

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



- 1 Introduction
- 2 FRET and Past Time Metric LTL
- 3 CoCoSpec & Simulink
- 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

- 1 Introduction
- 2 FRET and Past Time Metric LTL
- 3 CoCoSpec & Simulink
- 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

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: **C**ontract based **C**ompositional verification of **S**imulink models.

CoCoSim is an automated analysis and code generation framework for Simulink and Stateflow models.

CoCoSim Team



Hamza Bourbough



Pierre-Loic Garoche

... and many others from
The University of Iowa,
Onera - France,
Khanh Trinh (NASA Ames)

FRET-CoCoSim workflow

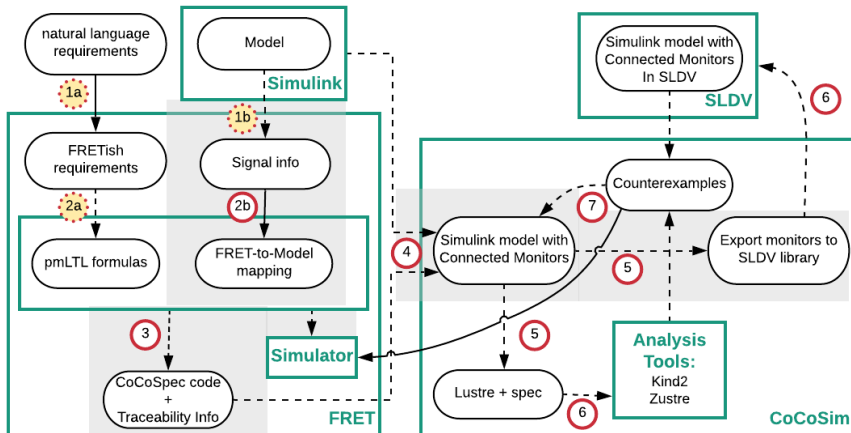


Figure: FRET-Workflow

Outline

- 1 Introduction
- 2 FRET and Past Time Metric LTL
- 3 CoCoSpec & Simulink
- 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

FRET and Past Time Metric LTL

- 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.

Example

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 operators (Y, O, H, S)

- Y (for 'Yesterday'): Yf is true iff f holds at the previous time instant.
- O (for 'Once'): Of is true iff f is true at some past time instant including the present time.
- H (for 'Historically'): Hf is true iff f is always true in the past.
- S (for 'Since'): fSg is true iff g holds somewhere at point t in the past and f is true from that point on.

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 CoCoSpec & Simulink**
- 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 \rightarrow (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
p	11	12	13	14
q	0	2	4	6
$p \rightarrow q$	11	2	4	6

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 $H A \Rightarrow G'$, where $G' = G \cup \{R_i \Rightarrow E_i\}$.

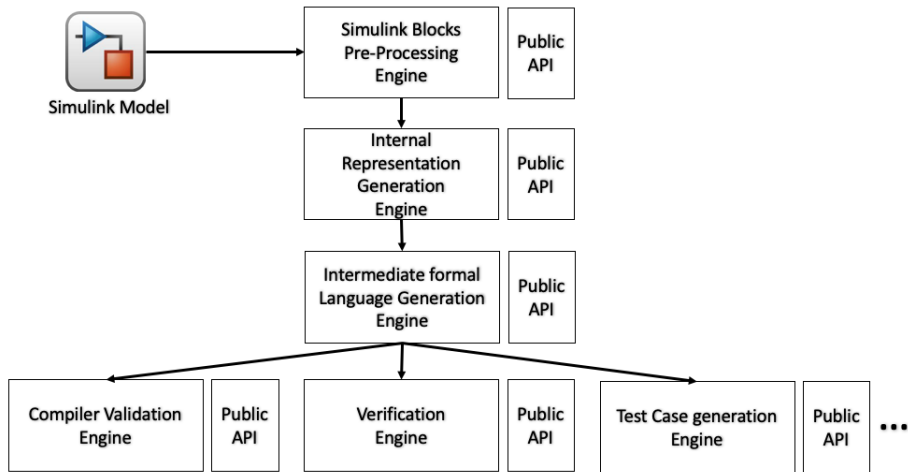
Example: Stopwatch implementation

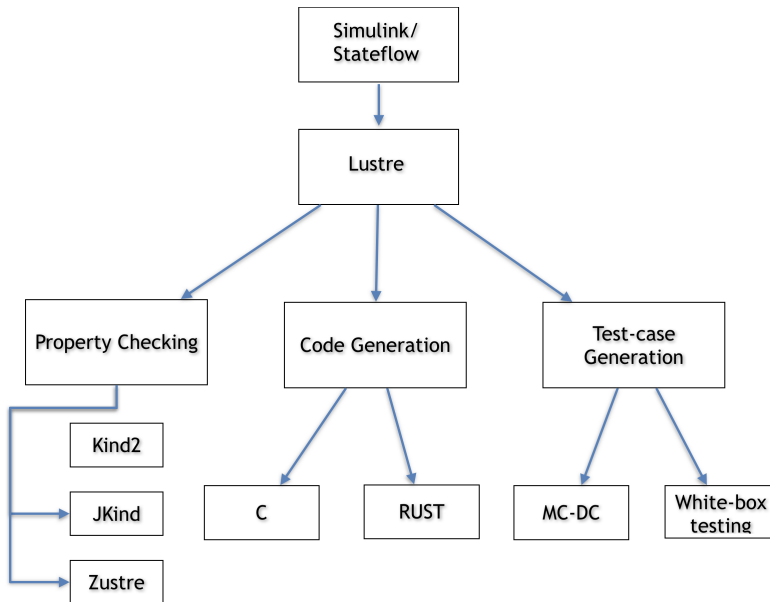
```
node stopwatch ( toggle, reset : bool ) returns (
    count : int );
(*@contract import stopwatchSpec(toggle, reset )
    returns (count) ; *)
var running : bool;
let
    running = (false -> pre running) <> toggle ;
    count =
        if reset then 0
        else if running then 1 -> pre count + 1
        else 0 -> pre count ;
tel
```

Example: Stopwatch Specification

```
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 CoCoSpec & Simulink
- 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)

Library	# supp. Blocks	% supp. Blocks	Unsupported blocks
Discontinuities	11	91%	Backlash
Discrete	19	90%	Discrete PID Controller, Discrete PID Controller (2DOF)
Logic & Bit Operations.	18	95%	Extract Bits
Lookup Tables.	9	100%	
Math Operations.	31	83%	Algebraic Constraint, Complex to Magnitude-Angle, Complex to Real-Imag, Find, Magnitude-Angle to Complex, Real-Imag to Complex

CoCoSim: Unsupported blocks (2/4)

Library	# supp. Blocks	% supp. Blocks	Unsupported blocks
Model Verif.	11	100%	
Ports & Sub-systems.	29	93%	While Iterator Subsystem, While Iterator
Signal Att.	13	93%	Unit Conversion
Signal Routing.	13	52%	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

CoCoSim: Unsupported blocks (3/4)

Library	# supp. Blocks	% supp. Blocks	Unsupported blocks
Sinks.	9	100%	
Sources.	15	57%	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

CoCoSim: Unsupported blocks (4/4)

Library	# supp. Blocks	% supp. Blocks	Unsupported blocks
User-Defined Functions.	1	6%	Argument Inport, Argument Output, Event Listener, Function Caller, Initialize Function, MATLAB Function, Interpreted MATLAB Function, Level-2 MATLAB S-Function, MATLAB System, Reset Function, S-Function, S-Function Builder, Simulink Function, Terminate Function

- 1 Introduction
- 2 FRET and Past Time Metric LTL
- 3 CoCoSpec & Simulink
- 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

- 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*, *atan2*, *asin*. Challenge 9 (EUL) uses *sin* and *cos*.
- **Nonlinearities and Discontinuous Math:** Such as *Abs*, *MinMax*, *Saturation*, *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 *Delay* and *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

Model	# Blocks	Block Types used
0_triplex	479	' Abs ', 'Action Port', 'Constant', ' Delay ', 'Demux', 'From', 'Goto', ' If ', 'Inport', ' Logic ', ' Merge ', 'Mux', 'Outport', ' Product ', ' Relational Operator ', ' Selector ', 'Signal Conversion', 'Subsystem', ' Sum ', ' Switch ', 'Terminator'
1_fsm	279	'Action Port', 'Constant', 'Demux', 'From', 'Goto', ' If ', 'Inport', ' Logic ', ' Merge ', 'Mux', 'Outport', ' Relational Operator ', ' Signal Conversion ', 'Subsystem', ' Switch ', ' Unit Delay '

Type of Simulink blocks used in the Challenges

Model	# Blocks	Block Types used
2_tustin	45	'DataType Duplicate', 'Data Type Propagation', 'From', 'Gain', 'Goto', 'Inport', 'Outport', ' Product ', ' Relational Operator ', ' Saturation Dynamic ', 'Subsystem', ' Sum ', ' Switch ', ' Unit Delay '
3_regulators	271	' BusCreator ', ' BusSelector ', 'Constant', 'From', 'Gain', 'Goto', 'Inport', ' Lookup_nD ', 'Math', ' Memory ', 'Outport', ' Product ', ' Relational Operator ', ' Saturate ', ' Saturation Dynamic ', ' Signal Conversion ', 'SubSystem', 'Sum', ' Switch ', 'Terminator', ' UnitDelay '

Type of Simulink blocks used in the Challenges

Model	# Blocks	Block Types used
4_nlguid	355	'ActionPort', 'Constant', 'Demux', 'Display', ' DotProduct ', 'From', ' Gain ', 'Goto', ' If ', 'Inport', 'InportShadow', ' Logic ', ' Math ', ' Merge ', 'Mux', 'Outport', ' Product ', ' Relational Operator ', ' Selector ', ' Sqrt ', 'SubSystem', ' Sum ', 'Terminator'
5_nn	699	'ActionPort', 'Constant', 'Demux', ' Gain ', ' If ', 'Inport', ' Merge ', 'Mux', 'Outport', ' Product ', ' Saturate ', 'SubSystem', ' Sum '
6_eb	75	'Constant', 'Display', 'Inport', ' Math ', 'Outport', ' Product ', ' Relational Operator ', ' Reshape ', ' Selector ', 'SubSystem', ' Sum ', ' Switch '

Type of Simulink blocks used in the Challenges

Model	# Blocks	Block Types used
7_autopilot	1357	'Abs', 'BusCreator', 'BusSelector', 'Concatenate', 'Constant', 'Data Type Conversion', 'Demux', 'Display', 'DotProduct', 'Fcn', 'From', 'Gain', 'Goto', 'Ground', 'Inport', 'InportShadow', 'Logic', 'Lookup_nD', 'Math', 'MinMax', 'Mux', 'Outport', 'Product', 'RateLimiter', 'Relational Operator', 'Reshape', 'Rounding', 'Saturate', 'Scope', 'Selector', 'Signum', 'Sqrt', 'SubSystem', 'Sum', 'Switch', 'Terminator', 'Trigonometry', 'UnitDelay', 'CMBBlock', 'Create 3x3 Matrix', 'Passive', 'Quaternion Modulus', 'Quaternion Norm', 'Quaternion Normalize', 'Rate Limiter Dynamic'

Type of Simulink blocks used in the Challenges

Model	# Blocks	Block Types used
8_swim	141	'ActionPort', 'Constant', 'Display', ' Gain ', 'If', 'Inport', ' Logic ', ' Merge ', 'Outport', ' Relational Operator ', ' Sqrt ', 'SubSystem', ' Sum ', ' UnitDelay '
9_euler	97	'Concatenate', 'Fcn', 'Inport', 'Mux', 'Outport', ' Product ', ' Reshape ', 'SubSystem', ' Trigonometry ', ' Create 3x3 Matrix '

Finite State Machine Requirement Example

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 not **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:

limits & autopilot



pullup



limits & autopilot



pullup



Second interpretation:

limits & autopilot



pullup

limits & autopilot

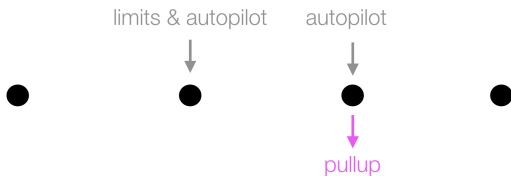


pullup

Finite State Machine Requirement Example

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?



Finite State Machine Requirement Example

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

First interpretation:

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

$((\text{limits} \ \& \ \text{autopilot}) \Rightarrow \text{pullup}) \ \text{S} \ (((\text{limits} \ \& \ \text{autopilot}) \Rightarrow \text{pullup}) \ \& \ \text{FTP})$

```
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)));
tel
```

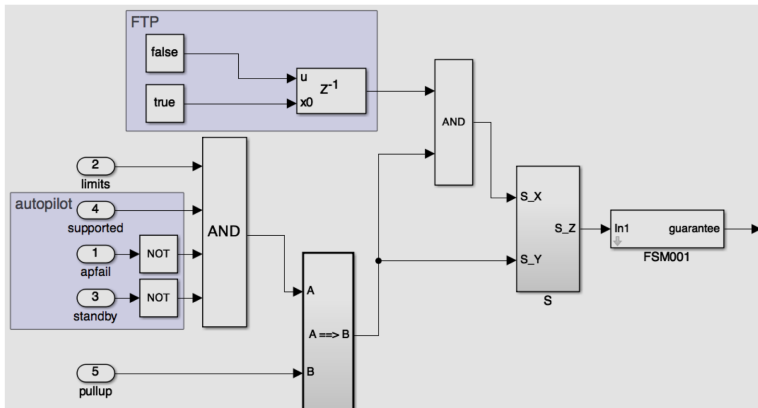
Finite State Machine Requirement Example

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

First interpretation:

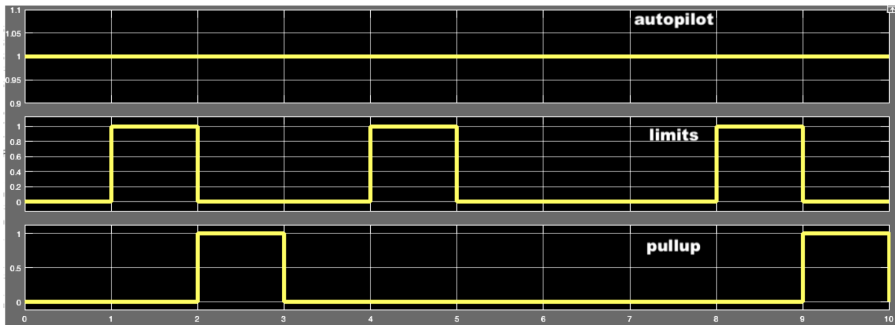
FSM shall always satisfy $(\text{limits} \ \& \ \text{autopilot}) \Rightarrow \text{pullup}$

$((\text{limits} \ \& \ \text{autopilot}) \Rightarrow \text{pullup}) \ S \ (((\text{limits} \ \& \ \text{autopilot}) \Rightarrow \text{pullup}) \ \& \ \text{FTP})$

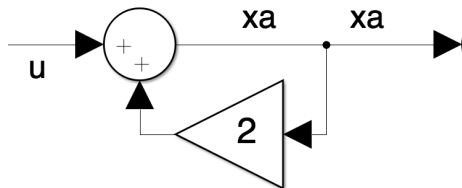


Finite State Machine Requirement Example

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



Algebraic loop



Example of an algebraic loop accepted by Simulink.

$$xa = u + 2*xa;$$

The generated Lustre that will be rejected because of the circular dependency.

Figure: A simple example of an algebraic loop.

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

Name	# Req	# Form	# An	Kind2	SLDV
				V/IN/UN	V/IN/UN
Triplex Monitor	6	6	6	5/1/0	5/1/0
FSM	13	13	13	7/6/0	7/6/0
Tustin Integrator	4	3	3	2/0/1	2/0/1
Regulators	10	10	10	0/5/5	0/0/10
Feedforward NN	4	4	4	0/0/4	0/0/4
Effector Blender	4	3	3	0/0/3	0/0/0
6DoF Autopilot	14	13	8	5/3/0	4/0/4
Sys. Safety Monitor (SWIM)	3	3	3	2/1/0	0/1/2
Euler Transf.	8	7	7	2/5/0	1/0/6
Total	66	62	57	23/21/13	19/8/27

Outline

- 1 Introduction
- 2 FRET and Past Time Metric LTL
- 3 CoCoSpec & Simulink
- 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

- Domain expertise: It 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

- 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 CoCoSpec & Simulink
- 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

- 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

