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

Robust Software Engineering Group
NASA Ames Research Center, CA, USA

Hamza Bourbouh
hamza.bourbouh@nasa.gov

06/20/2019
Outline

1. Introduction
2. FRET and Past Time Metric LTL
3. Lustre & CoCoSpec
4. CoCoSim
5. Lockheed Martin Challenge Problems
   - LM challenge 2: Finite State Machine
   - LM challenge 8: 6DOF with DeHavilland Beaver Autopilot
   - LM challenges results
6. Lessons learned
7. Conclusion
Outline

1. Introduction
2. FRET and Past Time Metric LTL
3. Lustre & CoCoSpec
4. CoCoSim
5. Lockheed Martin Challenge Problems
   - LM challenge 2: Finite State Machine
   - LM challenge 8: 6DOF with DeHavilland Beaver Autopilot
   - LM challenges results
6. Lessons learned
7. Conclusion
Survey on Model-Based Software Engineering and Auto-Generated Code

Figure: Types of bugs observed in the models and auto-generated code (responses to each part of question ranged from 11 to 35)

1NASA/TM-2016-219443
Introduction

Safety-critical development process

- High-level requirements are incrementally refined.
- Verification and validation at each level.
- Development process preserves the requirements.

Challenge

Difficult to make a formal connection between specifications and software artifacts.

Motivation

- Providing requirements written in restricted natural languages with formal semantic (FRET).
- Attaching system requirements to software artifacts (FRET-CoCoSim).
- Analyzing the model against those requirements (CoCoSim).
FRET: **Formal Requirements Elicitation Tool**

FRET is a framework for the elicitation, formalization, and understanding of requirements.

**FRET Team**

- Anastasia Mavridou
- Dimitra Giannakopoulou
- Tom Pressburger
- Johann Schumann
CoCoSim: **Contract based Compositional verification of Simulink models.**

CoCoSim is 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, Khanh Trinh (NASA Ames)
FRET-CoCoSim workflow

**Figure: FRET-Workflow**

- **1a**: Natural language requirements
- **1b**: Simulink model
- **2a**: FRETish requirements
- **2b**: Signal info
- **pmLTL formulas**: FRET-to-Model mapping
- **3**: Simulator
- **4**: CoCoSpec code + Traceability Info
- **5**: Lustre + spec
- **6**: Export monitors to SLDV library
- **7**: Counterexamples

Next Steps:

- Analysis Tools: Kind2 Zustre
- Simulink model with Connected Monitors in SLDV

-- hamza.bourbouh@nasa.gov --

---

FRET-CoCoSim

06/20/2019 8 / 51
Introduction

FRET and Past Time Metric LTL

Lustre & CoCoSpec

CoCoSim

Lockheed Martin Challenge Problems
  - LM challenge 2: Finite State Machine
  - LM challenge 8: 6DOF with DeHavilland Beaver Autopilot
  - LM challenges results

Lessons learned

Conclusion
Users enter system requirements in a restricted English-like natural language called FRETish.

FRETish contains up to six fields: **scope**, **condition**, **component***, **shall***, **timing**, and **response***. Mandatory fields are indicated with an asterisk.

- **scope** field specifies the period where the requirement holds. If omitted, the requirement is deemed to hold universally.
- **condition** field is a Boolean expression that further constrains when the requirement response shall occur.
- **component** field specifies the component that the requirement refers to.
- **timing** field specifies when the response shall happen. For instance: *immediately, always, after N time units*, etc.
- **response** is either an action that the component must execute, or a Boolean condition that the component’s behavior must satisfy.
Syntax: `scope, component, shall, timing, response`

**AP-002:** In roll_hold mode RollAutopilot shall *always* satisfy autopilot_engaged & no_other_lateral_mode
For each requirement, FRET generates two LTL-based formalizations in:

1. pure Future Time Metric LTL; and
2. pure Past Time Metric LTL (we refer to it as pmLTL).

The syntax of the generated formulas is compatible with the NuSMV model checker.
Past Time Metric LTL

Past time operators (Y, O, H, S)

- Y (for ‘Yesterday’): At any non-initial time, $\text{Y}f$ is true iff $f$ holds at the previous time instant.
- O (for ‘Once’): $\text{O}f$ is true iff $f$ is true at some past time instant including the present time.
- H (for ‘Historically’): $\text{H}f$ is true iff $f$ is always true in the past.
- S (for ‘Since’): $\text{S}_t g f$ is true iff $g$ holds somewhere at point $t$ in the past and $f$ is true from that point on.
Past Time Metric LTL

**Time-constrained versions of past time operators**

\( O_p \ [l, r] \ f \), where \( O_p \in \{0, H, S\} \) and \( l, r \in \mathbb{N}^0 \).

- \( H \ [l, r] \ f \) is true at time \( t \) iff \( f \) holds in all previous time instants \( t' \) such that \( t - r \leq t' \leq t - l \).
- \( 0 \ [l, r] \ f \) is true at time \( t \) iff \( f \) was true in at least one of the previous time instants \( t' \) such that \( t - r \leq t' \leq t - l \).
- \( f \ S \ [l, r] \ g \) is true at time \( t \) iff \( g \) holds at point \( t' \) in the past such that \( t - r \leq t' \leq t - l \) and \( f \) is true from that point on.
Outline

1 Introduction

2 FRET and Past Time Metric LTL

3 Lustre & CoCoSpec

4 CoCoSim

5 Lockheed Martin Challenge Problems
   - LM challenge 2: Finite State Machine
   - LM challenge 8: 6DOF with DeHavilland Beaver Autopilot
   - LM challenges results

6 Lessons learned

7 Conclusion
Lustre synchronous dataflow language

- Lustre code consists of a set of *nodes* that transform infinite streams of *input* flows to streams of *output* flows.
- A symbolic “abstract” universal clock is used to model system progress.
- Two important Lustre operators are
  - Right-shift `pre` (for *previous*) operator: at time \( t = 0 \), `pre p` is undefined, while for each time instant \( t > 0 \) it returns the value of \( p \) at \( t - 1 \). Example:
    
    | t  | 0 | 1 | 2 | 3 |
    |----|---|---|---|---|
    | p  | 11| 12| 13|14 |
    | pre(p) | - | 11| 12|13 |

  - Initialization `→` (for *followed-by*) operator: At time \( t = 0 \), \( p \rightarrow q \) returns the value of \( p \) at \( t = 0 \), while for \( t > 0 \) it returns the value of \( q \) at \( t \).
    
    | t  | 0 | 1 | 2 | 3 |
    |----|---|---|---|---|
    | x0 | 0 | 2 | 4 | 6 |
    | p  | 11| 12| 13|14 |
    | x0 → pre(q) | 0 | 11|12|13 |

hamza.bourbouh@nasa.gov  
FRET-CoCoSim  
06/20/2019  16 / 51
Example of pmLTL operators in Lustre

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

- Since
  
  --Y S X
  node S(X,Y: bool) returns (Z:bool);
  let
      Z = X or (Y and (false -> pre Z));
  tel

- Once
  
  node O(X:bool) returns (Y:bool);
  let
      Y = X or (false -> pre Y);
  tel
CoCoSpec extends Lustre with constructs for the specification of assume-guarantee contracts.

CoCoSpec assume-guarantee contracts are pairs of past time LTL predicates.

A CoCoSpec contract can have:
- internal variable declarations
- assume \((A)\) statements
- guarantee \((G)\) statements
- mode declarations consist of require \((R)\) and ensure \((E)\) statements

A node satisfies a contract \(C = (A, G')\) if it satisfies \(\square A \Rightarrow G'\), where \(G' = G \cup \{R_i \Rightarrow E_i\}\).
node stopwatch ( toggle, reset : bool ) returns ( count : int );
(*@contract import stopwatchSpec(toggle, reset ) returns (count) ; *)
var running : bool;
let
    running = (false -> pre running) <> toggle ;
count =
    if reset then 0
    else if running then 1 -> pre count + 1
    else 0 -> pre count ;
tel
contract stopwatchSpec( toggle, reset : bool ) returns 
  ( time : int ) ;
let
  var on: bool = toggle -> (pre on and not toggle)
  or (not pre on and toggle) ;
assume not (toggle and reset) ;
guarantee time >= 0 ;
mode resetting ( 
  require reset ;
  ensure time = 0 ;
);
mode running ( 
  require (not reset) and on;
  ensure true -> time = pre time + 1 ;
);
mode stopped ( 
  require (not reset) and (not on) ;
  ensure true -> time = pre time ;
); tel
1 Introduction
2 FRET and Past Time Metric LTL
3 Lustre & CoCoSpec
4 CoCoSim
5 Lockheed Martin Challenge Problems
   - LM challenge 2: Finite State Machine
   - LM challenge 8: 6DOF with DeHavilland Beaver Autopilot
   - LM challenges results
6 Lessons learned
7 Conclusion
## CoCoSim: Unsupported blocks (1/4)

<table>
<thead>
<tr>
<th>Library</th>
<th># supp. Blocks</th>
<th>% supp. Blocks</th>
<th>Unsupported blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discontinuities</td>
<td>11</td>
<td>91%</td>
<td>Backlash</td>
</tr>
<tr>
<td>Discrete</td>
<td>19</td>
<td>90%</td>
<td>Discrete PID Controller, Discrete PID Controller (2DOF)</td>
</tr>
<tr>
<td>Logic &amp; Bit Operations.</td>
<td>18</td>
<td>95%</td>
<td>Extract Bits</td>
</tr>
<tr>
<td>Lookup Tables.</td>
<td>9</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Math Operations.</td>
<td>31</td>
<td>83%</td>
<td>Algebraic Constraint, Complex to Magnitude-Angle, Complex to Real-Imag, Find, Magnitude-Angle to Complex, Real-Imag to Complex</td>
</tr>
<tr>
<td>Library</td>
<td># supp. Blocks</td>
<td>% supp. Blocks</td>
<td>Unsupported blocks</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------</td>
<td>----------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>Model Verif.</td>
<td>11</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>Ports &amp; Subsystems.</td>
<td>29</td>
<td>93%</td>
<td>While Iterator Subsystem, While Iterator</td>
</tr>
<tr>
<td>Signal Att.</td>
<td>13</td>
<td>93%</td>
<td>Unit Conversion</td>
</tr>
<tr>
<td>Signal Routing.</td>
<td>13</td>
<td>52%</td>
<td>Data Store Memory/Read/Write, Env. Controller, Goto Tag Visibility, Index Vector, State Reader, State Writer, Variant Source, Variant Sink, Manual Variant Source, Manual Variant Sink</td>
</tr>
</tbody>
</table>
## CoCoSim: Unsupported blocks (3/4)

<table>
<thead>
<tr>
<th>Library</th>
<th># supp. Blocks</th>
<th>% supp. Blocks</th>
<th>Unsupported blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sinks.</td>
<td>9</td>
<td>100%</td>
<td>Band-Limited White Noise, Counter Free-Running, Counter Limited, From File, From Spreadsheet, Repeating Sequence, Repeating Sequence Interpolated, Repeating Sequence Stair, Signal Editor, Signal Generator, Waveform Generator</td>
</tr>
<tr>
<td>Sources.</td>
<td>15</td>
<td>57%</td>
<td></td>
</tr>
</tbody>
</table>

hamza.bourbouh@nasa.gov
### CoCoSim: Unsupported blocks (4/4)

<table>
<thead>
<tr>
<th>Library</th>
<th># supp. Blocks</th>
<th>% supp. Blocks</th>
<th>Unsupported blocks</th>
</tr>
</thead>
</table>
Outline

1. Introduction
2. FRET and Past Time Metric LTL
3. Lustre & CoCoSpec
4. CoCoSim
5. Lockheed Martin Challenge Problems
   - LM challenge 2: Finite State Machine
   - LM challenge 8: 6DOF with DeHavilland Beaver Autopilot
   - LM challenges results
6. Lessons learned
7. Conclusion
Lockheed Martin Challenge Problems

- 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. The challenges can be found in https://github.com/hbourbouh/lm_challenges
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
Type of Simulink blocks used in the Challenges

Some of the blocks make verification difficult due to:

- **Transcendental Functions**: Such as the trigonometric functions. Challenge 7 (AP) uses $\cos$, $\sin$, $\text{atan2}$, $\text{asin}$. Challenge 9 (EUL) uses $\sin$ and $\cos$.

- **Nonlinearities and Discontinuous Math**: Such as $\text{Abs}$, $\text{MinMax}$, $\text{Saturation}$, $\text{Switch}$. Inverse of Matrix (3 by 3 and 5 by 5 Matrices) are used in Challenge 6 (EB) and 7 (AP).

- **Multidimensional Arrays**: Challenges 6 (EB) and 7 (AP) use the inverse of matrices, which is abstracted in Lustre. Additionally, challenge 7 (AP) manipulates Quaternions with some advanced Quaternion operations (e.g. Quaternion Modulus, Quaternion Norm and Quaternion Normalize).

- **States**: Blocks such as $\text{Delay}$ and $\text{Unit Delay}$ are used in the majority of LMCPS. They are used to access memories of signals up to n steps back (n=1 for UnitDelay).
## Type of Simulink blocks used in the Challenges

<table>
<thead>
<tr>
<th>Model</th>
<th># Blocks</th>
<th>Block Types used</th>
</tr>
</thead>
<tbody>
<tr>
<td>0_triplex</td>
<td>479</td>
<td>'Abs', 'Action Port', 'Constant', 'Delay', 'Demux', 'From', 'Goto' 'If', 'Inport', 'Logic', 'Merge', 'Mux', 'Outport', 'Product', 'Relational Operator', 'Selector', 'Signal Conversion', 'Subsystem', 'Sum', 'Switch', 'Terminator'</td>
</tr>
<tr>
<td>1_fsm</td>
<td>279</td>
<td>'Action Port', 'Constant', 'Demux', 'From', 'Goto', 'If', 'Inport', 'Logic', 'Merge', 'Mux', 'Outport', 'Relational Operator', 'Signal Conversion', 'Subsystem', 'Switch', 'Unit Delay'</td>
</tr>
<tr>
<td>Model</td>
<td># Blocks</td>
<td>Block Types used</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>2_tustin</td>
<td>45</td>
<td>'DataType Duplicate', 'Data Type Propagation', 'From', 'Gain', 'Goto', 'Inport', 'Outport', 'Product', 'Relational Operator', 'Saturation Dynamic', 'Subsystem', 'Sum', 'Switch', 'Unit Delay'</td>
</tr>
</tbody>
</table>
## Type of Simulink blocks used in the Challenges

<table>
<thead>
<tr>
<th>Model</th>
<th># Blocks</th>
<th>Block Types used</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_nn</td>
<td>699</td>
<td>'ActionPort', 'Constant', 'Demux', 'Gain', 'If', 'Inport', 'Merge', 'Mux', 'Outport', 'Product', 'Saturate', 'SubSystem', 'Sum'</td>
</tr>
<tr>
<td>6_eb</td>
<td>75</td>
<td>'Constant', 'Display', 'Inport', 'Math', 'Output', 'Product', 'Relational Operator', 'Reshape', 'Selector', 'SubSystem', 'Sum', 'Switch'</td>
</tr>
<tr>
<td>Model</td>
<td># Blocks</td>
<td>Block Types used</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
### Type of Simulink blocks used in the Challenges

<table>
<thead>
<tr>
<th>Model</th>
<th># Blocks</th>
<th>Block Types used</th>
</tr>
</thead>
<tbody>
<tr>
<td>8_swim</td>
<td>141</td>
<td>'ActionPort', 'Constant', 'Display', Gain, If, Inport, Logic, Merge, Outport,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Relational Operator, Sqrt, SubSystem, Sum, UnitDelay</td>
</tr>
<tr>
<td>9_euler</td>
<td>97</td>
<td>'Concatenate', 'Fcn', Inport, Mux, Outport, Product, Reshape, SubSystem,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trigonometry, Create 3x3 Matrix</td>
</tr>
</tbody>
</table>
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**).

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

**autopilot** = !**standby** & !**apfail** & **supported**
Finite State Machine Requirement Example

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

**First interpretation:**

![Finite State Machine Diagram 1](image1.png)

**Second interpretation:**

![Finite State Machine Diagram 2](image2.png)
Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is in **autopilot**.

**Third interpretation:** Does autopilot should stay active when latching a pullup?
Exceeding sensor \textbf{limits} shall latch an autopilot \textbf{pullup} when the pilot is in \textbf{autopilot}.

\textbf{First interpretation:}

\textbf{FSM} shall \textit{always} satisfy (limits & autopilot) $\Rightarrow$ pullup

\[
((\text{limits} \& \text{autopilot}) \Rightarrow \text{pullup}) \Rightarrow ((\text{limits} \& \text{autopilot}) \Rightarrow \text{pullup}) \& \text{FTP}
\]

\begin{verbatim}
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)))

end
\end{verbatim}
Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is in **autopilot**.

**First interpretation:**

\[
\text{FSM shall always satisfy (limits & autopilot) } \Rightarrow \text{ pullup}
\]

\[
((\text{limits & autopilot}) \Rightarrow \text{pullup}) \text{ S } (((\text{limits & autopilot}) \Rightarrow \text{pullup}) \& \text{FTP})
\]
Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is in **autopilot**.
Example of an algebraic loop accepted by Simulink.

Figure: A simple example of an algebraic loop.

\[ xa = u + 2*xa; \]

The generated Lustre that will be rejected because of the circular dependency.
Examples of requirements we needed domain expert help.

- **AP-004a:** Steady state roll commands shall be tracked within 1 degree in calm air.

- **AP-004b:** Response to roll step commands shall not exceed 10% overshoot in calm air.

Example of a requirement we could not formalize.

- **AP-004c:** Small signal ($<3$ degree) roll bandwidth shall be at least 0.5 rad/sec.
## 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 Monitor</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5/1/0</td>
<td>5/1/0</td>
</tr>
<tr>
<td>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</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2/0/1</td>
<td>2/0/1</td>
</tr>
<tr>
<td>Regulators</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0/5/5</td>
<td>0/0/10</td>
</tr>
<tr>
<td>Feedforward NN</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0/0/4</td>
<td>0/0/4</td>
</tr>
<tr>
<td>Effector Blender</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</td>
<td>14</td>
<td>13</td>
<td>8</td>
<td>5/3/0</td>
<td>4/0/4</td>
</tr>
<tr>
<td>Sys. 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 Transf.</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>
1. Introduction

2. FRET and Past Time Metric LTL

3. Lustre & CoCoSpec

4. CoCoSim

5. Lockheed Martin Challenge Problems
   - LM challenge 2: Finite State Machine
   - LM challenge 8: 6DOF with DeHavilland Beaver Autopilot
   - LM challenges results

6. Lessons learned

7. Conclusion
Lessons learned

- Domain expertise is needed
- Frequently used patterns: used only 8/120 FRET patterns, mainly invariants
- 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
Lessons learned

- Reasoning for violated properties: two ways

  \[ H(A \Rightarrow B) \]

- Check a weaker property by strengthening the preconditions \( A' \subset A \) and check \( H(A' \Rightarrow B) \)
- Check feasibility of \( B \) with bounded model checking \( H(\neg B) \) and return counterexamples to help construct stronger preconditions for which \( B \) is satisfied
Outline

1 Introduction
2 FRET and Past Time Metric LTL
3 Lustre & CoCoSpec
4 CoCoSim
5 Lockheed Martin Challenge Problems
   - LM challenge 2: Finite State Machine
   - LM challenge 8: 6DOF with DeHavilland Beaver Autopilot
   - LM challenges results
6 Lessons learned
7 Conclusion
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
Bridging the Gap Between Requirements and Model Analysis: Evaluation on Cyber-Physical Challenge Problems

Robust Software Engineering Group
NASA Ames Research Center, CA, USA

Hamza Bourbouh
hamza.bourbouh@nasa.gov

06/20/2019