Bridging the Gap Between Requirements and Model Analysis: Evaluation on Ten Cyber-Physical Challenge Problems

Anastasia Mavridou
Robust Software Engineering Group
NASA Ames Research Center
Safety-critical industry

• Guaranteeing proper system behavior can be challenging

• Very strict development process
  • High-level requirements are incrementally refined
  • Verification and validation at each level
  • Development process preserves the requirements
Safety-critical industry

• Guaranteeing proper system behavior can be challenging

• Very strict development process

  • High-level requirements are incrementally refined

  • Verification and validation at each level

  • Development process preserves the requirements

Difficult to make a formal connection between specifications and software artifacts
Lockheed Martin Case Study

• LM Aero Developed Set of 10 V&V Challenge Problems
• Each challenge includes:
  • Simulink model
  • Parameters
  • Documentation Containing Description and Requirements
  • Difficult due to transcendental functions, nonlinearities and discontinuous math, vectors, matrices, states
• Challenges built with commonly used blocks
• Publicly available case study
Overview of Challenge Problems

• Triplex Signal Monitor
• Finite State Machine
• Tustin Integrator
• Control Loop Regulators
• NonLinear Guidance Algorithm
• Feedforward Cascade Connectivity Neural Network
• Abstraction of a Control (Effector Blender)
• 6DoF with DeHavilland Beaver Autopilot
• System Safety Monitor
• Euler Transformation
Overview of Challenge Problems

- Triplex Signal Monitor
- Finite State Machine
- Tustin Integrator
- Control Loop Regulators
- NonLinear Guidance Algorithm
- Feedforward Cascade Connectivity Neural Network
- Abstraction of a Control (Effector Blender)
- 6DoF with DeHavilland Beaver Autopilot
- System Safety Monitor
- Euler Transformation
Finite State Machine Requirement Example

• Natural language requirement:

Exceeding sensor limits shall latch an autopilot pullup when the pilot is not in control (not standby) and the system is supported without failures (not apfail).
Finite State Machine Requirement Example

• Natural language requirement:

*Exceeding sensor limits* shall latch an autopilot *pullup* when the pilot is not in control (not *standby*) and the system is *supported* without failures (not *apfail*).
Finite State Machine Requirement Example

• Natural language requirement:

Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is not in control (not **standby**) and the system is **supported** without failures (not **apfail**).
Finite State Machine Requirement Example

- Natural language requirement:

> Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is **in autopilot**. **not in control** (not **standby**) and the system is **supported** **without failures** (not apfail).

\[
\text{autopilot} = \neg \text{standby} \& \neg \text{apfail} \& \text{supported}
\]
Finite State Machine Requirement Example

• Natural language requirement:

*Exceeding sensor limits* shall latch an autopilot *pullup* when the pilot is in *autopilot*. 
Finite State Machine Requirement Example

- Natural language requirement:

Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is in **autopilot**.

**First interpretation**: if **autopilot** and **limits** are true at a time step, then **pullup** must always be true at the same time step.
Finite State Machine Requirement Example

• Natural language requirement:

Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.

**First interpretation**: if autopilot and limits are true at a time step, then pullup must always be true at the same time step.
Finite State Machine Requirement Example

- Natural language requirement:

*Exceeding sensor* limits *shall latch an autopilot pullup when the pilot is in autopilot.*

**First interpretation:** if autopilot and limits are true at a time step, then pullup must always be true at the same time step.
Finite State Machine Requirement Example

• Natural language requirement:

Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is in **autopilot**.

*First interpretation*: if **autopilot** and **limits** are true at a time step, then **pullup** must always be true at the same time step.
Finite State Machine Requirement Example

- Natural language requirement:

*Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.*

**Second interpretation**: if autopilot and limits are true at a time step, then pullup must always be true at the next time step.
Finite State Machine Requirement Example

- Natural language requirement:

  Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is in **autopilot**.

  **Second interpretation**: if **autopilot** and **limits** are true at a time step, then **pullup** must always be true at the next time step.
Finite State Machine Requirement Example

- Natural language requirement:

*Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.*

**Third interpretation:** if limits and autopilot are true at a time step, then pullup and autopilot must be true at the next time step.
Finite State Machine Requirement Example

- Natural language requirement:

Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.

*Third interpretation*: if limits and autopilot are true at a time step, then pullup and autopilot must be true at the next time step.
Finite State Machine Requirement Example

• Natural language requirement:

*Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.*

*Third interpretation:* if limits and autopilot are true at a time step, then pullup and autopilot must be true at the next time step.
We formalized all three interpretations with FRET
FRET for the elicitation, formalization, and understanding of system requirements

FRET Team

Dimitra Giannakopoulou  Tom Pressburger  Johann Schumann
Finite State Machine Requirement

- Natural language requirement:

*Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.*

Atomic propositions in generated temporal formula.
Finite State Machine Requirement

- Natural language requirement:

Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.

Atomic propositions in generated temporal formula. Meaningless when it comes to the model!
Finite State Machine Requirement

• Natural language requirement:

Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.

Atomic propositions in generated temporal formula. Meaningless when it comes to the model!

Additional challenge: How to bridge the gap between requirements and analysis tools?
An Important Gap Remains

- Between
  - formalized requirements
  - model/code that they target
- Atomic propositions of a formula must be connected to variable values or method executions in the target code.
- This work proposes to bridge this gap
  - Bridging FRET and Analysis tools
  - Highly automatic approach
  - Interpretation of counterexamples both at requirements and model levels
An automated analysis and code generation framework for Simulink and Stateflow models

CoCoSim Team

Hamza Bourbouh  Pierre-Loic Garoche

... and many others from The University of Iowa, Onera - France, Carnegie Mellon University.
Our work supports…

- Automatic extraction of Simulink model information
- Association of high-level requirements with target model signals and components
- Translation of temporal logic formulas into synchronous data flow specifications and Simulink monitors
- Interpretation of counterexamples both at requirement and model levels
None of the three interpretations of the Finite State Machine requirement were satisfied by the model!
Writing Requirements in FRET

- Users enter system requirements in a restricted English-like language
Writing Requirements in FRET Input in FRET

• Users enter system requirements in a restricted English-like language

Component that the requirement refers to

- e.g., Autopilot, Monitor
Writing Requirements in FRET

• Users enter system requirements in a restricted English-like language

The component’s behavior must conform to the requirement
Writing Requirements in FRET

• Users enter system requirements in a restricted English-like language

Either an action or a Boolean condition

e.g., satisfy autopilot_engaged
Writing Requirements in FRET

- Users enter system requirements in a restricted English-like language

The period where the requirement holds

e.g., in/before/after initialization mode
Writing Requirements in FRET

- Users enter system requirements in a restricted English-like language

A Boolean expression that further constrains when the response shall occur

E.g., if $x > 0$
Writing Requirements in FRET

• Users enter system requirements in a restricted English-like language

A Boolean expression that further constrains when the response shall occur

\[ \text{e.g., if } x > 0 \]
Unambiguous Requirements with FRET

FSM shall always satisfy (limits & autopilot) => pullup

- Clear, unambiguous semantics in many different forms
  - Metric Temporal Logic
    - Pure Past time
    - Pure Future time
FRET Semantic Patterns

- FRET generates semantics based on templates.
- Each template is represented by a quadruple: [scope, condition, timing, response]

**FSM shall always satisfy (limits & autopilot) => pullup**

- [in, null, within, satisfaction] pattern

**Pure FT formula:** \( G (\text{first\_in\_$scope\_mode$} \rightarrow ((P I (\text{last\_in\_$scope\_mode$} I (X P))) I (F[<= $duration$] (P I (\text{last\_in\_$scope\_mode$} I (X P)))))) \)

**Pure PT formula:** \( H ((((!$post\_condition$) & $scope\_mode$) S (((!$post\_condition$) & $scope\_mode$) & \text{first\_in\_$scope\_mode$})) \rightarrow (\text{first\_in\_$scope\_mode$} I (O[<= $duration$] \text{first\_in\_$scope\_mode$})))) \)
Exporting Simulink Model Information

- Can be directly imported into FRET

```json
{
    "id": "fsm_12B/limits",
    "variable_name": "limits",
    "portType": "Import",
    "component_name": "fsm_12B",
    "dataType": [
        "boolean"
    ],
    "dimensions": [
        1, 1
    ],
    "width": 1
}
```
Linking requirement variables to Simulink signals

- FSM shall always satisfy (limits & autopilot) => pullup
Linking requirement variables to Simulink signals

- **FSM shall always satisfy (limits & autopilot) => pullup**
Translation of LTL to CoCoSpec

• Library of past time temporal operators

```coocospec
--Historically
node H(X:bool) returns (Y:bool);
let
    Y = X -> (X and (pre Y));
tel

node OT(const N:int; X:bool;) returns (Y:bool); --Timed Once
var C:int;
let
    C = if X then 0
        else (-1 -> pre C + (if pre C <0 then 0 else 1));
    Y = 0 <= C and C <= N;
tel
```
Generating CoCoSpec Contracts

```plaintext
contract FSMSpec(apfail: bool; limits: bool; standby: bool;
  supported: bool;) returns (pullup: bool;);
let
var FTP: bool = true -> false;
var autopilot: bool = supported and not apfail and not standby;
guarantee "FSM001" S( (((limits and autopilot) => (pullup))
  and FTP), (((limits and autopilot) => (pullup))));
```
Importing CoCoSpec to CoCoSim

FrontEnd
- Pre-Processing
- Optimization
- Traceability

Simulink / Stateflow

Compiler
- Lustre
- Automata
- Optimization
- Traceability

Pre-processed model

Lustre

Interface to Solvers
- Zustre
- Kind2
- JKind
- LustreC

45
Generating Simulink Observers

FSM shall **always** satisfy (limits & autopilot) => pullup
Tracing Counterexamples

FSM shall always satisfy (limits & autopilot) => pullup
natural language requirements

Model

Simulink

FRETish requirements

Signal info

FRET-to-Model mapping

pmLTL formulas

CoCoSpec code + Traceability Info

FRET

Simulator

CoCoSim
natural language requirements

Model

Simulink

FRETish requirements

Signal info

FRET-to-Model mapping

pmLTL formulas

CoCoSpec code + Traceability Info

Simulink model with Connected Monitors

CoCoSim
## Challenge Problem Analysis Results

<table>
<thead>
<tr>
<th>Name</th>
<th># Req</th>
<th># Form</th>
<th># An</th>
<th>Kind2 V/IN/UN</th>
<th>SLDV V/IN/UN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triplex Signal Monitor (TSM)</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5/1/0</td>
<td>5/1/0</td>
</tr>
<tr>
<td>Finite State Machine (FSM)</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>7/6/0</td>
<td>7/6/0</td>
</tr>
<tr>
<td>Tustin Integrator (TUI)</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2/0/1</td>
<td>2/0/1</td>
</tr>
<tr>
<td>Control Loop Regulators (REG)</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0/5/5</td>
<td>0/0/10</td>
</tr>
<tr>
<td>Feedforward Neural Network (NN)</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0/0/4</td>
<td>0/0/4</td>
</tr>
<tr>
<td>Control Allocator Effector Blender (EB)</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>0/0/3</td>
<td>0/0/0</td>
</tr>
<tr>
<td>6DoF Autopilot (AP)</td>
<td>14</td>
<td>13</td>
<td>8</td>
<td>5/3/0</td>
<td>4/0/4</td>
</tr>
<tr>
<td>System Safety Monitor (SWIM)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2/1/0</td>
<td>0/1/2</td>
</tr>
<tr>
<td>Euler Transformation (EUL)</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>2/5/0</td>
<td>1/0/6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>66</td>
<td>62</td>
<td>57</td>
<td>23/21/13</td>
<td>19/8/27</td>
</tr>
</tbody>
</table>
Lessons Learned

• Domain expertise: It is needed

• Frequently used patterns: used only 8/120 FRET patterns, mainly invariants

• What we gained by using CoCoSpec: modes introduce structure

• Reasoning for violated properties: two main ways 1) checking a weaker property; 2) check feasibility of stronger property.
Lessons Learned

• Incomplete Requirements: requirements were not mutually exclusive

• Scalability of the approach: tool-set keeps model hierarchy, contracts deployed at different levels

• Comparison of analysis tools: Kind2 faster usually than SLDV, also returned results in more cases due to modular analysis

• Optimization of FRET generated formulas
Thank you for your attention