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

<table>
<thead>
<tr>
<th>Natural language</th>
<th>Mathematical notations</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Ambiguous</td>
<td>• Unambiguous</td>
</tr>
<tr>
<td>• No formal analysis</td>
<td>• Various analysis techniques</td>
</tr>
</tbody>
</table>
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).*

\[
\text{autopilot} = \neg \text{standby} \& \text{supported} \& \neg \text{apfail}
\]
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**.*
Exceeding sensor **limits** shall latch an autopilot **pullup** when the pilot is in **autopilot**.
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) $\Rightarrow$ pullup

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

Future time

• Future time operators
  • X, F, G, U

Past time

• Past time operators
  • Y, O, H, S

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

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

$\mathbf{X} \phi$ 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:

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

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

\(\phi \ U \ \psi\) is true iff \(\psi\) 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: **S** \((\text{Since})\)
Future time Operators

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

**F** φ is true iff φ 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:

\[ F \phi \text{ is true iff } \phi \text{ is true at some future time point including the present time} \]
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

Dual past time operator: \( O \) (Once)
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
Future time Operators

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

\( \textbf{G} \phi \) is true iff \( \phi \) is always true in the future

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

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

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

- [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:** \(((H ((((((\neg FTP) S ((autopilot & limits) \& ((Y \neg (autopilot & limits))) | FTP))) \& (O[\leq1] ((autopilot & limits)\& ((Y \neg (autopilot & limits))) | FTP)))) -> (! (pullup))) \& (((autopilot & limits) \& FTP) -> (! (pullup)))))) \& (H ((O[=1+1] (((autopilot & limits) \& ((Y \neg (autopilot & limits))) | FTP)) \& (! (pullup)))) -> (O[<1+1] (FTP | (pullup)))))))
FRET Semantic Patterns

If autopilot & limits Autopilot shall after 1 step satisfy pullup

• [null, regular, after, satisfaction] pattern

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 (limits & autopilot) => pullup

• Pure PT: ((limits & autopilot) => pullup) S ((( limits & autopilot ) => pullup) & FTP)
Exporting Simulink Model Information

- Can be directly imported into FRET
Linking requirement variables to Simulink signals

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

<table>
<thead>
<tr>
<th>FRET Project</th>
<th>FRET Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM_requirements</td>
<td>FSM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Component</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fsm_12B</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FRET Variable</th>
<th>Variable Type</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>limits</td>
<td>Input</td>
<td></td>
</tr>
</tbody>
</table>

- **apfail**
- **limits**
- **standby**
- **supported**
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

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

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

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

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

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

CocoSpec

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 \((\text{limits} \& \text{autopilot}) \Rightarrow \text{pullup}\)

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

**Input variables**

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

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

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

Output variable

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) => (pulluppaypal)))
tel
Lustre & CoCoSpec

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

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

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

Internal variable

guarantee "FSM001": \[((\text{limits} \& \text{autopilot}) \Rightarrow \text{pullup}) \land \text{FTP}\), ((\text{limits} \& \text{autopilot}) \Rightarrow \text{pullup})\);
Lustre & CoCoSpec

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

Translated past time LTL formula

\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{verbatim}
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 (limits & autopilot) => pullup

(((limits & autopilot) => pullup) S (((limits & autopilot) => pullup) & FTP))
Generating Simulink Observers

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

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

<table>
<thead>
<tr>
<th>Number of blocks</th>
<th>Types of Blocks</th>
</tr>
</thead>
</table>
## Challenge Problem Complexity

### Number of blocks

| 7_autopilot | 1357 |

### Types of Blocks


**Transcendental functions**
## Challenge Problem Complexity

<table>
<thead>
<tr>
<th>Number of blocks</th>
<th>Types of Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>7autopilot</strong></td>
<td>1357</td>
</tr>
<tr>
<td><code>Abs</code>, <code>BusCreator</code>, <code>BusSelector</code>, <code>Concatenate</code>, <code>Constant</code>, <code>Data Type Conversion</code>, <code>Demux</code>, <code>Display</code>, <code>DotProduct</code>, <code>Fcn</code>, <code>From</code>, <code>Gain</code>, <code>Goto</code>, <code>Ground</code>, <code>Import</code>, <code>Inport</code>, <code>InportShadow</code>, <code>Logic</code>, <code>Lookup_nD</code>, <code>Math</code>, <code>MinMax</code>, <code>Mux</code>, <code>Outport</code>, <code>Product</code>, <code>RateLimiter</code>, <code>Relational Operator</code>, <code>Reshape</code>, <code>Rounding</code>, <code>Saturate</code>, <code>Scope</code>, <code>Selector</code>, <code>Signum</code>, <code>Sqrt</code>, <code>SubSystem</code>, <code>Sum</code>, <code>Switch</code>, <code> Terminator</code>, <code>Trigonometry</code>, <code>UnitDelay</code>, <code>CMBlock</code>, <code>Create 3x3 Matrix</code>, <code>Passive</code>, <code>Quaternion Modulus</code>, <code>Quaternion Norm</code>, <code>Quaternion Normalize</code>, <code>Rate Limiter Dynamic</code></td>
<td></td>
</tr>
</tbody>
</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>Total</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</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</th>
<th>SLDV</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>Total</td>
<td>66</td>
<td>62</td>
<td>57</td>
<td>23/21/13</td>
<td>19/8/27</td>
</tr>
</tbody>
</table>

Algebraic loop!
Abstraction of trigonometric, non-linear functions and allows local analysis

<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>66</td>
<td>62</td>
<td>57</td>
<td><strong>23/21/13</strong></td>
<td><strong>19/8/27</strong></td>
</tr>
</tbody>
</table>
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!