Formal Verification for safety critical requirements

From Unit Test to HIL
What if your PC could understand your requirements?
Agenda

- **Formal Specification of safety critical requirements**
  - The EmbeddedSpecifier Method
  - Universal Pattern
  - Requirements Coverage

- **Using Formal Specifications in the verification process**
  - Automatic Requirements-based Test Generation
  - Simulation-based Formal Verification
  - Formal Verification using Model Checking

- **Conclusion**
Agenda

- **Formal Specification of safety critical requirements**
  - The EmbeddedSpecifier Method
  - Universal Pattern
  - Requirements Coverage
- **Using Formal Specifications in the verification process**
  - Automatic Requirements-based Test Generation
  - Simulation-based Formal Verification
  - Formal Verification using Model Checking
- **Conclusion**
A Hierarchy of Notation Methods is defined

The more safety critical a function is, the more formal the notation and verification is recommended
### Challenges when specifying requirements in a formal way

- **Problem 1**: Some languages that might be used to express requirements are not formal.

- **Problem 2**: Formal methods are often considered to be too mathematical and too difficult to learn.

- **Solution**: A tool and a method that allows engineers to take their textual requirements and intuitively derive semi-formal and formal notations.

---

Example of a formal specification in LTL:

\[
S \rightarrow x.(!(![x+n,x+n] TRUE) \rightarrow \Box (y.([TSE \rightarrow (TC U[y+min,y+max] TEE]) \rightarrow z.([z+min,z+max] ASE \wedge ([z+min,z+max] ASE \rightarrow r.(AC U[r+min,r+max] AEE)))))))
\]
Example:

Req1: If the window moves up and an obstacle is detected, then the window has to start moving down in less than 10 ms.
1. Macro Definition

2. Structuring

3. Interface Mapping

Req1: If the window moves up and an obstacle is detected, then the window has to start moving down in less than 10 ms.
Req1: If the window moves up and an obstacle is detected, then the window has to start moving down in less than 10 ms.
Formal Specification Process

1. Macro Definition
2. Structuring
3. Interface Mapping

Req1: If the window moves up and an obstacle is detected, then the window has to start moving down in less than 10 ms.
Universal Pattern - A constructive approach

Speciation

Verification

Pre-Condition  Trigger 1  ...  Trigger n

[sequentiell | parallel]

ActMin

Action 1  ...  Action m

[sequentiell | parallel]

Implies eventually

or

not before, only after
Pattern Artifacts offer important detailed Information on Requirements

- Idea: Combine / Derive from these Artifacts (arbitrarily) to obtain Requirement Coverage Notions

Approach:

- consider all Valuations such that Artifacts/Conditions/Events evaluate to “true”
- Example: TriggerEvent: a||b → Goals: a&&b, !a&&b, a&&b
Agenda

- Formal Specification of safety critical requirements
  - The EmbeddedSpecifier Method
  - Universal Pattern
  - Requirements Coverage
- **Using Formal Specifications in the verification process**
  - Automatic Requirements-based Test Generation
  - Simulation-based Formal Verification
  - Formal Verification using Model Checking
- Conclusion
Requirements-based Test case generation

**Problem:**
- Manual creation of test cases is time consuming
- It is often unclear, which and how many test cases are needed to fully cover a requirement

**Solution:**
- Use Model Checking to generate test cases automatically from formalized requirements
- Ensure completeness of the generated test cases thanks to the clear definition of requirements coverage
Motivation for simulation-based formal verification

**Problem:**
- Testcases are typically created per requirement.
- It might stay undetected, if e.g. Test No.1 violates Requirement No.4.

**Solution:**
- Use a Requirement Observer to automatically observe the status of each requirement during the complete test process.
Simulation-based Formal Verification

Real-Time

BTC EmbeddedSpecifier

- Formal Specification

Test Environment

- Export
- Requirement Observer
  - Requirement Status: Fullfilled / Violated
- Observe
- System Under Test

Recorded Test Data

BTC EmbeddedValidator BASE

- Import
- Test Data
- Formal Specification

- Test case XY violated Requirement 5
- Requirement fulfilled

Test Environment

- Test Generation
  - Simulation-based
  - Complete Analysis

Specification

Verification
Example: Simulation-based Formal Verification in real-time

**Controller**

- Driver:
  - Up
  - Down

- Passenger:
  - Up
  - Down

- Obstacle Position

**Environment**

- Sequence Name: RTTSequence_0

<table>
<thead>
<tr>
<th>Activate</th>
<th>Status</th>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✅</td>
<td>REQ_PW_1_1</td>
<td>If the driver up switch is pressed, the window has to start moving up in less than 50 [ms]. Assumption: The driver does not push the down switch, the window is not at the top and there is no obstacle on the way.</td>
</tr>
<tr>
<td></td>
<td>✅</td>
<td>REQ_PW_1_2</td>
<td>If the driver down switch is pressed, the window has to start moving down in less than 50 [ms]. Assumption: The driver does not push the up switch, the window is not at the bottom.</td>
</tr>
<tr>
<td></td>
<td>✅</td>
<td>REQ_PW_2_1</td>
<td>If the driver or the passenger up switch is pressed for at most auto_up_time, the auto-up mode is activated and the window continues to move up. Assumption: The driver or the passenger does not push any other switch, the window is not at the top end and there is no obstacle on the way.</td>
</tr>
<tr>
<td></td>
<td>✅</td>
<td>REQ_PW_2_2</td>
<td>If the driver down or the passenger down switch is pressed for at most auto_down_time, the auto-down mode is activated and the window continues to move down. Assumption: The driver or the passenger does not push any other switch and the window is not at the bottom end.</td>
</tr>
<tr>
<td>✅</td>
<td>✅</td>
<td>REQ_PW_4_1</td>
<td>If the window moves up and an obstacle is detected, the window has to start moving down in less than 10 [ms]. Assumption: N/A</td>
</tr>
</tbody>
</table>

**dSPACE ControlDesk Next Generation**
Model Checking vs. Testing

- **Problem:**
  - A testcase only represents one possible path through the system
  - It is impossible to cover all paths with test cases

- **Solution:**
  - Model checking analyses all possible paths and guarantees a bug-free system
EmbeddedValidator - Method

Specification ➔ Verification

Test Generation | Simulation-based | Complete Analysis

dSPACE TargetLink ➔ Safety Requirements ➔ BTC EmbeddedSpecifier

BTC EmbeddedValidator ➔ Formal Requirement

TargetLink Code ➔ BTC EmbeddedValidator

Safety Requirements

Code does not fulfill the requirement + Counter Example

Code fulfills requirement
Agenda

- Formal Specification of safety critical requirements
  - The EmbeddedSpecifier Method
  - Universal Pattern
  - Requirements Coverage

- Using Formal Specifications in the verification process
  - Automatic Requirements-based Test Generation
  - Simulation-based Formal Verification
  - Formal Verification using Model Checking

- Conclusion
What if your PC could understand your requirements?

Better requirements!

Better verification!
Thank you.