Software Verification and Validation

Verification of Implementations of the IOT protocol MQTT via Active Automata Learning

Baseem AL-Twajre
baseem@hit.bme.hu

BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATICS (VIK)
DOCTORAL SCHOOL OF INFORMATICS

Software Verification and Validation

Model-Based Testing IoT Communication via Active Automata Learning

Martin Tappler Bernhard K. Aichernig
Institute of Software Technology
Graz University of Technology, Austria
{martin.tappler, aichernig}@ist.tugraz.at

Roderick Bloem
Institute of Applied Information Processing and Communications
Graz University of Technology, Austria
roderick.bloem@iaik.tugraz.at

BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATICS (VIK)
DOCTORAL SCHOOL OF INFORMATICS

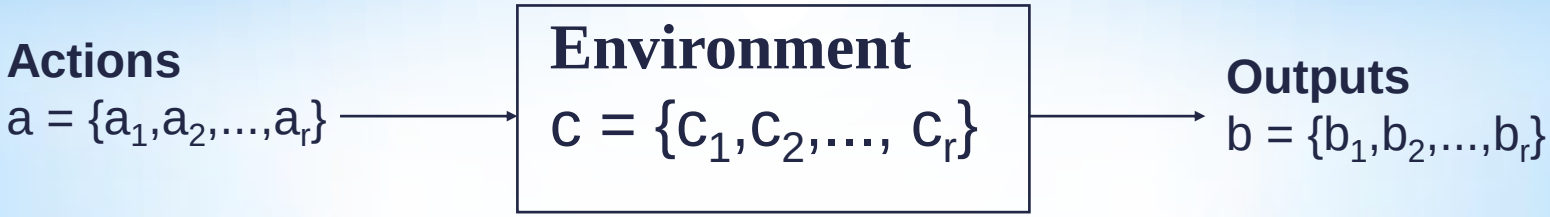
Overview

- Introduction.
- The Approach.
- The Architecture of learning setup.
- The Case Study :
  - Learning.
  - Conformance Checking.
  - Bug Hunt.
- The main contribution and Limitation.

Introduction

Learning Automata

An Automaton which selects on of it's actions related to its past experiences and rewards or punishments from the Environment.



Learning Automata

Applications:

- Network Routing.
- Priority assignment in Queuing System.
- Task Scheduling.
- Multiple-Access Networks.
- Image Compression.
- Pattern Classification.

Mealy state machine

A Mealy machine is a 6-tuple (Q , q_0 , I ,O ,δ , λ) where :

- Q is a finite set of states.
- q_0 is the initial state.
- I/O is a finite set of input/outputs symbols.
- $\delta : Q \times I \rightarrow Q$ is the state transition function.
- $\lambda : Q \times I \rightarrow O$ is the output function.

As we need a basis for determining whether two Mealy machines are equivalent. Equivalence is usually defined with respect to outputs, i.e. two deterministic Mealy machines are equivalent if they produce the same outputs for all input sequences.

We say that a Mealy machine Q 2

$$\langle Q_2, q_{0_2}, I, O, \delta_2, \lambda_2 \rangle$$

is equivalent to another Mealy machine Q 1

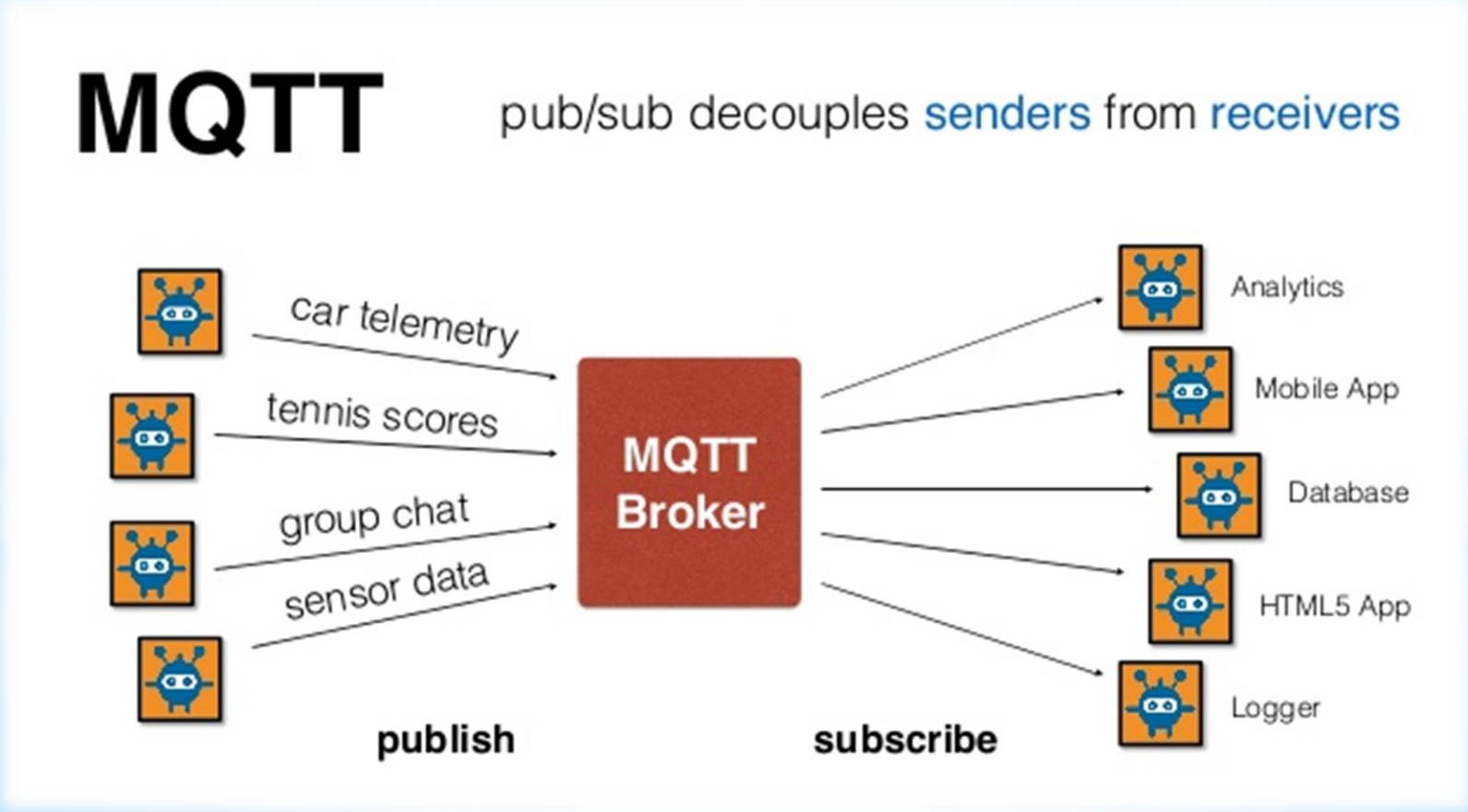
$$\langle Q_1, q_{0_1}, I, O, \delta_1, \lambda_1 \rangle$$

Iff :

$$\forall s \in I^* : \lambda_1(q_{0_1}, s) = \lambda_2(q_{0_2}, s)$$

MQTT Protocol

The MQTT protocol is a lightweight publish/subscribe protocol and therefore well-suited for resource-constrained environments such as the Internet of Things (IoT).



The Approach

The approach is a learning-based approach to detecting failures in reactive systems.

The technique is based on inferring models of multiple implementations of a common specification which are pair-wise cross-checked for equivalence.

Any counterexample to equivalence is flagged as suspicious and has to be analyzed manually.

Hence, it is possible to find failures in a semi-automatic way without prior modeling.

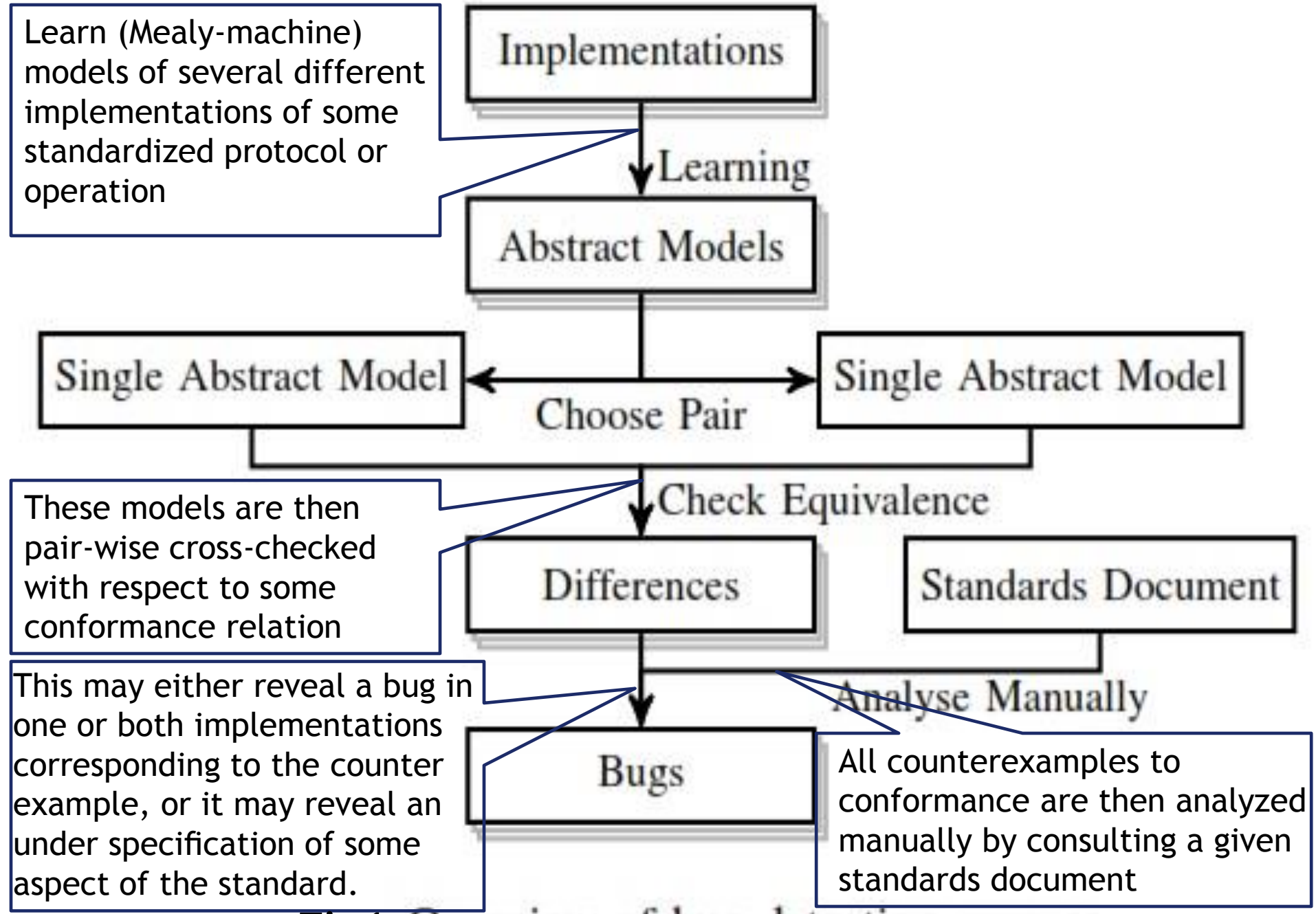


Fig.1. Overview of bug-detection process.

This approach apparently cannot detect all possible faults because :

- Specific faults may be implemented by all examined implementations.
- The fault-detection capabilities are limited by the level of abstraction used for learning.

This gives rise to two research questions they aim at answering :

- Is it possible to effectively detect nontrivial faults using this approach despite the necessary severe abstraction?
- What are the limitations of the approach?

Active Automata Learning

consider learning algorithms operating in the minimally adequate teacher (MAT) framework proposed by Angluin. These algorithms infer models of black-box systems, also referred to as systems under learning (SULs).

- Minimally Adequate Teacher Framework (MAT) :

The interaction is carried out via two types of queries posed by the learning algorithm and answered by a minimally adequate teacher. These two types of queries are usually called membership queries and equivalence queries.

- Learning Mealy Machines:

They want to learn the behavior of a black-box SUL of which they only know the input and output interface. Hence, output queries are conceptually simple: inputs are provided to the SUL and it produces some outputs.

Active Automata Learning

Since we are dealing with a black-box system, we normally cannot check for equivalence with a hypothesis. In practice, it is thus necessary for a teacher used in a learning algorithm to approximate equivalence queries somehow. This can for instance be achieved via **conformance testing** as implemented in **LearnLib**.

To summarize, a learning algorithm for Mealy machines relies on three operations:

- **Reset**
  resets the SUL
- **output query**
  performs a single test executing a sequence of inputs and recording the outputs
- **equivalence query**
  performs a set of tests comparing the outputs of the SUL and the current hypothesis

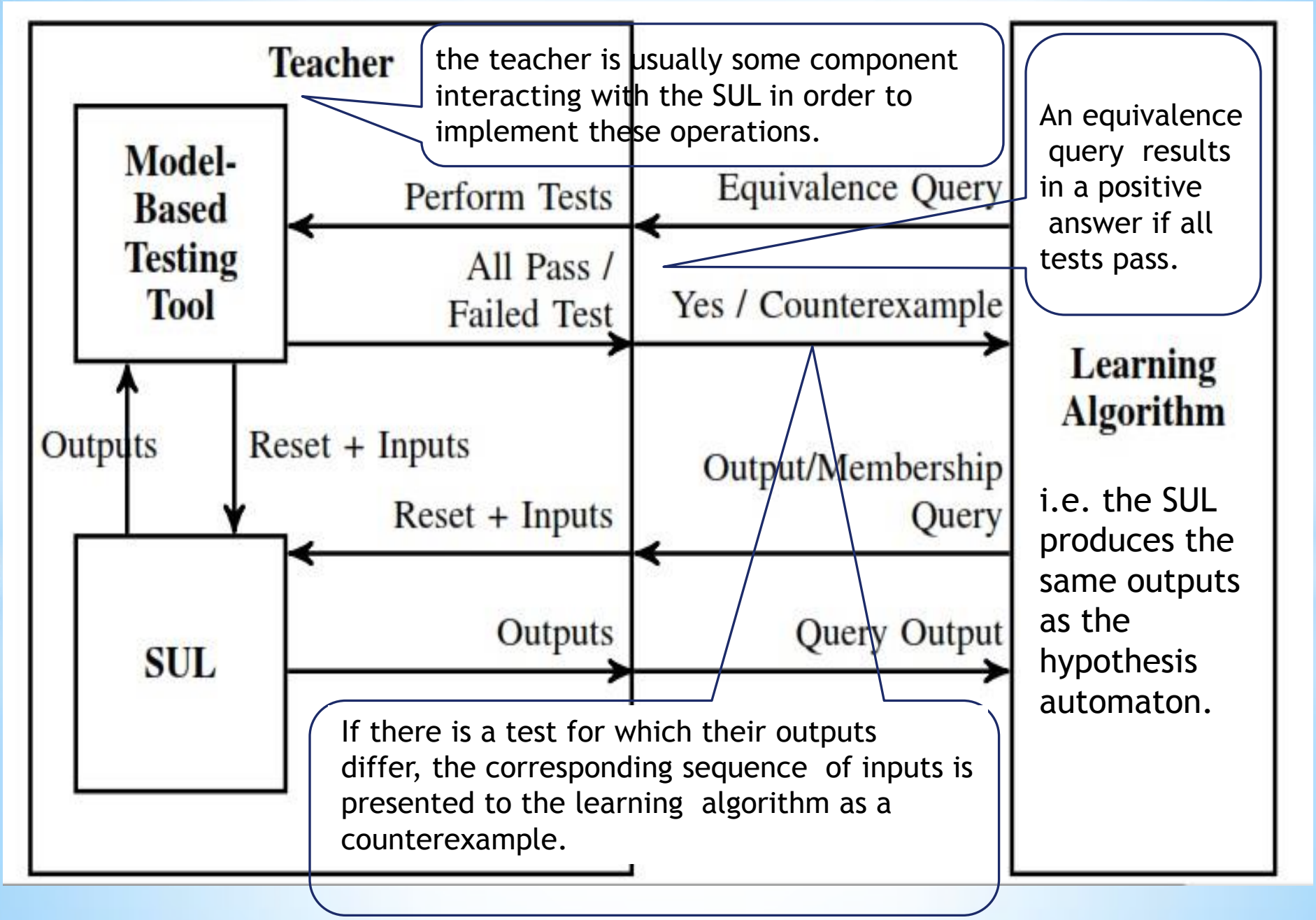


Fig. 2. The interaction between SUL, teacher and learning algorithm

The architecture of a learning environment for MQTT

Two aspects influence the architecture of a learning environment for MQTT :

- **Firstly**, the need to account for dependencies between clients. Unlike in pure client/server settings, like in the TLS protocol or the TCP , it is not sufficient to simulate one client to adequately infer a model of the server/broker in MQTT. They need to control multiple clients and record the messages each one has received.
- **Secondly**, the need to cope with the enormous amount of possible inputs, i.e. the large number of packets they can send to the brokers. This, however, is a general problem of active automata learning in the MAT framework and an issue for learning almost all non-trivial systems.

To deal with this problem, they introduce a **mapper** component performing **abstraction** and **concretization**.

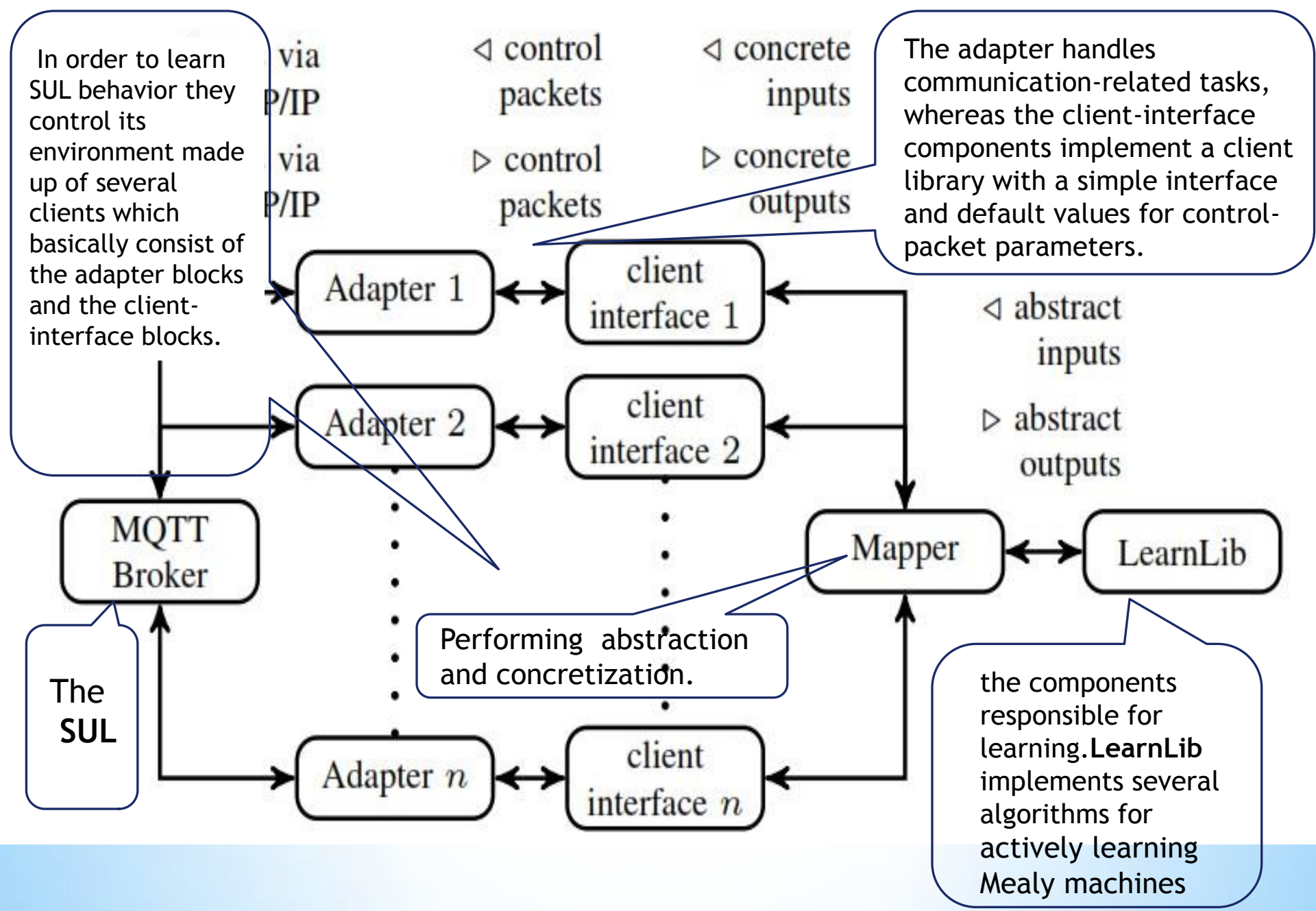


Fig. 3. The architecture used for learning of Mealy-machine models of MQTT brokers.

Model-based testing process

For each input alphabet:

1) Learn a model *mi* of each implementation *i* .
2) For each pair (*mi* , *mj*) of learned models:
   - Check equivalence.
   - For each counterexample c to equivalence.
     1) Test c on implementations *i* and *j* .
     2) Analyze manually if outputs of *i* and *j* are correct.

Note that if they find a counterexample to equivalence, i.e. a suspicious trace c, they test it on the corresponding implementations. They do so to ensure that c actually shows a difference between the implementations and is not the result from an error introduced by learning. Although active automata learning is in general sound, this may happen because they only approximate equivalence queries by conformance testing.

As they check conformance on model-level, they can also use techniques other than testing. They could, e.g., use external tools such as CADP to check equivalence , encode the problem as reachability problem and use satisfiability modulo theories (SMT) solvers for the task, or use a graph based approach. They will actually use a graph-based approach, whereby they roughly interpret Mealy machines as labeled transition systems (LTSs).

The Case Study

They carried out experiments in which they learned models of five implementations of MQTT brokers/servers. Examining these models, they found several violations of the MQTT specification.

- Learning

  The learning part was implemented in Scala using the Java-library LearnLib for active automata learning. Most of the learning-related functionality was thus already implemented and they only had to implement application-specific components such as mappers, and a component responsible for the configuration of learning experiments.

❑Conformance Checking

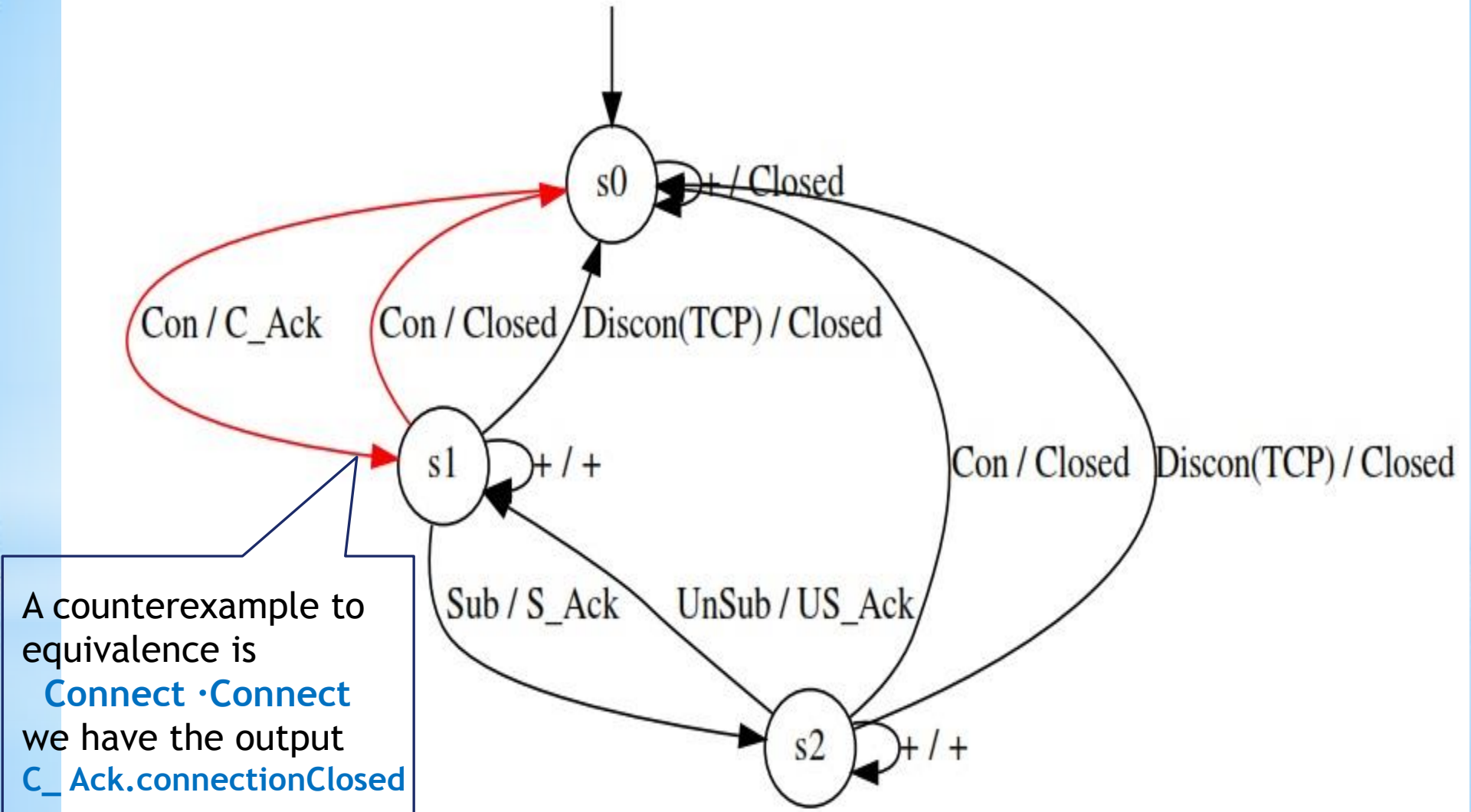
They actually check for equivalence and either output that the models are equivalent or present all found counterexamples to the user.

This is accomplished through the application of bisimulation checks.
”On the Fly”-Check: they interpreted Mealy machines as LTSs whereby they interpreted input-output pairs labeling a transition in a Mealy machine as a single transition label in the LTS-interpretation.

❑ Bug Hunt

with respect to error detection in implementations. Altogether they found 17 bugs in all implementations combined whereby they did not find any in the Mosquitto broker.

A simple example of a violation of the protocol specification can be found by considering the behavior of the HBMQTT broker with respect to the functionality covered by the Simple mapper.



A counterexample to equivalence is
Connect ·Connect
we have the output
C_ Ack.connectionClosed

Fig.4. Mosquitto model
learned by observing the Mosquitto broker with abbreviated action labels

the output
C _Ack · Empty

HBMQTT acknowledges the first connection attempt and ignores the second by not producing any output and it actually does not change its state as well.

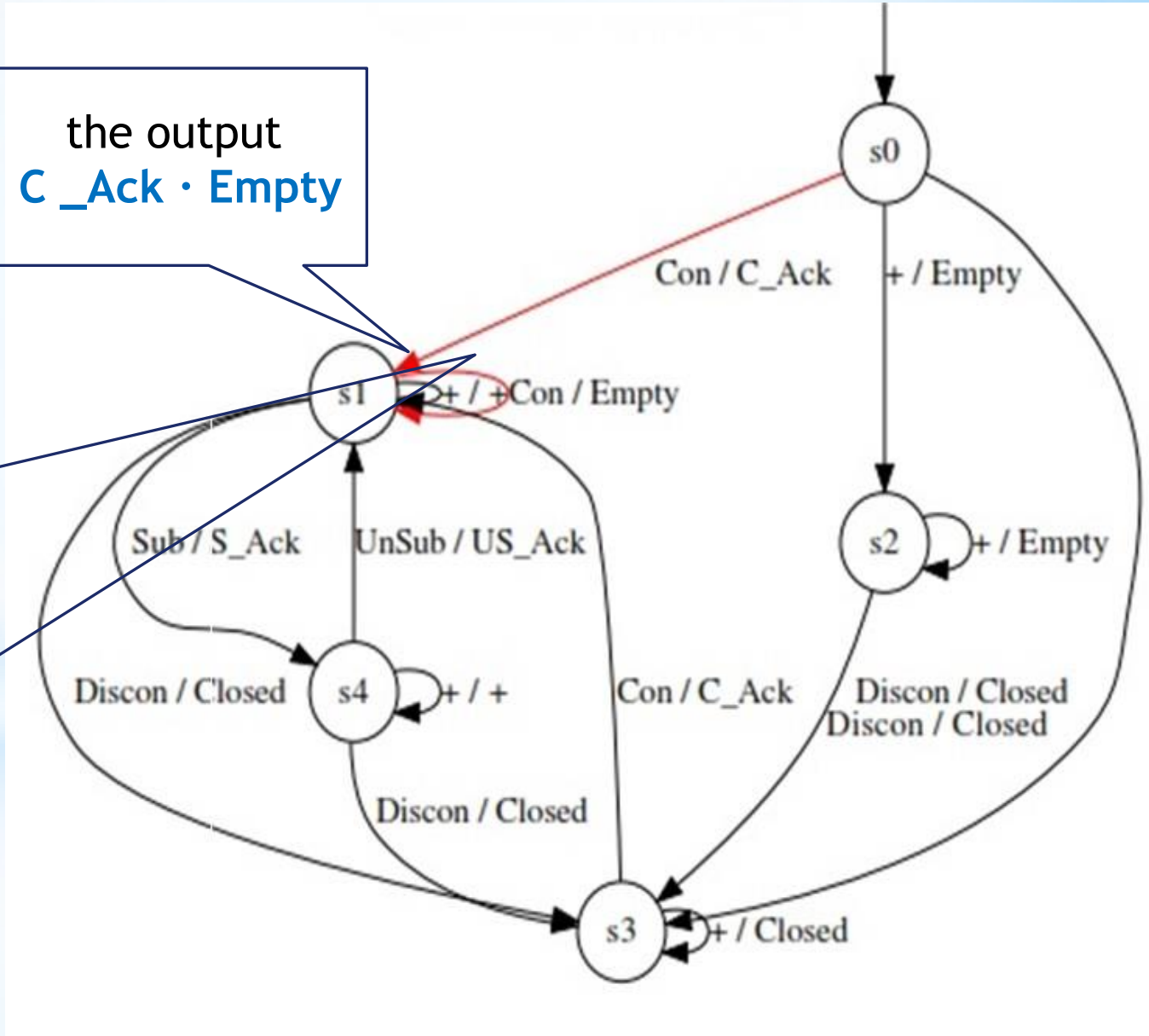


Fig.5. HBMQTT model
learned by observing the HBMQTT broker with abbreviated action labels

The MQTT specification states that Mosquitto's behaviour is correct whereas HBMQTT behaves in an incorrect way

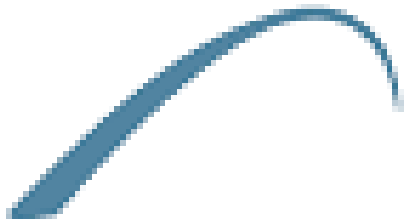
> A Client can only send the CONNECT Packet once over a Network Connection. The Server MUST process a second CONNECT Packet sent from a Client as a protocol violation and disconnect the Client [MQTT-3.1.0-2].

➢ The main contribution

The main contribution of this paper is thus the presentation and empirical evaluation of the mentioned approach based on learning experiments with five different black-box systems.
More concretely, they learned models of five different MQTT brokers, systems used in IoT communication.

➢ What are the limitations of the approach?

This approach apparently cannot detect all possible faults because specific faults may be implemented by all examined implementations. In addition to that, the fault-detection capabilities are limited by the level of abstraction used for learning.



THANK YOU
FOR
YOUR ATTENTION


ANY
QUESTIONS
?