Formalizing and Analyzing Requirements with FRET

Anastasia Mavridou

Robust Software Engineering Group
SGT Inc., KBR / NASA Ames Research Center
Requirements engineering

• Central step in the development of safety-critical systems

• 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).*
Requirements engineering

Natural language
- Ambiguous
- No formal analysis

Mathematical notations
- Unambiguous
- Various analysis techniques
Despite the ambiguity of unrestricted natural language, it is unrealistic to expect developers to write requirements in mathematical notations.
Autopilot 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).
Autopilot 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).*
Autopilot 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 (and not apfail).*
Autopilot 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).

autopilot = !standby & supported & !apfail
Autopilot Requirement Example

- Natural language requirement:

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

• Natural language requirement:

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

- Natural language requirement:

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

• Natural language requirement:

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

- Natural language requirement:

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

• Natural language requirement:

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

- Natural language requirement:

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

![Diagram showing limits & autopilot pullups](image-url)
Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is in **autopilot**.
Autopilot Requirement Example

• Natural language requirement:

Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.
None of the three interpretations of the Autopilot requirement were satisfied by the model!
FRETish

- Restricted natural language for writing requirements
  - Intuitive
  - Unambiguous
  - Based on a grammar
  - Underlying semantics are determined by specific fields.
Writing Requirements in FRETish

- Users enter system requirements in a structured English-like language
Writing Requirements in FRETish

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

Component that the requirement refers to

e.g., Autopilot, Monitor
Writing Requirements in FRETish

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

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

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

A Boolean expression

e.g., satisfy autopilot_engaged
Writing Requirements in FRETish

- 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 FRETish

- 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 FRETish

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

Specifies when the response shall happen, relative to the scope and condition

e.g., always, immediately, after n time steps
Unambiguous Requirements with FRET

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

• Clear, unambiguous semantics in many different forms
  • Linear Temporal Logic
    • Pure Past time
    • Pure Future time
Temporal logics

<table>
<thead>
<tr>
<th>Future time</th>
<th>Past time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Future time operators</td>
</tr>
<tr>
<td></td>
<td>X, F, G, U</td>
</tr>
<tr>
<td></td>
<td>Past time operators</td>
</tr>
<tr>
<td></td>
<td>Y, O, H, S</td>
</tr>
</tbody>
</table>

A future time formula is satisfied by an execution, if the formula holds at the **initial** state of the execution.

A past time formula is satisfied by an execution, if the formula holds at the **final** state of the execution.
Future time Operators

$X$ (Next) refers to the next time step:

$X \phi$ is true iff $\phi$ holds at the next time step
Future time Operators

\( X \) (Next): refers to the next time step:

\( X \phi \) is true iff \( \phi \) holds at the next time step
Future time Operators

\( \mathbf{X} \) (Next): refers to the next time step:

\( \mathbf{X} \phi \) is true iff \( \phi \) holds at the next time step

Dual past time operator: \( \mathbf{Y} \) (Yesterday)
Future time Operators

\( U \) (Until) refers to multiple time steps:

\[ \phi U \psi \text{ is true iff } \psi \text{ holds at some time step } t \text{ in the future and for all time steps } t' \text{ such that } t' < t \text{ } \phi \text{ is true.} \]
Future time Operators

**U** (Until): refers to multiple time steps

φ U ψ is true iff ψ holds at some time step t in the future and for all time steps t’ (such that t’ < t) φ is true.
Future time Operators

\( \mathbf{U} \) (Until): refers to multiple time steps

\( \phi \mathbf{U} \psi \) is true iff \( \psi \) holds at holds at some time step \( t \) in the future and for all time steps \( t' \) (such that \( t' < t \)) \( \phi \) is true.

Dual past time operator: \( \mathbf{S} \) (Since)
Future time Operators

\( F \) (eventually): refers to at least one time step in the future:

\( F \phi \) is true iff \( \phi \) is true at some future time point including the present time
Future time Operators

\( F \) (eventually): refers to at least one time step in the future:

\( F \phi \) is true iff \( \phi \) is true at some future time point including the present time
Future time Operators

\(F\) (eventually): refers to at least one time step in the future:

\(F \phi\) is true iff \(\phi\) is true at some future time point including the present time
Future time Operators

$\mathbf{G}$ (Globally): refers to all future steps of an execution

$\mathbf{G} \phi$ is true iff $\phi$ is always true in the future
Future time Operators

\( G \) (Globally): refers to all future steps of an execution

\( G \phi \) is true iff \( \phi \) is always true in the future
Future time Operators

\( G \) (Globally): refers to all future steps of an execution

\( G \, \phi \) is true iff \( \phi \) is always true in the future

Dual past time operator: \( H \) (Historically)
FRET Semantic Patterns

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

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

- [null, null, always] pattern

  - Pure FT: $G (( \text{limits} \& \text{autopilot}) \Rightarrow \text{pullup})$
  - Pure PT: $H (( \text{limits} \& \text{autopilot}) \Rightarrow \text{pullup})$
FRET Semantic Patterns

If autopilot & limits Autopilot shall after 1 step satisfy pullup

- [null, regular, after, satisfaction] pattern

- Pure PT: 
  \[
  \]
FRET Semantic Patterns

If autopilot & limits Autopilot shall after 1 step satisfy pullup

• [null, regular, after, satisfaction] pattern

• Pure PT: \((\text{H}(((\neg \text{FTP}) \ S \ ((\text{autopilot} \ & \ \text{limits}) \ & \ ((Y (\neg (\text{autopilot} \ & \ \text{limits}))) \ | \ \text{FTP}))) \ & \ ((O[<=1] ((\text{autopilot} \ & \ \text{limits}) & ((Y (\neg (\text{autopilot} \ & \ \text{limits}))) \ | \ \text{FTP})))) \rightarrow (\neg (\text{pullup}))) \ & \ ((\text{autopilot} \ & \ \text{limits}) \ & \ \text{FTP}) \rightarrow (\neg (\text{pullup})))) \ & \ (\text{H}((O[=2] (((\text{autopilot} \ & \ \text{limits}) \ & ((Y (\neg (\text{autopilot} \ & \ \text{limits}))) \ | \ \text{FTP}) \ & \ (\neg (\text{pullup})))) \rightarrow (O[<2] (\text{FTP} \ | \ (\text{pullup}))))))

Time-constrained versions of past-time operators
How do we make the connection with analysis tools?
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 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 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 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 must be mapped to model signal values or method executions in the target code.
• To breach this gap:
  • Connect FRET with Analysis tools (CoCoSim, NuSMV, etc)
  • Highly automated approach
  • Interpretation of counterexamples both at requirements and models level
Mapping propositions to model signals

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

- **Pure PT:** ((( limits & autopilot ) => pullup) S ((( limits & autopilot ) => pullup) & FTP))
Mapping propositions to model signals

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

- Pure PT: \(((\text{limits} \& \text{autopilot}) \Rightarrow \text{pullup}) \& \text{FTP})\)
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**
Lustre & CoCoSpec

- A synchronous, declarative language that operates on **streams**
- A Lustre program is called a **node** and has a cyclic behavior
- At the $n$th execution cycle of the program, all the involved streams take their $n$th value
- Variables represent input, output, and locally defined streams
- CoCoSpec: a mode-aware **assume-guarantee-based contract language** built as an extension of the Lustre language.

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

    ((( limits & autopilot ) => pullup) S ((( limits & autopilot ) => pullup) & FTP)
Lustre & CoCoSpec

- A synchronous, declarative language that operates on streams
- A Lustre program is called a node and has a cyclic behavior
- At the $n$th execution cycle of the program, all the involved streams take their $n$th value
- Variables represent input, output, and locally defined streams

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

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

```
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)));
```
Lustre & CoCoSpec

Autopilot shall **always** satisfy \((\text{limits} \& \text{autopilot}) \implies \text{pullup}\)
\[
((\text{limits} \& \text{autopilot}) \implies \text{pullup}) \land ((\text{limits} \& \text{autopilot}) \implies \text{pullup}) \land \text{FTP})
\]

**CocoSpec**

```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)));
tel
```
Autopilot shall always satisfy \((\text{limits} \& \text{autopilot}) \implies \text{pullup}\)

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

Input variables

```haskell
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
```
Lustre & CoCoSpec

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

(( limits & autopilot ) => pullup) \( S \) ((( limits & autopilot ) => pullup) & FTP)

Output variable

```javascript
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)));
```
Lustre & CoCoSpec

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

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

```plaintext
contract FSMSpec(apfail:bool; limits:bool; standby:bool; supported:bool; ) returns (pullup: bool; );
let
var FTP:bool=true \Rightarrow false;
var autopilot:bool=supported and not apfail and not standby

\text{guarantee} "FSM001" ( ( ((\text{limits} \& \text{autopilot}) \Rightarrow \text{pullup})
  
  and FTP), ((\text{limits and autopilot}) \Rightarrow \text{pullup}) ));

tel
```
Lustre & CoCoSpec

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

(( limits & autopilot ) => pullup) S ((( limits & autopilot ) => pullup) & FTP)

**Translated past time LTL formula**

```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)));
tel
```
Translation of LTL to CoCoSpec/Lustre

- Library of past time temporal operators

```plaintext
--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 Simulink Observers

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

\(((\text{limits} \& \text{autopilot}) \Rightarrow \text{pullup}) \& \text{FTP}\)
Generating Simulink Observers

Autopilot shall always satisfy \((\text{limits} \& \text{autopilot}) \implies \text{pullup}\)

\(((\text{limits} \& \text{autopilot}) \implies \text{pullup}) \land ((\text{limits} \& \text{autopilot}) \implies \text{pullup}) \land \text{FTP})\)
Generating Simulink Observers

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

((( limits & autopilot ) => pullup) S ((( limits & autopilot ) => pullup) & FTP))
Autopilot shall always satisfy (limits & autopilot) => pullup

(((limits & autopilot) => pullup) S (((limits & autopilot) => pullup) & FTP))
Tracing Counterexamples

If autopilot & limits Autopilot shall after 1 step satisfy autopilot & pullup
Tracing Counterexamples

If autopilot & limits Autopilot shall after 1 step satisfy autopilot & pullup

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

Very different from the initial requirement!
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
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
## Challenge Problem Complexity

### Number of blocks

| 7_autopilot | 1357 |

### Types of Blocks

# Challenge Problem Complexity

<table>
<thead>
<tr>
<th>Number of blocks</th>
<th>Types of Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>7_autopilot</td>
<td>1357</td>
</tr>
</tbody>
</table>
## Challenge Problem Complexity

<table>
<thead>
<tr>
<th>Number of blocks</th>
<th>Types of Blocks</th>
</tr>
</thead>
</table>

**Nonlinearities & Discontinuous math**
# 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>
# 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><strong>66</strong></td>
<td><strong>62</strong></td>
<td><strong>57</strong></td>
<td><strong>23/21/13</strong></td>
<td><strong>19/8/27</strong></td>
</tr>
</tbody>
</table>
## Challenge Problem Analysis Results

<table>
<thead>
<tr>
<th>Name</th>
<th># Req</th>
<th># Form</th>
<th># An</th>
<th>Kind2</th>
<th>SLDV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>V/IN/UN</td>
<td>V/IN/UN</td>
</tr>
<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>
## 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><strong>66</strong></td>
<td><strong>62</strong></td>
<td><strong>57</strong></td>
<td><strong>23/21/13</strong></td>
<td><strong>19/8/27</strong></td>
</tr>
</tbody>
</table>

**Algebraic loop!**
### Challenge Problem Analysis Results

<table>
<thead>
<tr>
<th>Name</th>
<th># Req</th>
<th># Form</th>
<th># An</th>
<th>V/IN/UN</th>
<th>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><strong>66</strong></td>
<td><strong>62</strong></td>
<td><strong>57</strong></td>
<td><strong>23/21/13</strong></td>
<td><strong>19/8/27</strong></td>
</tr>
</tbody>
</table>

Abstraction of trigonometric, non-linear functions and allows local analysis
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
Thank you for your attention!