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

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

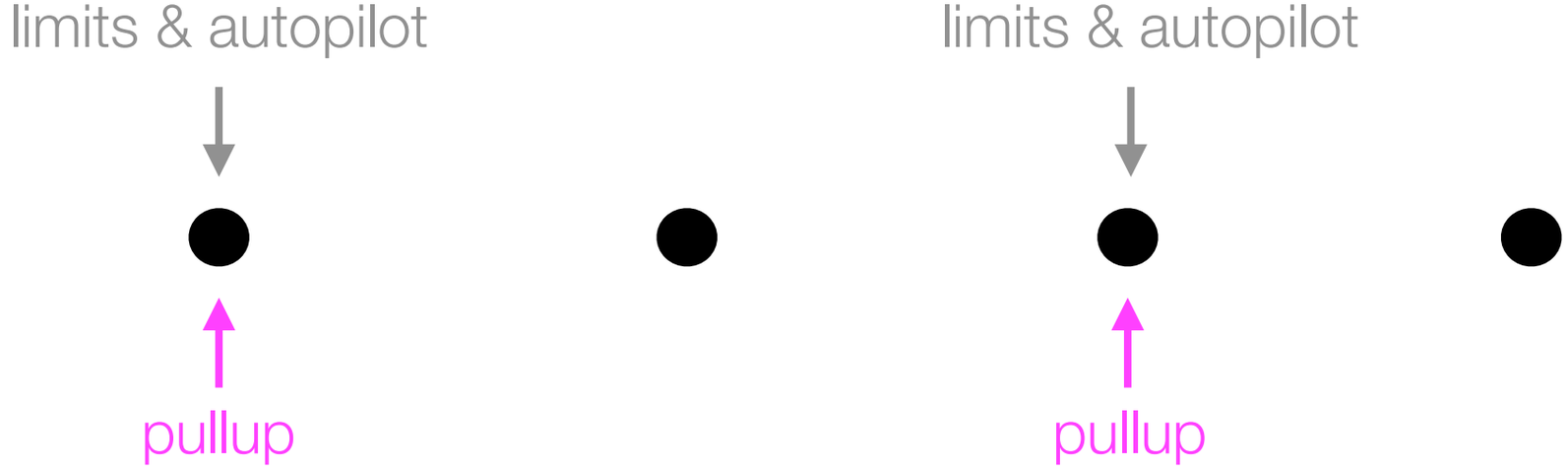
- Natural language requirement:

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

Autopilot Requirement Example

- Natural language requirement:

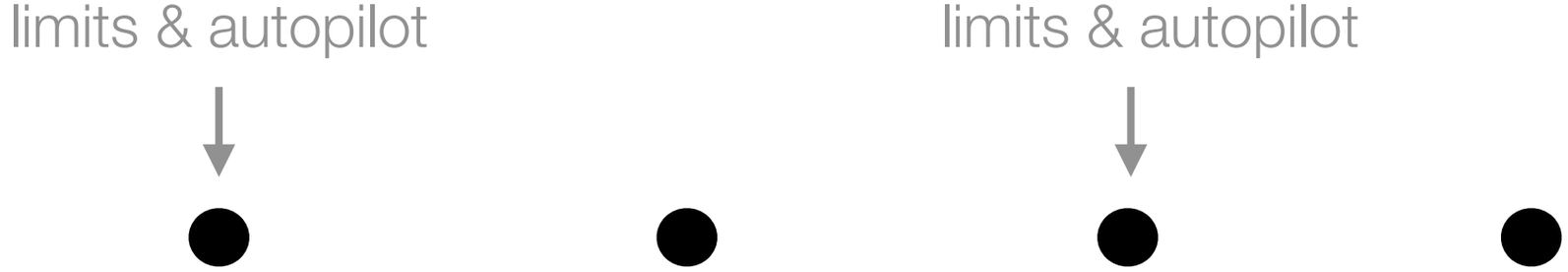
Exceeding sensor <u>limits</u> shall latch an autopilot <u>pullup</u> when the pilot is in <u>autopilot</u>.



Autopilot Requirement Example

- Natural language requirement:

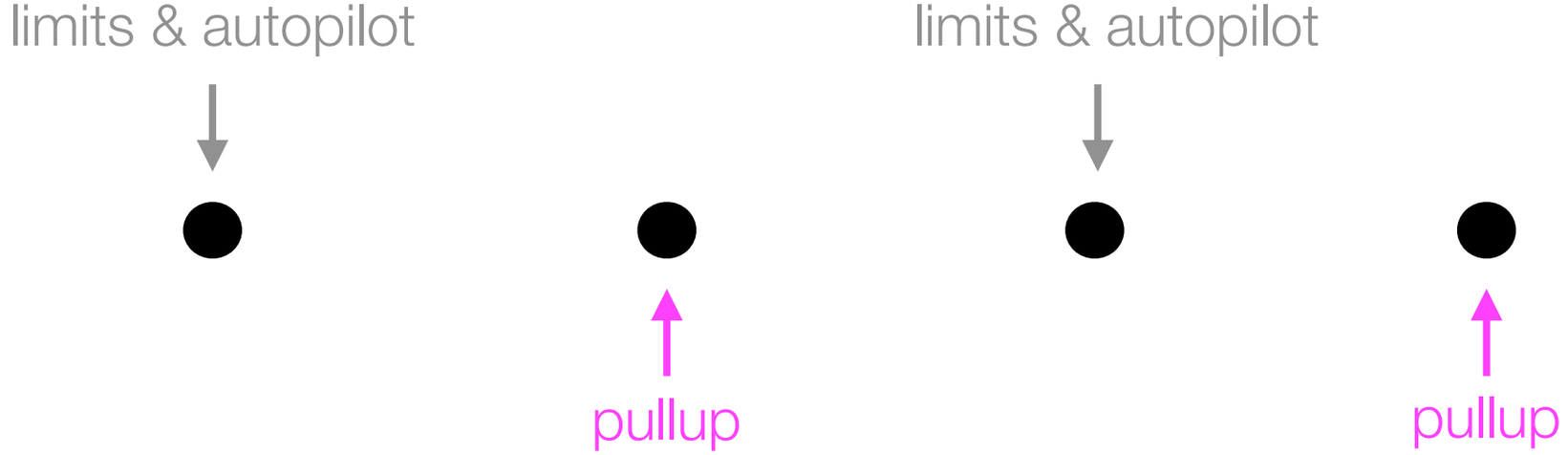
Exceeding sensor limits shall latch an autopilot pullup when the pilot is in autopilot.



Autopilot Requirement Example

- Natural language requirement:

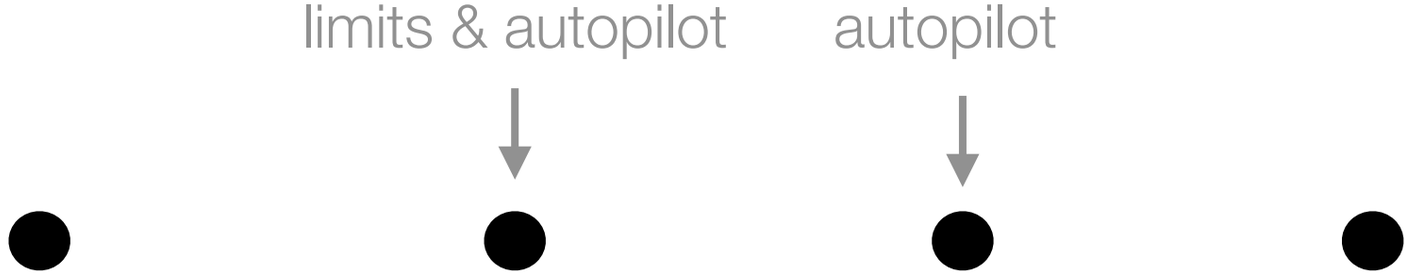
Exceeding sensor <u>limits</u> shall latch an autopilot <u>pullup</u> when the pilot is in <u>autopilot</u>.



Autopilot Requirement Example

- Natural language requirement:

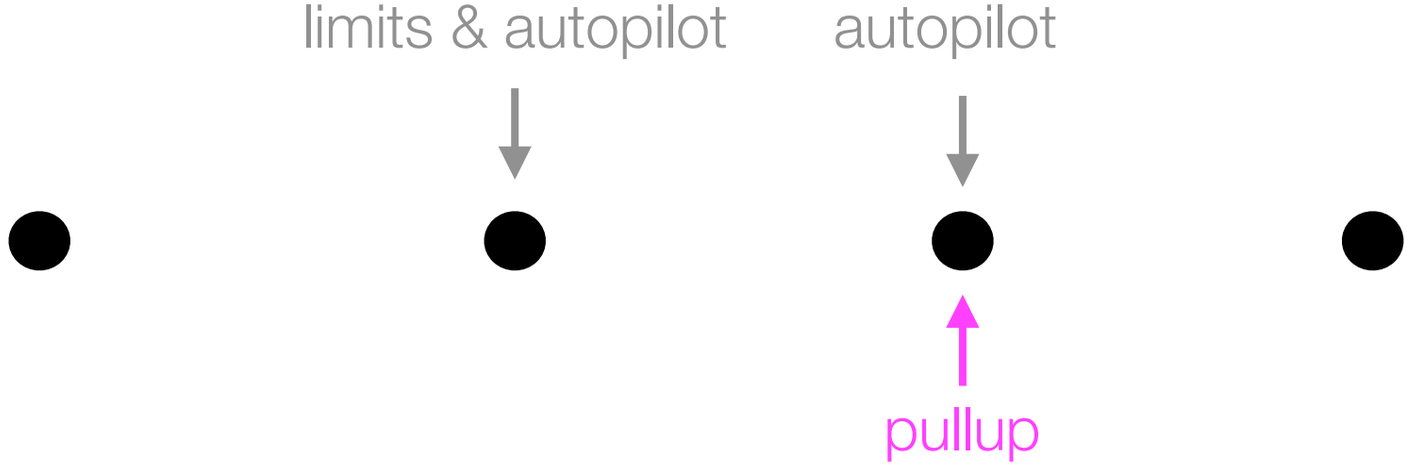
Exceeding sensor <u>limits</u> shall latch an autopilot <u>pullup</u> when the pilot is in <u>autopilot</u>.



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

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

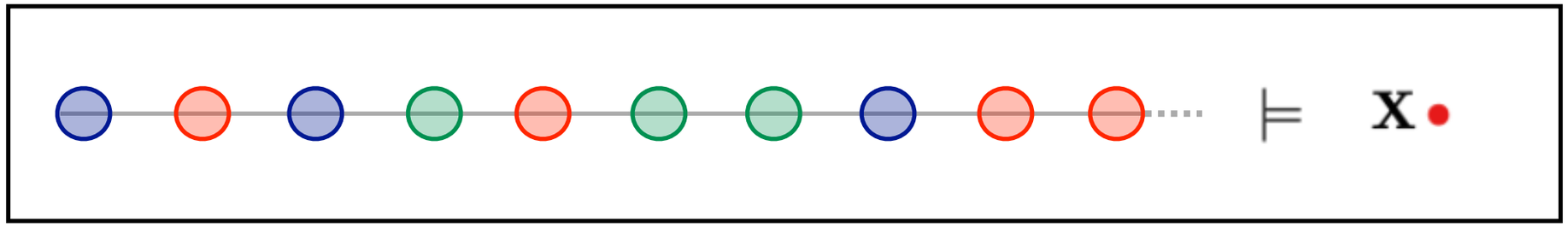
X (Next) refers to the next time step:

X ϕ is true iff ϕ holds at the next time step

Future time Operators

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

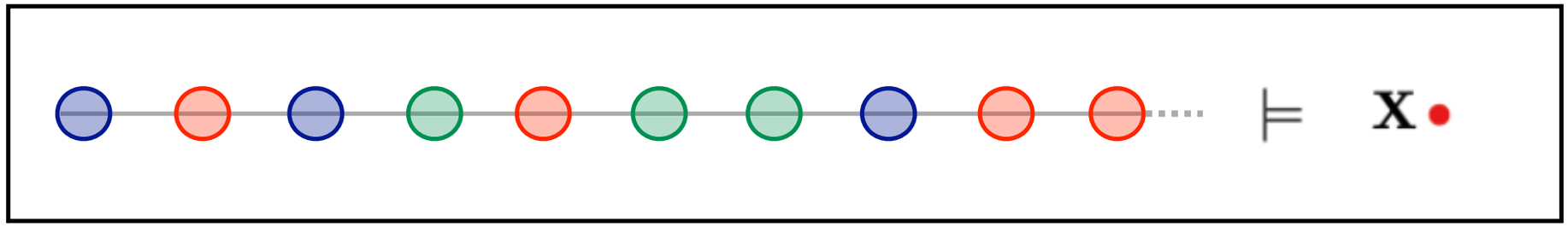
X ϕ is true iff ϕ holds at the next time step



Future time Operators

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

X ϕ is true iff ϕ holds at the next time step



Dual past time operator: **Y** (Yesterday)

Future time Operators

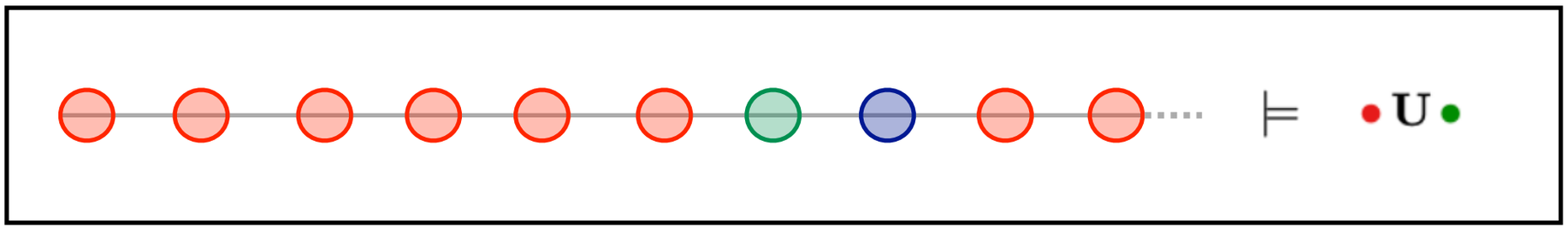
U (Until) refers to multiple time steps:

ϕ **U** ψ is true iff ψ holds at 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 (Until): refers to multiple time steps

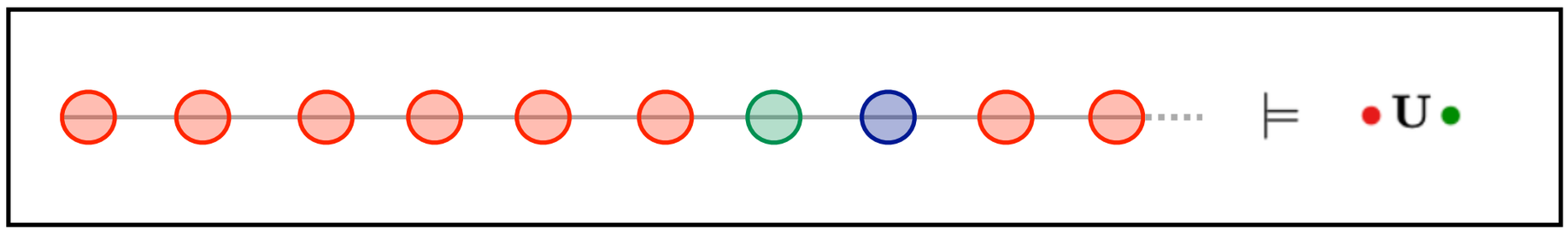
ϕ **U** ψ is true iff ψ holds at holds at some time step t in the future and for all time steps t’ (such that t’ < t) ϕ is true.



Future time Operators

U (Until): refers to multiple time steps

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



Dual past time operator: **S** (Since)

Future time Operators

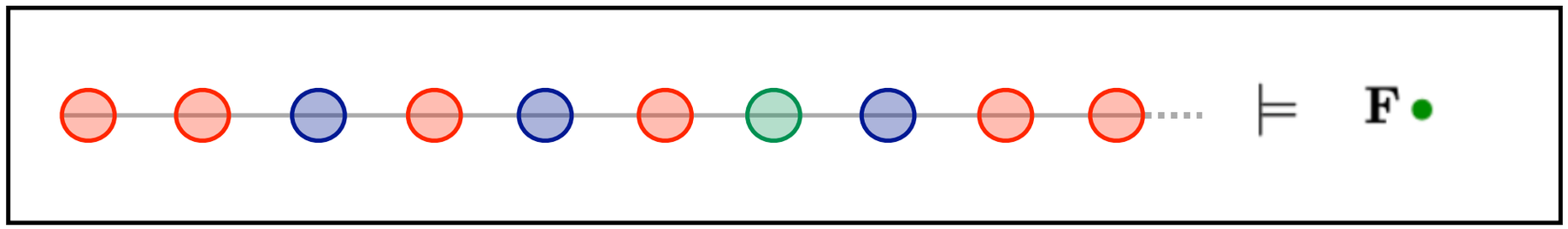
F (eventually): refers to at least one time step in the future:

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:

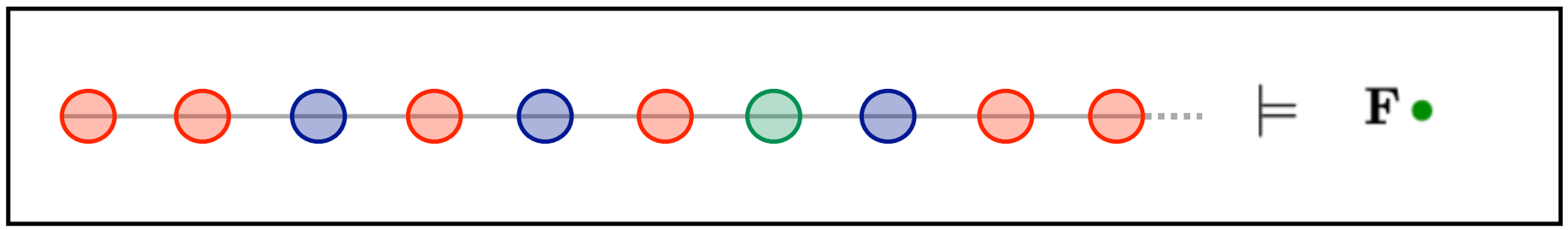
F ϕ is true iff ϕ is true at some future time point including the present time



Future time Operators

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

F ϕ is true iff ϕ is true at some future time point including the present time



Dual past time operator: **O** (Once)

Future time Operators

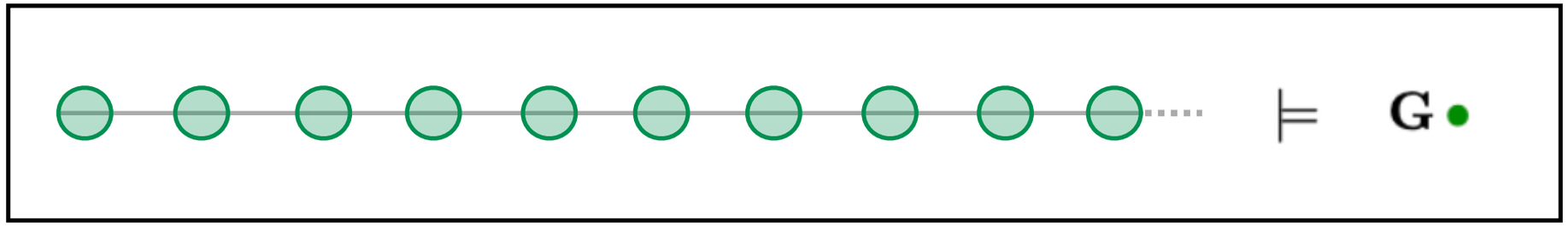
G (Globally): refers to all future steps of an execution

G ϕ is true iff ϕ is always true in the future

Future time Operators

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

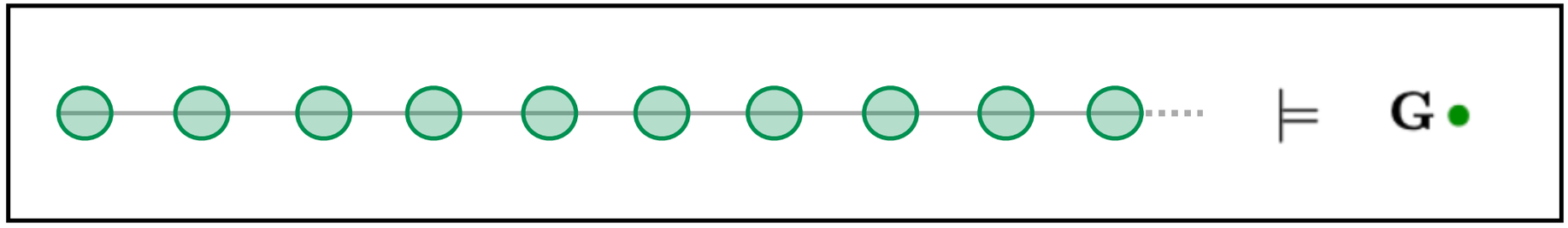
G ϕ is true iff ϕ is always true in the future



Future time Operators

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

G ϕ is true iff ϕ 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:
  [scope,condition,timing,response]

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

  - [null, null, always] pattern

- **Pure FT**: G ((limits & autopilot) => pullup)
- **Pure PT**: H ((limits & autopilot) => pullup)

FRET Semantic Patterns

If autopilot & limits Autopilot shall after 1 step satisfy pullup

- [null, regular, after, satisfaction] pattern

- **Pure PT:** ((H (((((! FTP) S ((autopilot & limits) & ((Y (! (autopilot & limits))) | FTP))) & (O[<=1] ((autopilot & limits)& ((Y (! (autopilot & limits))) | FTP)))) -> (! (pullup))) & (((autopilot & limits) & FTP) -> (! (pullup))))) & (H ((O[=1+1] (((autopilot & limits) & ((Y (! (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

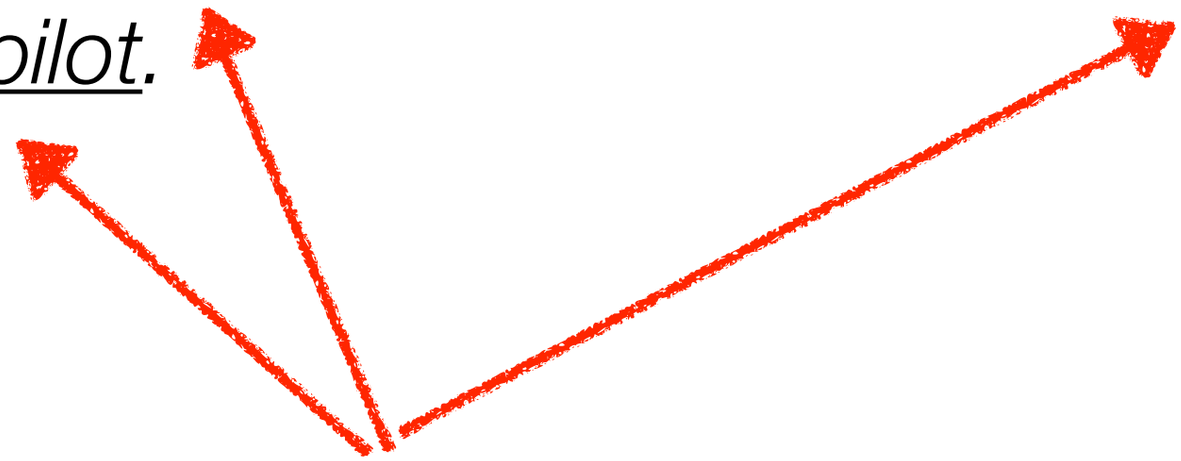
- **Pure PT:** ((H (((((! FTP) S ((autopilot & limits) & ((Y (! (autopilot & limits))) | FTP))) & (O[<=1] ((autopilot & limits)& ((Y (! (autopilot & limits))) | FTP)))) -> (! (pullup))) & (((autopilot & limits) & FTP) -> (! (pullup))))) & (H ((O[=2] (((autopilot & limits) & ((Y (! (autopilot & limits))) | FTP)) & (! (pullup)))) -> (O[<2] (FTP | (pullup))))))

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.

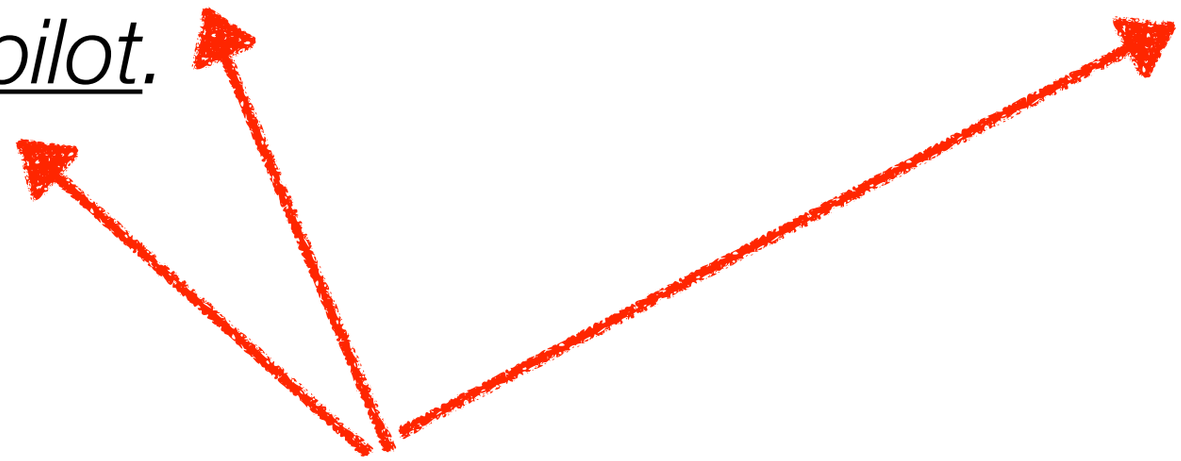


Atomic propositions in generated formula.

Finite State Machine Requirement

- Natural language requirement:

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

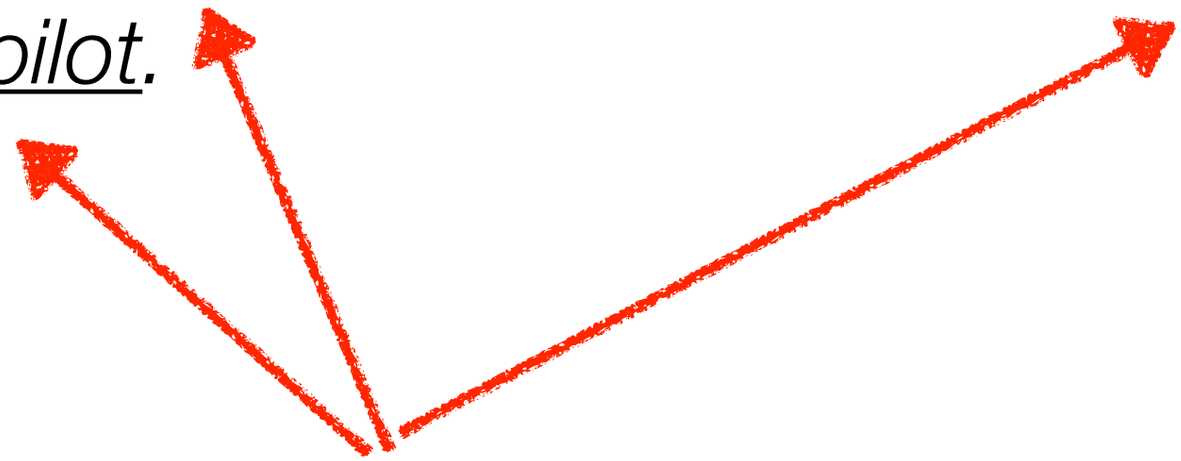


Atomic propositions in generated formula.
Meaningless when it comes to the model!

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)

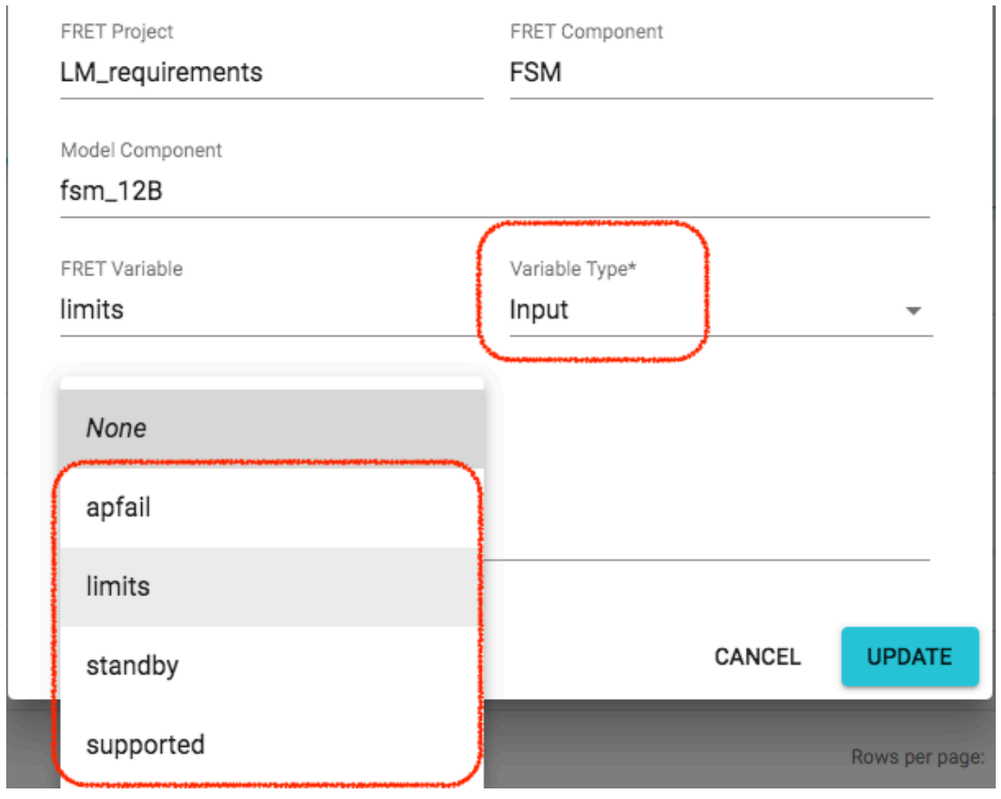
Exporting Simulink Model Information

- Can be directly imported into FRET


```
{
  "id": "fsm_12B/limits",
  "variable_name": "limits",
  "portType": "Inport",
  "component_name": "fsm_12B",
  "dataType": [
    "boolean"
  ],
  "dimensions": [
    1, 1
  ],
  "width": 1
},
```

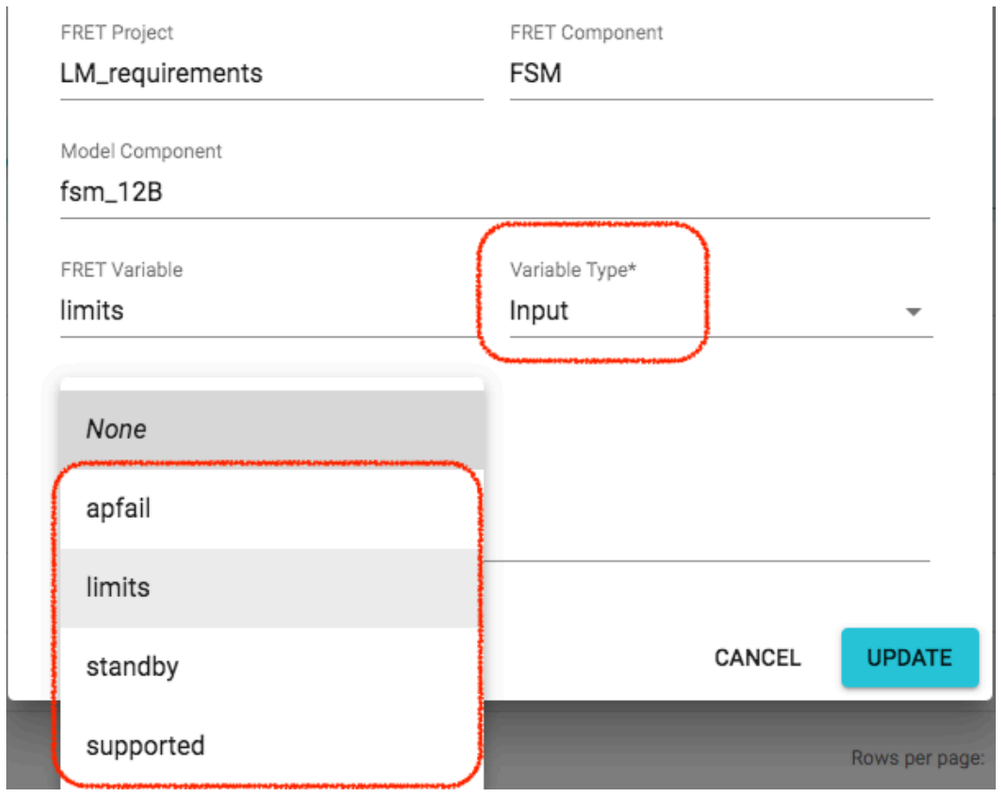
Linking requirement variables to Simulink signals

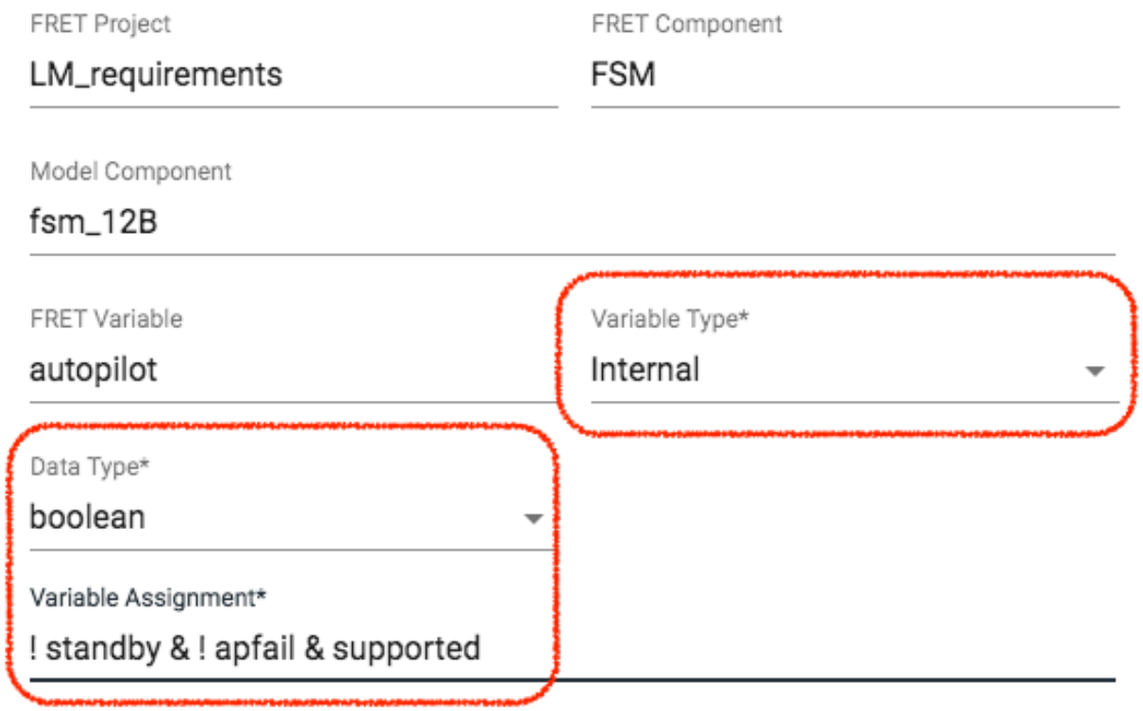
- FSM shall always satisfy (limits & autopilot) => pullup



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
- 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)

```
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)

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 (limits & autopilot) => pullup

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

Input variables

```
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

```
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)

```
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
```

Internal variable

Lustre & CoCoSpec

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

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

Translated past time LTL formula

```
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

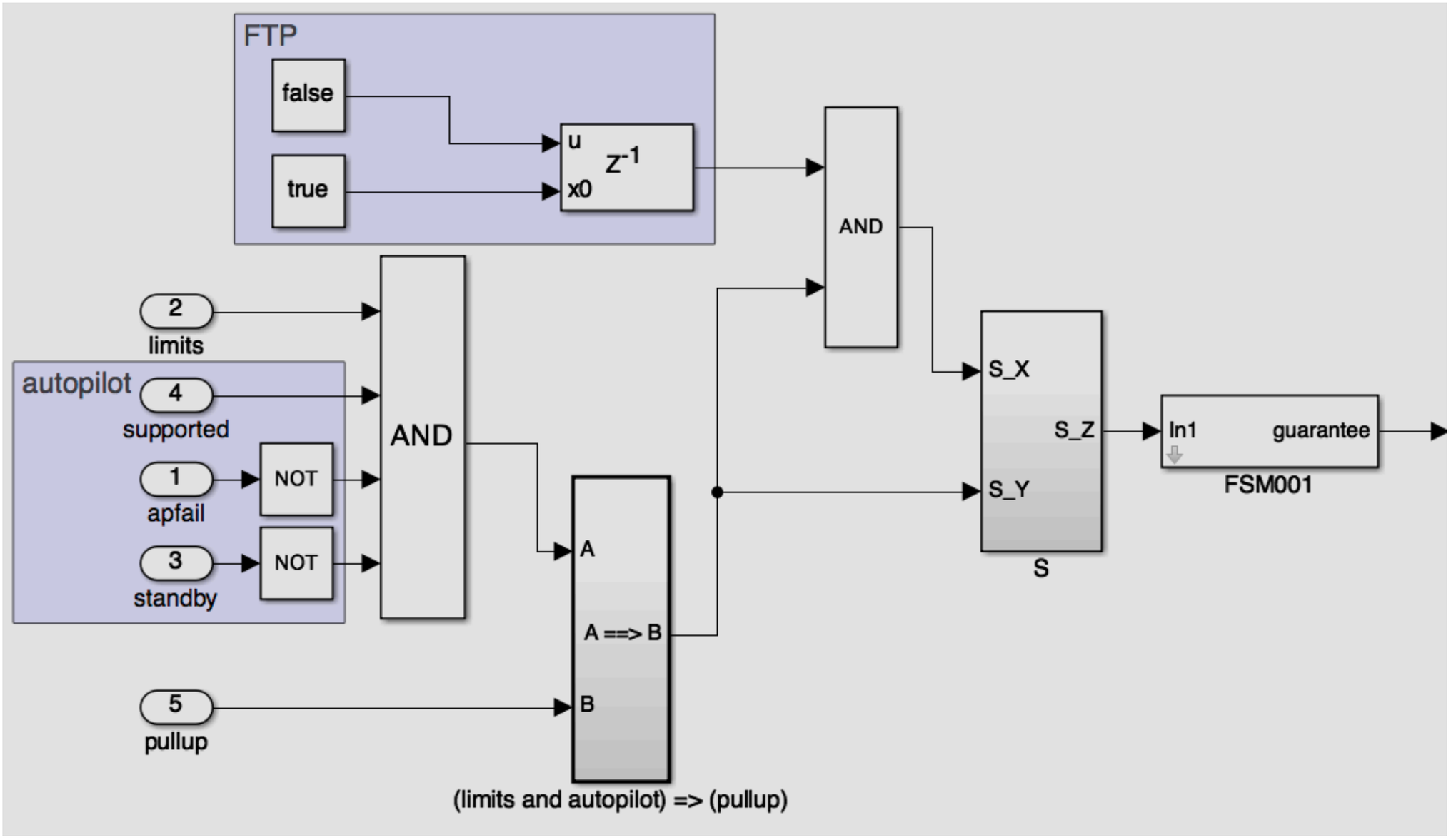
```
--Historically
node H(X:bool) returns (Y:bool);
let
     Y = X -> (X and (pre Y));
tel
```

```
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

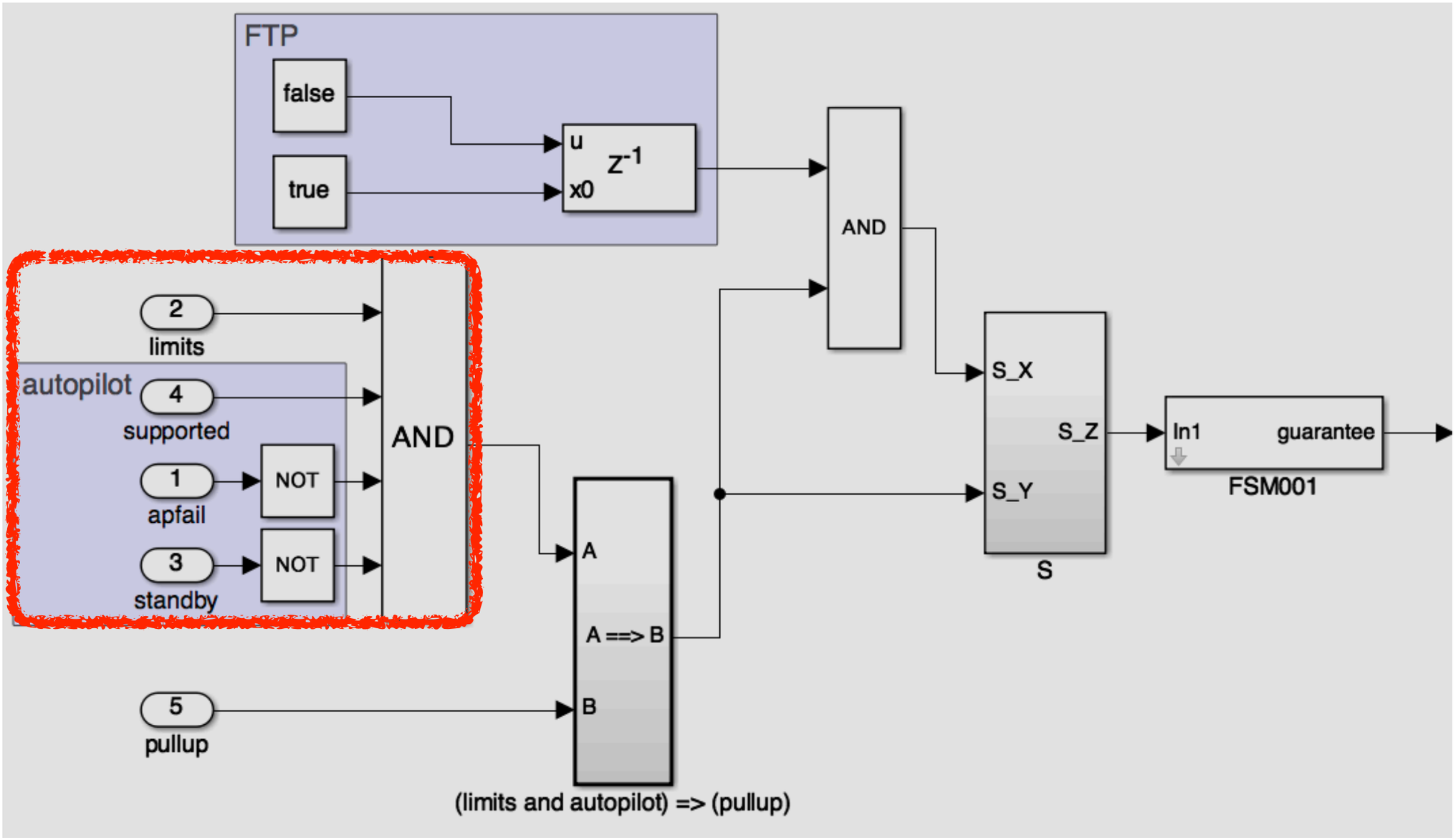
((limits & autopilot) => pullup) S (((limits & autopilot) => pullup) & FTP)



Generating Simulink Observers

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

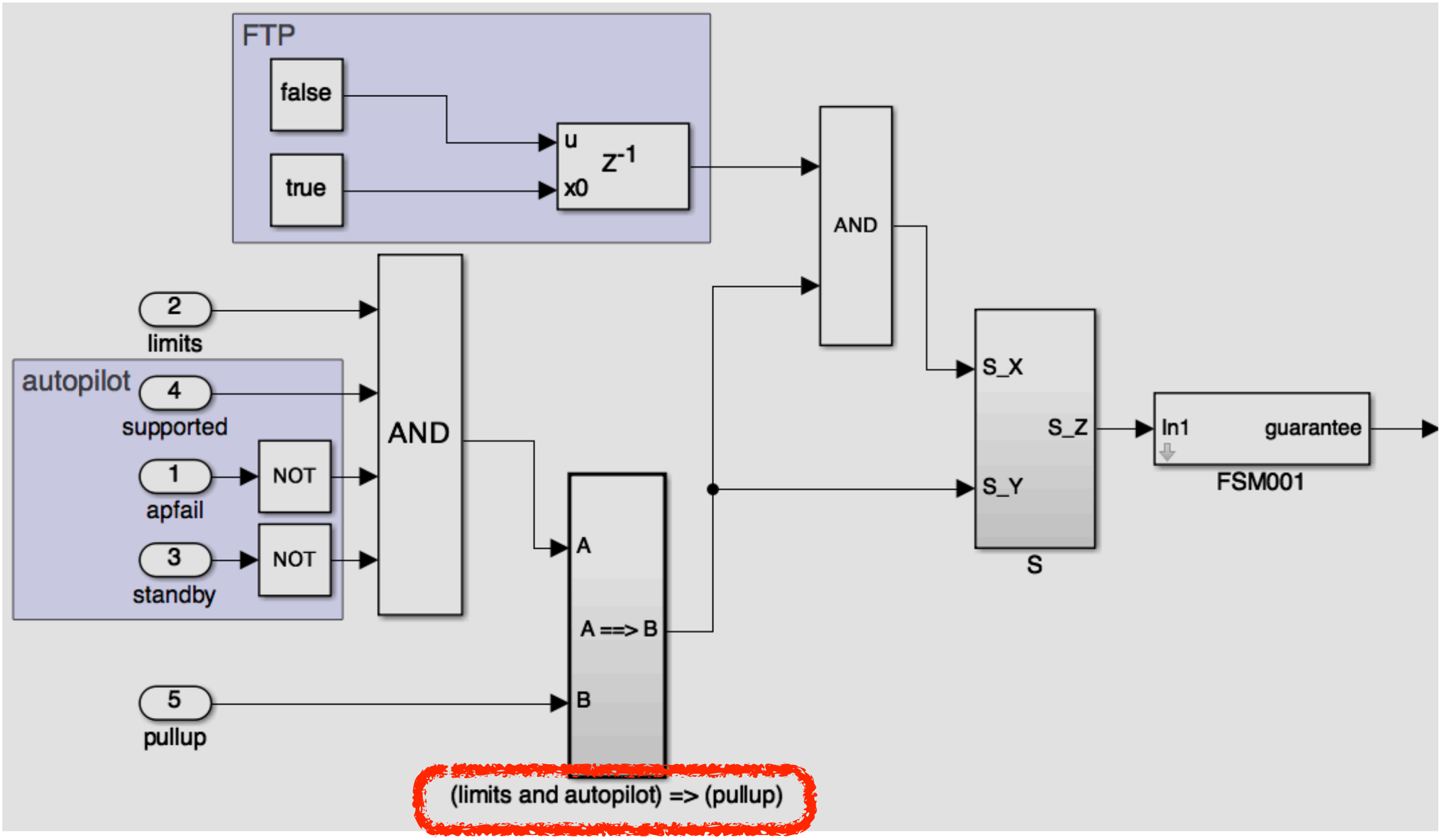
((limits & autopilot) => pullup) S (((limits & autopilot) => pullup) & FTP)



Generating Simulink Observers

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

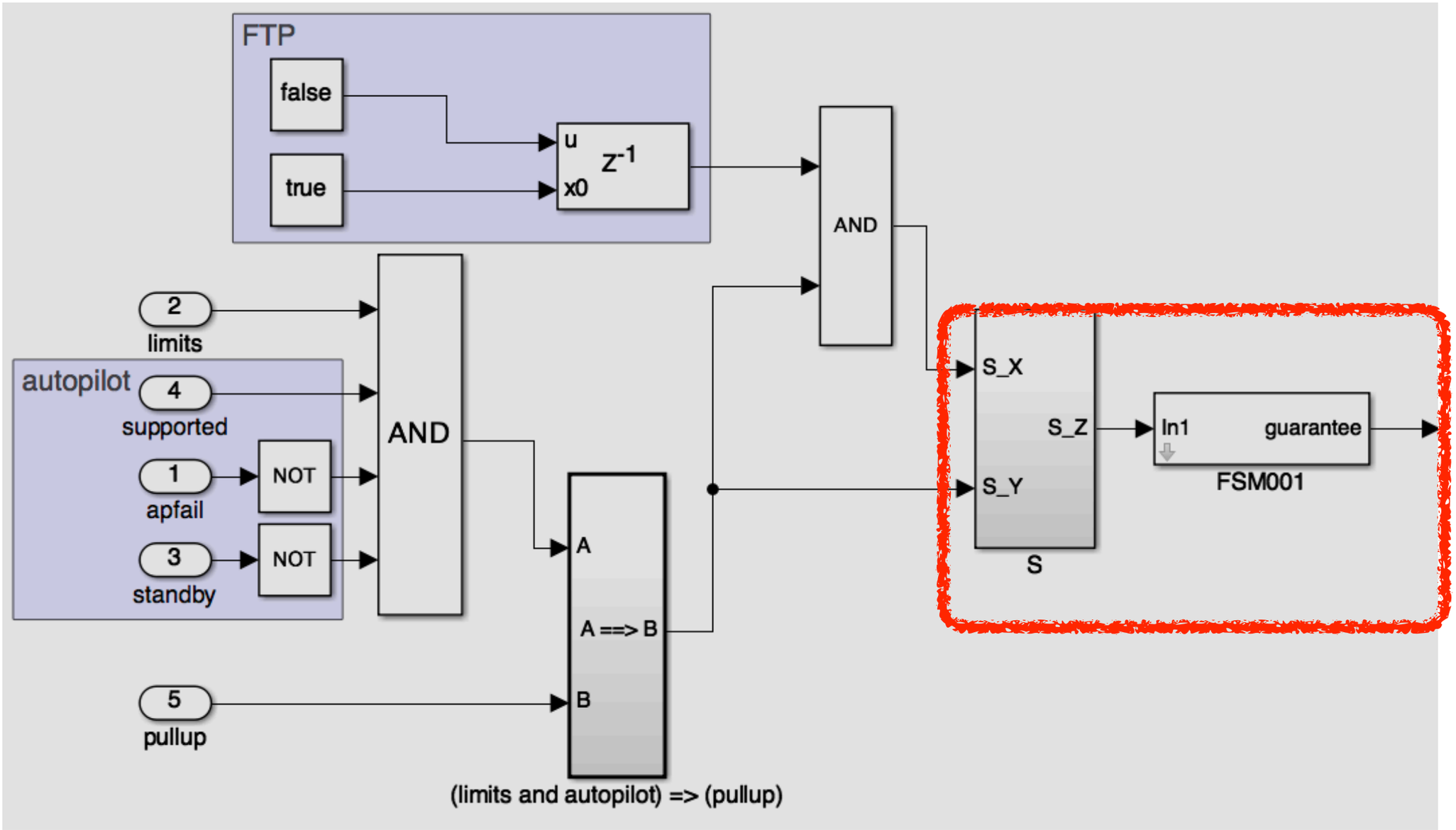
((limits & autopilot) => pullup) S (((limits & autopilot) => pullup) & FTP)



Generating Simulink Observers

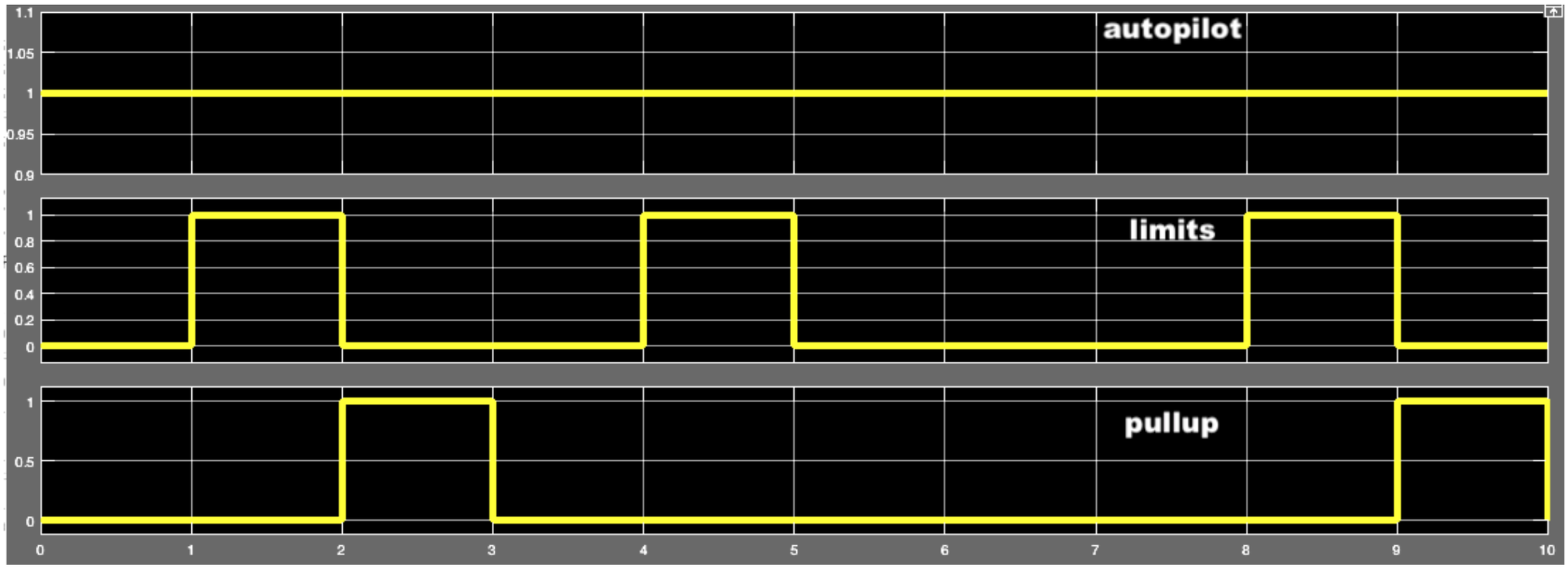
Autopilot shall always satisfy (limits & autopilot) => pullup

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



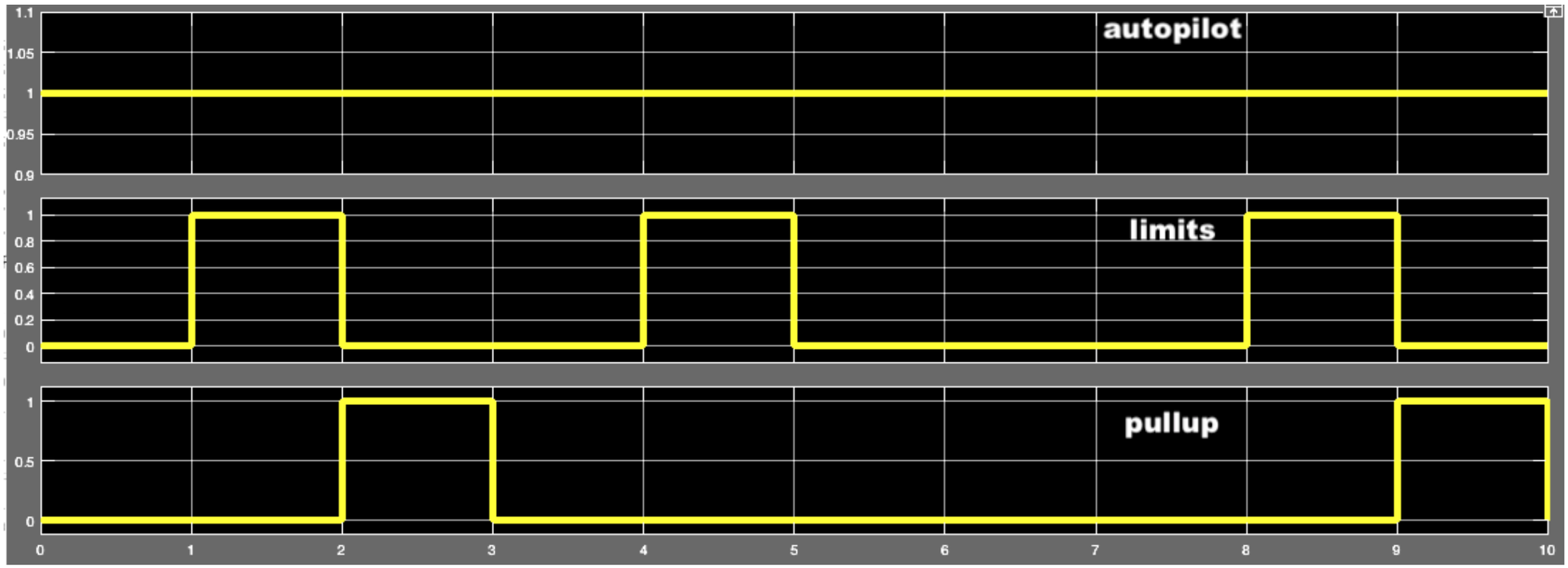
Tracing Counterexamples

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

Types of Blocks

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', 'CMBlock', 'Create 3x3 Matrix', 'Passive', 'Quaternion Modulus', 'Quaternion Norm', 'Quaternion Normalize', 'Rate Limiter Dynamic'

Challenge Problem Complexity

Number of blocks

Types of Blocks

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', 'CMBlock', 'Create 3x3 Matrix', 'Passive', 'Quaternion Modulus', 'Quaternion Norm', 'Quaternion Normalize', 'Rate Limiter Dynamic'

Transcendental functions

Challenge Problem Complexity

Number of blocks

Types of Blocks

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'**, **'CMBlock'**, **'Create 3x3 Matrix'**, **'Passive'**, **'Quaternion Modulus'**, **'Quaternion Norm'**, **'Quaternion Normalize'**, 'Rate Limiter Dynamic'

Nonlinearities & Discontinuous math

Challenge Problem Analysis Results

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

Challenge Problem Analysis Results

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

Challenge Problem Analysis Results

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

Challenge Problem Analysis Results

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

Algebraic loop!

Challenge Problem Analysis Results

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

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!