

INTELLIGENT  
SYSTEMS  
DIVISION



# FRET Tutorial

Formal Requirements Elicitation Tool

*Presented by*

Tom Pressburger

May 02, 2022

# Lockheed Martin Cyber-Physical System Challenge, component FSM

The 10 Cyber-Physical V&V Challenges were created by Lockheed Martin Aeronautics to evaluate and improve the state-of-the-art in formal method toolsets. Each challenge problem includes:

- a high-level description

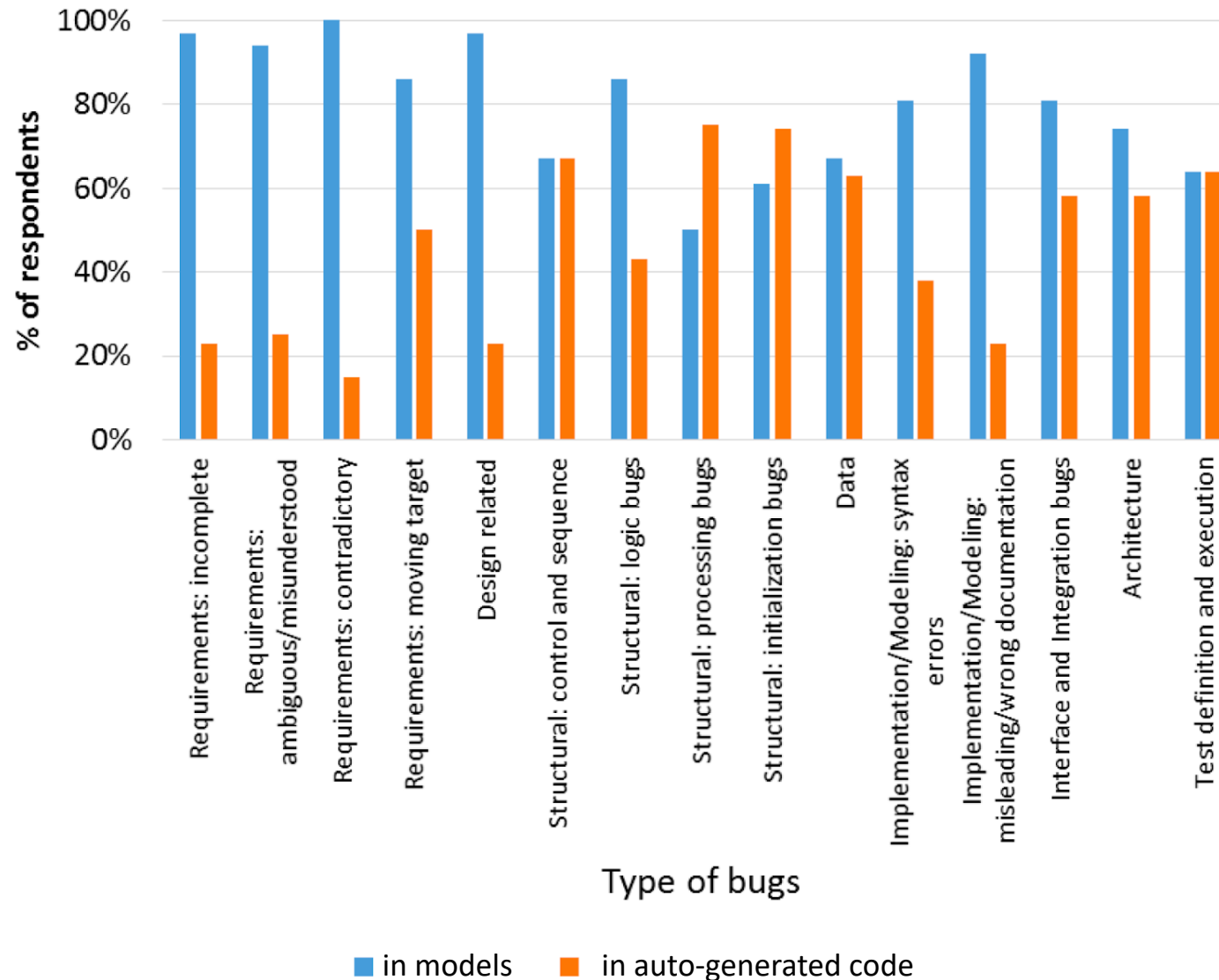
- a set of requirements written in plain English;

- a Simulink model;

- a set of parameters (in .mat format) for simulating the model.

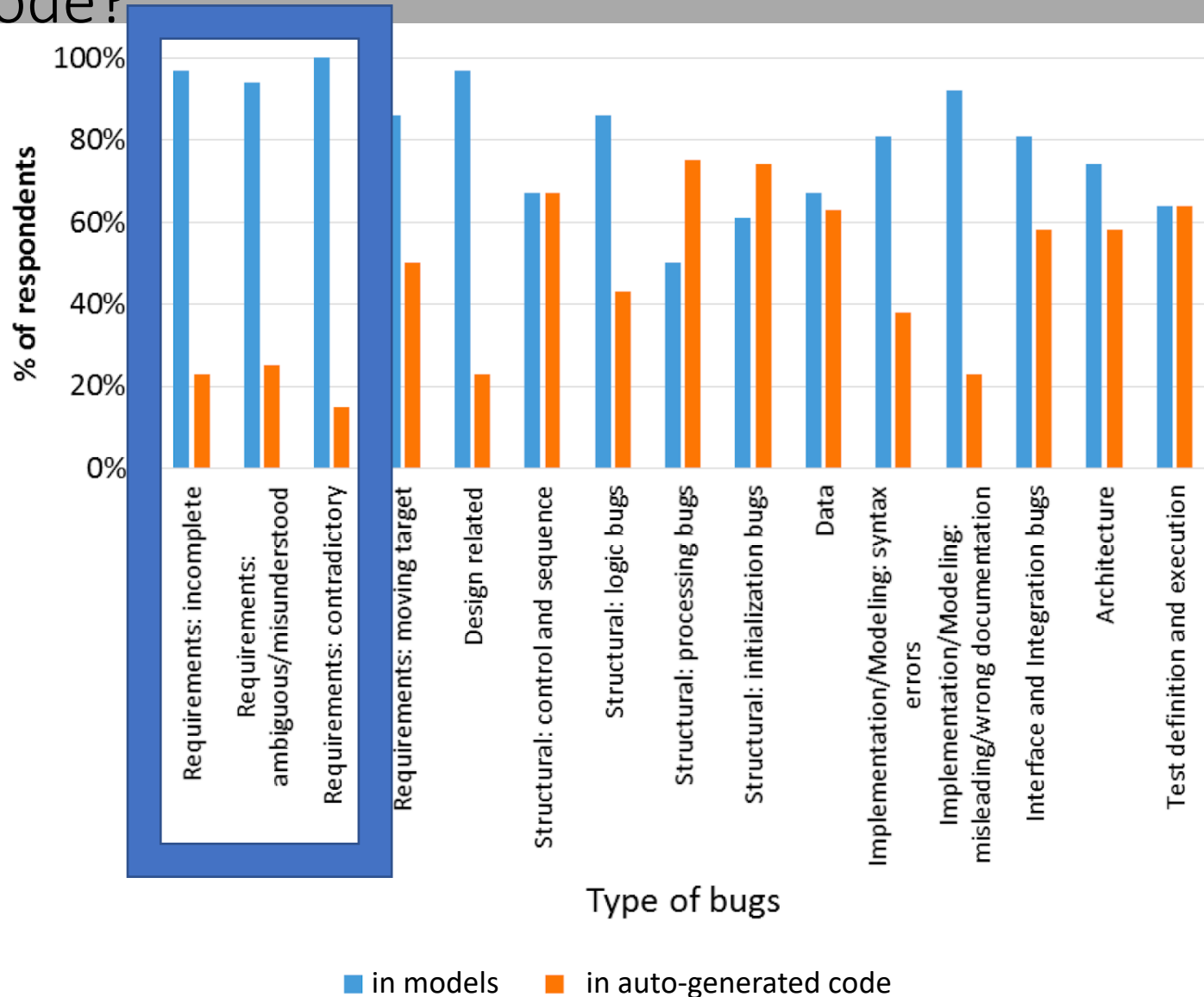
FSM: represents an abstraction of an advanced autopilot system responsible for commanding a safety maneuver in the event of a hazard.

# What types of bugs are found in models and code?



Johann Schumann, Matt Knudsen, Teme Kahsai, Noble Nkwocha, Katerina Goseva-Popstojanova, Thomas Kyanko, "Report: Survey on Model-Based Software Engineering and Auto-Generated Code", NASA/TM-2016-219443, 2016.

# What types of bugs are found in models and code?



Johann Schumann, Matt Knudsen, Teme Kahsai, Noble Nkwocha, Katerina Goseva-Popstojanova, Thomas Kyanko, "Report: Survey on Model-Based Software Engineering and Auto-Generated Code", NASA/TM-2016-219443, 2016.

# language of developers forced to write reqs

- 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).
- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data is not good.
- The autopilot shall change states from NOMINAL to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from MANEUVER to STANDBY when the pilot is in control (standby) and sensor data is good.
- ...

# language of developers forced to write reqs

## Lockheed Martin Cyber-Physical System Challenge, component FSM:

- 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 a fail).

At every timepoint where these conditions hold or only when they **become** true?

- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data is not good.
- The autopilot shall change states from NOMINAL to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from MANEUVER to STANDBY when the pilot is in control (standby) and sensor data is good.
- ...

# language of developers forced to write reqs

## Lockheed Martin Cyber-Physical System Challenge, component FSM:

- 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 a fail).

At every timepoint these conditions hold or only when they **become** true?

- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data is not good.
- The autopilot shall change states from NOMINAL to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from MANEUVER to STANDBY when the pilot is in control (standby) and sensor data is good.
- ...

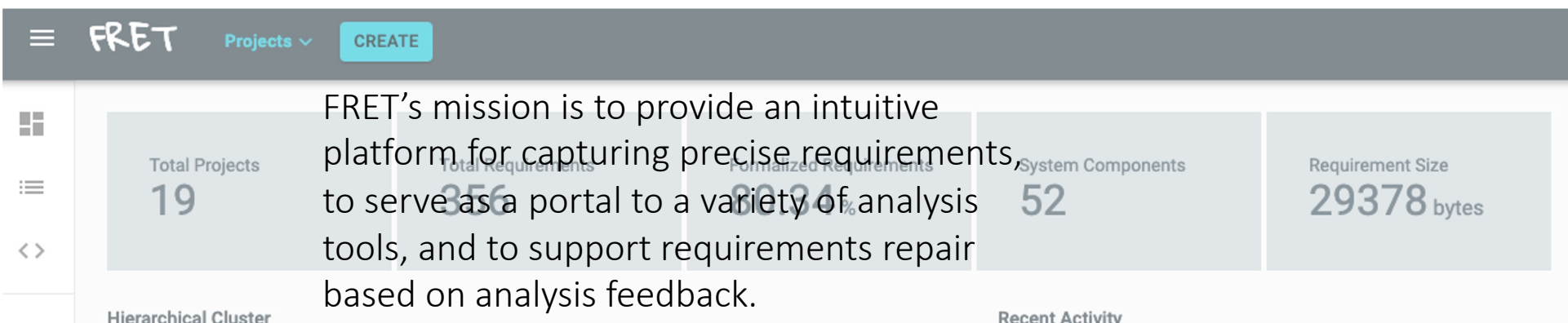
Are the requirements consistent?

Does my model/code satisfy the requirements?

# language formal analysis tools understand

```
var autopilot: bool = (not standby) and supported and (not
  apfail);
var pre_autopilot: bool = false -> pre autopilot;
var pre_limits: bool = = false -> pre limits;
guarantee "FSM-001v2" S((((autopilot and pre_autopilot and
  pre_limits) and (pre (not (autopilot and pre_autopilot and
  pre_limits)))) or ((autopilot and pre_autopilot and
  pre_limits) and FTP)) => (pullup)) and FTP), (((autopilot
  and pre_autopilot and pre_limits) and (pre (not (autopilot
  and pre_autopilot and pre_limits)))) or ((autopilot and
  pre_autopilot and pre_limits) and FTP)) => (pullup));
```





FRET's mission is to provide an intuitive platform for capturing precise requirements, to serve as a portal to a variety of analysis tools, and to support requirements repair based on analysis feedback.

**Welcome to FRET**  
<https://github.com/NASA-SW-VnV/fret>



[anastasia.mavridou@nasa.gov](mailto:anastasia.mavridou@nasa.gov)  
[andreas.katis@nasa.gov](mailto:andreas.katis@nasa.gov)  
[tom.pressburger@nasa.gov](mailto:tom.pressburger@nasa.gov)

**Team (ARC):** Andreas Katis, Anastasia Mavridou, Tom Pressburger, Johann Schumann, Khanh Trinh

**Alumni:** David Bushnell, Tanja DeJong, Dimitra Giannakopoulou, George Karamanolis, David Kooi, Julian Rhein, Nija Shi

**Collaborators (LaRC):** Swee Balanchandran, Esther Conrad, Aaron Dutle, Alwyn Goodloe, Ivan Perez, Laura Titolo

# FRET bridges the gap

- **Captures** requirements in a restricted natural language with **unambiguous semantics**
- **Explains** formal **semantics** in various forms: natural language, diagrams, interactive simulation
- **Assists** in writing requirements through requirement **templates**
- **Formalizes** requirements in a **compositional** (hence maintainable and extensible) manner
- **Checks consistency** of requirements and provides feedback
- **Connects** with **analysis tools** and **exports verification code**
  - ✓ for model checking Simulink models with CoCoSim
  - ✓ for model checking Lustre code with Kind2
  - ✓ for efficient runtime monitoring with Copilot

# FRET bridges the gap

- **Captures** requirements in a restricted natural language with **unambiguous semantics**: *FRETish*
- **Explains** formal **semantics** in various forms: natural language, diagrams, interactive simulation
- **Assists** in writing requirements through requirement **templates**
- **Formalizes** requirements in a **compositional** (hence maintainable and extensible) manner
- **Checks consistency** of requirements and provides feedback
- **Connects** with **analysis tools** and **exports verification code**
  - ✓ for model checking Simulink models with CoCoSim
  - ✓ for model checking Lustre code with Kind2
  - ✓ for efficient runtime monitoring with Copilot

# capturing requirements in FRETish

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected

*scope* if altitude\_hold\_selected *condition* the altitude\_hold\_autopilot *component* shall *always* *timing* satisfy *response* maintain\_altitude

# capturing requirements in FRETish

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected

if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

scope                      condition                      component                      timing                      response



Q: Upon which part of the system is the requirement being levied?  
A: The altitude\_hold\_autopilot.




# capturing requirements in FRETish

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected

if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

scope                      condition                      component                      timing                      response

Q: What do we want the system to achieve?  
A: maintain\_altitude



# capturing requirements in FRETish

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected

if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

The diagram shows the requirement sentence "if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude" with colored boxes highlighting different parts. Below each box is a handwritten label: "scope" (red box), "condition" (yellow box), "component" (green box), "timing" (blue box), and "response" (light green box). A blue arrow points from the "scope" label to the question below.

Q: During what portion of the execution is the requirement enforced?

A: It has been omitted, meaning *globally, always*

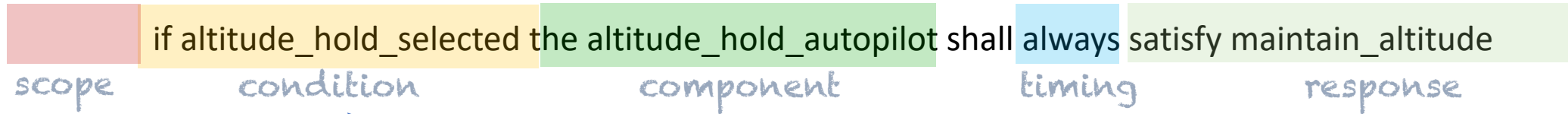
*Note: the portion can be expressed relative to a system mode.*

# capturing requirements in FRETish

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected

if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

scope                      condition                      component                      timing                      response



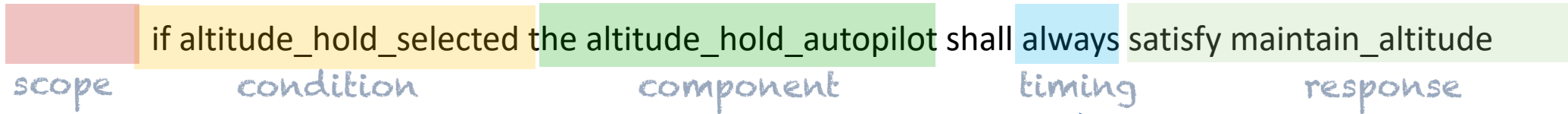
Q: What condition triggers the response?

A: altitude\_hold\_selected becoming true, within the scope



# capturing requirements in FRETish

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected



Q: Where does the response happen, relative to the scope and trigger?

A: always, meaning *thenceforth*

# capturing requirements in FRETish

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected

if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

scope                      condition                      component                      timing                      response

**SCOPE** in, before, after, notin, onlyIn, onlyBefore, onlyAfter; when omitted, global

**CONDITION** null, regular

**TIMING** immediately, next, always, never, eventually, until, before, for, within, after

**RESPONSE** satisfaction

# capturing requirements in FRETish

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected

if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

scope                      condition                      component                      timing                      response

**SCOPE** in, before, after, notin, onlyIn, onlyBefore, onlyAfter; when omitted, global

**CONDITION** null, regular

**TIMING** immediately, next, always, never, eventually, until, before, for, within, after

**RESPONSE** satisfaction

8 \* 2 \* 10 = 160 semantic templates /  
template keys!

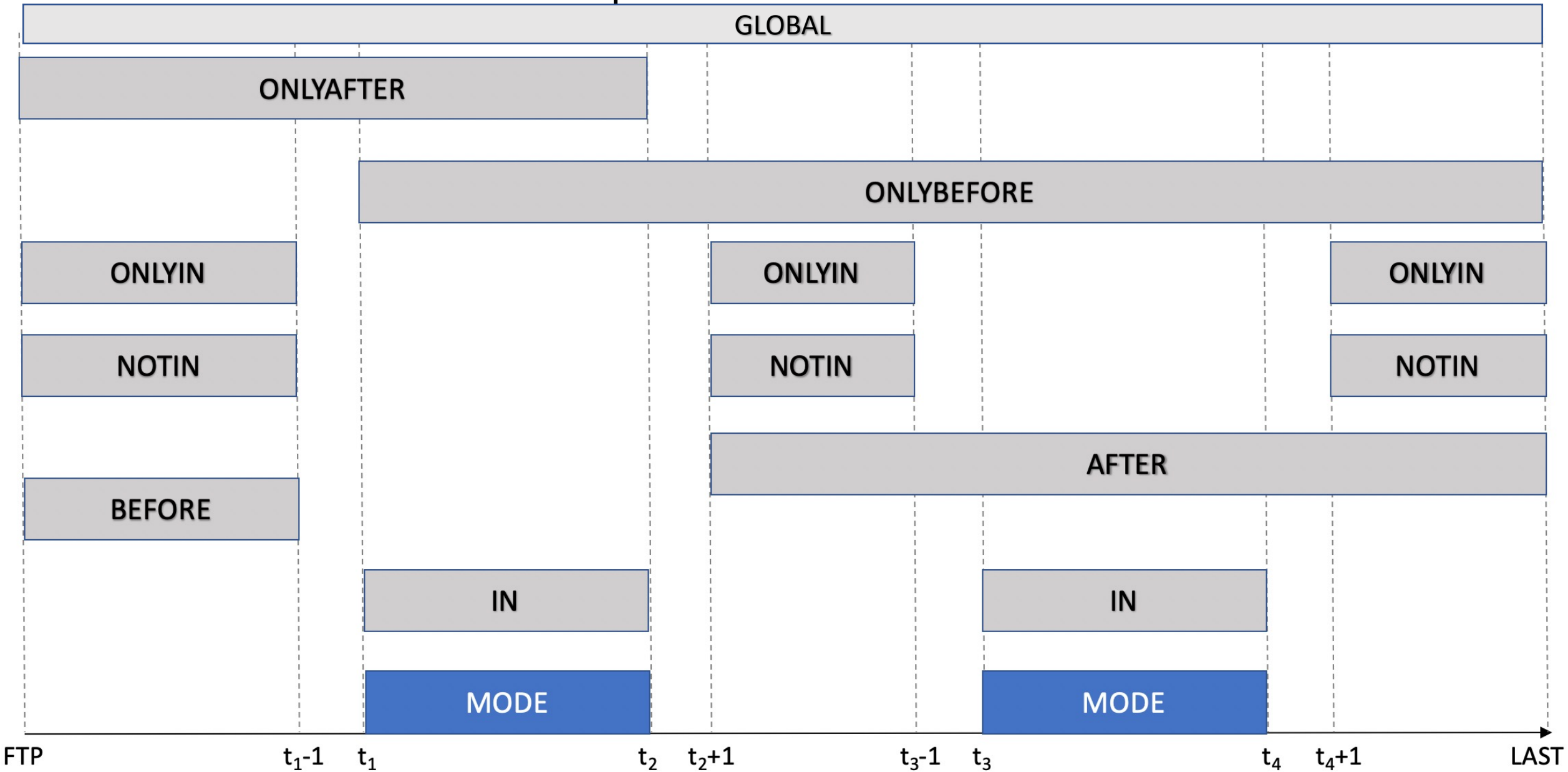
# Expressions

- Boolean
  - $!, \&, |, \Rightarrow, \text{if\_then\_}, \Leftrightarrow, p(x,y,z)$
  - $\text{preBool}(init,p)$ ,
  - $\text{persisted}(n,p), \text{occurred}(n,p)$
  - $\text{persists}(n,p), \text{occurs}(n,p)$
- Arithmetic
  - $=, \neq, <, >, \leq, \geq$
  - $+, -, *, /, ^, f(x,y)$
  - $\text{preInt}(init,n), \text{preReal}(init,x)$

# Scope condition component timing response

- **(global)** The system shall always satisfy `count >= 0`
- **In** landing mode the system shall eventually satisfy `decrease_speed`
- **Before** energized mode the system shall always satisfy `energized_indicator_off`
- **After** boot mode the system shall immediately satisfy `prompt_for_password`
- When **not in** initialization mode the system shall always satisfy `commands_accepted`
- **Only in** landing mode shall the system eventually satisfy `landing_gear_down`
- **Only before** energized mode shall the system eventually satisfy `manually_touchable`
- **Only after** arming mode shall the system eventually satisfy `fired`

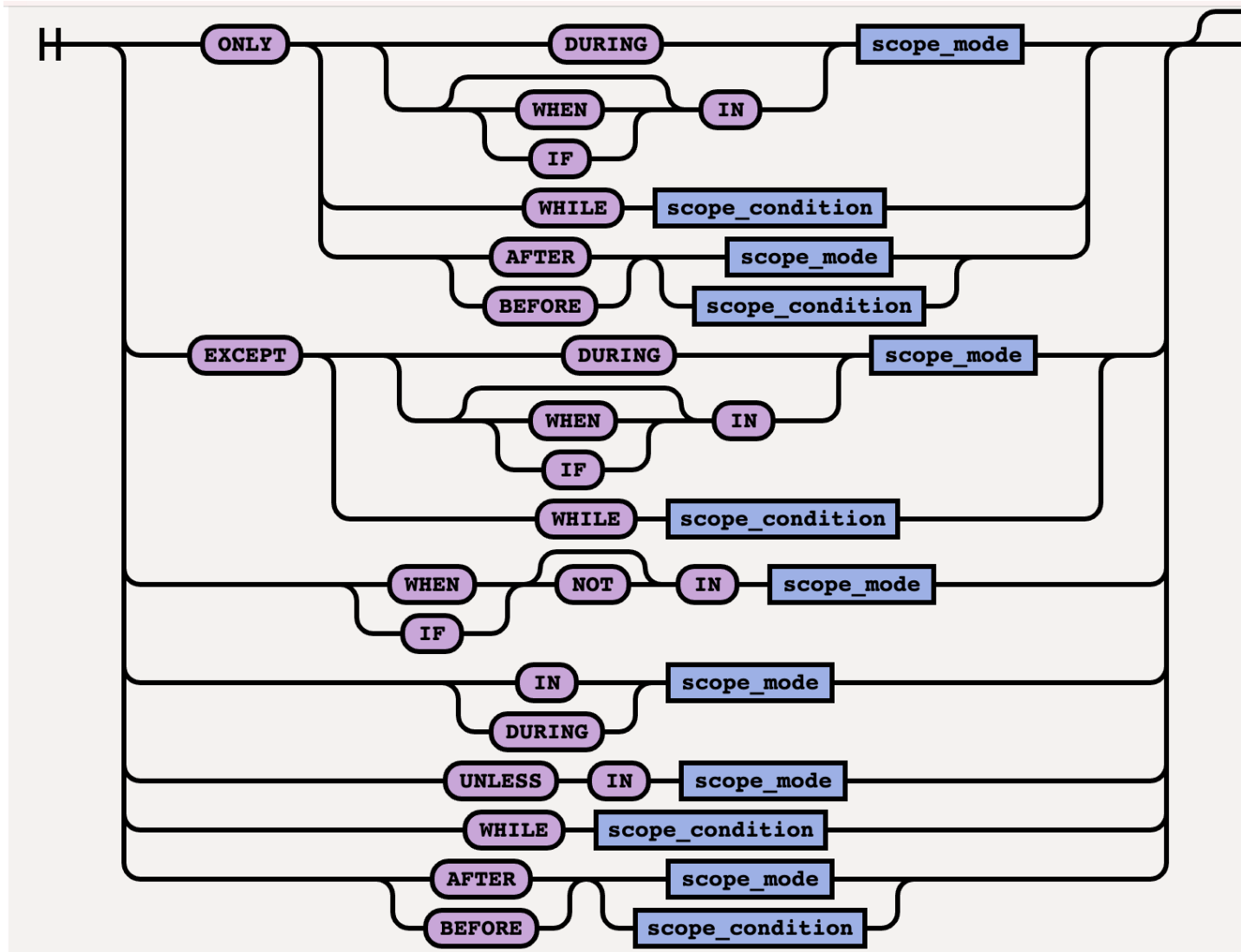
# Scope Intervals



## Scope (contd)

- **While** mode = 4 the watch shall always satisfy alarm\_icon\_on
- **While** persisted(4,high\_temperature) the monitor shall until shutoff satisfy alarm\_on
- **Before** taxiing & receivedClearance the plane shall never satisfy takeoff

# Scope grammar



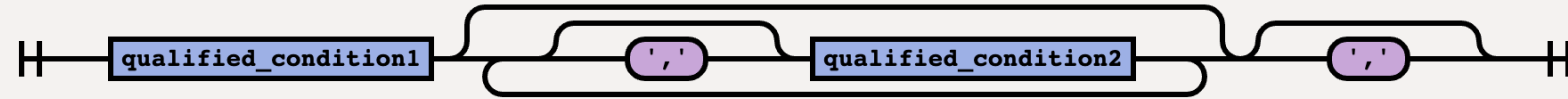


# scope **Condition** component timing response

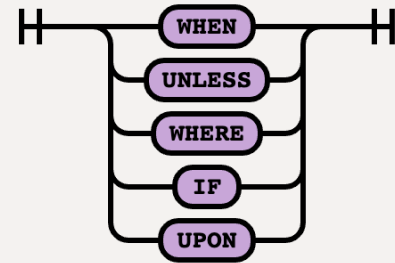
- *upon, if, when, where* `BOOL_EXP`
- *unless* `BOOL_EXP` (equivalent to “*upon ! BOOL\_EXP*”)
- Trigger: **upon** the Boolean expression becoming true from being false in the scope, or being true at the beginning of the scope.

# Condition grammar

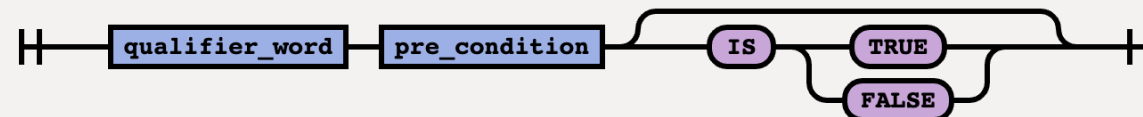
regular\_condition



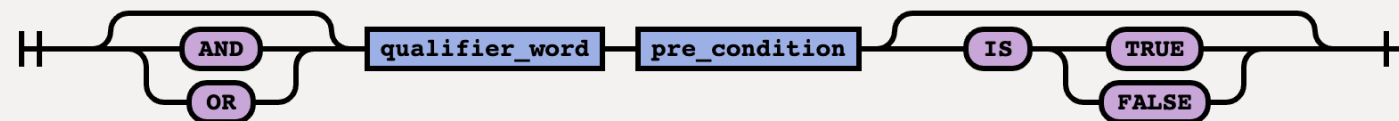
qualifier\_word



qualified\_condition1



qualified\_condition2

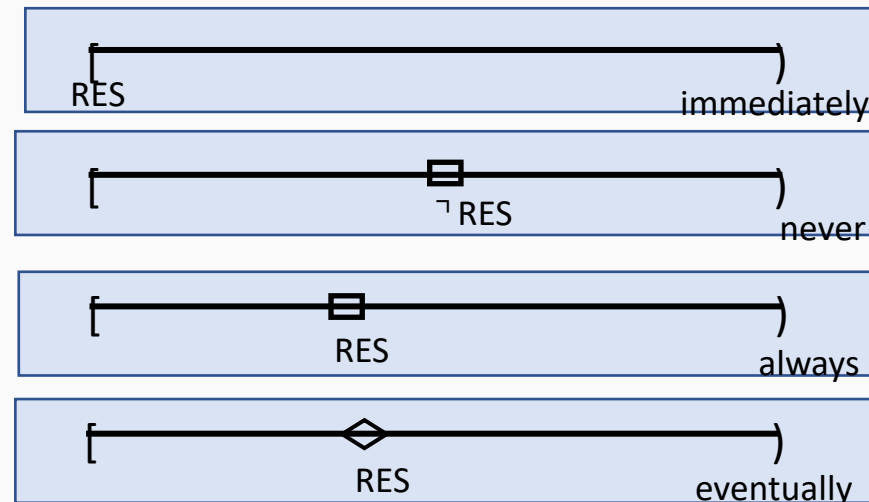
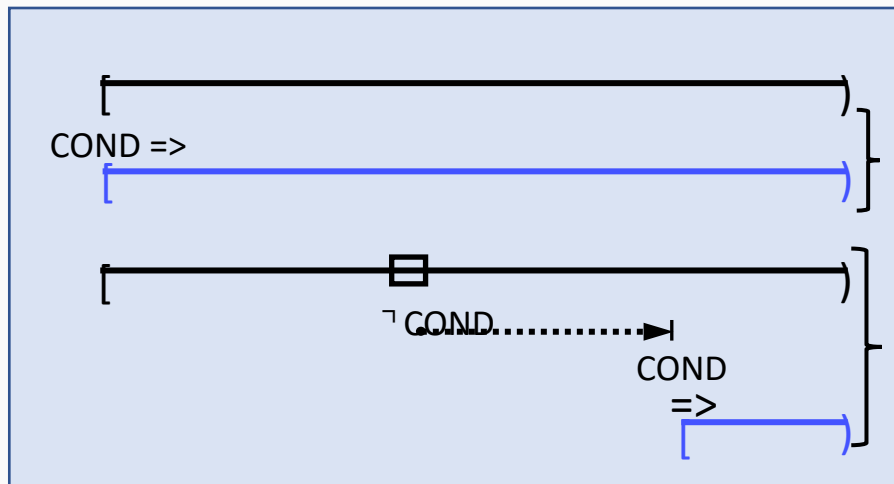


# scope condition component **Timing** response

- In roll\_hold mode RollAutopilot shall **immediately** satisfy if  $(\text{roll\_angle} < 6.0 \ \& \ \text{roll\_angle} > -6.0)$  then  $\text{roll\_hold\_reference} = 0.0$
- When currentOverload the circuitBreaker shall, **at the next timepoint**, satisfy shutoff
- In landingMode the system shall **eventually** satisfy LandingGearLowered
- The autopilot shall **always** satisfy if allGood then state = nominal
- In drivingMode the system shall **never** satisfy cellPhoneOn & !cellPhoneHandsFree
- When errorCondition, the system shall, **for** 4 ticks, satisfy alarmOn
- In landing mode, the the system shall **within** 2 ticks satisfy is\_stable
- When input = 1, the integrator shall, **after** 10 ticks, satisfy output = 10
- In CountdownMode the system shall, **until** Count = 0, satisfy Count > 0
- The system shall, **before** TakeOff, satisfy CheckListTasksCompleted

# FRET is rigorous and extensible

- Semantic templates have RTGIL semantics. RTGIL = Real-Time Graphical Interval Logic
- FRET generates formulas in *future-* (finite and infinite-trace) and *past-time* linear-time metric temporal logics, and CoCoSpec/Lustre. Discrete time.
- A verification framework within FRET ensures correctness of formalization algorithms.
- All aspects of our approach are compositional – based on requirement fields.



# FRET bridges the gap

**Captures** requirements in a restricted natural language with **unambiguous semantics**

**Explains** formal **semantics** in various forms: natural language, diagrams, interactive simulation

- **Assists** in writing requirements through requirement **templates**

- **Formalizes** requirements in a **compositional** (hence maintainable and extensible) manner: *past, future linear temporal logic, Lustre*

- **Checks consistency** of requirements and provides feedback

- **Connects** with **analysis tools** and **exports verification code**

- ✓ for model checking Simulink models with CoCoSim
- ✓ for model checking Lustre code with Kind2
- ✓ for runtime analysis of C programs with Copilot

# Capturing, explaining and formalizing requirements

**CREATE**

**Create Requirement**

Project: Demo-FSM

Requirement ID: Parent Requirement ID

Rationale and Comments

Requirement Description

A requirement follows the sentence structure displayed below, where fields are optional unless indicated with "\*". For information on a field format, click on its corresponding bubble.

SCOPED | CONDITIONS | COMPONENT\* | SHALL\* | TIMING | RESPONSES\*

SEMANTICS

**CANCEL** **CREATE**

**ASSISTANT** TEMPLATES GLOSSARY

Ready to speak FRETish?

Please use the editor on your left to write your requirement or pick a predefined template from the TEMPLATES tab.

LM\_AUTOPILOT REG\_YAW\_ACC\_REQ

## Update Requirement

Requirement ID	Parent Requirement ID	Project
Test-ALTHOLD		LM_requirements

### Rationale and Comments

#### Rationale

#### Comments

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected

### Requirement Description

A requirement follows the sentence structure displayed below, where fields are optional unless indicated with "\*". For information on a field format, click on its corresponding bubble.

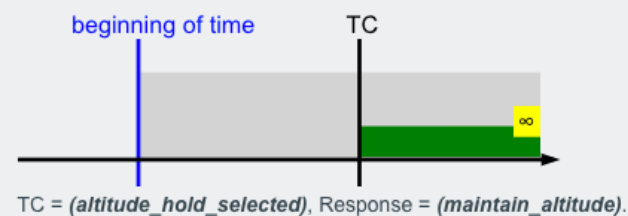


if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

SEMANTICS

## Semantics

ENFORCED: in the interval defined by the entire execution. TRIGGER: first point in the interval if (*altitude\_hold\_selected*) is true and any point in the interval where (*altitude\_hold\_selected*) becomes true (from false). REQUIRES: for every trigger, RES must hold at all time points between (and including) the trigger and the end of the interval.



### Diagram Semantics

### Formalizations

#### Future Time LTL

```
((LAST V (((! (altitude_hold_selected)) & ((! LAST) & (X (altitude_hold_selected)))) -> (X (LAST V (maintain_altitude)))))) & ((altitude_hold_selected) -> (LAST V (maintain_altitude))))
```

Target: *altitude\_hold\_autopilot* component.

#### Past Time LTL

```
(H ((H (! (altitude_hold_selected))) | (maintain_altitude)))
```

Target: *altitude\_hold\_autopilot* component.

## Update Requirement

Requirement ID: Test-ALTHOLD      Project: LM\_requirements

### Rationale and Comments

#### Rationale

#### Comments

the altitude hold autopilot shall maintain altitude whenever altitude hold is selected

### Requirement Description

A requirement follows the sentence structure displayed below, where fields are optional unless indicated with "\*". For information on a field format, click on its corresponding bubble.

SCOPED    CONDITIONAL    COMPONENT\*    SHALL\*    TIMING    RESPONSES\*

if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

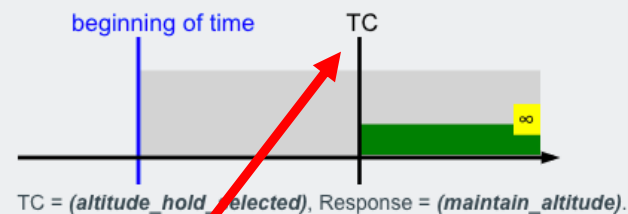


but this is not what I mean...

SEMANTICS

### Semantics

ENFORCED: in the interval defined by the entire execution. TRIGGER: first point in the interval if (*altitude\_hold\_selected*) is true and any point in the interval where (*altitude\_hold\_selected*) becomes true (from false). REQUIRES: for every trigger, RES must hold at all time points between (and including) the trigger and the end of the interval.



### Diagram Semantics

### Formalizations

#### Future Time LTL

```
((LAST V (((! (altitude_hold_selected)) & ((! LAST) & (X (altitude_hold_selected)))) -> (X (LAST V (maintain_altitude)))))) & ((altitude_hold_selected) -> (LAST V (maintain_altitude))))
```

Target: *altitude\_hold\_autopilot* component.

#### Past Time LTL

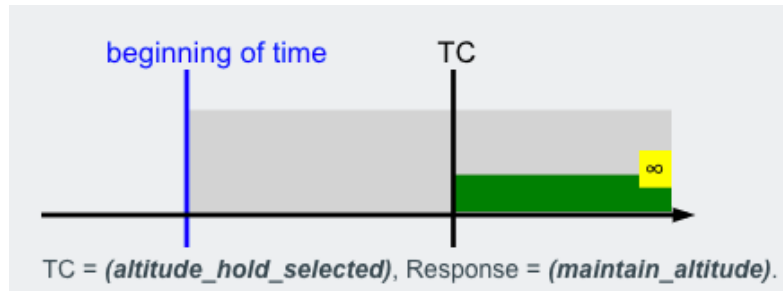
```
(H ((H (! (altitude_hold_selected)) | (maintain_altitude))))
```

Target: *altitude\_hold\_autopilot* component.



# getting to the right requirement

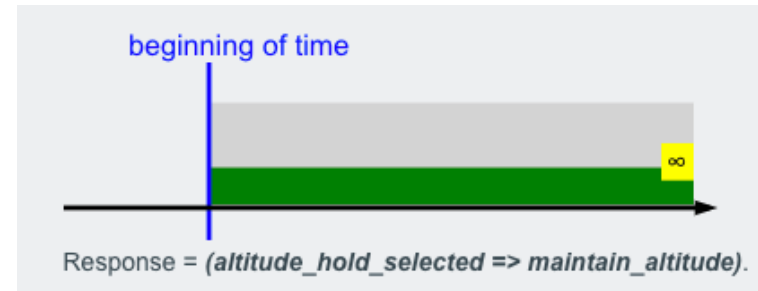
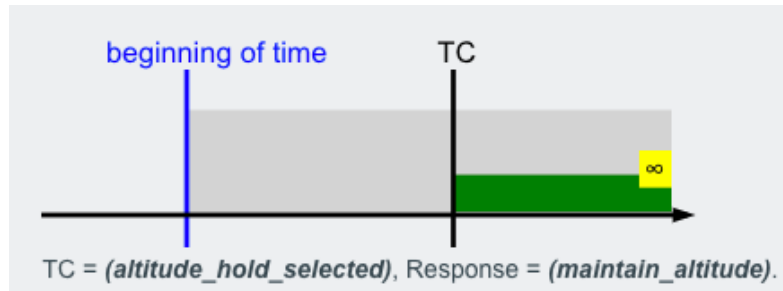
**TAKE1:** if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude



# getting to the right requirement

**TAKE1:** if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

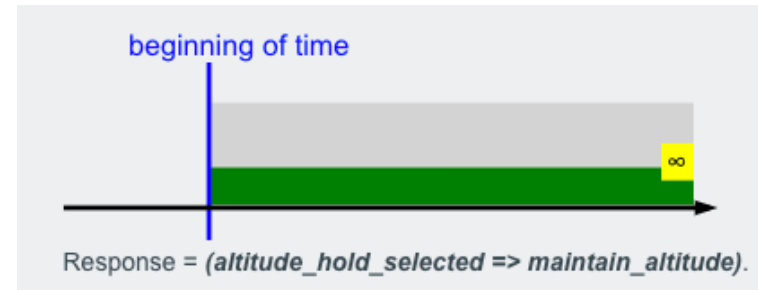
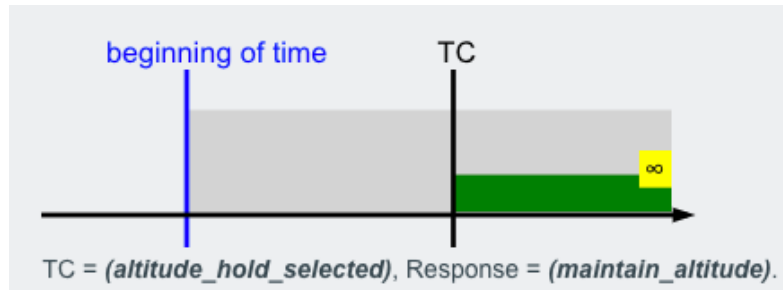
**TAKE2:** the altitude\_hold\_autopilot shall always satisfy if altitude\_hold\_selected then maintain\_altitude



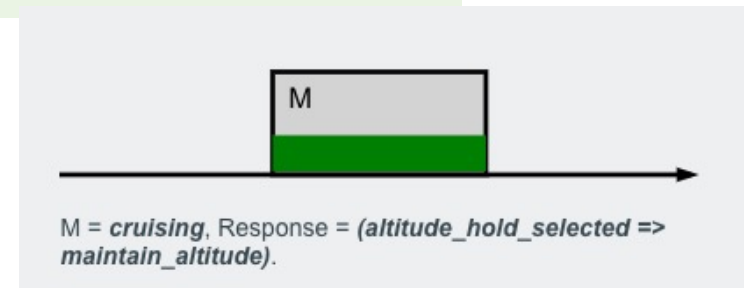
# getting to the right requirement

**TAKE1:** if altitude\_hold\_selected the altitude\_hold\_autopilot shall always satisfy maintain\_altitude

**TAKE2:** the altitude\_hold\_autopilot shall always satisfy if altitude\_hold\_selected then maintain\_altitude



**TAKE3:** when in cruising mode, the altitude\_hold\_autopilot shall always satisfy if altitude\_hold\_selected then maintain\_altitude



# FRET bridges the gap

- **Captures** requirements in a restricted natural language with **unambiguous semantics**
- **Explains** formal **semantics** in various forms: natural language, diagrams, interactive simulation
- **Assists** in writing requirements through requirement **templates**
- **Formalizes** requirements in a **compositional** (hence maintainable and extensible) manner
- **Checks consistency** of requirements and provides feedback
- **Connects** with **analysis tools** and **exports verification code**
  - ✓ for model checking Simulink models with CoCoSim
  - ✓ for model checking Lustre code with Kind2
  - ✓ for efficient runtime monitoring with Copilot

# Assistance: Requirement templates

## Lockheed Martin Cyber-Physical System Challenge, component FSM:

- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data is not good.
- The autopilot shall change states from NOMINAL to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from MANEUVER to STANDBY when the pilot is in control (standby) and sensor data is good.

# Requirement templates

## Lockheed Martin Cyber-Physical System Challenge, component FSM:

- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data is not good.
- The autopilot shall change states from NOMINAL to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from MANEUVER to STANDBY when the pilot is in control (standby) and sensor data is good.

# Requirement templates

The screenshot displays the FRET software interface. At the top, there is a navigation bar with 'FRET', 'Projects', and a 'CREATE' button. The main window is titled 'Create Requirement' and contains a form with the following fields:

- Requirement ID: FSM 002
- Parent Requirement ID: (empty)
- Project: LM\_requirements

Below these fields is a 'Rationale and Comments' section with two text areas:

- Rationale: (empty)
- Comments: The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).

Underneath is the 'Requirement Description' section, which includes a paragraph explaining the sentence structure and a visual grammar tool. The tool consists of a grid with colored bubbles for 'SCOPE', 'CONDITIONS', 'COMPONENT\*', 'SHALL\*', 'TIMING', and 'RESPONSES\*'. Below the grid is a text box containing the template sentence: 'component shall always satisfy if (input\_state & condition) then output\_state'.

On the right side, there is a sidebar with two tabs: 'ASSISTANT' and 'TEMPLATES'. The 'TEMPLATES' tab is active, showing a 'Change State' template. The template description states: 'This template describes how the state of a finite-state-machine component changes. It describes the input state and some conditions based on which the change must occur. The corresponding output state must reflect the required change. The input and output states have a pre - post- relationship'. Below this, an 'Examples' section shows a code snippet: 'FSM\_Autopilot shall always satisfy if (state = ap\_standby\_state & ! standby & ! apfail) then STATE = ap\_transition\_state'.



# FRET bridges the gap

- **Captures** requirements in a restricted natural language with **unambiguous semantics**
- **Explains** formal **semantics** in various forms: natural language, diagrams, interactive simulation
- **Assists** in writing requirements through requirement **templates**
- **Formalizes** requirements in a **compositional** (hence maintainable and extensible) manner
- **Checks consistency** of requirements and provides feedback
- **Connects** with **analysis tools** and **exports verification code**
  - ✓ for model checking Simulink models with CoCoSim
  - ✓ for model checking Lustre code with Kind2
  - ✓ for efficient runtime monitoring with Copilot



# Checking Consistency

Lockheed Martin Cyber-Physical System Challenge, component FSM:

**Definition of a *Realizable* set of requirements:** A system exists that satisfies the requirements for *every* valid environment input.

- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.

# Checking Realizability

Lockheed Martin Cyber-Physical System Challenge, component FSM:

- The autopilot shall change states from **TRANSITION** to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from **TRANSITION** to NOMINAL when the system is supported and sensor data is good.

Input state: **TRANSITION**

# Checking Realizability

Lockheed Martin Cyber-Physical System Challenge, component FSM:



- The autopilot shall change states from **TRANSITION** to **STANDBY** when the pilot is in control (standby).
- The autopilot shall change states from **TRANSITION** to **NOMINAL** when the system is supported and sensor data is good.

Input state: **TRANSITION** ✓  
Condition 1: pilot is in control ✓  
Condition 2: system is supported ✓  
                  sensor data is good ✓

# Checking Realizability

## Lockheed Martin Cyber-Physical System Challenge, component FSM:

- The autopilot shall change states from **TRANSITION** to **STANDBY** when the pilot is in control (standby).
- The autopilot shall change states from **TRANSITION** to **NOMINAL** when the system is supported and sensor data is good.

Input state: **TRANSITION** ✓  
Condition 1: pilot is in control ✓  
Condition 2: system is supported ✓  
                  sensor data is good ✓  
Output state 1: **STANDBY**   
Output state 2: **NOMINAL** 

The system must be consistent for any valid environmental input.

# Checking Realizability

- Realizable requirements: A system exists that satisfies the requirements for *every* valid environment input
- Unrealizable requirements: Diagnostic analysis
  - Identify minimal sets of unrealizable requirements in specification
  - Counterexamples
  - Simulation of conflicting requirements
- Compositional Realizability Checking
  - *Connected Components (CC)*: sets of requirements where the sets can be analyzed independently

Giannakopoulou, Dimitra, Andreas Katis, Anastasia Mavridou, and Thomas Pressburger. "Compositional realizability checking within FRET." (NASA/TM-20210013008).

Mavridou, Anastasia, Andreas Katis, Dimitra Giannakopoulou, David Kooi, Thomas Pressburger, and Michael W. Whalen. "From Partial to Global Assume-Guarantee Contracts: Compositional Realizability Analysis in FRET." FM 2021

# Variable declaration

- Variable name in requirement
- Variable Type:
  - Input (the system monitors the variable)
  - Output (the system controls the variable)
  - Internal: just a name for a Lustre expression, like a macro.
- Datatype
  - Boolean, integer, double, unsigned integer, single

# Variable Declaration/Mapping Dialog

The screenshot shows a software interface for managing variables. At the top, there is a header "FSM\_Autopilot" with an "EXPORT" button. Below this is a search bar labeled "Corresponding Model Component" with an "IMPORT" button. A table lists variables with columns: "FRET Variable Name", "Model Variable Name", "Variable Type", "Data Type", and "Description". The table contains entries like "ap\_maneuver\_state", "ap\_nominal\_state", "ap\_standby\_state", "ap\_transition\_state", "apfail", "good", "standby", "STATE", "state", and "supported".

An "Update Variable" dialog box is open in the foreground. It contains the following fields:

- FRET Project: LM\_requirements
- FRET Component: FSM\_Autopilot
- Model Component: (empty)
- FRET Variable: apfail
- Variable Type\*: (dropdown menu)
- Description: (empty)

At the bottom of the dialog are "CANCEL" and "UPDATE" buttons. The background table is dimmed, and the "Rows per page: 10" and "1-10 of 10" information is visible at the bottom right.

File View Help

FRET Projects CREATE

VARIABLE MAPPING REALIZABILITY

System Component \*  
FSM  Compositional  Monolithic Timeout (seconds) 900 CHECK DIAGNOSE EXPORT HELP

CC0 CC1 CC2

ID ↑	Summary
FSM001	FSM shall always satisfy (limits & !standby & !apfail & supported) => pullup
FSM002	FSM shall always satisfy (standby & state = ap_transition_state) => STATE = ap_standby_state
FSM003	FSM shall always satisfy (state = ap_transition_state & good & supported) => STATE = ap_nominal_state
FSM004	FSM shall always satisfy (! good & state = ap_nominal_state) => STATE = ap_maneuver_state
FSM005	FSM shall always satisfy (state=ap_nominal_state & standby) => STATE = ap_standby_state
FSM006	FSM shall always satisfy (state = ap_maneuver_state & standby & good) => STATE = ap_standby_state
FSM007	FSM shall always satisfy (state = ap_maneuver_state & supported & good) => STATE = ap_transition_state
FSM008	FSM shall always satisfy (state = ap_standby_state & !standby) => STATE = ap_transition_state
FSM009	FSM shall always satisfy (state = ap_standby_state & apfail )=> STATE = ap_maneuver_state
FSM010	FSM shall always satisfy (senstate = sen_nominal_state & limits) => SENSTATE = sen_fault_state

Rows per page: 10 1-10 of 13

Anastasia Mavridou, Andreas Katis, Dimitra Giannakopoulou, David Kooi, Thomas Pressburger, Michael W. Whalen: *From Partial to Global Assume-Guarantee Contracts: Compositional Realizability Analysis in FRET*. FM 2021.



# Simulation of Counterexample



# FRET bridges the gap

- **Captures** requirements in a restricted natural language with **unambiguous semantics**
- **Explains** formal **semantics** in various forms: natural language, diagrams, interactive simulation
- **Assists** in writing requirements through requirement **templates**
- **Formalizes** requirements in a **compositional** (hence maintainable and extensible) manner
- **Checks consistency** of requirements and provides feedback

**Connects** with **analysis tools** and **exports verification code**

- ✓ for model checking Simulink models with CoCoSim
- ✓ for model checking Lustre code with Kind2
- ✓ for efficient runtime monitoring with Copilot

# Variable mapping

- In target model/code: e.g., the corresponding signal in Simulink model
  - Simulink architectural information can be imported into FRET so user can navigate/choose among possibilities

# Connection with analysis tools

The screenshot displays the FRET software interface. At the top, there is a navigation bar with the FRET logo, a 'Projects' dropdown, and a 'CREATE' button. Below this, the main content area is titled 'Requirement Variables to Model Mapping: Demo-FSM'. There are two tabs: 'VARIABLE MAPPING' (active) and 'REALIZABILITY'. An 'Export Language \*' dropdown is visible. Below that, there is a section for 'FSM' with an 'EXPORT' button. A light blue bar contains a 'Corresponding Model Component' dropdown and an 'IMPORT' button. The main part of the interface is a table with the following data:

FRET Variable Name ↑	Model Variable Name	Variable Type	Data Type	Description
ap_maneuver_state		Internal	double	value 2.0
ap_nominal_state		Internal	double	value 1.0
ap_standby_state		Internal	double	value 3.0
ap_transition_state		Internal	double	value 0.0
apfail	apfail	Input	boolean	
good	good	Input	boolean	
limits	limits	Input	boolean	
pullup	pullup	Output	boolean	
request	request	Input	boolean	
sen_fault_state		Internal	double	value 2.0

At the bottom right of the table, there is a pagination control: 'Rows per page: 10' and '1-10 of 18' with navigation arrows.

# Connection with analysis tools

The image shows a MATLAB Simulink CoSim window for a file named 'fsm\_12B'. The interface includes a top menu bar with 'SIMULATION', 'DEBUG', 'MODELING', 'FORMAT', and 'APPS'. Below this is a toolbar with icons for 'New', 'Open', 'Save', 'Print', 'Library Browser', 'Log Signals', 'Add Viewer', 'Signal Table', 'Stop Time' (set to 10), 'Normal' (dropdown), 'Fast Restart', 'Step Back', 'Run', 'Step Forward', 'Step', and 'Data Inspector'. The main workspace contains a 'FiniteStateMachine' block with four inputs: 'standby' (boolean, labeled 1), 'apfail' (boolean, labeled 2), 'supported' (boolean, labeled 3), and 'limits' (boolean, labeled 4). The block has one output: 'pullup' (boolean, labeled 1). A text box at the top of the workspace reads: 'Cyber-Physical V&V Challenge Problems', 'LM Aeronautics Quantum Information Science Research Team 2015', and 'Copyright © 2015 Lockheed Martin Corporation'. The status bar at the bottom shows 'Automated Analysis Framework', '222%', and 'FixedStepDiscrete'.

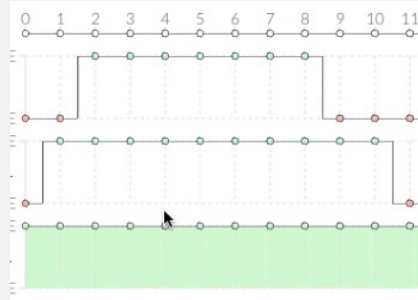
captures + assists



FRETish

when in cruising mode, the altitude\_hold\_;

explains



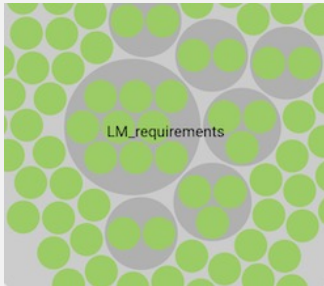
ENFORCED: In every interval where *cruising* holds. TRIGGER: first point in the interval. REQUIRES: for every trigger, RES must hold at all time points between (and including) the trigger and the end of the interval.



Diagram Semantics

- M Mode of operation (mentioned in Scope)
- Intervals
- Scope interval
- Response must hold at least somewhere in this interval
- Negation of response must hold at least somewhere in this interval
- Response must hold everywhere in this interval

stores + displays



AP-002A	+	when in roll_hold mode
AP-002B	+	in roll_hold mode RollA
AP-003	+	*This requirement is th

formalizes

Future Time LTL

`(LAST V (cruising -> (altitude_hold -> maintain_altitude)))`

Target: *altitude\_hold\_autopilot* component.

Past Time LTL

checks + diagnoses

connects + exports

FRET Variable Name ↑
ABSOF_ALT_MINUS_ALTIC
ALTITUDE_HOLD

	2	3	4	Step 5
rue		true	true	true
rue		true	true	true
rue		true	true	true

# Ready for FRETish?

FRET's mission is to provide an intuitive platform for capturing precise requirements, to serve as a portal to a variety of analysis tools, and to support requirements repair based on analysis feedback.

<https://github.com/NASA-SW-VnV/fret>

Andreas Katis, Anastasia Mavridou, Dimitra Giannakopoulou, Thomas Pressburger, Johann Schumann, *Capture, Analyze, Diagnose: Realizability Checking of Requirements in FRET*, CAV 2022 (conditionally accepted).

Esther Conrad, Laura Titolo, Dimitra Giannakopoulou, Thomas Pressburger, Aaron Dutle. *A Compositional Proof Framework for FRETish Requirements*. CPP 2022.

Ivan Perez, Anastasia Mavridou, Tom Pressburger, Alwyn Goodloe and Dimitra Giannakopoulou. *Automated Translation of Natural Language Requirements to Runtime Monitors*, TACAS 2022

Anastasia Mavridou, Andreas Katis, Dimitra Giannakopoulou, David Kooi, Thomas Pressburger, Michael W. Whalen: *From Partial to Global Assume-Guarantee Contracts: Compositional Realizability Analysis in FRET*. FM 2021.

Giannakopoulou, Dimitra, Andreas Katis, Anastasia Mavridou, and Thomas Pressburger. "Compositional realizability checking within FRET." (NASA/TM-20210013008).

Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Johann Schumann: *Automated Formalization of Structured Natural Language Requirements*. IST Journal, 2021.

Aaron Dutle, César A. Muñoz, Esther Conrad, Alwyn Goodloe, Laura Titolo, Iván Pérez, Swee Balachandran, Dimitra Giannakopoulou, Anastasia Mavridou, Thomas Pressburger: *From Requirements to Autonomous Flight: An Overview of the Monitoring ICAROUS Project*. FMAS 2020.

Anastasia Mavridou, Hamza Bourbouh, Dimitra Giannakopoulou, Thomas Pressburger, Mohammad Hejase, P-Loïc Garoche, Johann Schumann: *The Ten Lockheed Martin Cyber-Physical Challenges: Formalized, Analyzed, and Explained*. RE 2020.

Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Johann Schumann: *Generation of Formal Requirements from Structured Natural Language*. REFSQ 2020.

Anastasia Mavridou, Hamza Bourbouh, Pierre-Loïc Garoche, Dimitra Giannakopoulou, Thomas Pressburger, Johann Schumann: *Bridging the Gap Between Requirements and Simulink Model Analysis*. REFSQ 2020.

**Thank you**