Formal methods for embedded software systems: Two problems

Meenakshi D’Souza

IIIT-Bangalore.

24th January 2019.
Embedded Control Software

Figure: Robotics

Figure: Avionics

Figure: IoT

Figure: Simulink: Embedded Control Design
Embedded control software: Characteristics

- Runs on a proprietary real-time platform.
- Software tightly coupled with its environment.
- Distributed, real-time.
- Typically safety critical—subject to certification and regulatory requirements.
- Not feasible to shut down a malfunctioning system to restore safety or functionality.

One of the areas where formal methods is used by the industry.
Software for Control Automation
Software for automation engineering systems

- Software is used in automation systems to monitor and control various operations like batch processing, arc welding etc.
- Such software is implemented using domain-specific languages, most are proprietary in nature.
- Usually safety critical in nature, certification standards demand use of formal methods techniques.
Program analysis tools for automation software

- Standard program analysis tools (Coverity for C, PolySpace for C/C++/Ada, Klocwork etc.) are not known to work for such languages.
  - Complex data types, task-based asynchronous/parallel execution, real-time system interrupts.
  - Tools do not port well across various development environments.
  - Many tools deploy pattern based matching to detect code violations, not known to scale for industrial automation tools.
We work with three programming languages used in industry automation:

- **Rapid**, a domain-specific language for programming industrial robot arms.
- **IEC 61131-3** for PLC programming.
- **Electronic Device Description Language (EDDL)** used for configuration of field devices.
Rapid: A robotics programming language

- ABB robots are multi-axes industrial robots/robot manipulators.
- Typical actions done include welding, painting, picking, placing etc.
- **Rapid** is ABB proprietary language to program their robots.
- Robot instructions can be programmed using a teach pendant that generates Rapid code or directly using a textual interface.
Many standard datatypes are included in Rapid: `num`, `string`, `bool`, `array` etc.

In addition, complex datatypes support co-ordinates in 3-D space, target positions for the robot arm etc.

- `VAR pos := [500, 0, 940];`: Position in 3-D space.
- `VAR robtarget p15 := [[600, 500, 225.3], [1, 0, 0, 0], [1, 1, 0, 0], [11, 12.3, 9E9, 9E9, 9E9, 9E9]];`: Position of a robot.
  First tuple: position in 3-D space,
  Last three tuples specify orientation of the tool,
  axis-configuration of the robot and the position of external axes respectively.
Rapid in a nutshell: Instructions

- **WaitTime 200;**: Instructs the robot to wait for 200 seconds before doing any assigned work.
- **IDelete intr;**: Disables the interrupt variable *intr*.
- **MoveL p1,v500,z10,tool1;**: Moves the position of the robotic tool *tool1* linearly to the position *p1*, with velocity *v500* and zone data *z10*.
  - This internally calculates the torque that needs to be applied to each axis (motor) to move linearly to the position *p1*. 
Rapid in a nutshell: Program control flow

- Program flow: Written using standard imperative language constructs including relational and logical expressions, IF-THEN-ELSE statements and FOR and WHILE loops.
- Procedure calls are available, makes execution semantics complex.
- System generated interrupts and exceptions and their handling can alter the control flow of a program.
Rapid: An example

```plaintext
PROJECT TestProject

TASK T1

MODULE TestModule | ------ The main module

VAR num h; VAR var num n1; VAR intnum sig1;
VAR robtarget r1:=[[500,500,500],[1,0,0,0],[1,1,0,0],[500,9E9,9E9,9E9,9E9,9E9]];
VAR robtarget r2:=[[400,400,400],[1,0,0,0],[1,1,0,0],[500,9E9,9E9,9E9,9E9,9E9]];
VAR robtarget r3:=[[300,300,300],[1,0,0,0],[1,1,0,0],[500,9E9,9E9,9E9,9E9,9E9]];
VAR robtarget r4; VAR signalId d1; VAR toolData tool1;

PROC main() | ---- The main procedure ----
SIMpleBug;
11 := h - 10;
SettoZero;
CONNECT sig1 with SimpleError;
ISignalID d11, 1, sig1;
r4 := [[h,h,h], [1, 0, 0, 0], [1, 1, 0, 0], [500, 9E9, 9E9, 9E9, 9E9, 9E9]];
DrawShape;
ENDPROC

PROC SimpleBug()

<table>
<thead>
<tr>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHILE h &lt; 3.4E+38 - 2 DO</td>
</tr>
<tr>
<td>h := h+1;</td>
</tr>
<tr>
<td>ENDPWHILE</td>
</tr>
<tr>
<td>ENDPROC</td>
</tr>
</tbody>
</table>

PROC SettoZero()

<table>
<thead>
<tr>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 := 0;</td>
</tr>
<tr>
<td>ENDPROC</td>
</tr>
</tbody>
</table>

ENDMODULE
ENDTASK
ENDPROJECT
```
IEC 61131-3

- **IEC 61131** is an open international standard for Programmable Logic Controllers (PLC).
- IEC 61131-1 deals with architecture and programming languages of the control program within PLC.
- Several standard data types, user defined data types including a kind of strongly typed pointer, I/O variables amongst others, program organization units that structure the code in a modular way.
IEC 61131-3: An example

CreateArray (Array => Array_var,
    FirstIndex := 1,
    LastIndex := 2,
    ArrayElement := array_element,
    tatus => status);

index_val := status;
index_val := index_val +1;
putArray (Array := Array_var,
    Index := index_val,
    ArrayElement := array_element,
    Status => status);
index_val := index_val + 3;
putArray (Array := Array_var,
    Index := index_val,
    ArrayElement := array_element,
    Status => status);

while (run < 3) do
    if (run < 3) then
        Mountain_Height := 5;
    else
        Mountain_Height := 7;
    end_if;
exit;
run := run + 1;
end_while;
EDDL (Electronic Device Description Language), an IEC standard, is a language for describing the service and configuration of field devices for process and factory automation.

EDDL has data, communication (e.g. addressing information), user interfaces and operations (e.g., calibration).
EEDL: An example

MANUFACTURER HCF, DEVICE_TYPE SAMPLE1,
    DEVICE_REVISION 1, DD_REVISION 1
VARIABLE temperatureUSL {
    LABEL "Temp USL";
    HELP [upper_sensor_limit_help];
    HANDLING READ;
    TYPE FLOAT {DISPLAY_FORMAT " .3f";}
}
METHOD Sample_Method {
    LABEL "Sample Method";
    DEFINITION {
        int len1, x, y;
        char a[10], b[20], c[15];
        b=a+c;
        x=6;
        if(x<10) {
            len1=6;
            y=x/(x-len1);
        } else {
            if(y>10) {
                len1=20;
            }
        }
    }
}
Common characteristics

- All three languages, although from disparate domains, characterize domain-specific languages for automation engineering.
- All of them support
  - Task based execution and use a modular structure for code organization. Execution can change based on interrupts.
  - Code is used to monitor and control various devices and controllers. Interrupts come from the platform.
  - Variables can be primitive as well as structured data types.
- Other languages like PLCopen, KRL, etc. support similar programming structures, and can be analyzed as well.
Program analysis framework: Key contributions

- Generic datatype to represent the parsed information for the three languages.
- Flexible Data Flow Analysis (DFA) engine to encode data flow rules as needed by varying the domain.
- Flexible rule engine to process data for further analysis.
Rapid: Program analysis framework

Conversion Stage

Extraction

Project → Extract

Organization Tree

Units of code → Organization units → Variables → Scope table → Configuration

Processing

Parse → Generate CFG → Generate inter-procedural CFG → Inter-procedural CFG

Analysis Engine

Generic Analysis Engine

Pattern matching engine

Data Flow Analysis Engine

Rule Config → Rules → Analyze → Warnings
Generic analysis engine

- Abstract-Syntax Tree (AST) and Control Flow Graph (CFG) are generated the usual way.
- Inter-procedural CFG needs one or more executions as input if procedure calls are not clear from the code.
  - CFG and inter-procedural CFG are annotated with data flow results.
- Data Flow Analysis (DFA) uses abstract interpretation based on interval domain abstraction to define transfer functions that support all standard arithmetic, logical and relational operations.
  - Interval domain semantics defined for all special datatypes.
- Several standard syntactic errors can be detected using AST and CFG.
Rapid code: Annotated inter-procedural CFG
Rules: Classification

- **Generic Programming rules:** Depend solely on the analysis framework, generic across all languages.
  - Division by zero, array out of bounds.

- **Language specific rules:** Based on a specific language, may not exist for all languages.
  - Boundary violation check for a robot arm (Rapid), variable re-definition rule (EDDL).

- **User specified rules.** Rules defined by the user based on a specific project or application.
  - Nesting levels of code, based on quality requirements.
  - Conformance to NORSOK standards for oil and gas applications.
### Distribution of rules

<table>
<thead>
<tr>
<th>Rules</th>
<th>Language</th>
<th>Generic errors</th>
<th>Language based</th>
<th>User specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern-based</td>
<td>IEC 61131-3</td>
<td>11</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>EDDL</td>
<td>4</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>RAPID</td>
<td>11</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>34</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>Semantic</td>
<td>IEC 61131-3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>EDDL</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>RAPID</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>DFA</td>
<td>IEC 61131-3</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>EDDL</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>RAPID</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>10</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>
## Errors and rules: IEC 61131-3

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Category</th>
<th>Error</th>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incorrect Attribute</td>
<td>language specific</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Uninitialized variable</td>
<td>generic</td>
<td>0</td>
<td>76</td>
</tr>
<tr>
<td>Datatype mismatch</td>
<td>language specific</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>Divide-by-zero</td>
<td>generic</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Duplicate identifier</td>
<td>generic</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## Errors and rules: EDDL

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Category</th>
<th>Error</th>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide-by-zero</td>
<td>generic</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Missing Mandatory Menus</td>
<td>language specific</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>Unused variables</td>
<td>generic</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>Assignment for comparison</td>
<td>generic</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
## Errors and rules: Rapid

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Category</th>
<th>Error</th>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illegal wait statement</td>
<td>language specific</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Function side effects</td>
<td>language specific</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Routines not used</td>
<td>generic</td>
<td>12</td>
<td>52</td>
</tr>
<tr>
<td>Unused variable</td>
<td>project specific</td>
<td>41</td>
<td>15</td>
</tr>
<tr>
<td>Arithmetic overflow</td>
<td>generic</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Constant</td>
<td>project specific</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>Unreachable code</td>
<td>generic</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
Program analysis framework: Drawbacks

- Framework generates false positives.
  - Interval domain based abstraction is known to have this problem.
  - We are working on other abstraction techniques.
- Tool cannot handle recursion efficiently, we assume that worst case is reached when the function call stack exceeds a set limit.
- Works really well for Rapid, has handled code with $\geq 11000$ statements and high cyclomatic complexity. Yet to be tested on large EDDL code.
Joint work with Avijit Mandal and Raoul Jetley (ABB).

28th IEEE ISSRE 2017, industry track.

Modelling and verification of IoT protocols

1 Joint work with Maithily Diwan, Michael Butler
Introduction: Internet of Things

- Connects different computing devices, sensors, actuators, people and virtually any object.
- Prevalent in various industries like health care, automotive, manufacturing, power grid, domotics, etc.
- Gartner has predicted that there will be over 20 billion devices by 2020.
- Communication between these devices is an important aspect of IoT.
Introduction: IoT Communication Protocols

- Various protocols are used for communication in an IoT system. TCP/IP is a popular protocol used in lower layers.
- Protocols used in IoT systems possess properties like bandwidth efficient, light-weight and small code foot-print.
- Common IoT Protocol features: publish-subscribe, messaging layer, QoS (Quality of Service) levels, resource discovery, re-transmission, etc.
- Some protocols adapted for use in application layer in an IoT system - MQTT, MQTT-SN, CoAP, XMPP, AMQP.
Comparison of IoT communication protocols

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Protocol Feature</th>
<th>MQTT</th>
<th>MQTT-SN</th>
<th>CoAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Architecture</td>
<td>Asynchronous Message exchange</td>
<td>Asynchronous Message exchange</td>
<td>REST architecture Layered Approach</td>
</tr>
<tr>
<td>2</td>
<td>Transport Layer</td>
<td>TCP</td>
<td>Any</td>
<td>UDP</td>
</tr>
<tr>
<td>3</td>
<td>Communication type</td>
<td>UniCast</td>
<td>UniCast/Multicast</td>
<td>UniCast/Multicast</td>
</tr>
<tr>
<td>4</td>
<td>Addressing</td>
<td>ClientID, Server address</td>
<td>ClientID, Server address</td>
<td>URI Based</td>
</tr>
<tr>
<td>5</td>
<td>Messaging pattern</td>
<td>Publish Subscribe</td>
<td>Publish Subscribe</td>
<td>Request-Response Publish-Subscribe</td>
</tr>
<tr>
<td>6</td>
<td>QoS Levels</td>
<td>AtmostOnce, AtleastOnce, ExactOnce</td>
<td>AtmostOnce, AtleastOnce, ExactOnce</td>
<td>AtmostOnce, AtleastOnce</td>
</tr>
<tr>
<td>7</td>
<td>Persistent Session</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>Retained Message /Offline/Caching</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>Proxying/Caching</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>Resource Discovery</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>Sleep Mode</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>Security</td>
<td>Optional TLS</td>
<td>Optional TLS</td>
<td>Optional DTLS</td>
</tr>
</tbody>
</table>
Introduction to Event-B

- Uses set theory as a modeling notation and first order predicate calculus for writing axioms and invariants.
- Step by step refinement to represent systems at different abstraction levels and provides proofs to verify consistency of refinements.
- Has two types of components: contexts and machines.
Introduction to Event-B

Machine

- A machine has several events and can also define variables and its types.
- Can refine another machine to introduce new events, refine events, split or merge events.

Event

- An event consists of guards which need to be satisfied before the actions in events are executed.
- When an event is enabled and executed, the variables are updated as per the actions in the event.
- An invariant is a condition on the state variables that must hold permanently.
Rodin and ProB

**Rodin**
- Implements Event-B and is based on Eclipse platform.
- Has sophisticated automatic provers like PP, ML and SMT.
- Provides interactive proving mechanism for manual proofs.
- Offers various plug-ins: text editors, decomposition/modularization tools, simulator ProB, etc.

**ProB**
- Provides a simulation environment through animation.
- Run executes a sequence of events.
- System state gives values of variables, evaluates invariants, axioms and guards for all the events.
- Deadlocks, invariant violation and errors can be detected.
Mapping between communication protocol and Event-B model

**Communication Protocol**
- Channels, Servers, Clients, Messages
- Message Attributes and properties
- Transmit/Receive Messages, Error Event Attacker, Time Tracking
- Conditions for message transmission/consumption, Error Detection, Timer
- Buffers, Channel Properties, Timers
- Protocol Properties for IoT

**Event-B Model**
- **Context**
  - Sets in the Context
  - Constants and Axioms in the Contexts
- **Machine**
  - Events in the Machine
  - Guards of the Event
  - Variables in the Machine
  - Invariants in the Machine
Protocol Modeling and Decomposition Using Event-B

The protocol modeling is done in two major steps:

- Building a common abstract model encompassing the common features of various protocols.
- Refining this common abstract model into a concrete model of a particular IoT protocol.

Our modeling is done using the techniques of machine decomposition, refinement and atomicity decomposition in Event-B.
Common Abstract Model

Context

- Set MSG represents a message which is a basic communication entity.
- Attributes of a message are defined as relations over the set message and the sets defined for the attributes.
- A projection function is used to extract the value of an attribute for a given message.
Common Abstract Model

Atomicity Decomposition

Communication Channel

1st Refinement

MultiCast/Broadcast
UniCast

2nd Refinement

Resource Discovery
Service Discovery
Channel Establishment
Channel Conversation

3rd Refinement

NonConfirmable MessageSend
Confirmable MessageSend
Timer
Intruder

4th Refinement

Message Send
Message Acknowledge
Timer Increment
Timeout Detection
Common Abstract Model

Machine Decomposition

[Diagram showing the process flow with nodes such as Resource Discovery, Service Discovery, Connect, Disconnect, Channel Establishment, Non-Confirmable Msg Send, Confirmable Msg Send, Message Exchange, Timer Increment, Timeout Detection, Timer, Intruder, Intruder/Attacker, etc., with arrows indicating the flow and relationships between the components.]
Concrete Protocol Models

Atomicity Decomposition

1st Refinement
- Publish QoS1
- Publish QoS2
- Publish QoS2 Rcv
- Publish QoS2 Orig
- Publish QoS2 Dup
- Publish QoS2 Rcv Client
- Publish QoS2 Rcv Server

2nd Refinement
- Publish QoS2 Orig
- Publish QoS2 Rcv
- Publish QoS2 Dup
- Publish QoS2 Rcv Client
- Publish QoS2 Rcv Server
- Publish QoS2
- Publish Release QoS2
- Publish QoS2 Rcv
- Publish QoS2 Dup
- Subscribe
- Subscribe Send
- Subscribe Receive

3rd Refinement
- Publish QoS2 Orig
- Publish QoS2 Rcv
- Publish QoS2 Dup
- Publish QoS2 Rcv Client
- Publish QoS2 Rcv Server
- Publish QoS2
- Publish Release QoS2
- Publish QoS2 Rcv
- Publish QoS2 Dup
- Subscribe
- Subscribe Send
- Subscribe Receive
Concrete Protocol Models

Event for publishing message with QoS0

Event Client_Publish_QoS0 (ordinary) $\Leftarrow$

any

M10
ch
Topic

where

grd1: $\text{Msg\_Topic}(M10) = \text{Topic}$
grd2: $\text{IncrementTime} = \text{FALSE}$
grd3: $M10 \in \text{MSG} \land \text{Topic} \in \text{TOPIC}$
grd4: $\text{PayloadCounter} \in 0..9$
grd5: $(M10 \mapsto ((\text{Publish} \mapsto \text{AtmostOnce}) \mapsto \text{PayloadCounter})) \in \text{Msg\_Type\_QoS}$

then

act1: $\text{Client\_timer}(ch) := 1$
act2: $\text{IncrementTime} := \text{TRUE}$
act3: $\text{PayloadCounter} := \text{PayloadCounter} + 1$
act4: $\text{LimitTimer} := 1$
act5: $\text{Channel\_Direction}(ch) := \text{Client\_To\_Server}$
act6: $\text{MsgsInChannel}(ch) := \text{MsgsInChannel}(ch) \cup \{M10 \mapsto \text{PayloadCounter}\}$

end
Model Validation

- ProB is used for validating our model through simulation of events and checking LTL properties.
- Accuracy of the model can be obtained by executing different runs and observing the sequence of events and variable values in each of these events.
- ProB also reports invariant violation or error in events which is then corrected in the model.
- Model validation is also done by writing and verifying invariants.
Properties are verified by writing them as invariants that have to be satisfied for all the events in model.

**Message Ordering**

If both client and server make sure that no more than one message is "in-flight" at any one time, then no QoS1 message will be received after any later one.

\[
\forall ch \cdot \forall pc1 \cdot \forall pc2 \cdot ((pc1 \in 0 \cdot 9 \land pc2 \in 0 \cdot 9 \land ch \in establishChannel) \\
\land (pc1 \in Client_MsgSentQoS2(ch) \lor pc1 \in Client_MsgSentQoS1(ch)) \\
\land (pc2 \in Client_MsgSentQoS2(ch) \lor pc2 \in Client_MsgSentQoS1(ch)) \\
\land (time > SendTRange(pc2) + Response_Timeout) \\
\land pc1 \neq pc2 \land (SendTRange(pc1) < SendTRange(pc2)) \\
\Rightarrow (RcvTRange(pc1) \leq RcvTRange(pc2))
\]
Persistent Session

When a client reconnects with “CleanSession” set to 0, both the client and server must re-send any unacknowledged publish packets (where QoS > 0) and publish release packets using their original packet Identifiers. The variable RcvTRange is updated with current time only after the message is received. Hence it should be greater than the SendTRange time.

\[ \forall ch \cdot \forall pc \cdot ((pc \in 0 \cdot 9 \land ch \in establishChannel) \land Channel\_CleanSess(ch) = FALSE \land ((pc \in Client\_MsgSentQoS1(ch)) \lor (pc \in Client\_MsgSentQoS2(ch))) \land (time > (SendTRange(pc) + Response\_Timeout)) \Rightarrow (RcvTRange(pc) > SendTRange(pc))) \]
Verification of IoT Properties Using Event-B

**QoS of a message from Client1 to Client2**

The effective QoS of any message received by the subscriber is minimum of QoS with which the publishing client transmits this message and the QoS set by the subscriber while subscribing for the given topic.

\[
\forall ch \cdot \forall pc \cdot \forall chnl \cdot \forall msg \cdot ((pc \in 0 \cdot 9 \land ch \in \text{establishChannel}) \land \text{msg} \in \text{MSG} \land chnl \in \text{establishChannel} \land (pc \in \text{Client\_MsgSentQoS1}(ch)) \land (\text{msg} \mapsto ((\text{PUBLISH} \mapsto \text{AtleastOnce}) \mapsto pc)) \in \text{Msg\_Type\_QoS} \land ((\text{Msg\_Topic(msg)} \mapsto \text{ExactOnce}) \in \text{Channel\_TopicQoS}(chnl))) \land ((\text{time} - \text{SendTRange}(pc)) \geq \text{Response\_Timeout})) \Rightarrow (\exists QC \cdot ((QC \geq 1) \land \text{Client\_MsgReceived\_2}(chnl) = QC)))
\]
## Proof Obligations Results

<table>
<thead>
<tr>
<th>Sl.No.</th>
<th>Protocol Property</th>
<th>Proof Obligations</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Duplicate Channel</td>
<td>10</td>
<td>Passed</td>
</tr>
<tr>
<td>2</td>
<td>Message Ordering</td>
<td>34</td>
<td>Passed</td>
</tr>
<tr>
<td>3</td>
<td>Persistent Session</td>
<td>34</td>
<td>Passed</td>
</tr>
<tr>
<td>4</td>
<td>QoS1 in single channel</td>
<td>26</td>
<td>Passed</td>
</tr>
<tr>
<td>5</td>
<td>QoS2 in single channel</td>
<td>26</td>
<td>Passed</td>
</tr>
<tr>
<td>6</td>
<td>Retained QoS1 message</td>
<td>24</td>
<td>Passed</td>
</tr>
<tr>
<td>7</td>
<td>Retained QoS2 message</td>
<td>24</td>
<td>Passed</td>
</tr>
<tr>
<td>8</td>
<td>Effective QoS0 in Multi channel (3 cases)</td>
<td>66</td>
<td>Passed</td>
</tr>
<tr>
<td>9</td>
<td>Effective QoS1 in Multi channel (3 cases)</td>
<td>66</td>
<td>Passed</td>
</tr>
<tr>
<td>10</td>
<td>Effective QoS2 in Multi channel (3 cases)</td>
<td>72</td>
<td>Passed</td>
</tr>
<tr>
<td>11</td>
<td>Request-Response Matching and Timeout</td>
<td>39</td>
<td>Passed</td>
</tr>
<tr>
<td>12</td>
<td>Confirmable Message ID Matching and Timeout</td>
<td>39</td>
<td>Passed</td>
</tr>
<tr>
<td>13</td>
<td>Exponential Backoff</td>
<td>39</td>
<td>Passed</td>
</tr>
</tbody>
</table>
Related Work

- In his paper published in 2014, the author Aziz. B., shows that there are scenarios where MQTT has failed to adhere to the QoS requirement.

- Gawanmeh. A’s paper published in 2011, shows that a protocol used for IoT - Zigbee is verified for properties related to connection establishment properties using Event-B.

- Authors Lee, S., Kim, H., Hong, D. K., Ju, H, of paper written in 2013, give methods to evaluate performance of MQTT protocol with regards to different QoS levels used and compare with other IoT protocol CoAP.

- In their paper of 2013, Authors Che, X., Maag, S, test connection properties using passive testing for XMPP protocol in IoT.
Conclusion

- Proposed and demonstrated use of a framework based on Event-B for modelling some of the widely used IoT protocols MQTT, MQTT-SN and CoAP.
- Properties verified: QoS, persistent session, will, retain messages, resource discovery, two layered request-response architecture, caching, proxying and message deduplication.
- We show that the protocols work as intended in an uninterrupted network as well as with an intruder which consumes messages in the network.
- Proposed and used a model of time based on intervals of time points, a new feature in Event-B.
Thank you! Questions?

RATBERT, MY COMPANY IS HIRING FOR OUR QUALITY ASSURANCE GROUP. YOU’D BE PERFECT.

WHAT WOULD I HAVE TO DO?

YOU WOULD FIND FLAWS IN OUR NEW PRODUCT, THUS MAKING YOURSELF AN OBJECT OF INTENSE HATRED AND RIDICULE.

BUT THEN YOU’D FIX THOSE FLAWS... AND YOUR RESPECT FOR ME WOULD GROW INTO A SPECIAL BOND OF FRIENDSHIP, RIGHT?!

NO, THEN WE SHIP.