - S0 - ?b / !o(1) - S0 - ?a / !o(2) - S0 - ?a / !o(0) - S1$

# Test Cases & Test Suites (Cont.)

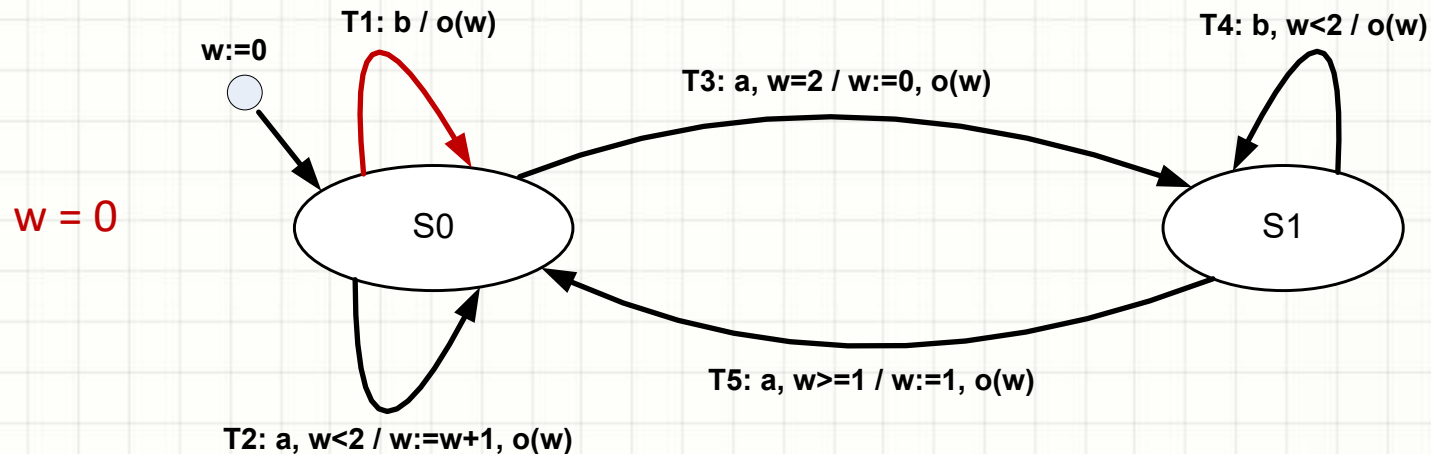


- Test Suite

- Test Case 1

$S0 - ?a/!o(1) - S0 - ?b/!o(1) - S0 - ?a/!o(2) - S0 -$   
 $?a/!o(0) - S1 - ?b/!o(0) - S1$

# Test Cases & Test Suites (Cont.)



- Test Suite

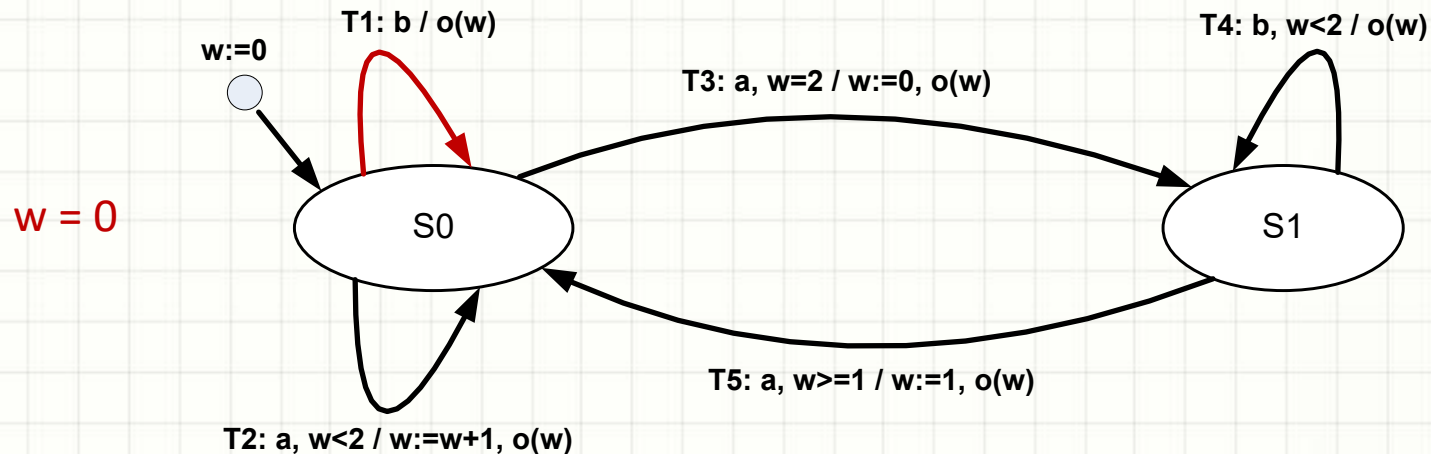
- Test Case 1

$S0 - ?a/!o(1) - S0 - ?b/!o(1) - S0 - ?a/!o(2) - S0 - ?a/!o(0) - S1 - ?b/!o(0) - S1$

- Test Case 2

$S0 - ?b/!o(0) - S0$

# Test Cases & Test Suites (Cont.)



- Test Suite

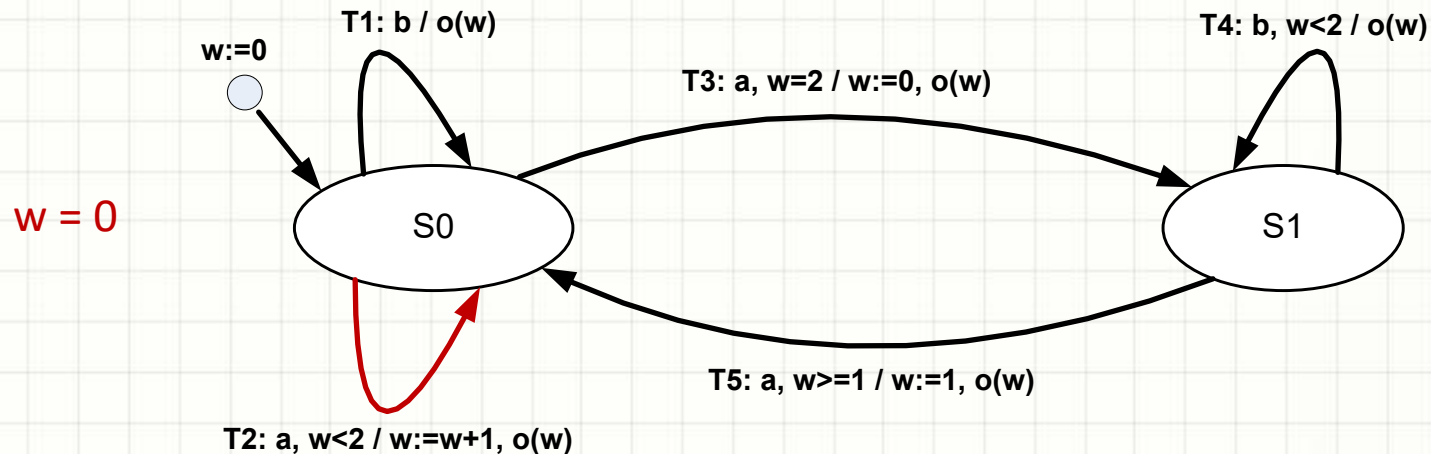
- Test Case 1

$S0 - ?a/!o(1) - S0 - ?b/!o(1) - S0 - ?a/!o(2) - S0 - ?a/!o(0) - S1 - ?b/!o(0) - S1$

- Test Case 2

$S0 - ?b/!o(0) - S0 - ?b/!o(0) - S0$

# Test Cases & Test Suites (Cont.)



- Test Suite

- Test Case 1

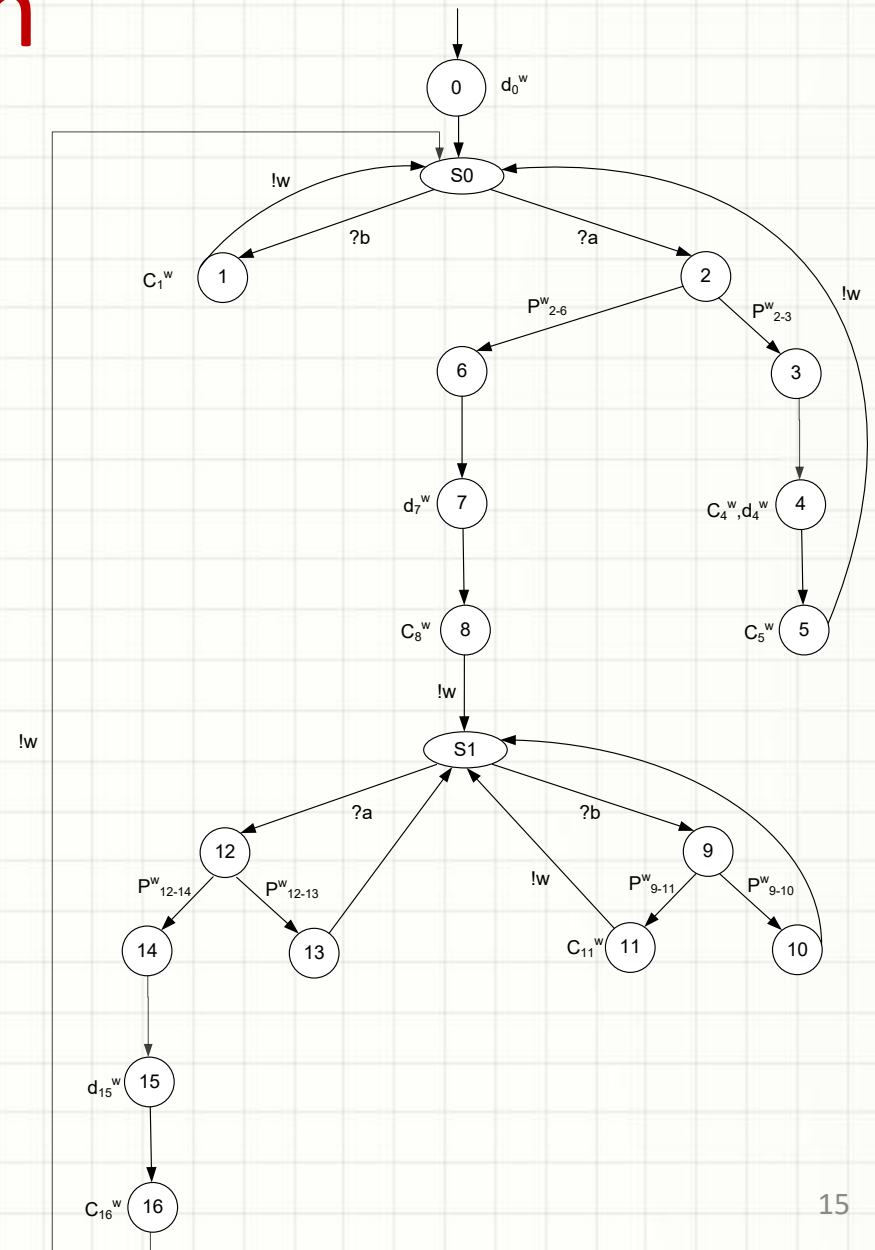
$S_0 - ?a / !o(1) - S_0 - ?b / !o(1) - S_0 - ?a / !o(2) - S_0 - ?a / !o(0) - S_1 - ?b / !o(0) - S_1$

- Test Case 2

$S_0 - ?b / !o(0) - S_0 - ?b / !o(0) - S_0 - ?a / !o(1) - S_0$

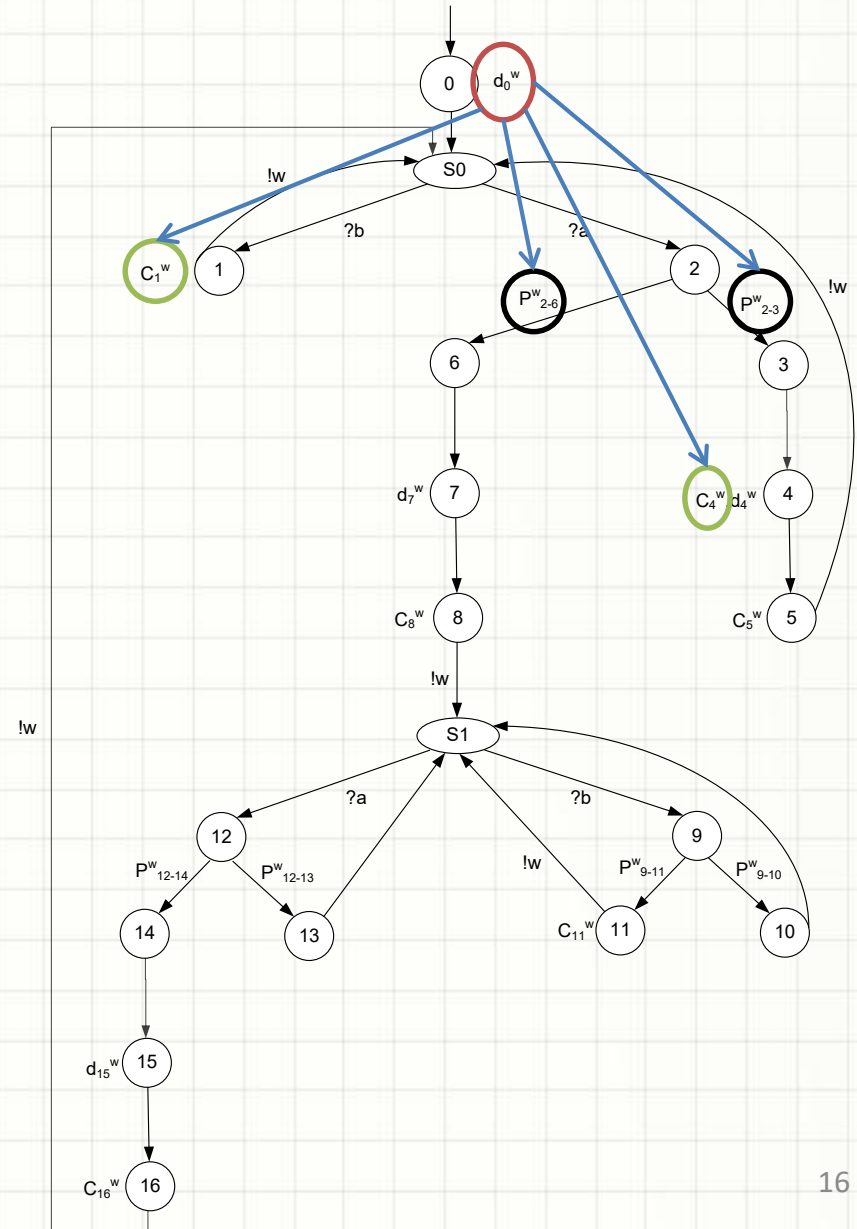
# EFSM Flow-Graph

EFSM Flow-Graph is a transformation of EFSM model with notations for variables' Definitions, C-Uses and P-Uses



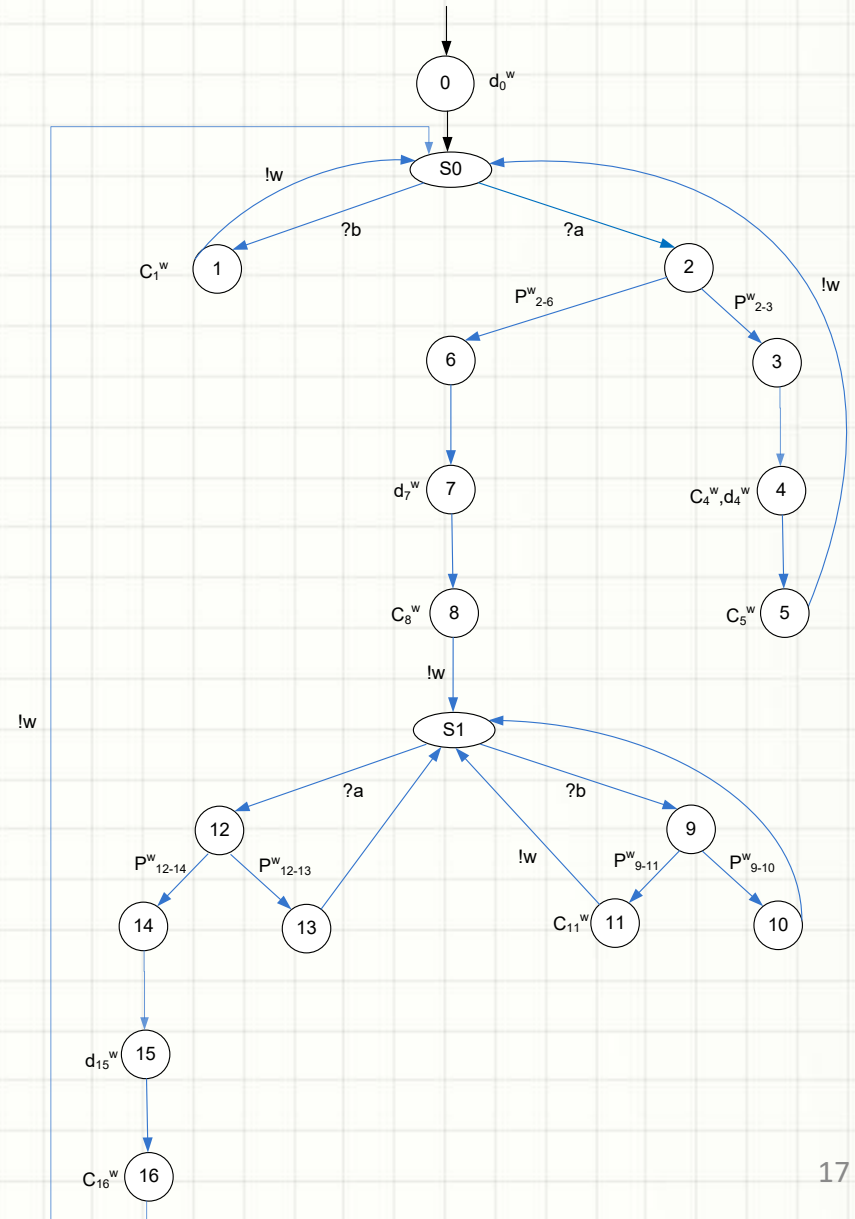
# Data-Flow, Control-Flow Test Suites

– All-Uses test suite.



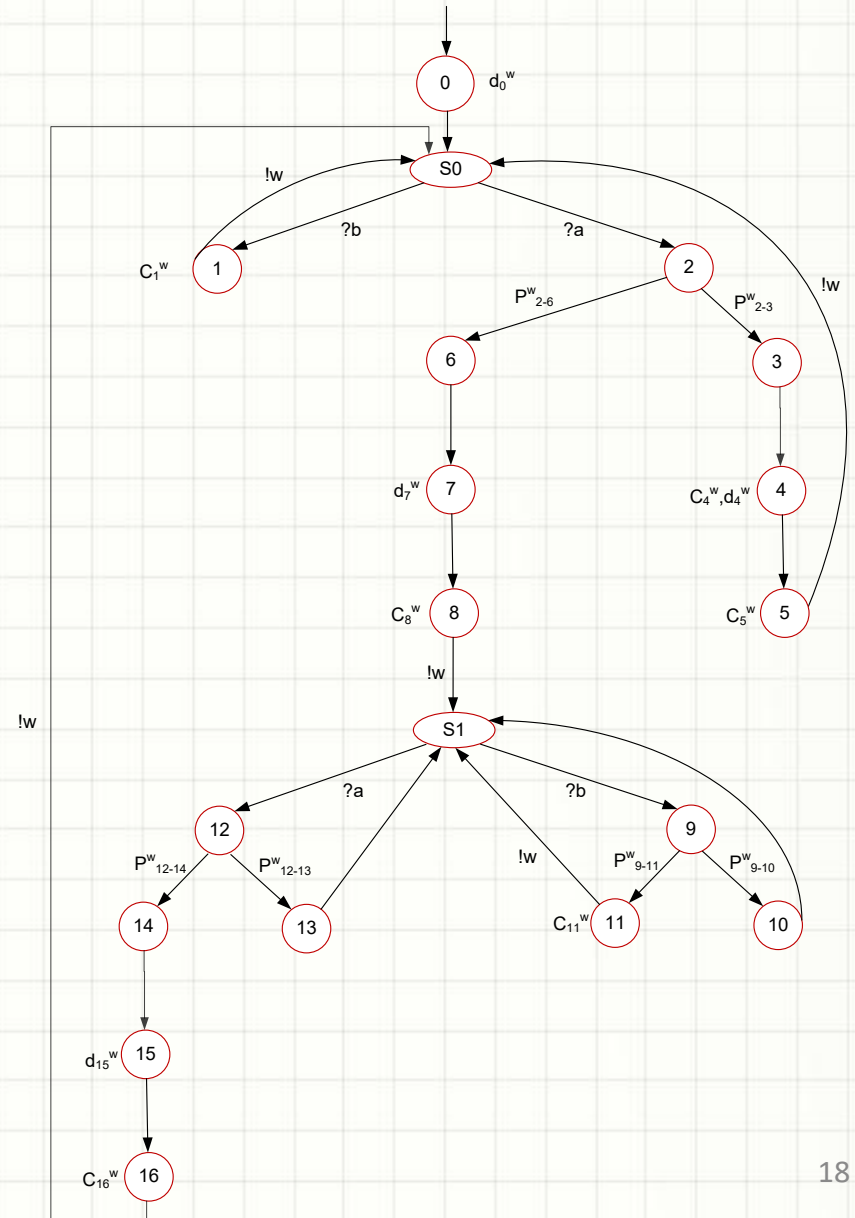
# Data-Flow, Control-Flow Test Suites

- All-Uses test suite.
- All-Edges test suite.



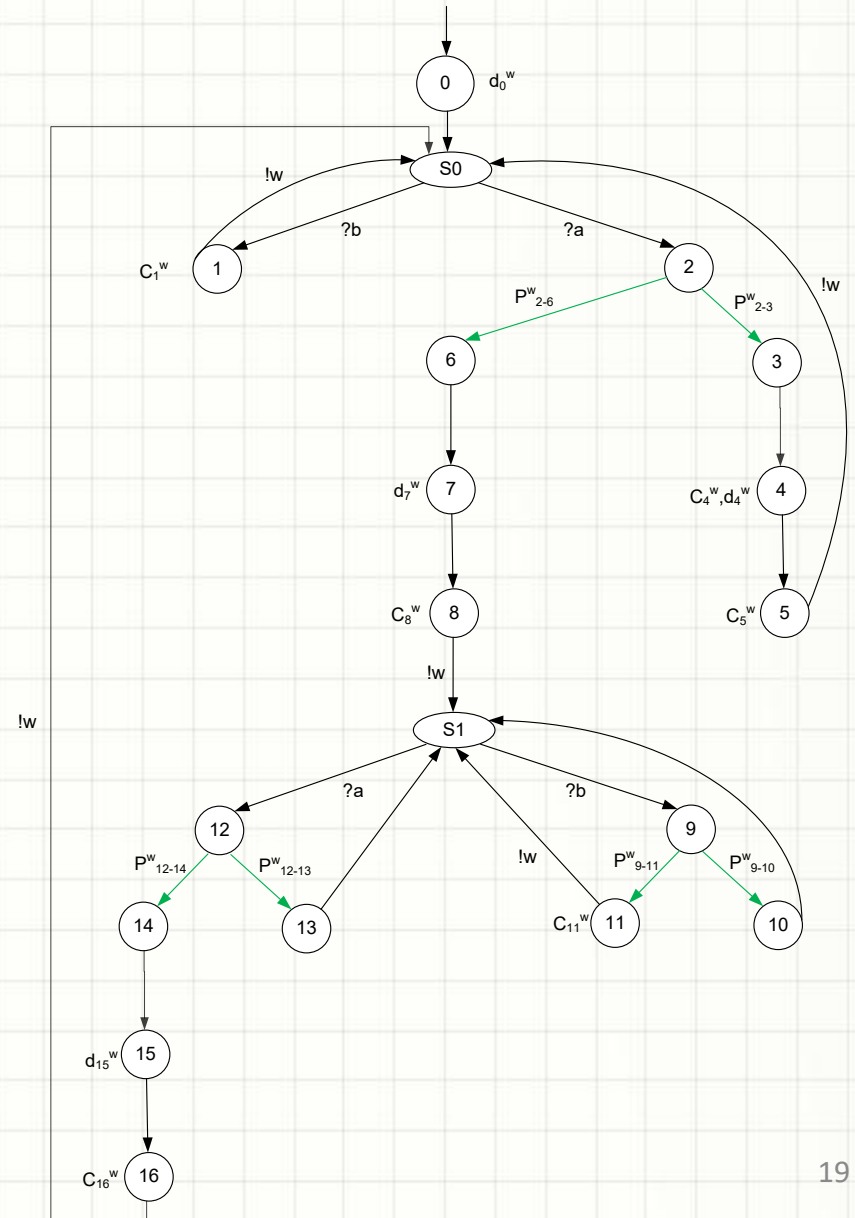
# Data-Flow, Control-Flow Test Suites

- All-Uses test suite.
- All-Edges test suite.
- All-Nodes test suite.



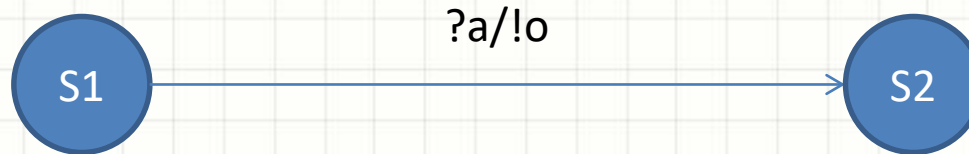
# Data-Flow, Control-Flow Test Suites

- All-Uses test suite.
- All-Edges test suite.
- All-Nodes test suite.
- All-Decisions test suite.



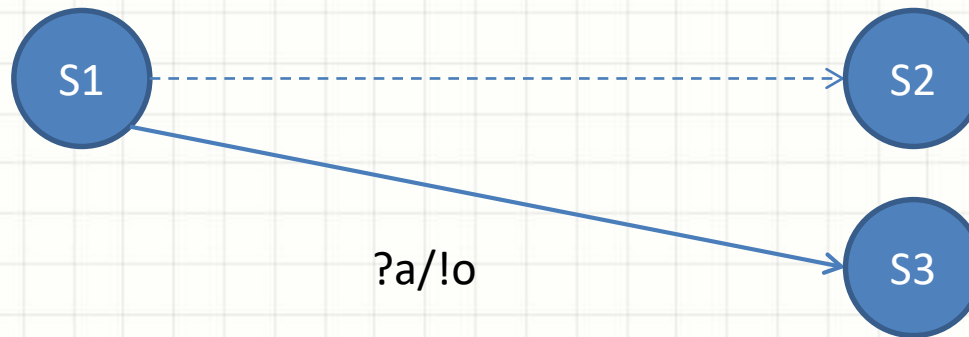
# EFSM Test Suites

- Single Transfer Fault (STF) test suite.



# EFSM Test Suites

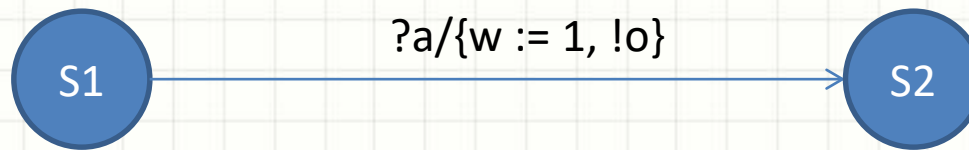
- Single Transfer Fault (STF) test suite.



- Double Transfer Fault (DTF) test suite.

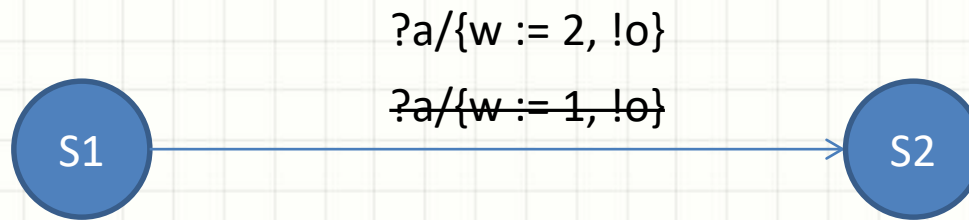
# EFSM Test Suites

- Single Assignment Fault (SAF) test suite.



# EFSM Test Suites

- Single Assignment Fault (SAF) test suite.



- Double Assignment Fault (DAF) test suite.

# Mutation Testing

- Mutation testing is a mechanism to evaluate and assess the quality of a test suite and to guarantee its efficiency by checking the **coverage of the test suite in terms of number of killed mutants** [1].
- **Expensive** but **efficient** in fault and error detection [2].
- Cost can be reduced by selecting mutation operators **carefully**.

[1] Tatiana Sugeta, Jose Carlos Maldonado, and W. Eric Wong, "Mutation Testing Applied to Validate SDL Specifications," LNCS, pp. 193-208, 2004.

[2] DeMillo R.A and Offutt A.J, "Experimental results from an automatic test case generator," ACM Trans<sub>4</sub> Software Engineering, vol. 2, pp. 109–127, 1993.

# Coverage Assessment of Mutation Testing

- Although Mutant generation was originally proposed as part of a **testing strategy**, Thevenod-Fosse et al. [3] used it as a method for generating faulty versions for experiments.
- James H. Andrews et al. [4] compared **four** different test suites, **Block, C-Use, Decision, and P-Use**, in which they found that C-Use and P-Use test suites were able to kill more faulty machines than the others.

[3] P. Thevenod-Fosse, H. Waeselynck, and Y. Crouzet, "An Experimental Study on Software Structural Testing: Deterministic versus Random Input Generation," Proc. 21st Int'l Symp. Fault-Tolerant Computing, pp. 410-417, June 1991.

[4] J.H. Andrews, L.C. Briand, Y. Labiche, and A.S. Namin, "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria," IEEE Transactions on Software Engineering, vol. 32, pp. 608-624, August 2006.

# Coverage Assessment of Mutation Testing

- Offutt J. et al. [5] compared four testing criteria, Mutation, Edge-Pair, All-Uses, and Prime Path Coverage:

	Tests	Faults	Cost/Benefit
All-uses	362	54	6.7
Mutation	269	75	3.6

[5] Li Nan, U. Praphamontripong, and J. Offutt, "An Experimental Comparison of Four Unit Test Criteria: Mutation, Edge-Pair, All-Uses and Prime Path Coverage," Software Testing, Verification and Validation Workshops, 2009. ICSTW '09. International Conference, pp. 220-229, April 2009.

# Coverage Assessment of Mutation Testing

- Frankl et al. compared the **effectiveness** of All-Uses vs. Mutation Testing in which they found that mutation testing did better than All-Uses [6].
- Kakarla et al. compared between Data-Flow testing and mutation testing [7] in which they found that Data-Flow testing outperformed mutation testing in the number of test cases required (**cost**), but mutation testing was twice as effective as Data-Flow testing in exposing faults (**benefit**).

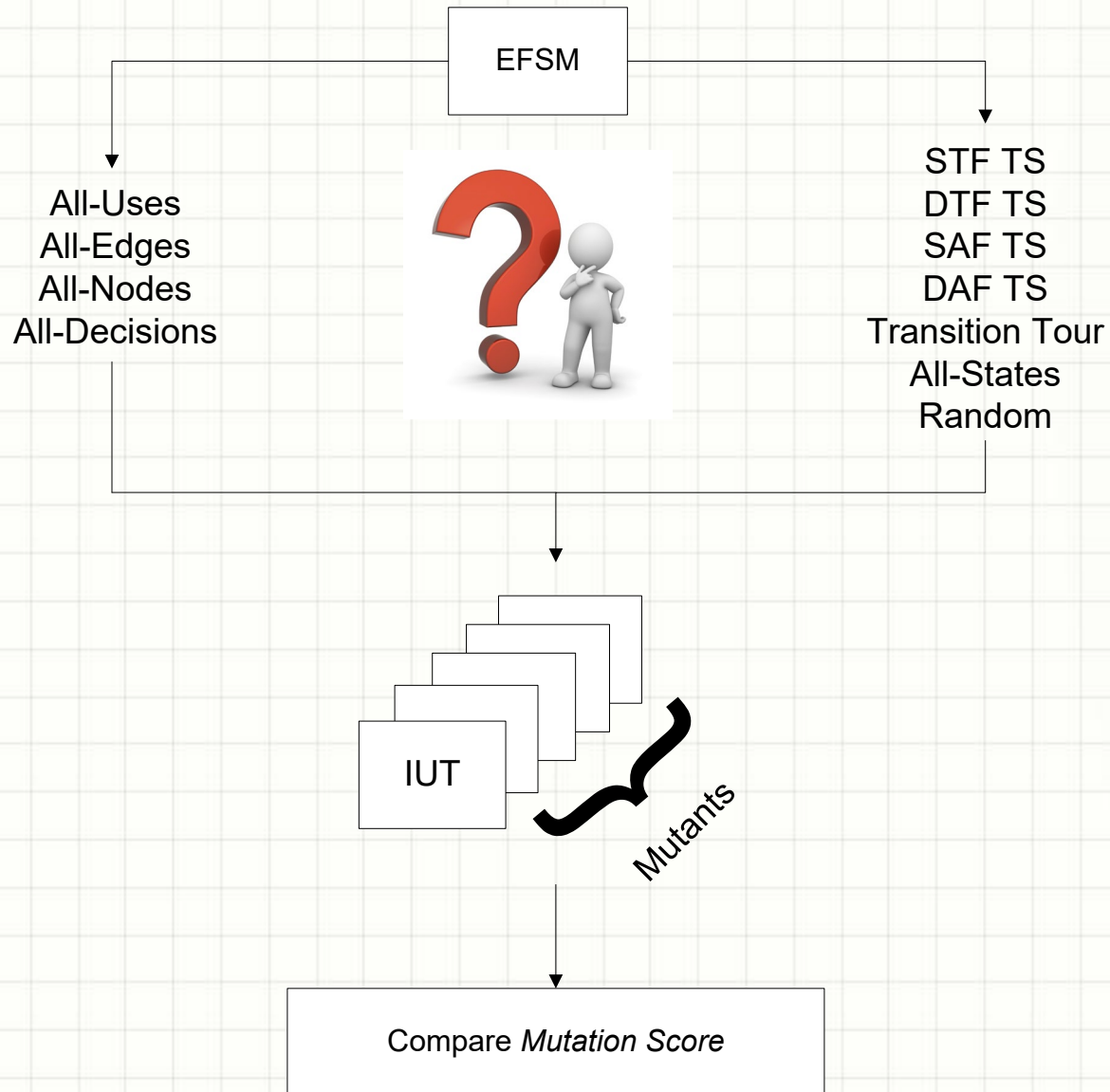
[6] Phyllis G. Frankl, Stewart N. Weiss, and Cang Hu, "All-Uses versus Mutation Testing: An Experimental Comparison of Effectiveness," June 1996.

[7] S. Kakarla, S. Momotaz, and A.S. Namin, "An Evaluation of Mutation and Data-Flow Testing: A Meta-analysis," Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference, pp. 366-375, March 2011.

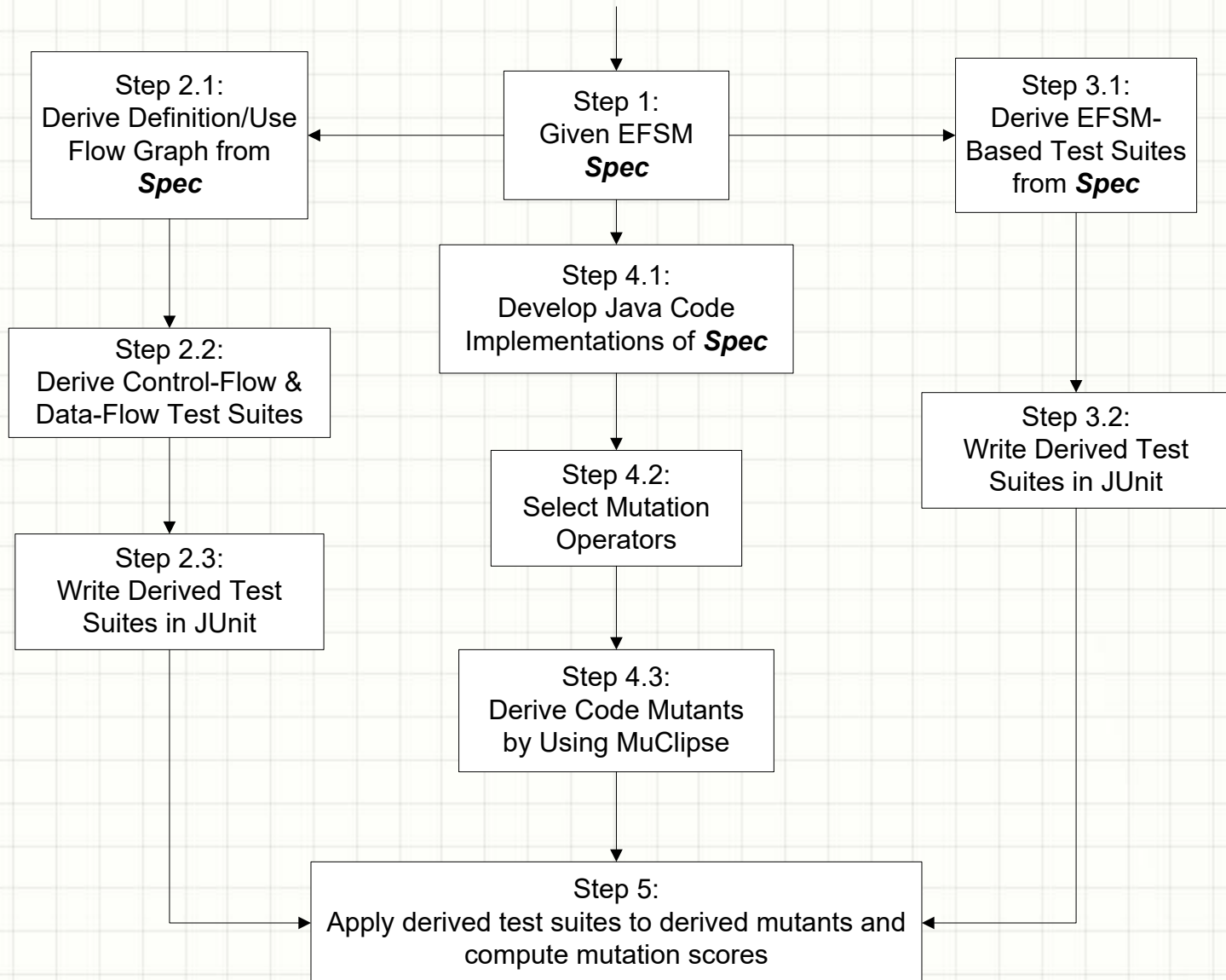


# **ASSESSING CONTROL-FLOW, DATA-FLOW AND EFSM BASED TEST SUITES**

# Research Objectives

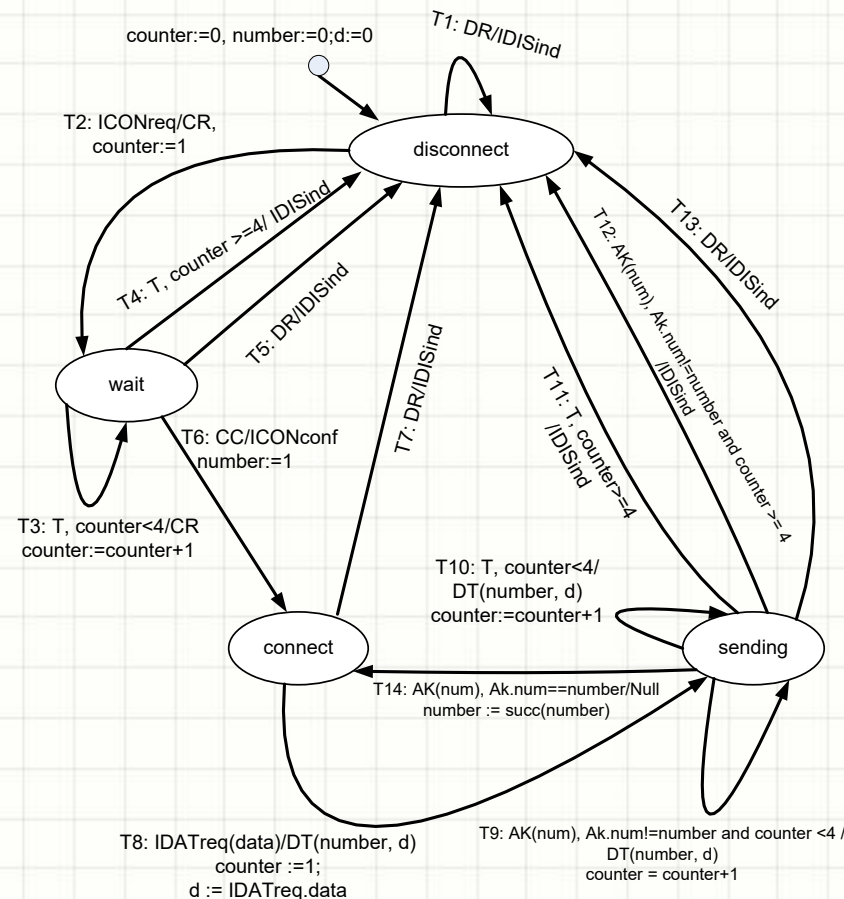


# Assessment Methodology



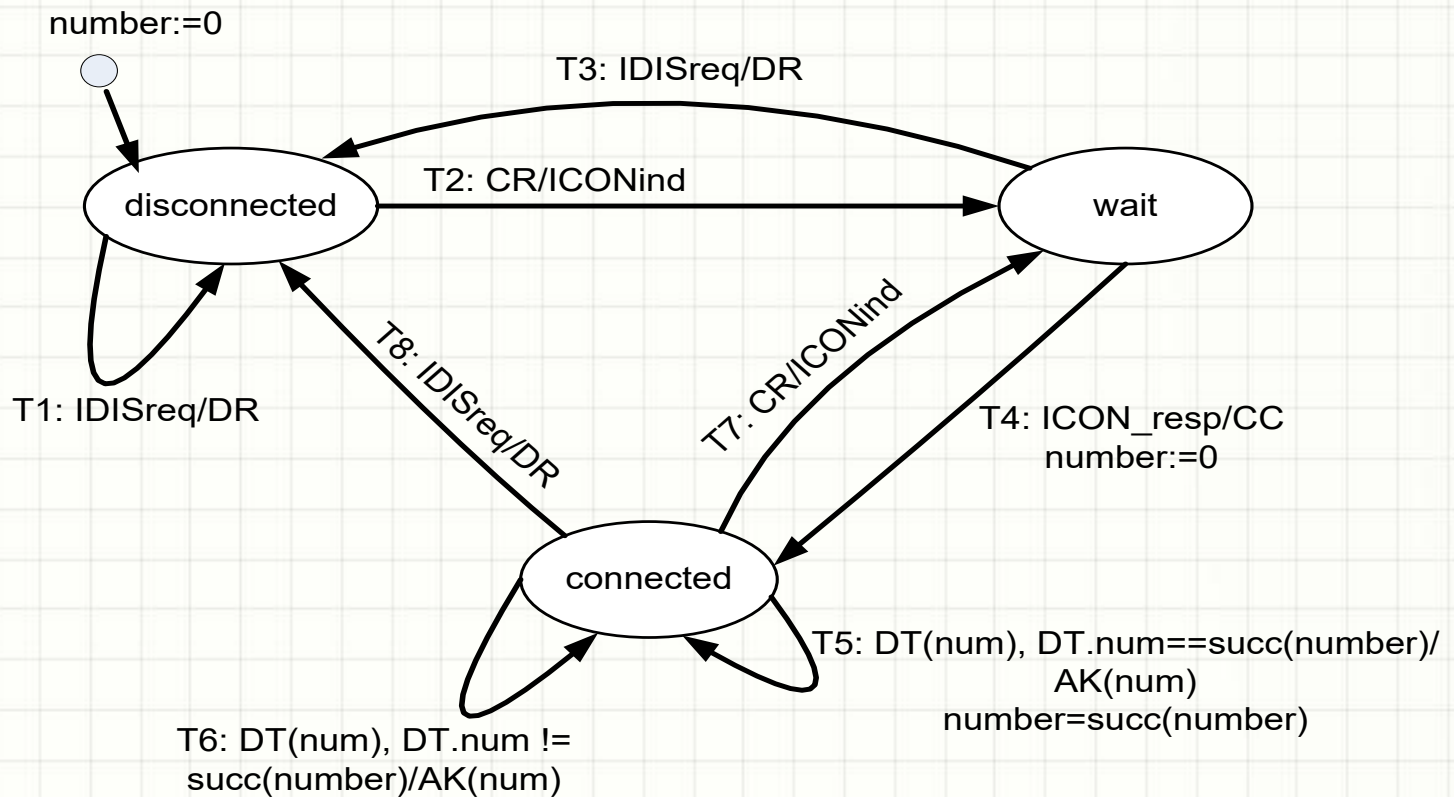
# Considered EFSM Specification

- Case Study 1: Initiator [8]



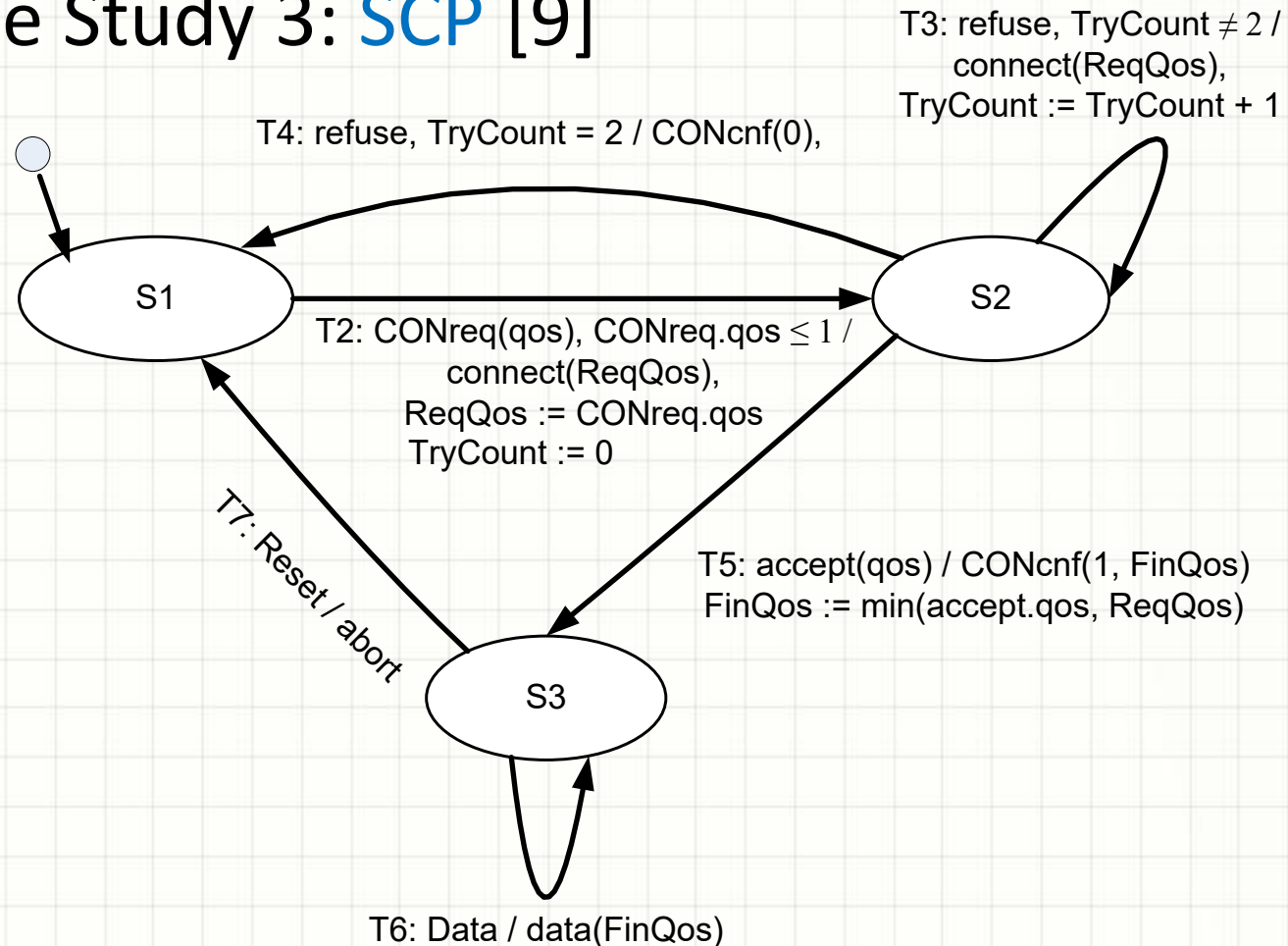
# Considered EFSM Specification (Cont.)

- Case Study 2: Responder [8]



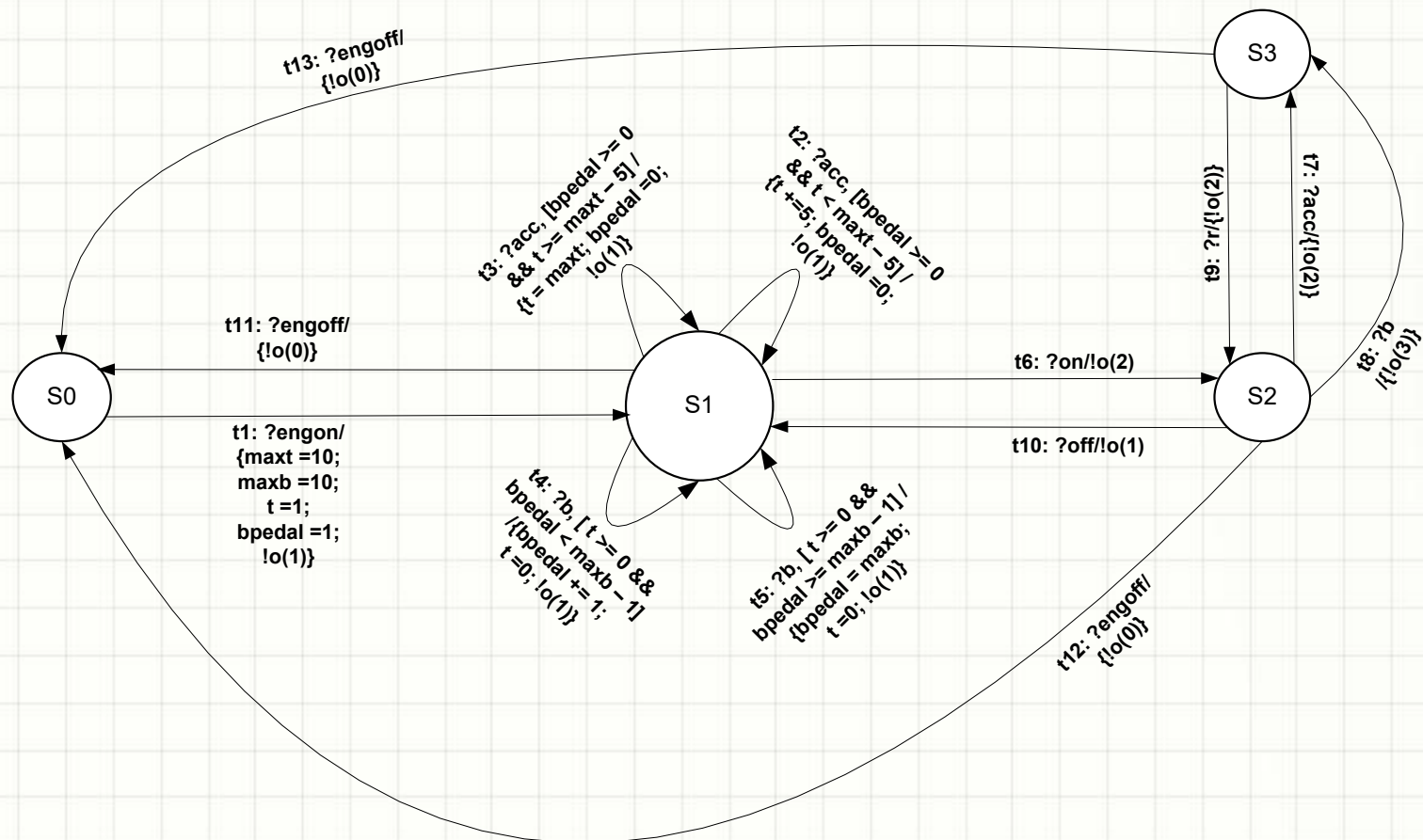
# Considered EFSM Specification (Cont.)

- Case Study 3: SCP [9]



# Considered EFSM Specification (Cont.)

- Case Study 4: Cruise Control [10]



# Random Test Suites

- A **Random Test Suite** is a test suite generated by a random walk through (or from a randomly generated path of) the EFSM Specification.
- Considered Random Test Suites:
  - Same length as the **All-Uses** test suite:
    - With **one** test case.
    - With the **same number** of test cases.
  - Same length as the **All-Edges** test suite:
    - With **one** test case.
    - With the **same number** of test cases.

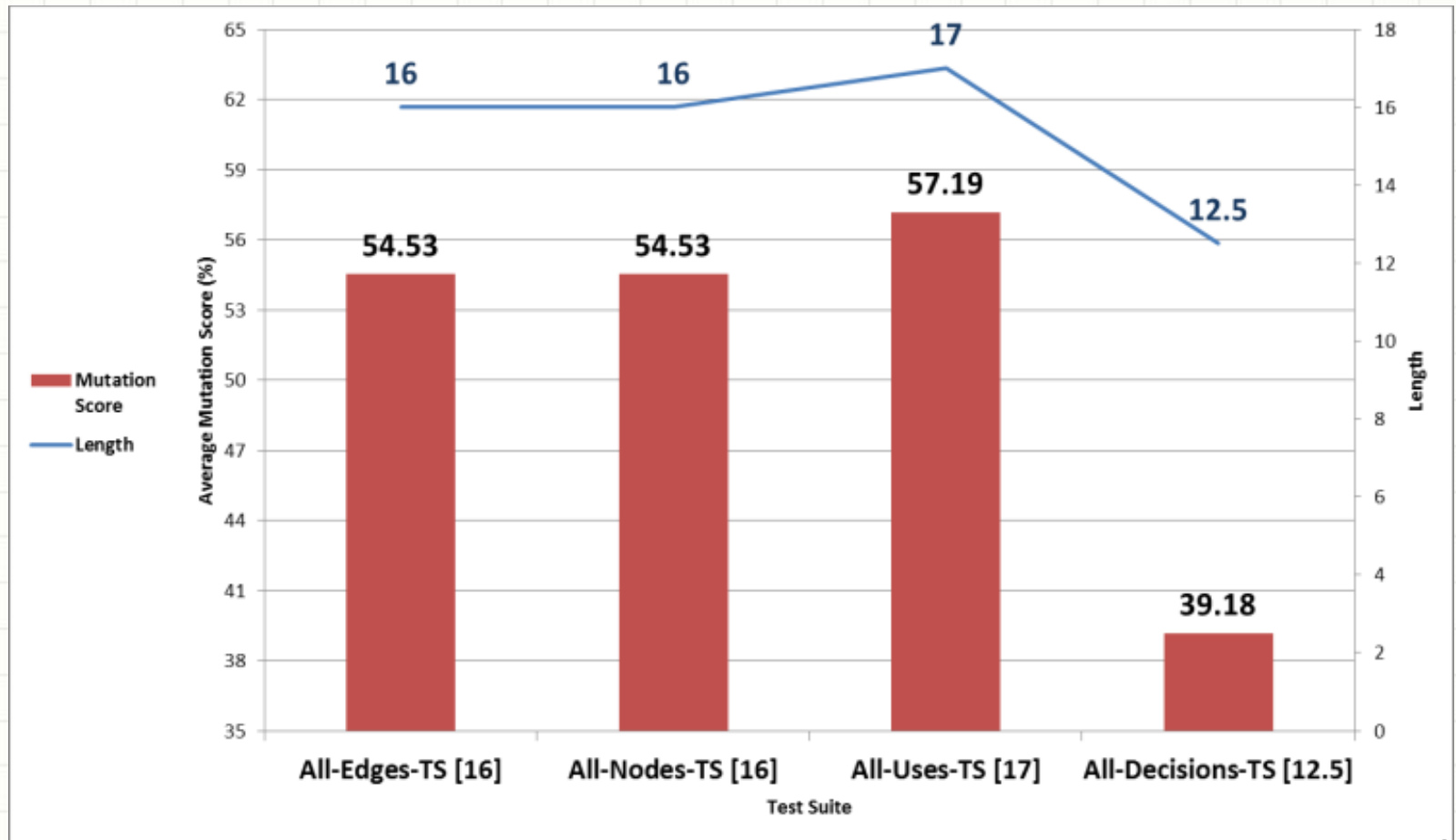
# Mutation Operators

- Code mutants are derived using the following well-known types of mutation operators [11]:

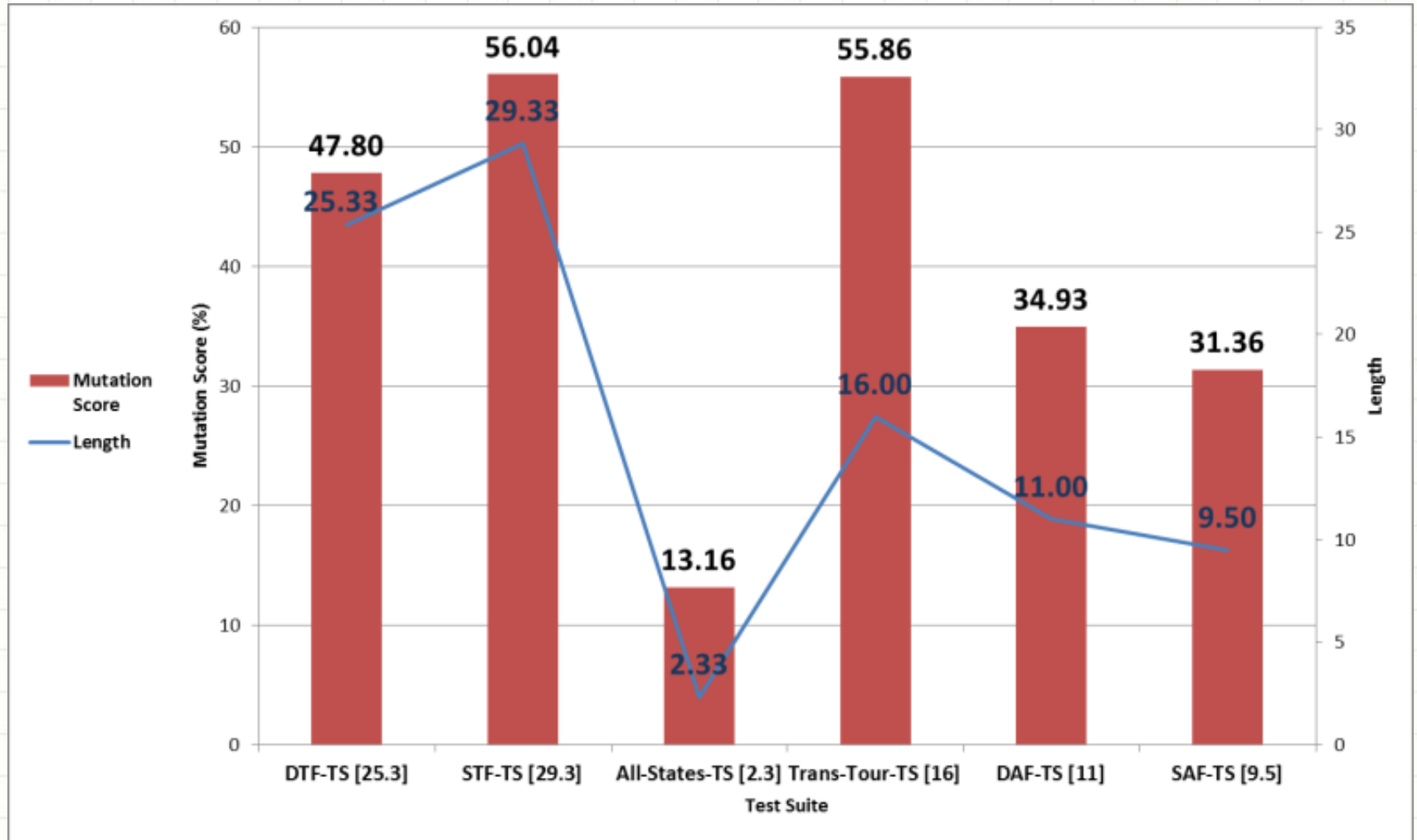
Operator	Description
AOR	Arithmetic Operator Replacement
AOI	Arithmetic Operator Insertion
AOD	Arithmetic Operator Deletion
ROR	Relational Operator Replacement
COR	Conditional Operator Replacement
COI	Conditional Operator Insertion
COD	Conditional Operator Deletion
SOR	Shift Operator Replacement
LOR	Logical Operator Replacement
LOI	Logical Operator Insertion
LOD	Logical Operator Deletion
ASR	Assignment Operator Replacement

# Experimental Results

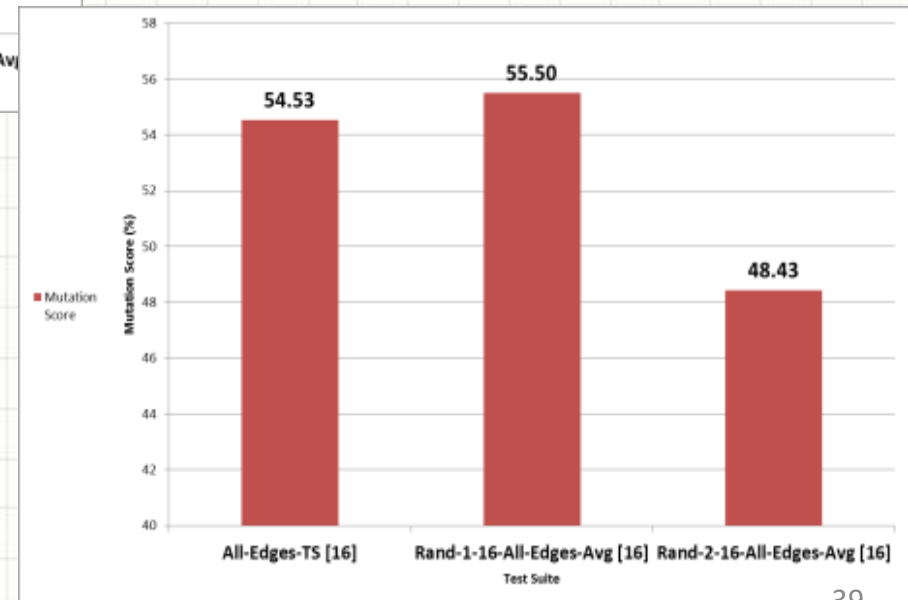
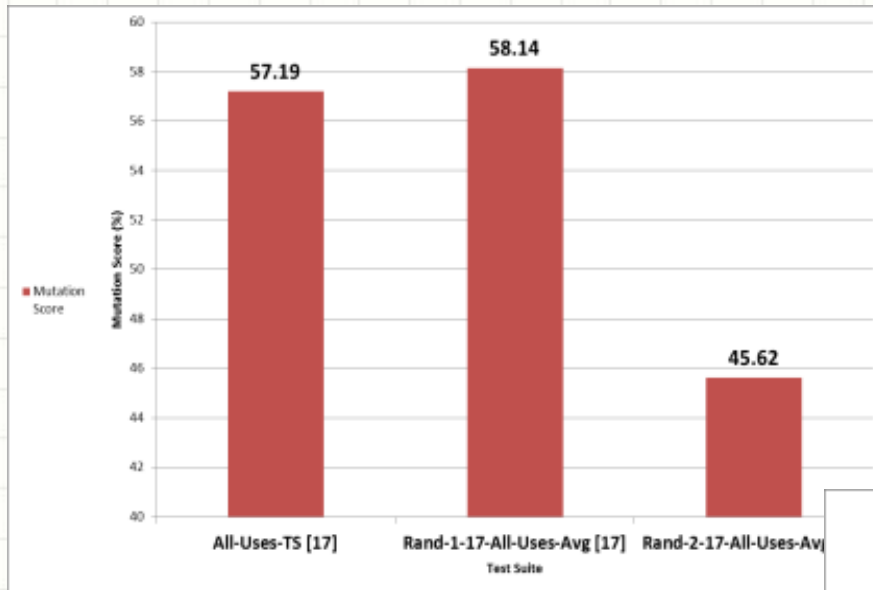
- Coverage of Control-Flow and Data-Flow Test Suites:



- Coverage of EFSM-Based Test Suites



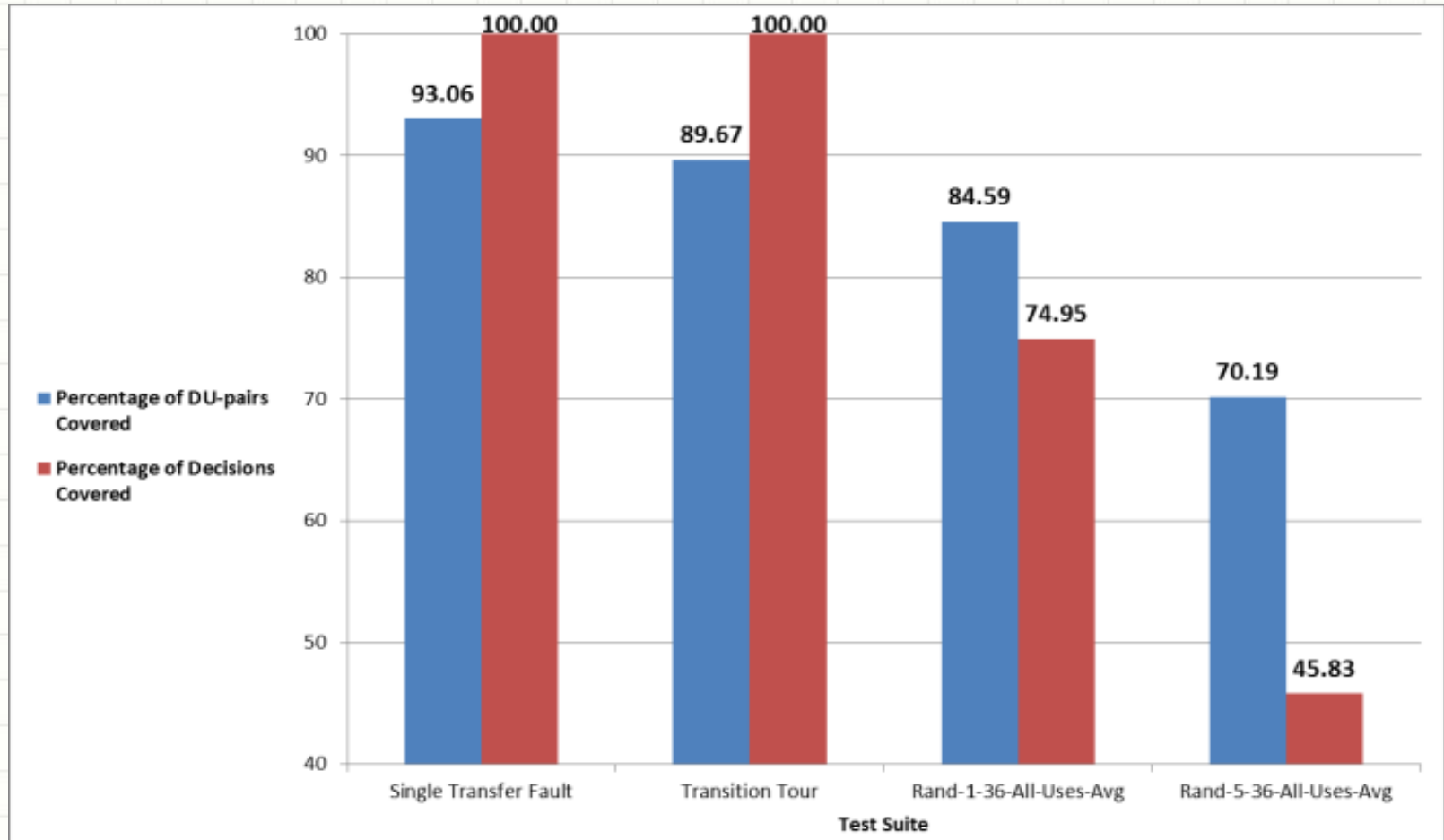
- Coverage of EFSM-Based Random Test Suites versus All-Uses and All-Edges Test Suites



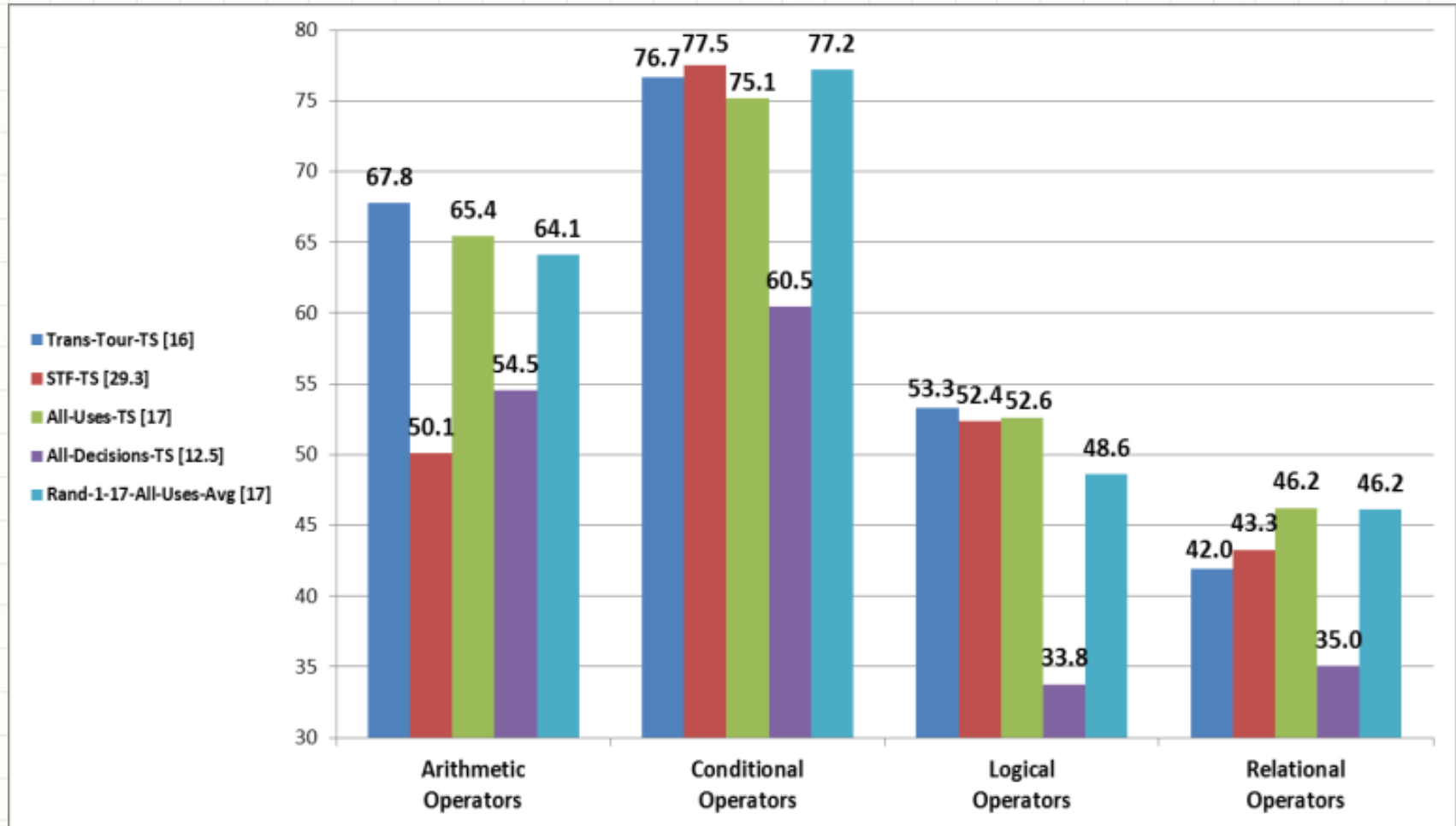
- Coverage of Best Control-Flow, Data-Flow and EFSM-Based Test Suites



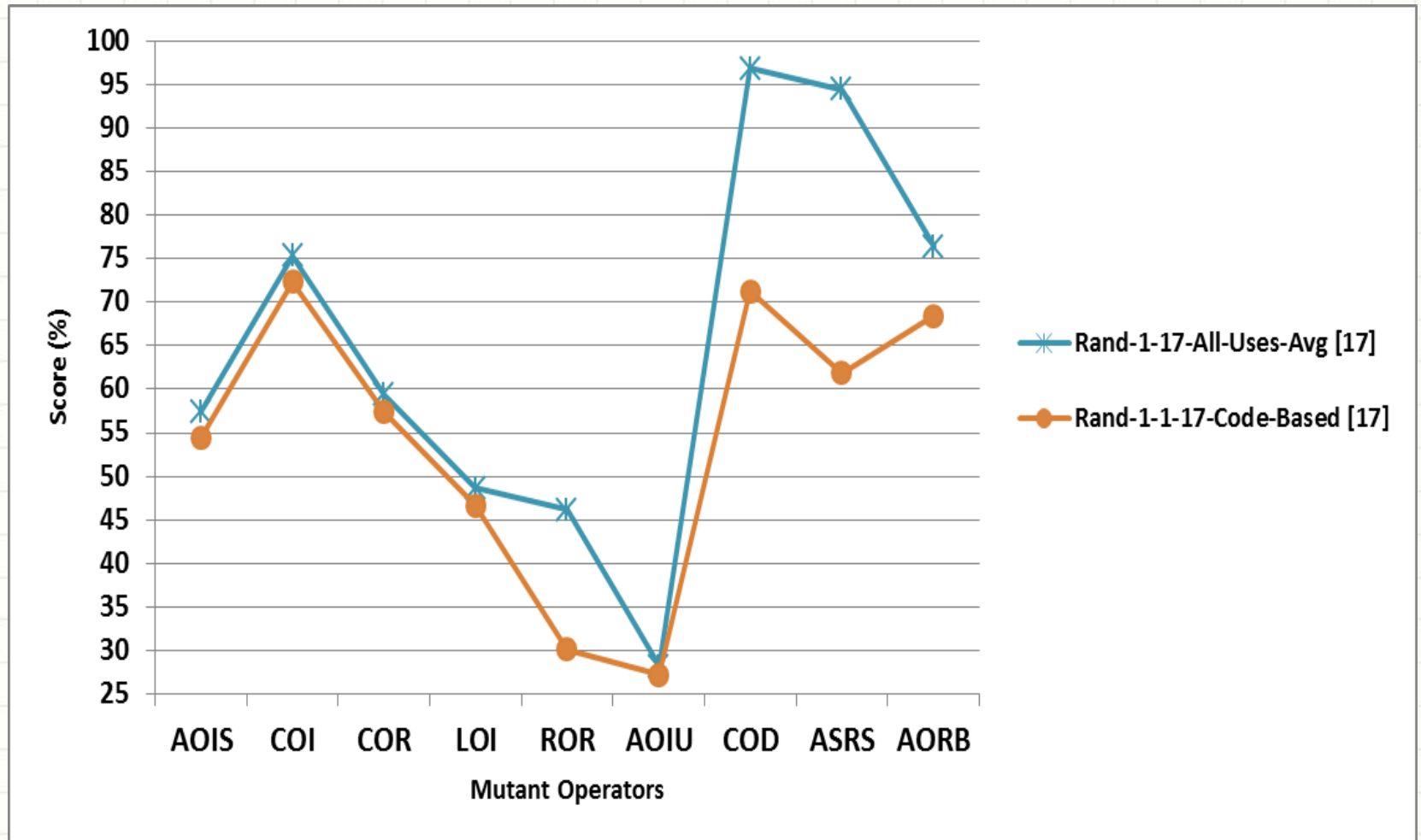
- Coverage of Best EFSM-Based Test Suites of All-Uses and All-Decisions Test Suites



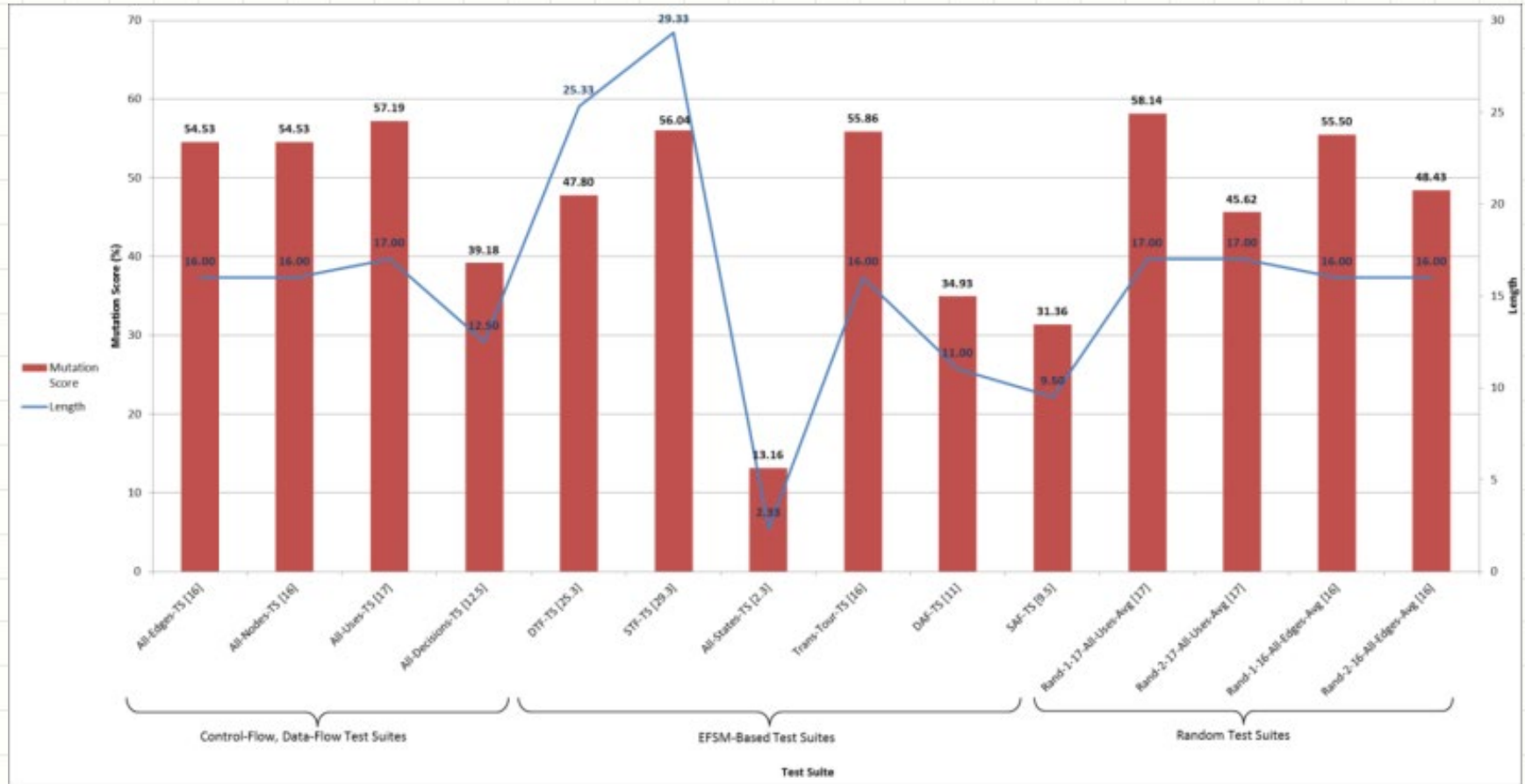
- Coverage of Best Control-Flow, Data-Flow and EFSM-Based test suites per each mutation operators category



- **Comparison of EFSM-Based versus Code-Based Random Test Suites**



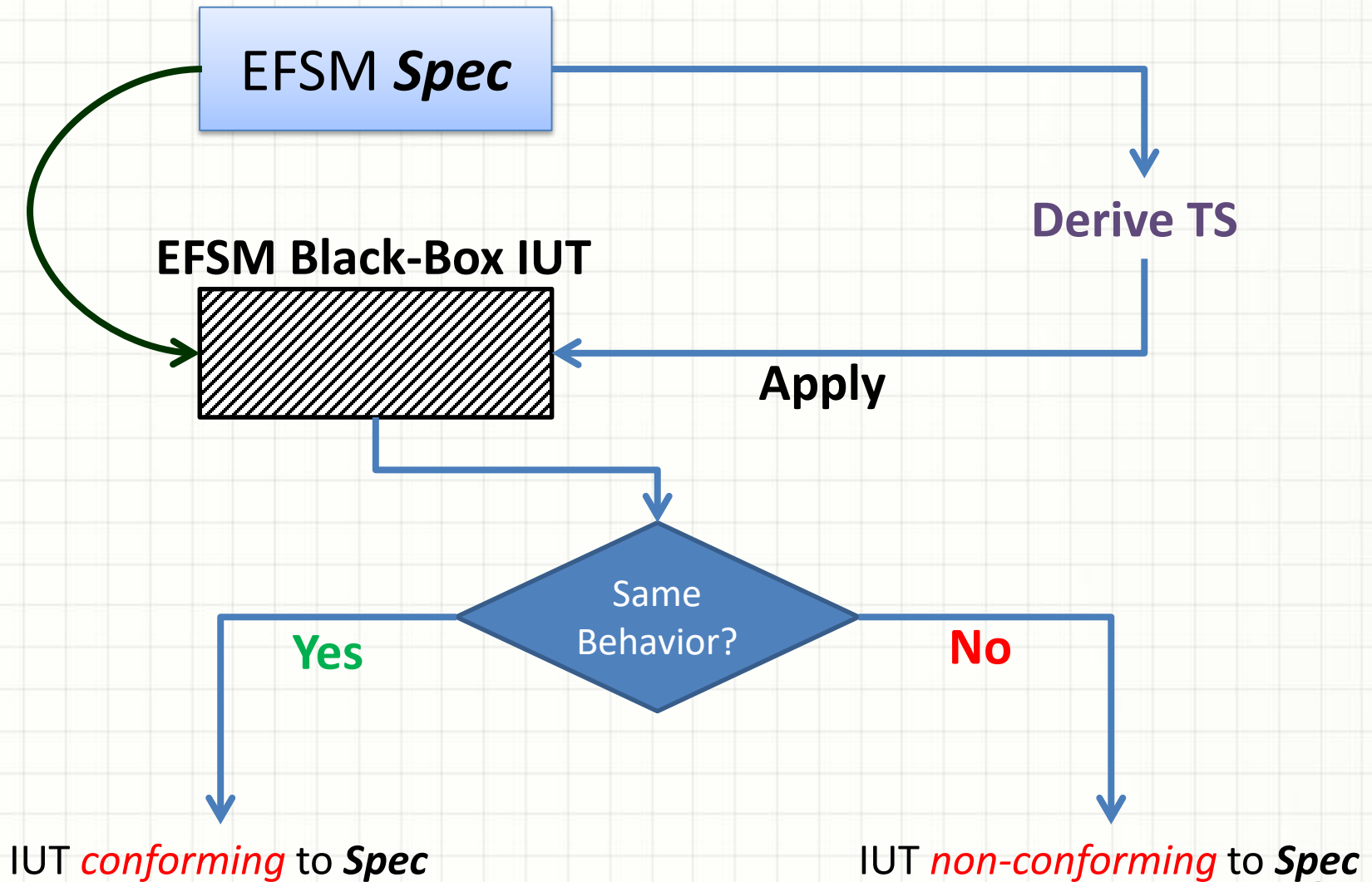
- Summary of All Results





# **TESTING WITH RESPECT TO TRANSFER FAULTS: A METHOD AND AN ASSESSMENT**

# Conformance Testing



# Assumptions

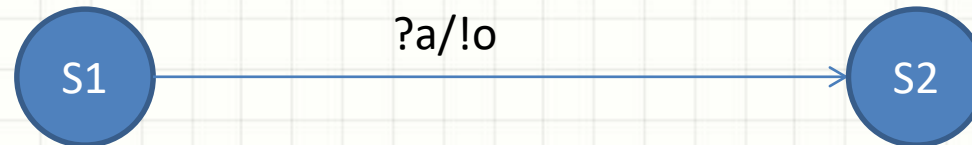
- We assume the following:

The EFSM Implementation Under Testing (IUT) has:

- the **same number of states** as the EFSM specifications.
- **No Guard Faults.**
- **No Assignment Faults.**

# Types of Considered EFSM Faults

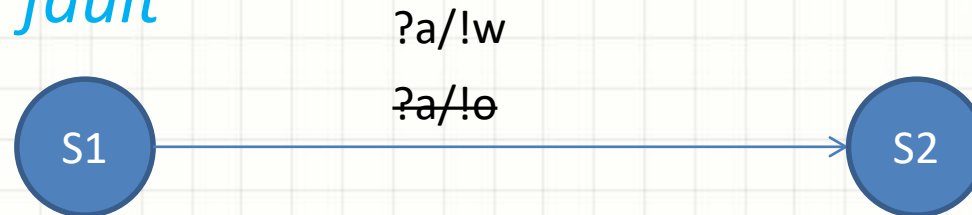
- We consider two types of EFSM faults:
  - *output fault*



# Types of Considered EFSM Faults

- We consider two types of EFSM faults:

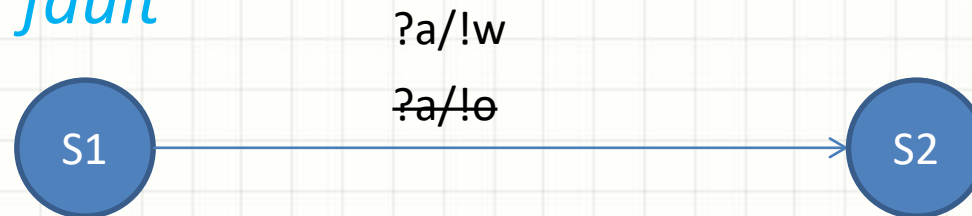
– *output fault*



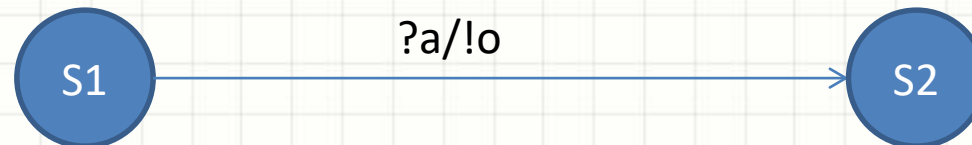
# Types of Considered EFSM Faults

- We consider two types of EFSM faults:

– *output fault*



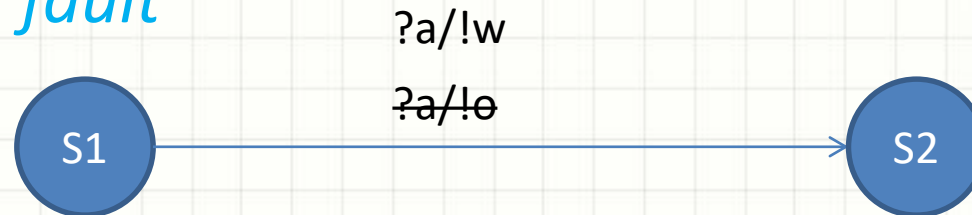
– *transfer fault*



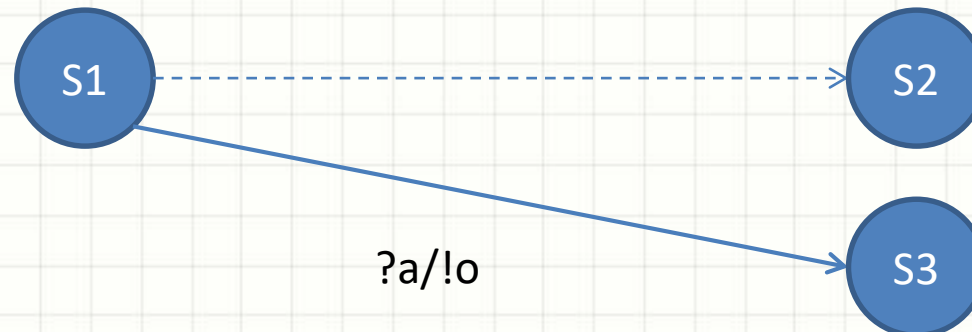
# Types of Considered EFSM Faults

- We consider two types of EFSM faults:

– *output fault*



– *transfer fault*



# Configuration Distinguishing

$(s, \mathbf{v}_1), (s, \mathbf{v}_2)$  are distinguishable if:

$\exists$  an input sequence  $\alpha$ , such that, outputs produced at  $(s, \mathbf{v}_1)$  and  $(s, \mathbf{v}_2)$  in response to  $\alpha$  are different:

$$\begin{array}{l} (s, \mathbf{v}_1) \xrightarrow{\alpha / \beta_1} \\ (s, \mathbf{v}_2) \xrightarrow{\alpha / \beta_2} \end{array} \quad \longrightarrow \quad \beta_1 \neq \beta_2$$

# State Distinguishing

$s_i, s_j$  are distinguishable if:

$\exists$  input sequence  $W_{ij}$ , such that:

$$(s_i, \mathbf{v}_1) \xrightarrow{\alpha_1 / \beta_1}$$

$$(s_i, \mathbf{v}_2) \xrightarrow{\alpha_2 / \beta_3}$$

$$W_{ij} = \{\alpha_1, \alpha_2\}$$

$$(s_j, \mathbf{v}_2) \xrightarrow{\alpha_1 / \beta_2}$$

$$(s_j, \mathbf{v}_3) \xrightarrow{\alpha_2 / \beta_4}$$

# *State-Reduced* EFSM

An EFSM is *state-reduced* if:

- Initialized: Every State is reachable from  $(s_0, \mathbf{v}_0)$ .
- $\forall$  two states  $s, s'$  are distinguishable.

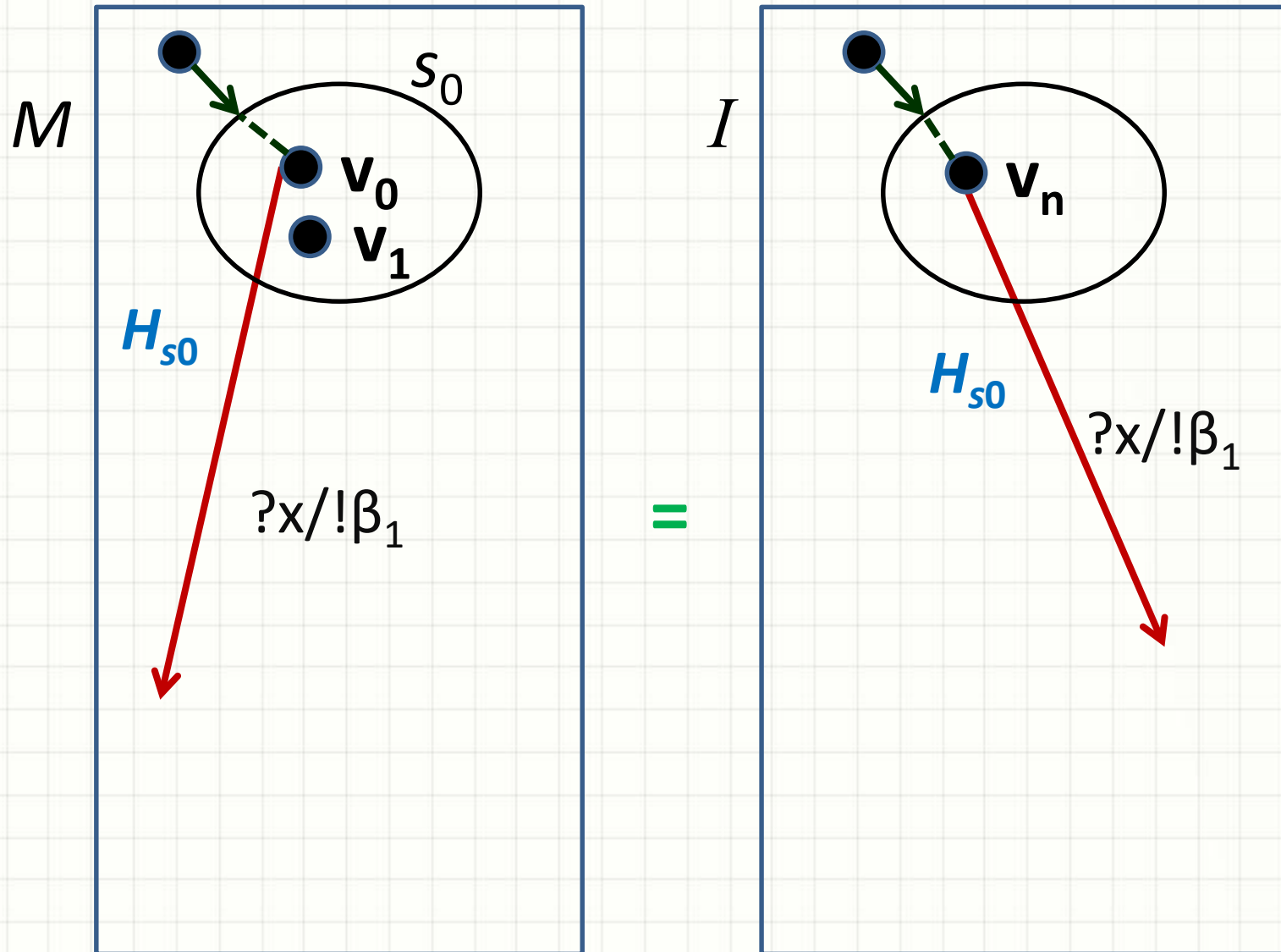
# Fault Model $(M, \cong, \mathcal{T})$

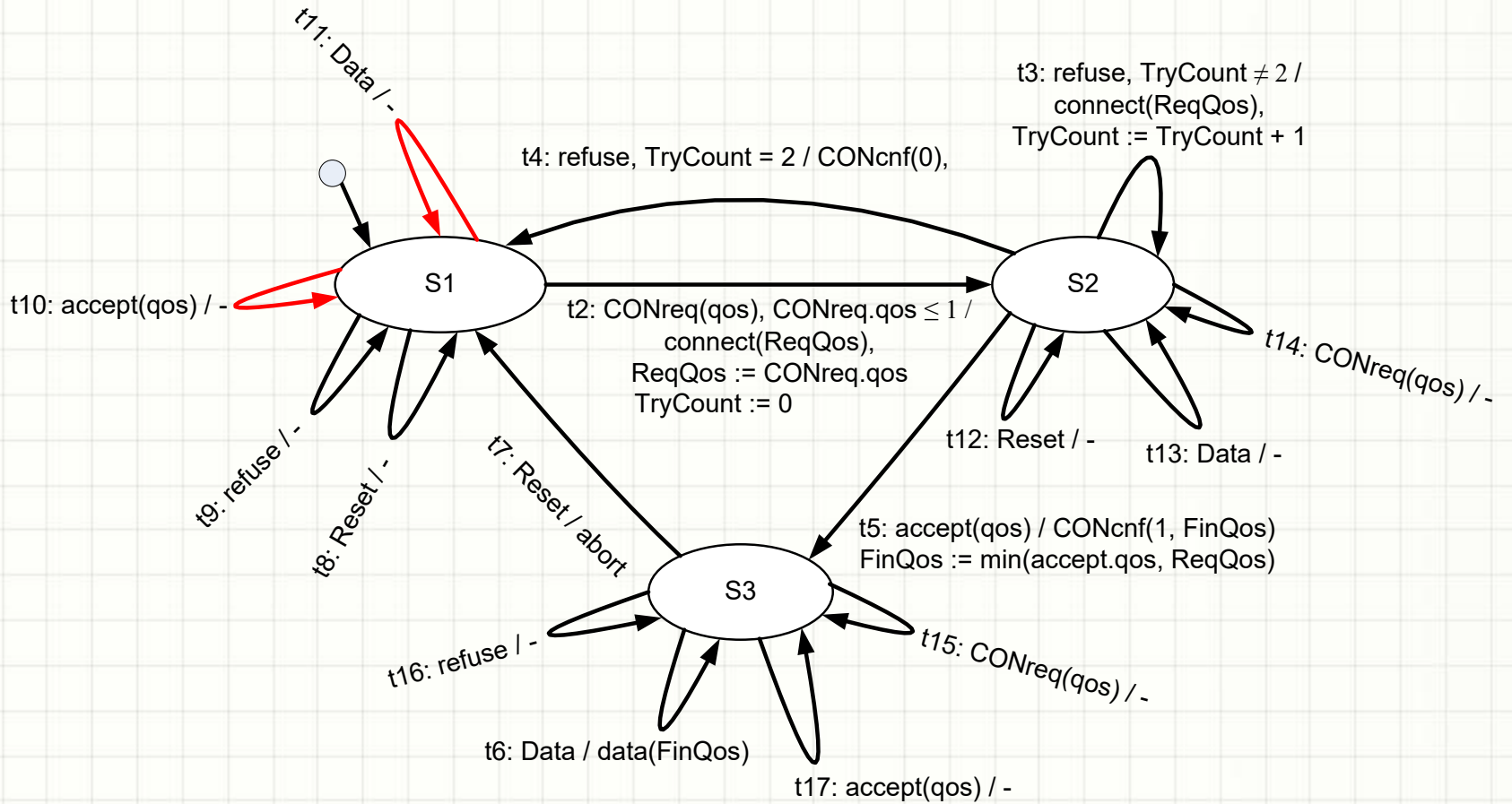
- EFSM specification  $M$  of a given system.
- *fault domain*  $\mathcal{T}$  that includes the set of all possible conforming or non-conforming EFSM implementations of the given system.
- Conformance relation  $\cong$ .

# Algorithm

- **Input:** EFSM specification  $M$ :
  - $n$  states
  - *state-reduced*
  - *Deterministic, complete, initialized*
  - *initially connected*
  - $F = \{H_{s_0}, H_{s_1}, \dots, H_{s_{n-1}}\}$  of the  $n$  states
- **Output:** A complete Test Suite  $TS$  w.r.t. the fault model  $(M, \cong, \mathcal{F})$ .

# Step-1: Verify Initial Configuration ( $s_0, \mathbf{v}_0$ )





?accept(0) / !-, ?Data / !-

$H_{s1}$

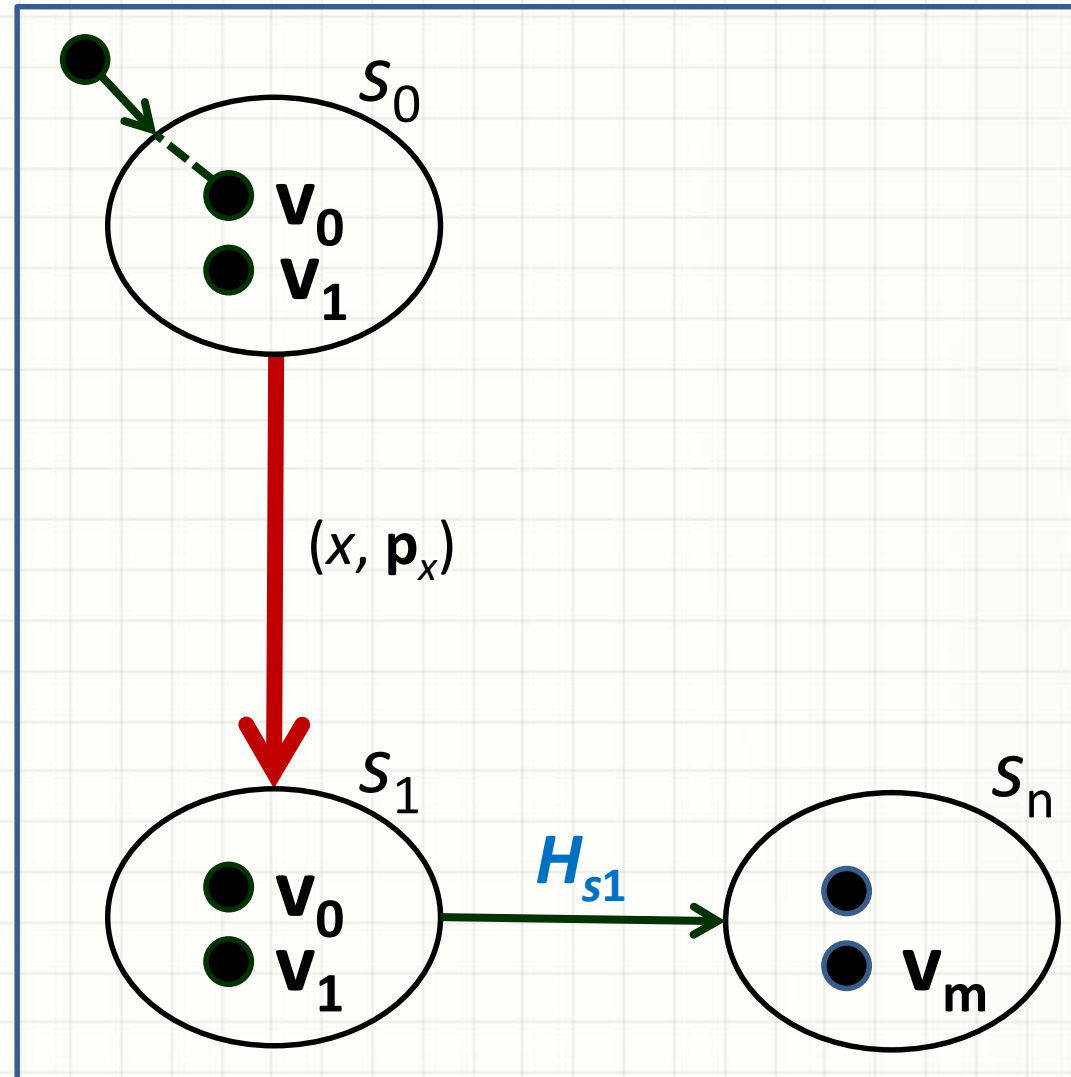
## Step-2: As possible check outgoing transitions of $s_0$ reachable from $(s_0, v_0)$

$$t = (s_0, x, P, op, y, up, s')$$

$$\exists \mathbf{p}_x: (v_0, \mathbf{p}_x) \models P$$

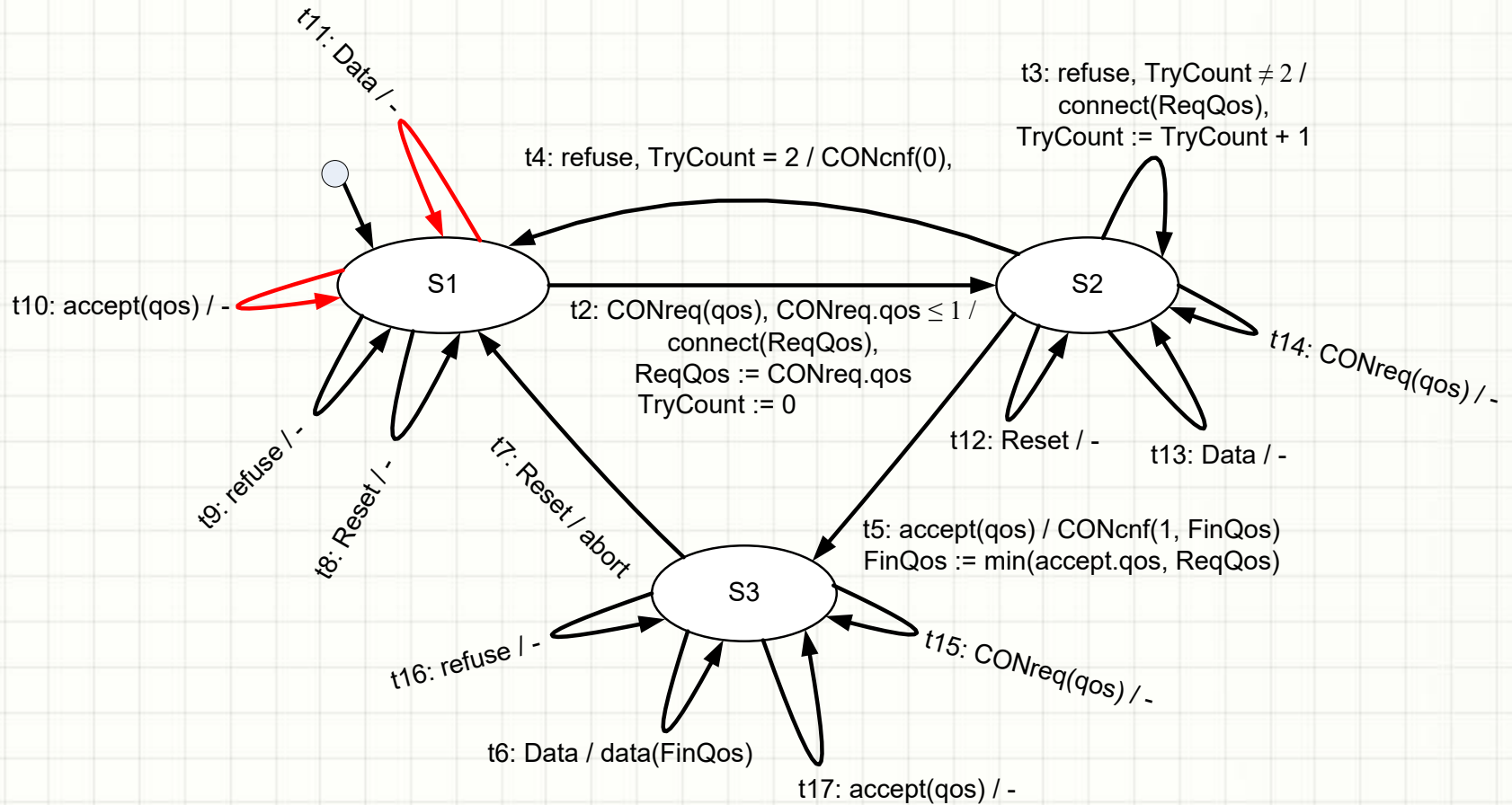
$$TC: (x, \mathbf{p}_x).H_{s'}$$

Add Transition  $t$  to  
*tested*.





Test Case to check  $t_8$ :  
 ?Reset / !-

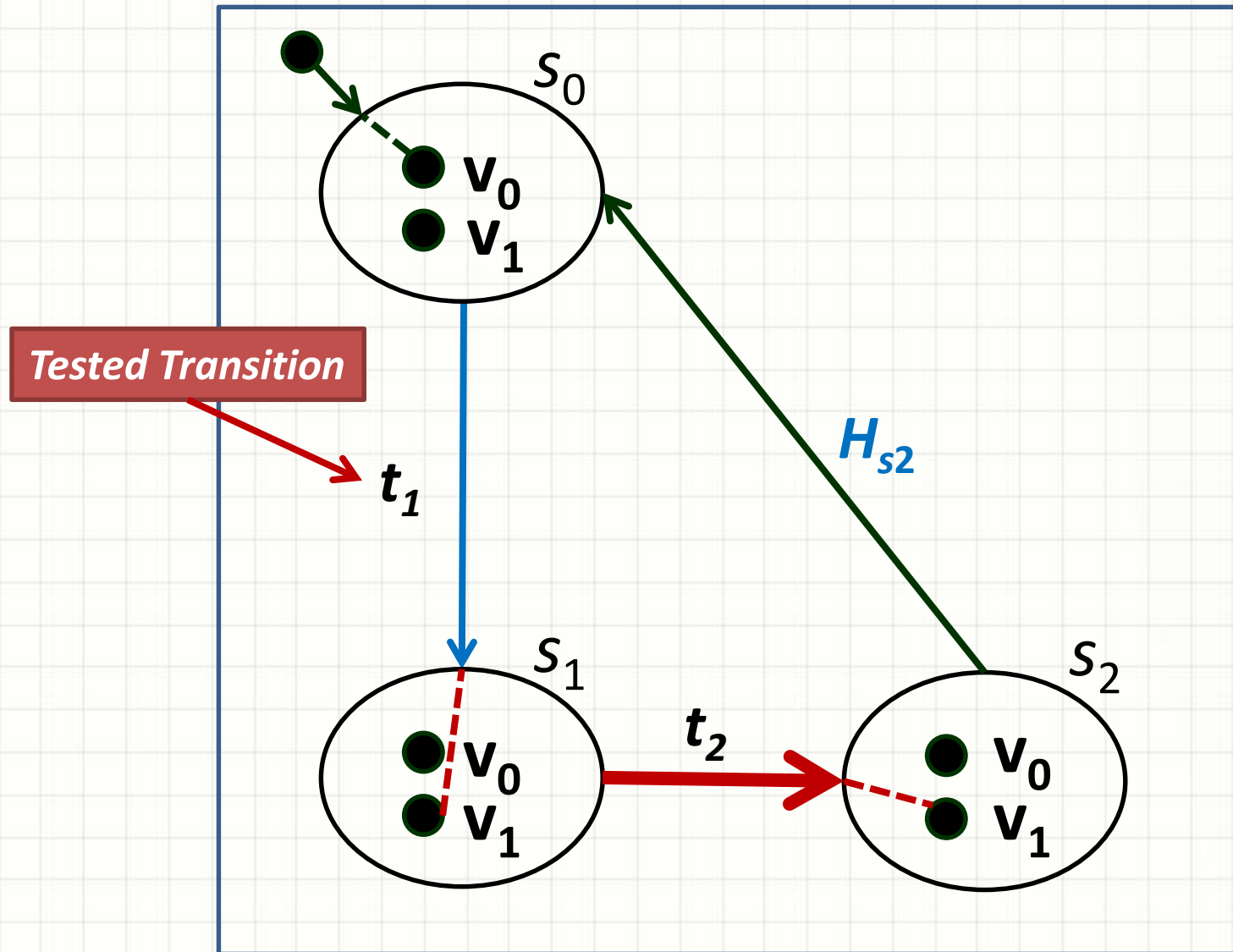


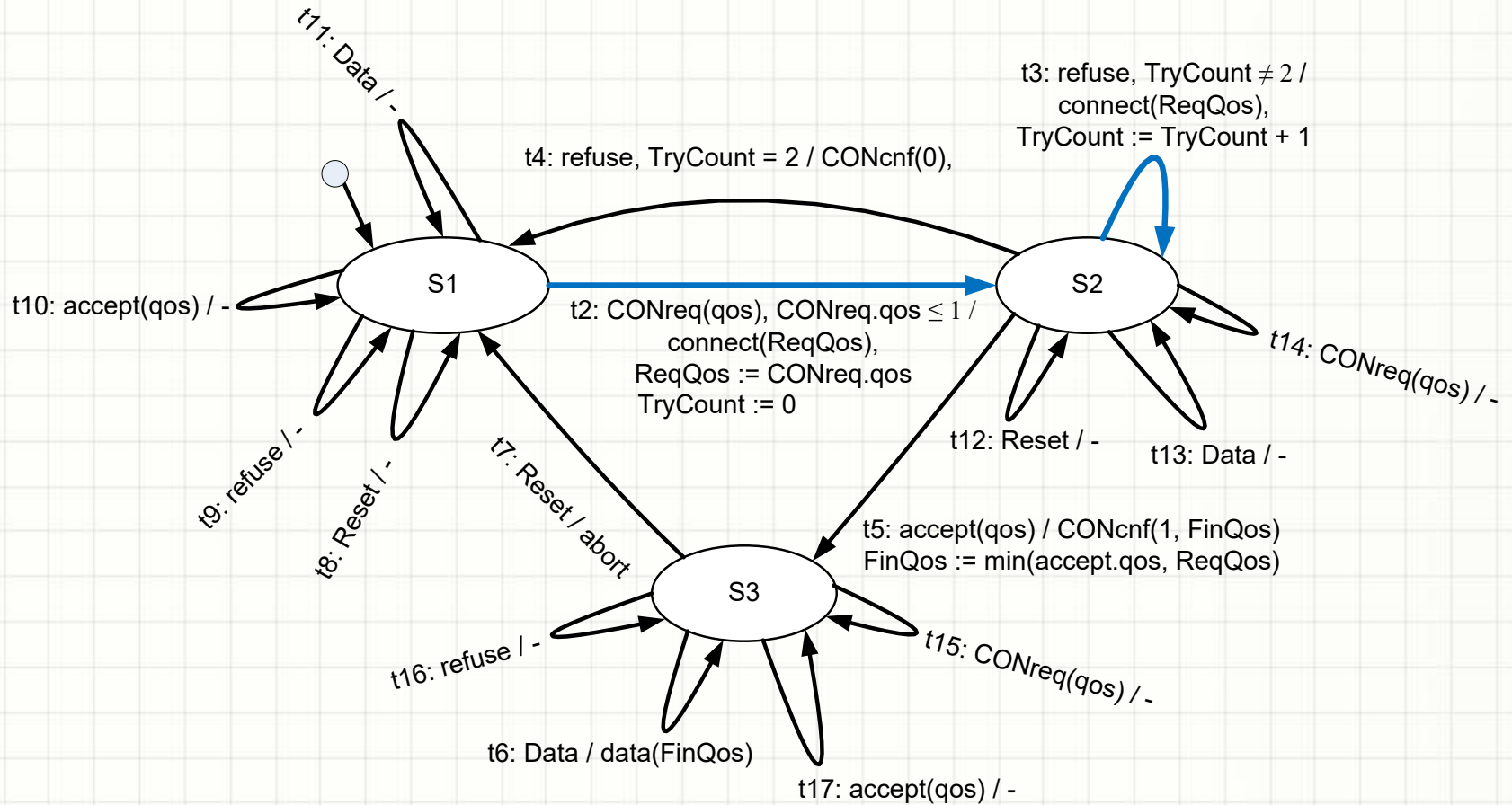
Test Case to check  $t_8$ :

?Reset / !-, ?accept(0) / !-, ?Data / !-

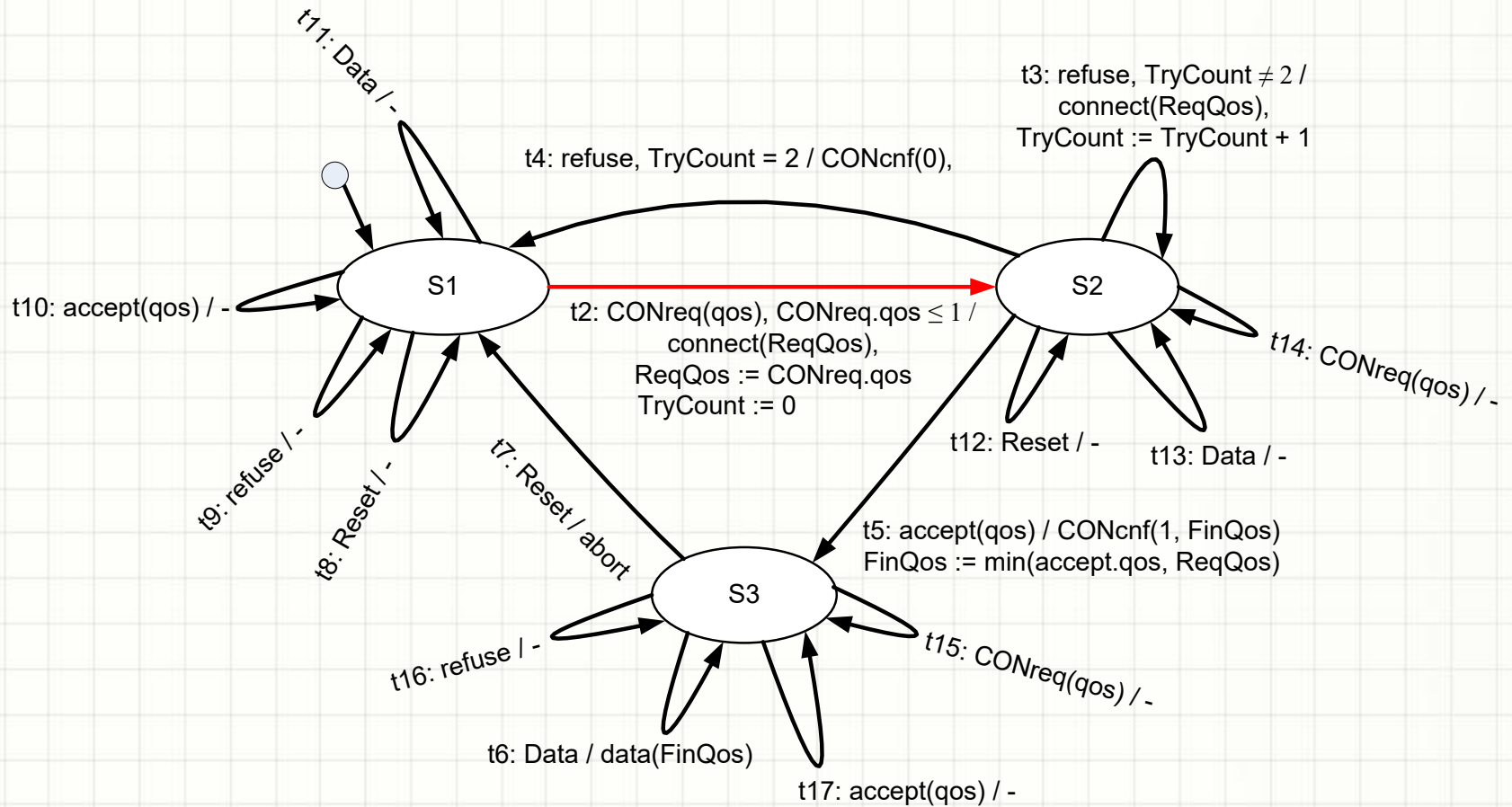
$H_{s1}$

# Step-3: Check outgoing transitions of other states reached using tested transitions



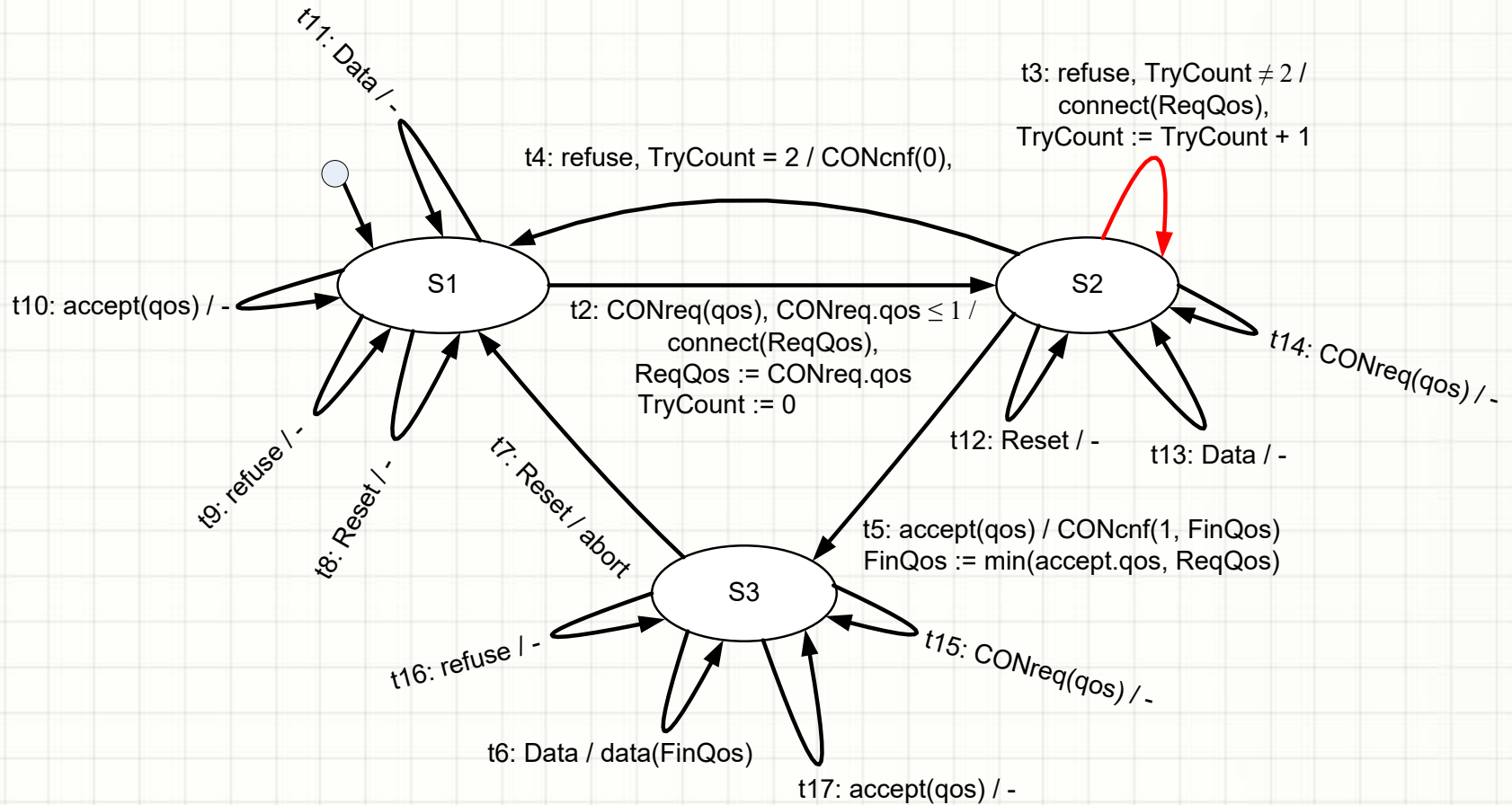


In order to test  $t_4$ ,  
 $t_2$  and  $t_3$  should be tested first.



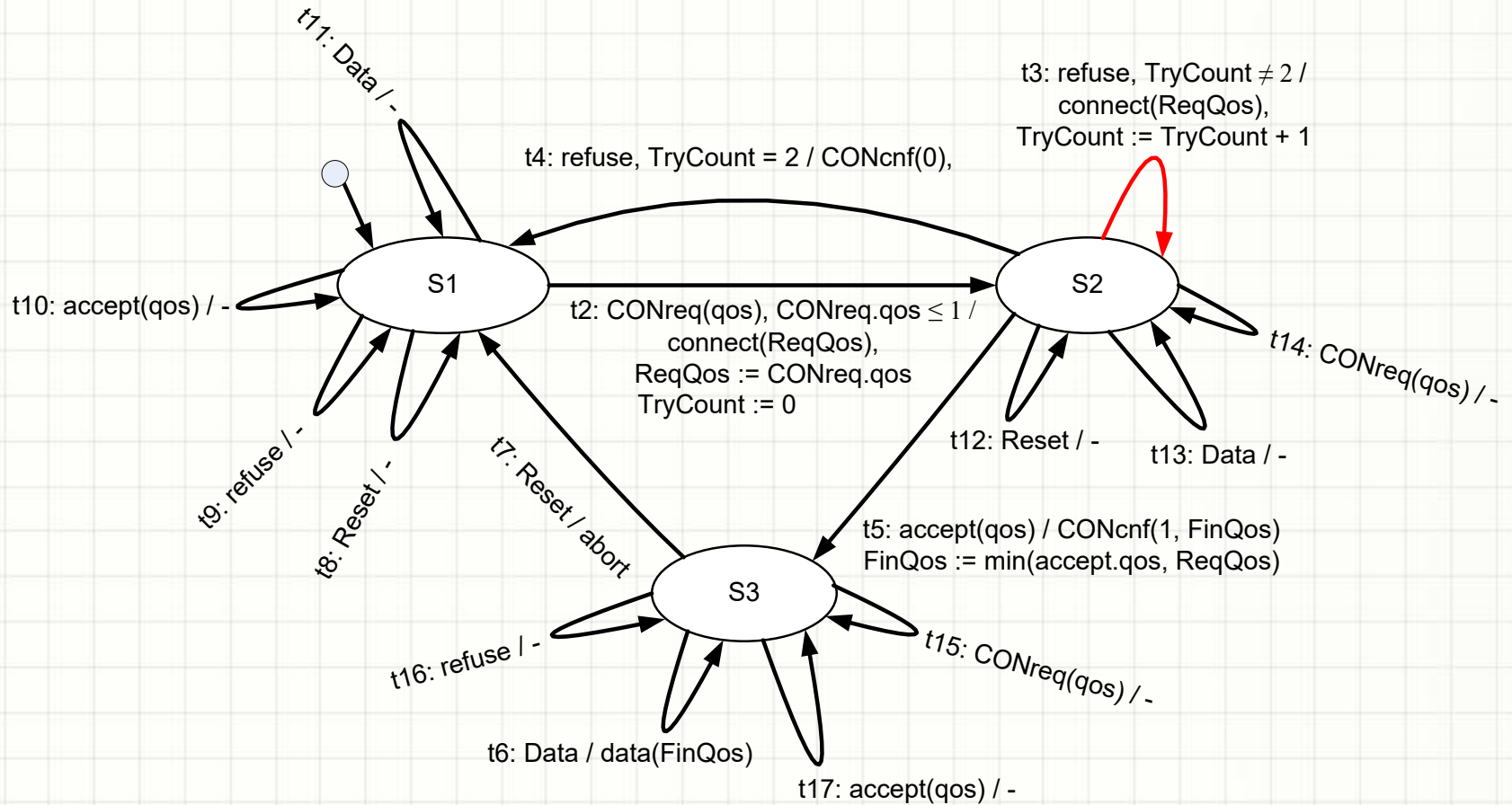
Test Case to check  $t_4$ :

r. ?CONreq(0)/!connect(0),



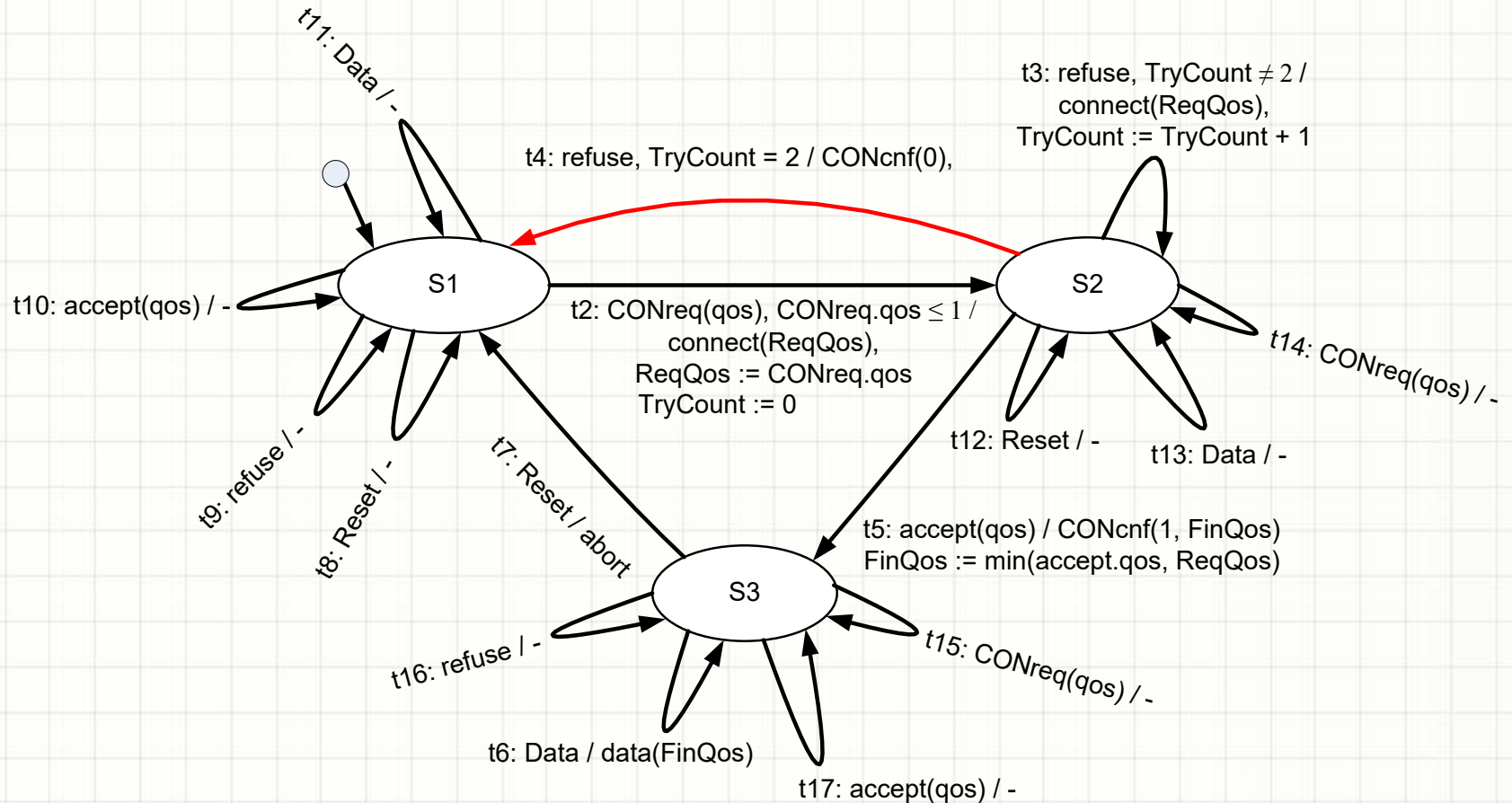
Test Case to check  $t_4$ :

r. ?CONreq(0)/!connect(0), ?refuse/!connect(0)



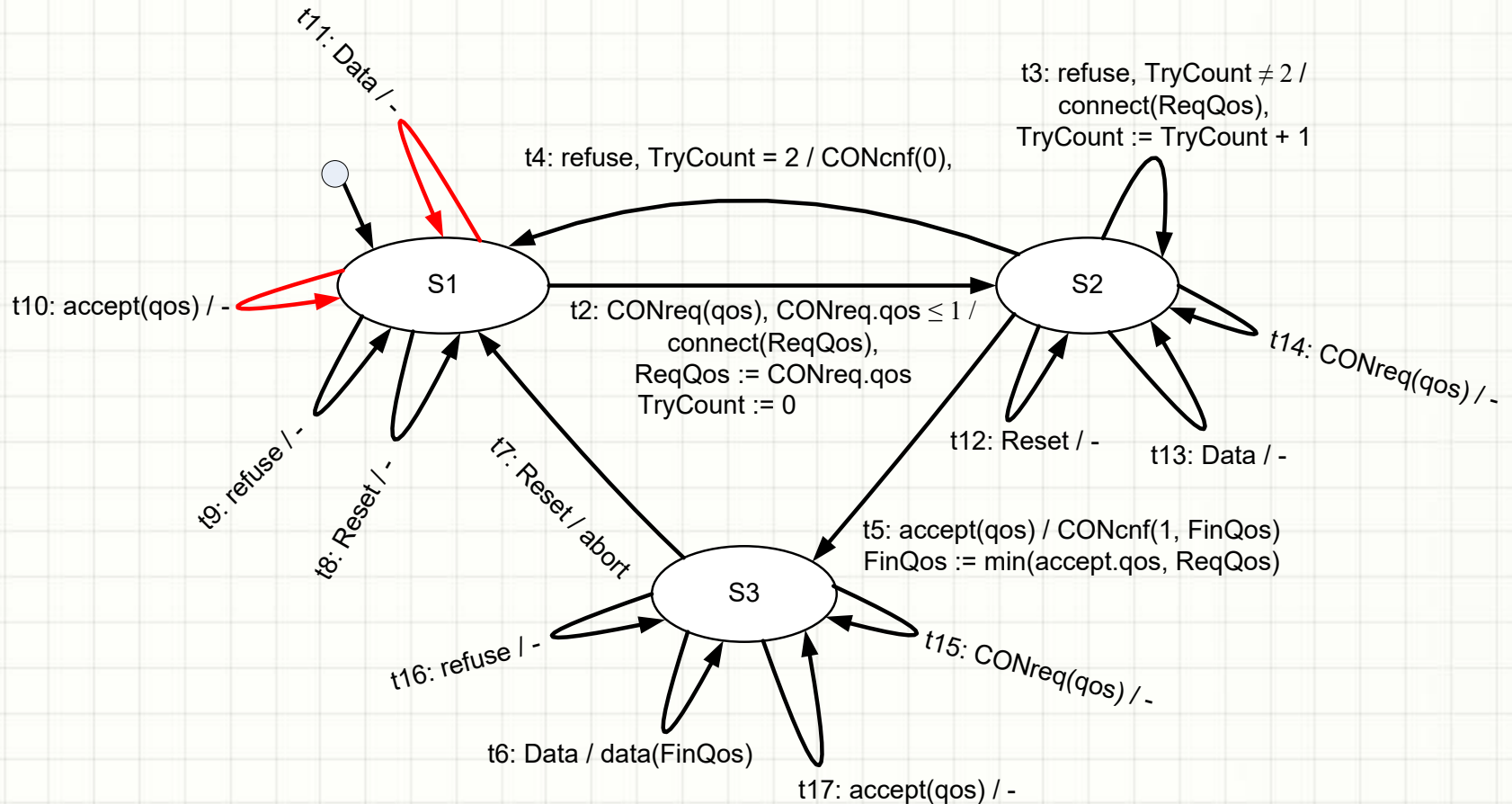
## Test Case to check $t_4$ :

r. ?CONreq(0)/!connect(0), ?refuse/!connect(0), ?refuse/!connect(0),



## Test Case to check $t_4$ :

r. ?CONreq(0)/!connect(0), ?refuse/!connect(0), ?refuse/!connect(0),  
?refuse/!CONcnf(0)

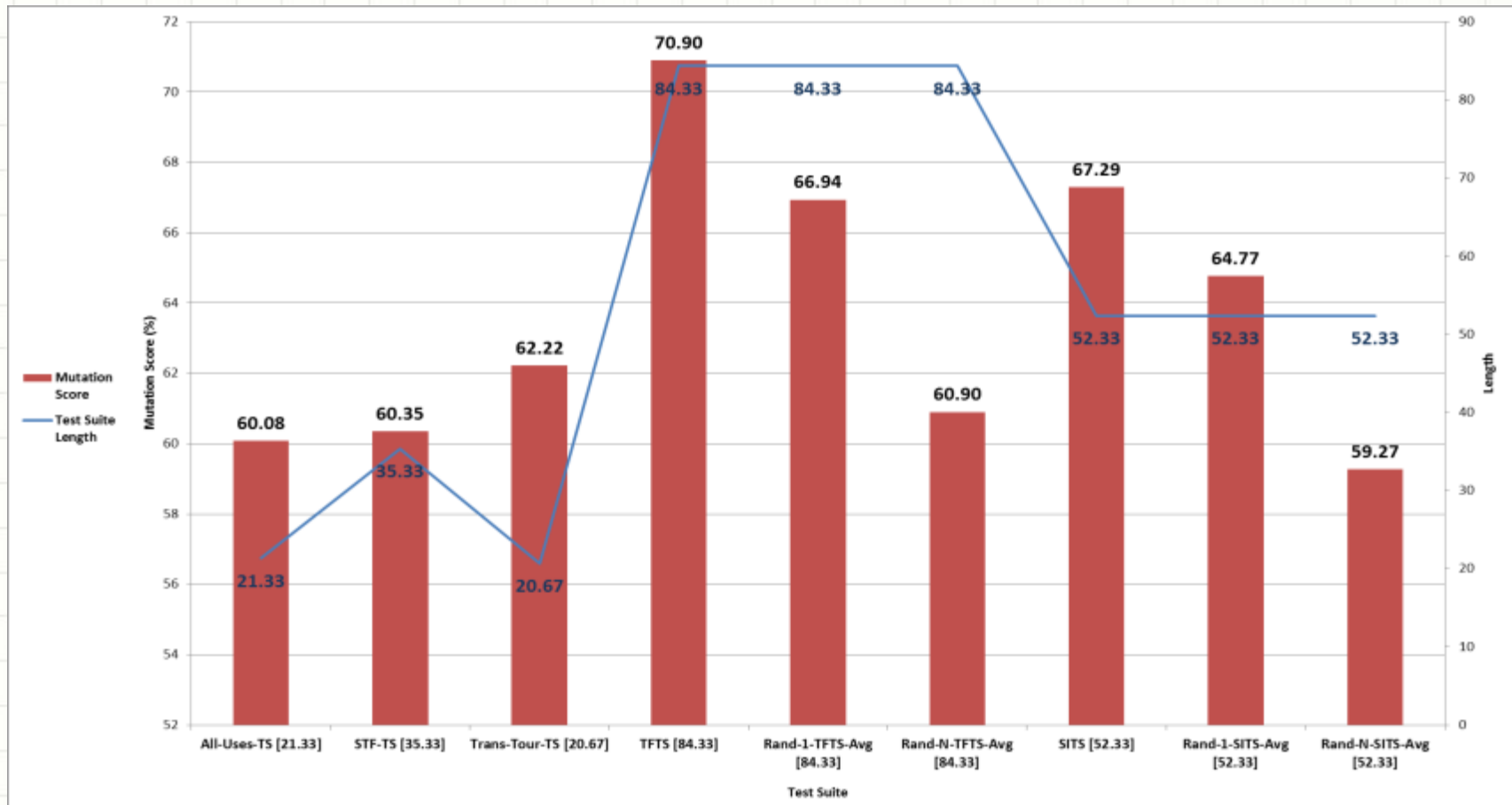


## Test Case to check $t_4$ :

r. ?CONreq(0)/!connect(0), ?refuse/!connect(0), ?refuse/!connect(0),  
 ?refuse/!CONcnf(0), ?accept(0)/!-, ?Data/!-

$H_{s1}$

# Experimental Results



# Conclusion

- ★ **Random All Uses** outperform EFSM-based, Data-Flow, and Control-Flow test suites.
- ★ The best EFSM-Based test suites are **Transition Tour** and **Single Transfer Fault** test suites.

# Conclusion

- ★ The best Data-Flow and Control-Flow test suite is **All-Uses** test suite.
- ★ Transition Tour, Single Transfer Fault, and All-Uses test suites are **comparable**.

# Conclusion

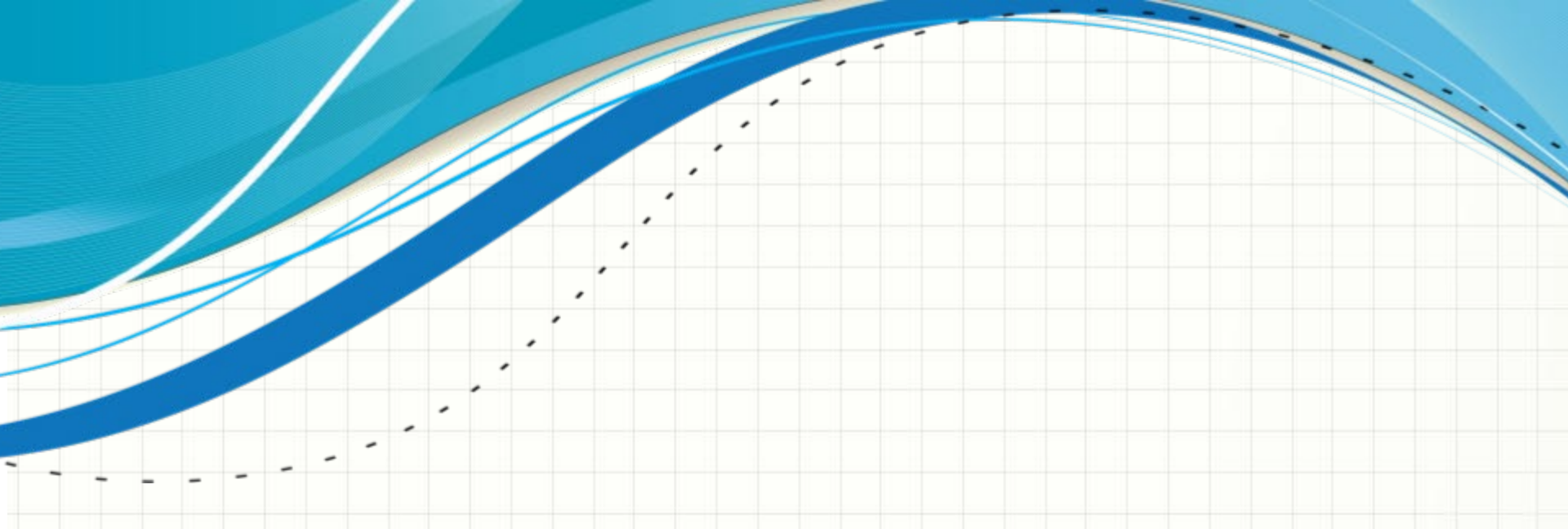
- ★ *Transfer Faults Based Test Suites (TFTSs)* outperform EFSM-based, random, and the traditional Data-Flow and Control-Flow test suites.

# Future Work

- Extending the testing with respect to transfer faults method to deal with **partial EFSMs** and consider related fault models and Conformance relations.



**QUESTIONS?**



**THANK YOU**