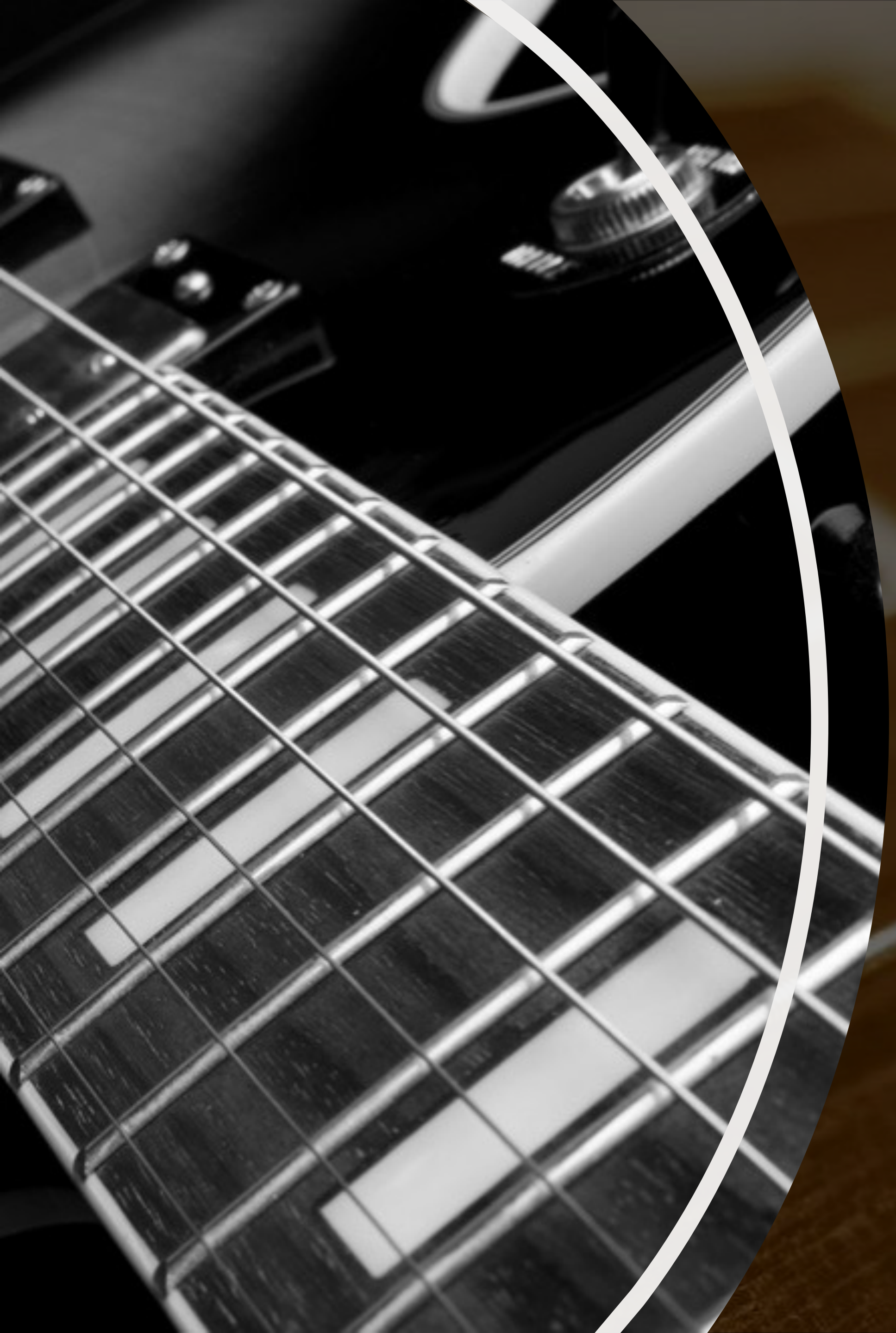
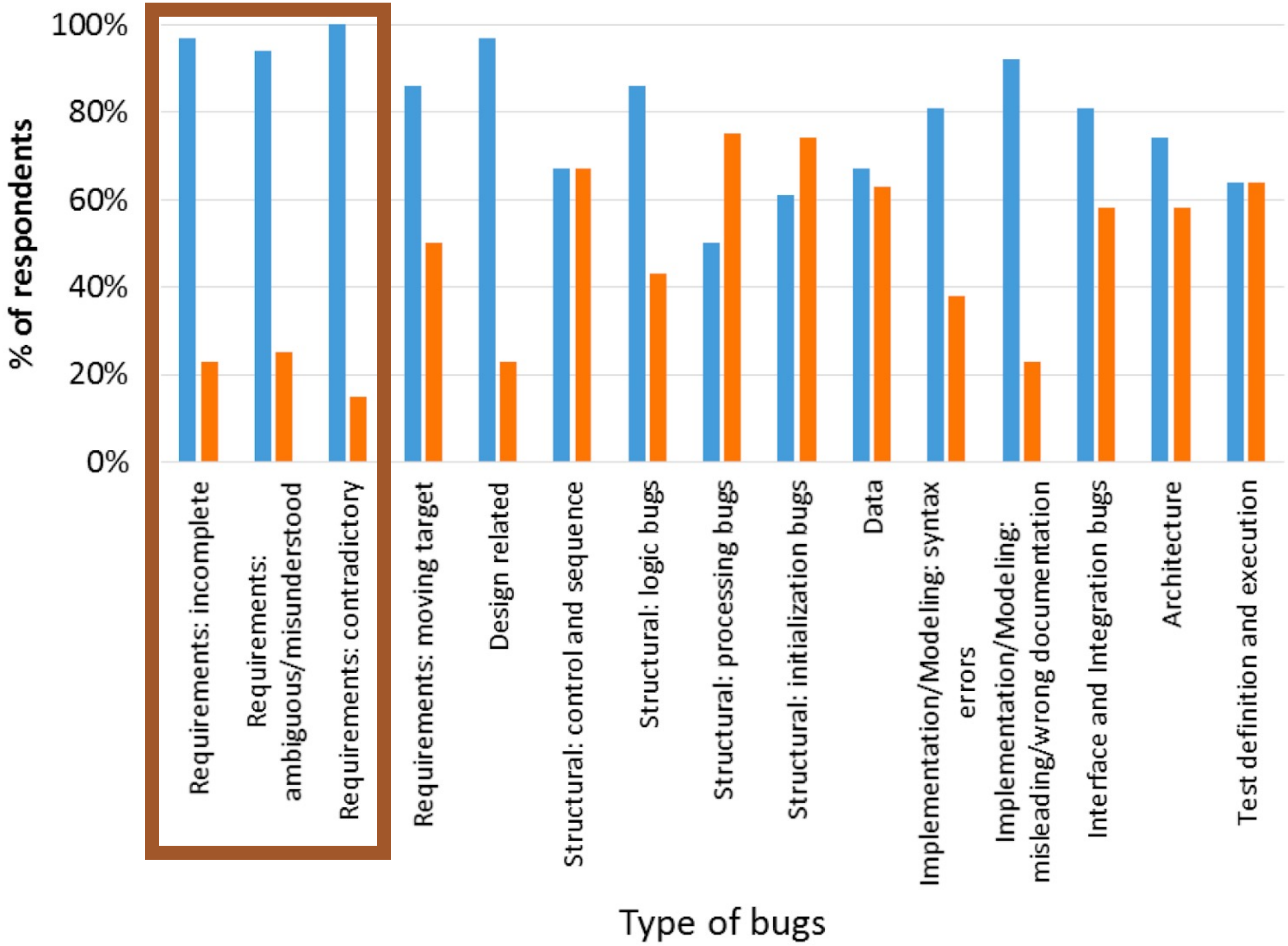


Capturing and Analyzing Requirements with FRET

Anastasia Mavridou

KBR, NASA Ames Research Center

Types of bugs 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.

how developers write requirements

10 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 apfail).
- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).

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

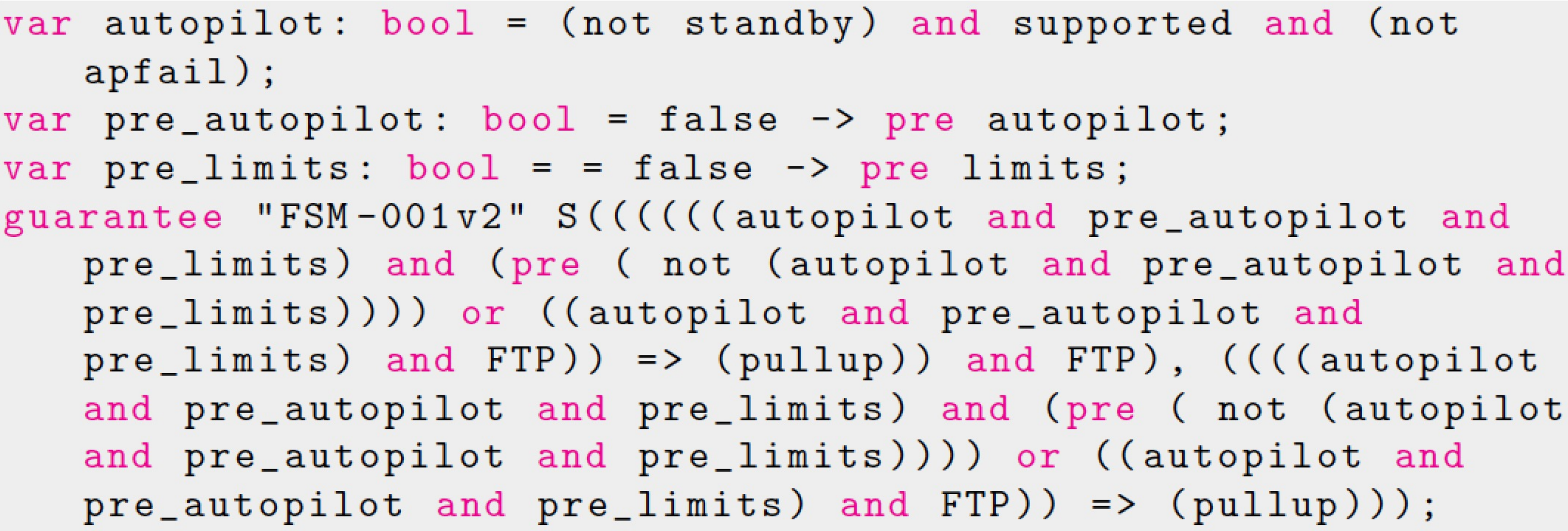
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data is not good.
- The autopilot shall change states from NOMINAL to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from MANEUVER to STANDBY when the pilot is in control (standby) and sensor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

Are the requirements consistent?

does my model/code satisfy the requirements?

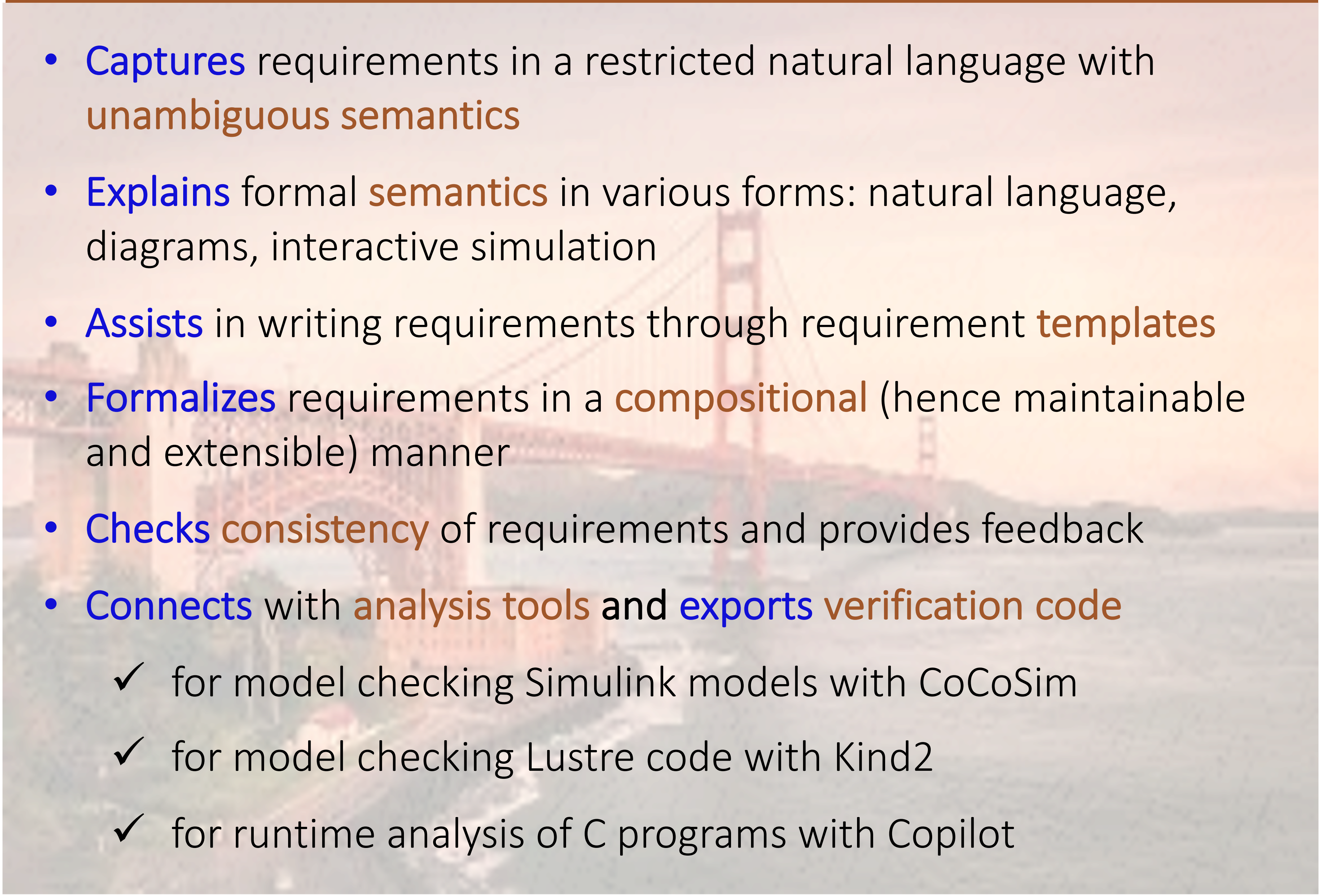
what formal analysis tools understand

Lockheed Martin Cyber-Physical System Challenge, component FSM:

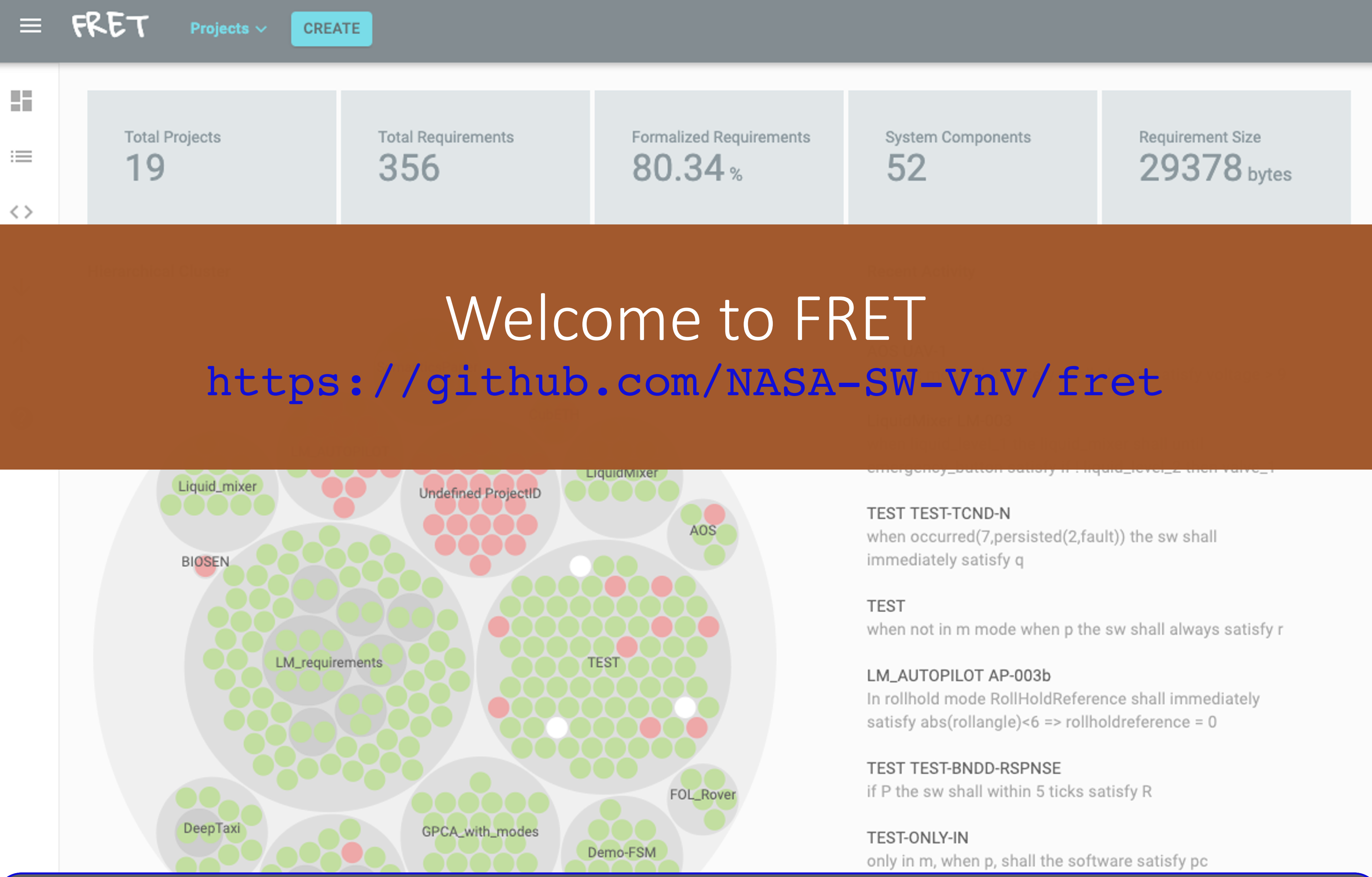


```
var autopilot: bool = (not standby) and supported and (not
    apfail);
var pre_autopilot: bool = false -> pre autopilot;
var pre_limits: bool = = false -> pre limits;
guarantee "FSM-001v2" S((((((autopilot and pre_autopilot and
    pre_limits) and (pre ( not (autopilot and pre_autopilot and
    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 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 runtime analysis of C programs with Copilot



Total Projects 19

Total Requirements 356

Formalized Requirements 80.34 %

System Components 52

Requirement Size 29378 bytes

TEST TEST-TCND-N
when occurred(7,persisted(2,fault)) the sw shall immediately satisfy q

TEST
when not in m mode when p the sw shall always satisfy r

LM_AUTOPILOT AP-003b
In rollhold mode RollHoldReference shall immediately satisfy abs(rollangle)<6 => rollholdreference = 0

TEST TEST-BNDD-RSPNSE
if P the sw shall within 5 ticks satisfy R

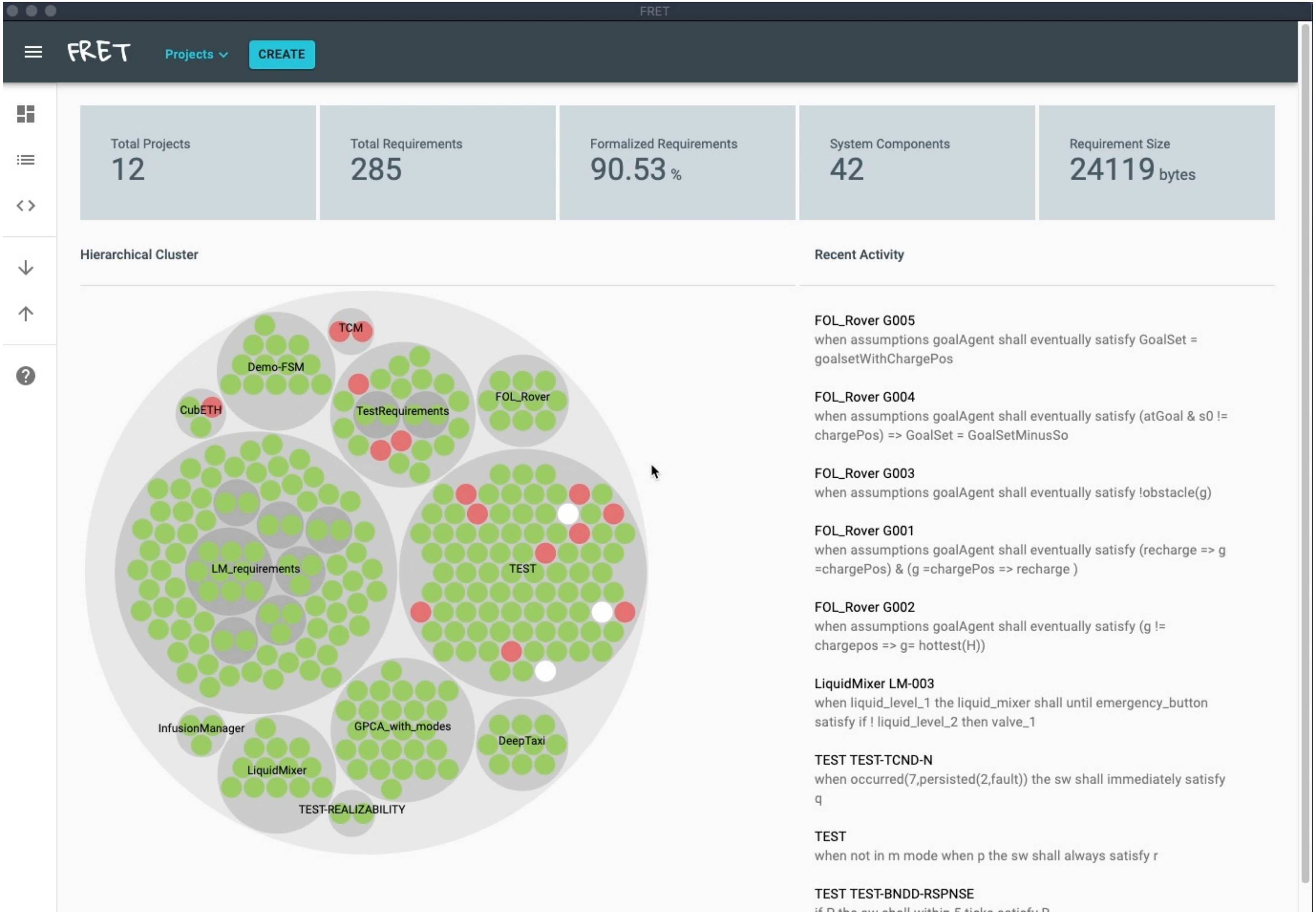
TEST-ONLY-IN
only in m, when p, shall the software satisfy pc

Welcome to FRET

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

Team: Andreas Katis, Anastasia Mavridou, Tom Pressburger, Johann Schumann, Khanh Trinh
Alumni & Interns: Milan Bhandari, David Bushnell, Tanja DeJong, Dimitra Giannakopoulou, Kelly Ho, George Karamanolis, David Kooi, Jessica Phelan, Julian Rhein, Daniel Riley, Nija Shi

Welcome to FRET



Total Projects
12

Total Requirements
285

Formalized Requirements
90.53 %

System Components
42

Requirement Size
24119 bytes

Hierarchical Cluster

Recent Activity

FOL_Rover G005
when assumptions goalAgent shall eventually satisfy GoalSet = goalsetWithChargePos

FOL_Rover G004
when assumptions goalAgent shall eventually satisfy (atGoal & s0 != chargePos) => GoalSet = GoalSetMinusSo

FOL_Rover G003
when assumptions goalAgent shall eventually satisfy !obstacle(g)

FOL_Rover G001
when assumptions goalAgent shall eventually satisfy (recharge => g =chargePos) & (g =chargePos => recharge)

FOL_Rover G002
when assumptions goalAgent shall eventually satisfy (g != chargepos => g= hottest(H))

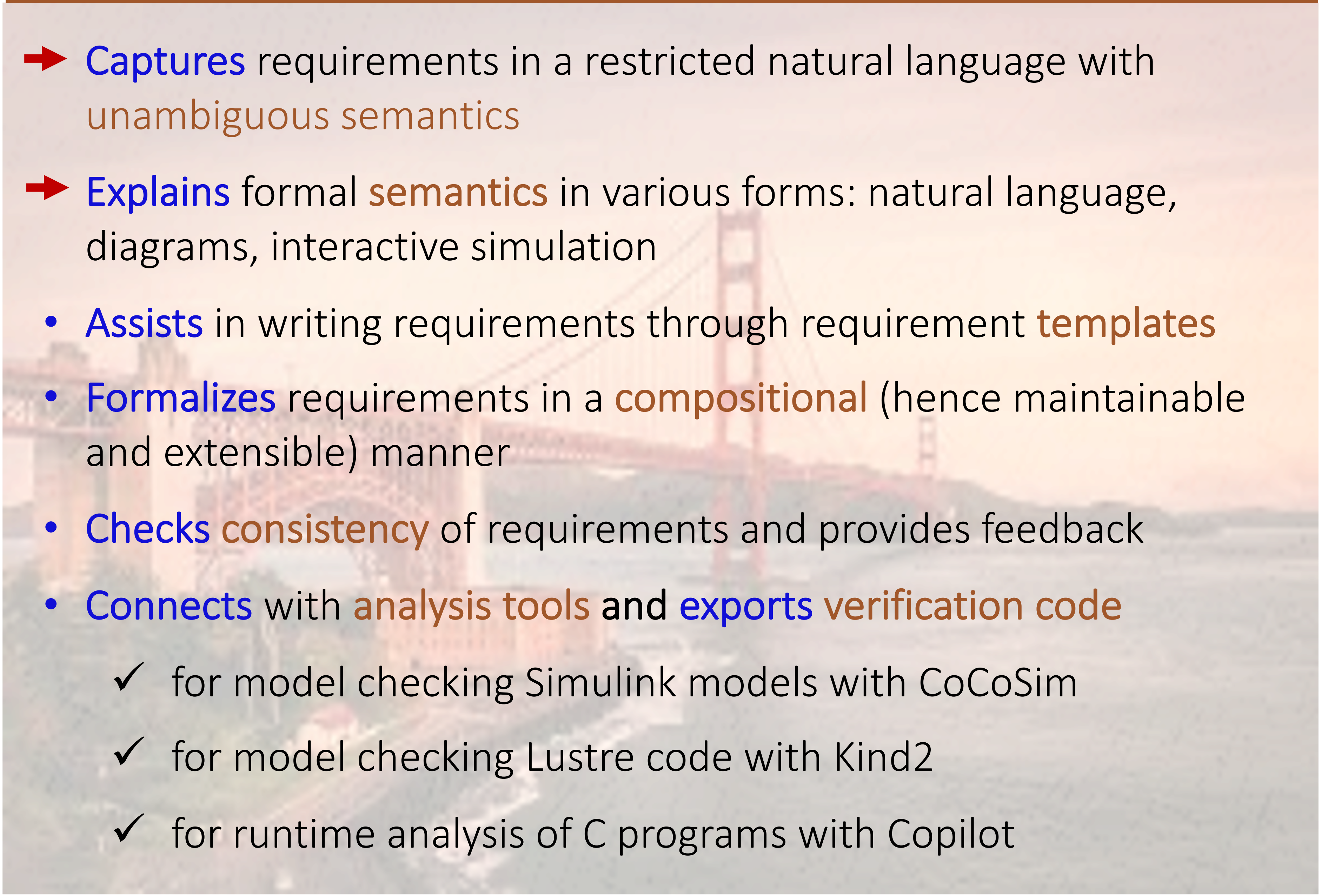
LiquidMixer LM-003
when liquid_level_1 the liquid_mixer shall until emergency_button satisfy if ! liquid_level_2 then valve_1

TEST TEST-TCND-N
when occurred(7,persisted(2,fault)) the sw shall immediately satisfy q

TEST
when not in m mode when p the sw shall always satisfy r

TEST TEST-BNDD-RSPNSE
if P the sw shall within 5 ticks satisfy R

FRET bridges the gap

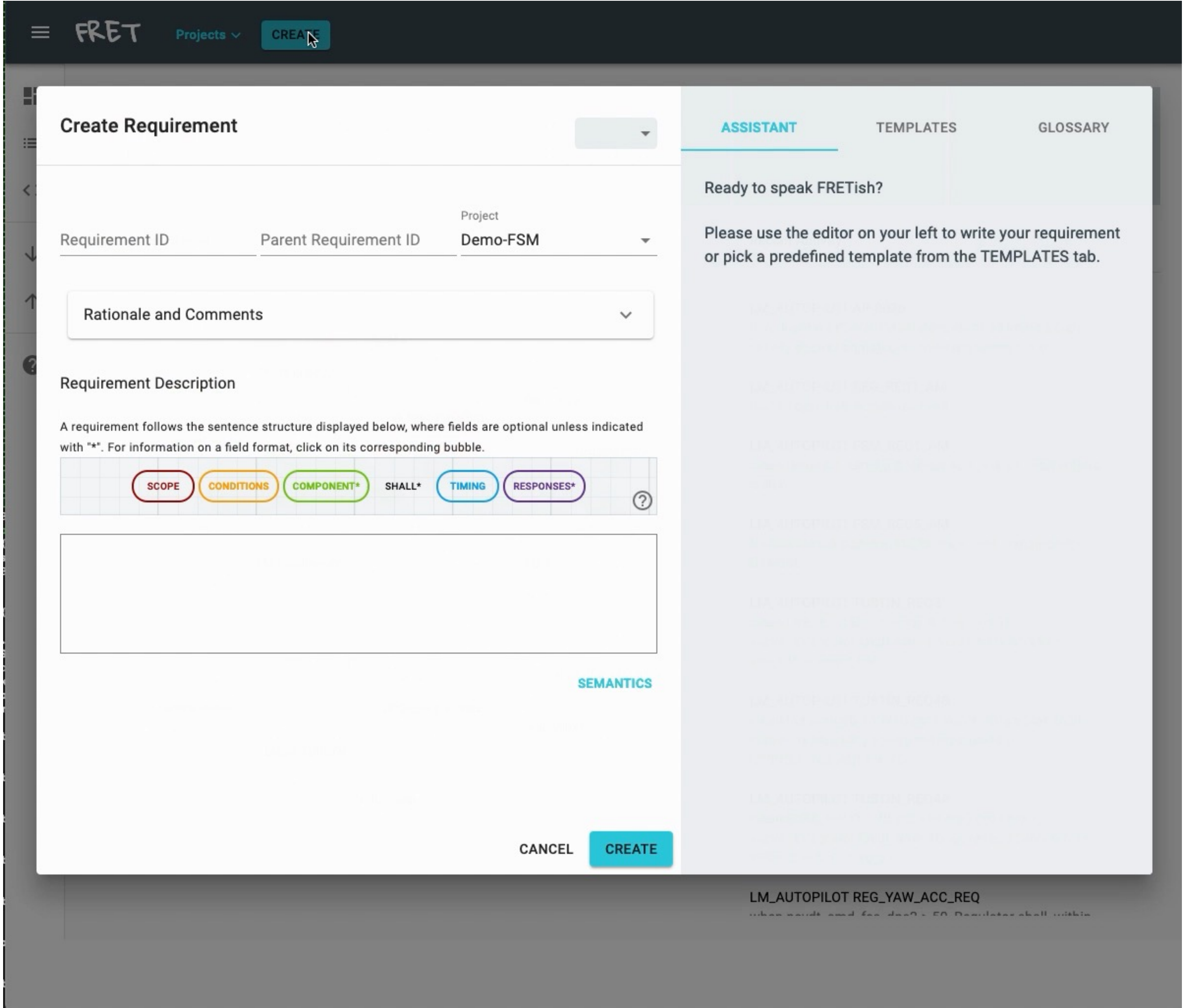


➔ Captures requirements in a restricted natural language with unambiguous semantics

➔ 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 runtime analysis of C programs with Copilot

Capturing and explaining requirements



Create Requirement

Requirement ID
Parent Requirement ID
Project: Demo-FSM

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.

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

Capturing requirements

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

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

FRETish:

if altitude_hold_selected the altitude_hold_autopilot shall always satisfy maintain_altitude

scope condition component* timing response*

Capturing requirements

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

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

FRETish:

	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

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

The altitude hold autopilot shall **maintain altitude** whenever altitude hold is selected

FRETish:

	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

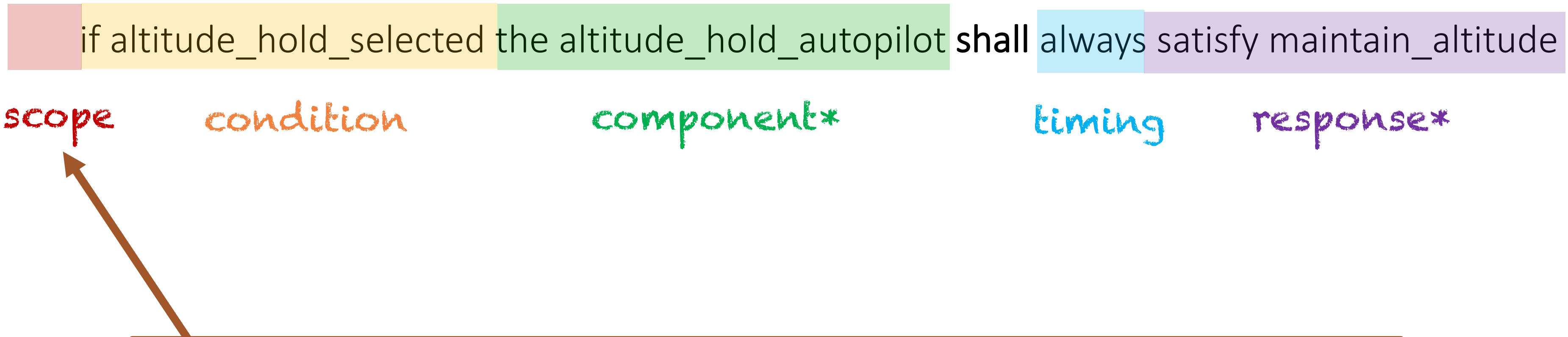
Capturing requirements

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

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

FRETish:



if altitude_hold_selected the altitude_hold_autopilot shall always satisfy maintain_altitude

scope condition component* timing response*

Q: During what portion of the execution is the requirement enforced?
A: During the whole execution: omit scope.

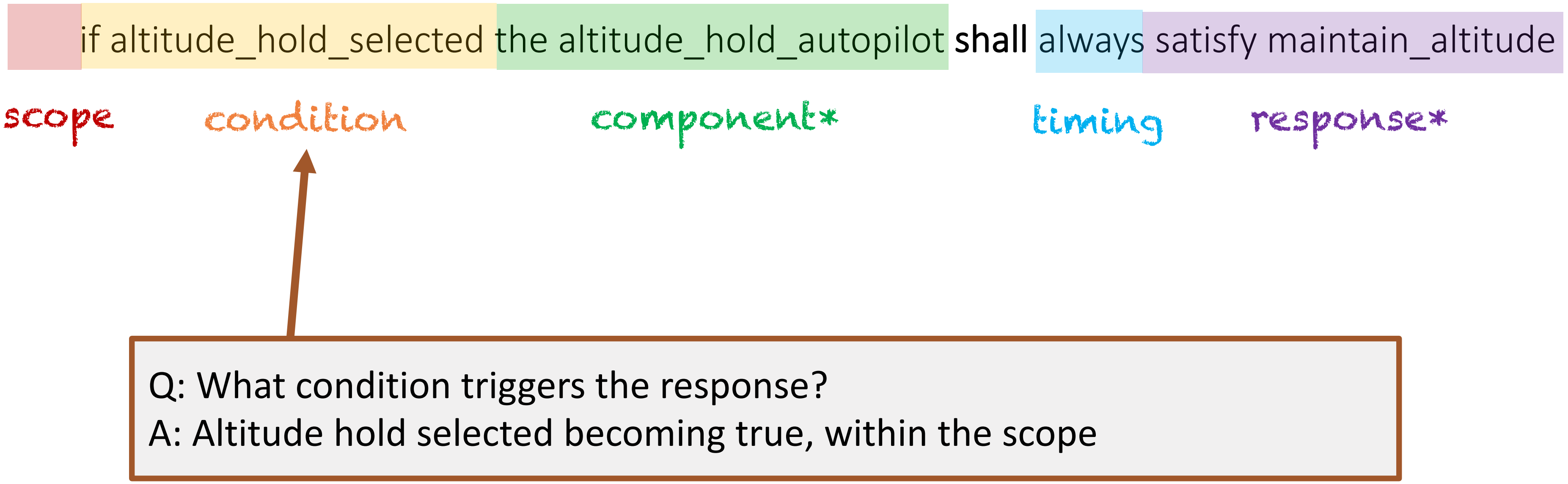
Capturing requirements

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

The altitude hold autopilot shall maintain altitude **whenever altitude hold is selected**

FRETish:



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

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

The altitude hold autopilot shall maintain altitude **whenever** altitude hold is selected

FRETish:

if altitude_hold_selected the altitude_hold_autopilot **shall** always satisfy maintain_altitude

scope condition component* timing response*

Q: When does the response happen, relative to the scope and condition?
A: Whenever (always afterwards) the condition is triggered

Capturing requirements

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

The altitude hold autopilot shall maintain altitude **whenever altitude hold is selected**

FRETish:

if altitude_hold_selected the altitude_hold_autopilot **shall** always satisfy maintain_altitude

scope condition component* timing response*

CONDITION and RESPONSE expressions:

Boolean

- !, &, |, =>, if_then_, <=>, p(x,y,z)
- preBool(init,p)
- persisted(n,p), occurred(n,p)
- Persists(n,p), occurs(n,p)

Arithmetic

- =, !=, <, >, <=, >=
- +, -, *, /, ^, f(x,y)
- preInt(init, n), preReal(init,x)

Capturing requirements

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

The altitude hold autopilot shall maintain altitude **whenever altitude hold is selected**

FRETish:

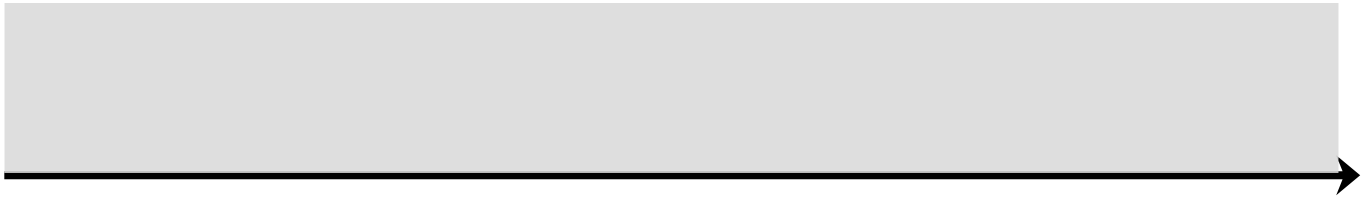
	if altitude_hold_selected	the altitude_hold_autopilot	**shall**	always	satisfy maintain_altitude
scope	condition	component*		timing	response*

SCOPE	null (global), in, before, after, notin, onlyIn, onlyBefore, onlyAfter
CONDITION	null, regular
TIMING	immediately, next, always, never, eventually, until, before, for, within, after

Capturing requirements

SCOPE

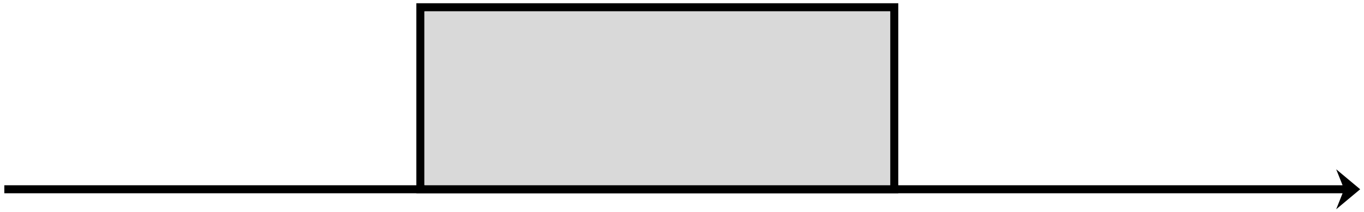
null (global), in, before, after, notin, onlyIn, onlyBefore, onlyAfter



- (global) The system shall always satisfy count >= 0

Capturing requirements

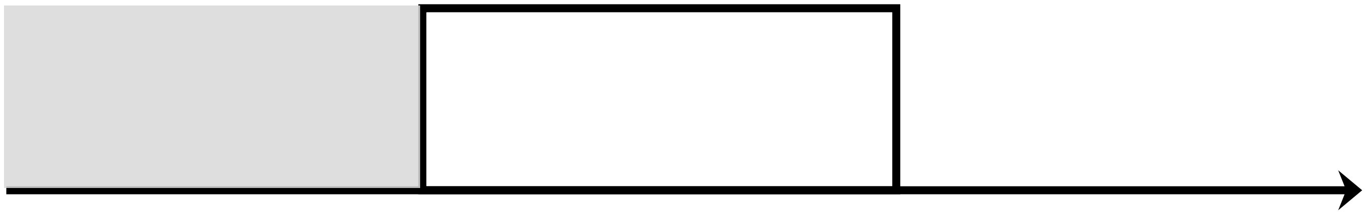
SCOPE null (global), in, before, after, notin, onlyIn, onlyBefore, onlyAfter



- **In** landing mode the system shall eventually satisfy decrease_speed

Capturing requirements

SCOPE null (global), in, **before**, after, notin, onlyIn, onlyBefore, onlyAfter

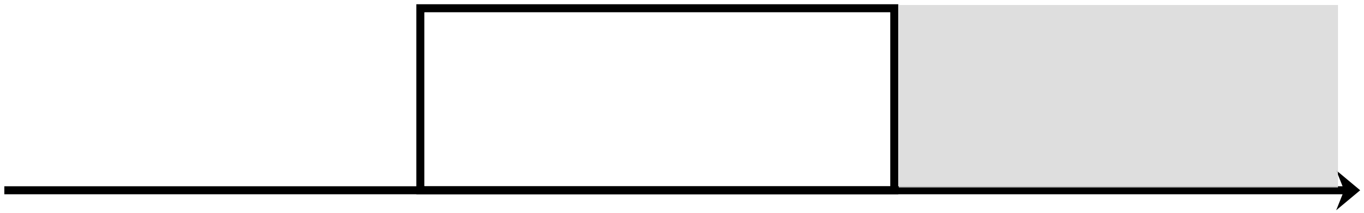


- **Before** energized mode the system shall always satisfy energized_indicator_off

Capturing requirements

SCOPE

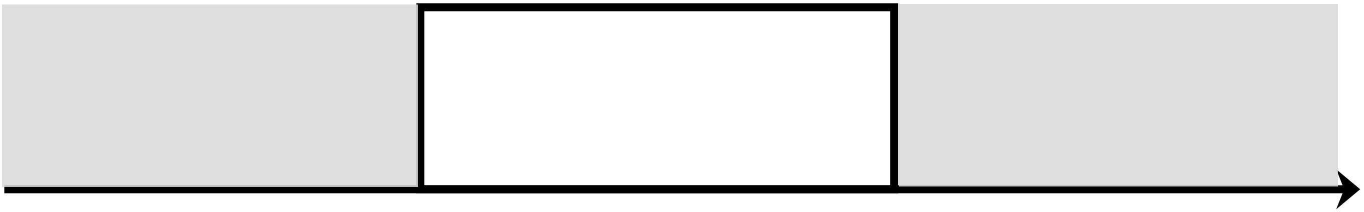
null (global), in, before, **after**, notin, onlyIn, onlyBefore, onlyAfter



- **After** boot mode the system shall immediately satisfy prompt_for_password

Capturing requirements

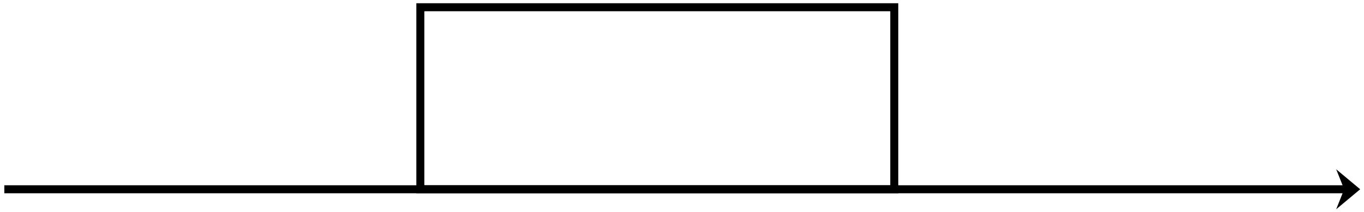
SCOPE null (global), in, before, after, **notin**, onlyIn, onlyBefore, onlyAfter



- When **not in** initialization mode the system shall always satisfy commands_accepted

Capturing requirements

SCOPE null (global), in, before, after, notin, onlyIn, onlyBefore, onlyAfter

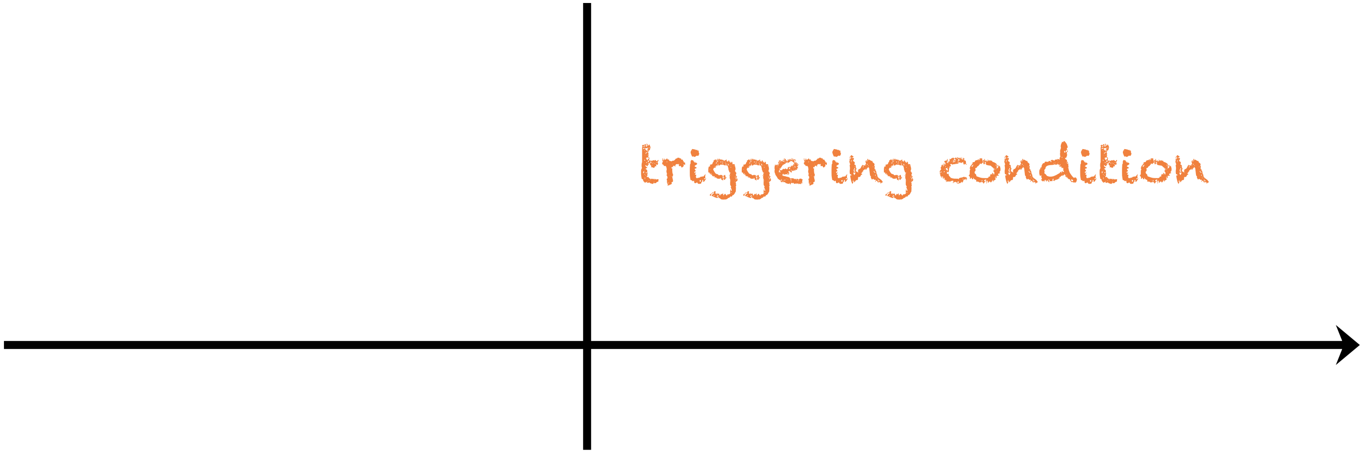


- (global) The system shall always satisfy count >= 0
- **In** landing mode the system shall eventually satisfy decrease_speed
- **Before** energized mode the system shall always satisfy energized_indicator_off
- **After** boot mode the system shall immediately satisfy prompt_for_password
- When **not 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

Capturing requirements

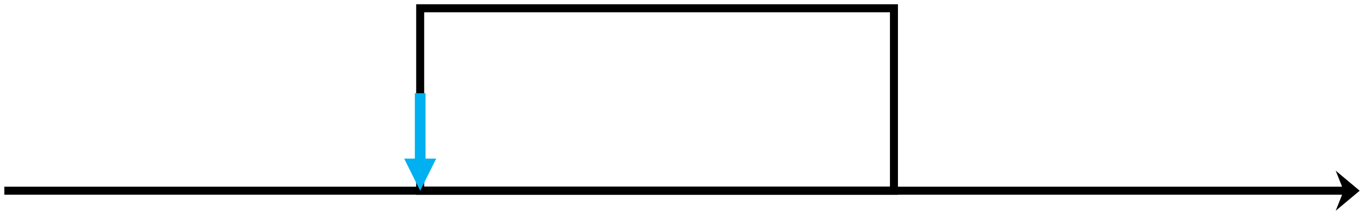
CONDITION | null, regular

- 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.



Capturing requirements

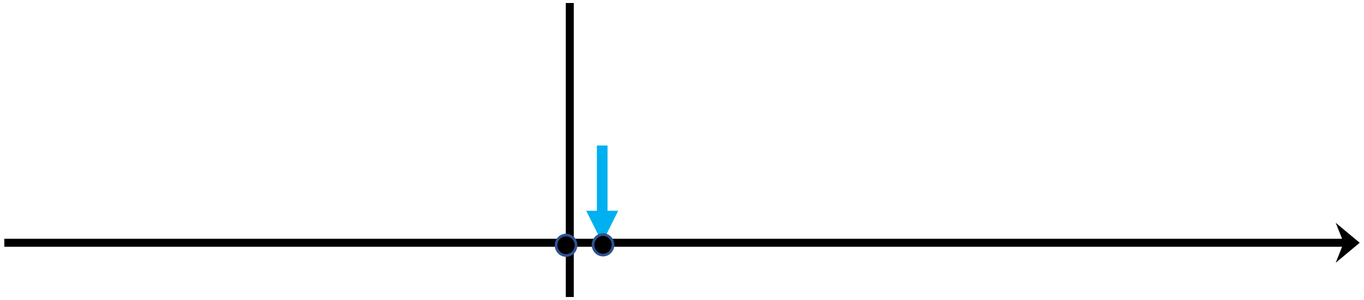
TIMING immediately, next, always, never, eventually, until, before, for, within, after



- In roll_hold mode RollAutopilot shall **immediately** satisfy roll_hold_reference = 0.0

Capturing requirements

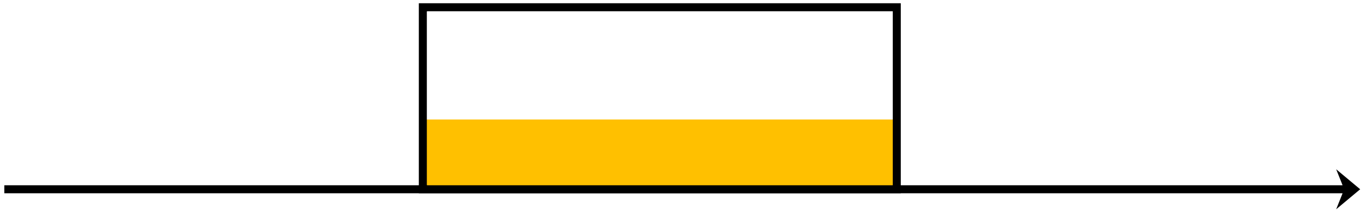
TIMING immediately, next, always, never, eventually, until, before, for, within, after



- When currentOverload the circuitBreaker shall, at the **next** timepoint, satisfy shutoff

Capturing requirements

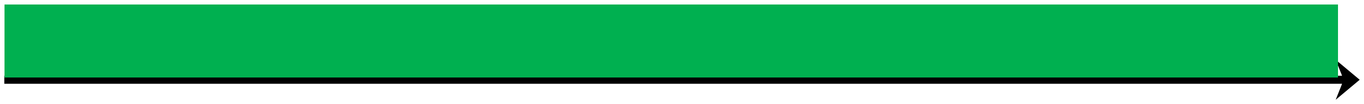
TIMING immediately, next, always, never, eventually, until, before, for, within, after



- In landingMode the system shall **eventually** satisfy LandingGearLowered

Capturing requirements

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



- The autopilot shall **always** satisfy if allGood then state = nominal

Capturing requirements

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

- In roll_hold mode RollAutopilot shall **immediately** satisfy if (roll_angle< 6.0 & roll_angle > -6.0) then 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

Let’s write a requirement together

Getting to the right requirement

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall set the roll hold reference to 30 degrees.

FRETish:

scope   condition   component*   shall*   timing   response*

Getting to the right requirement

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall set the roll hold reference to 30 degrees.

FRETish:

If abs(roll_angle) >30 & roll_hold_mode_engagement,

scope condition component* shall* timing response*

Getting to the right requirement

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, **the autopilot** shall set the roll hold reference to 30 degrees.

FRETish:

If abs(roll_angle) >30 & roll_hold_mode_engagement, autopilot

scope condition component* shall* timing response*

Getting to the right requirement

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot **shall** set the roll hold reference to 30 degrees.

FRETish:

If abs(roll_angle) >30 & roll_hold_mode_engagement, autopilot shall

scope condition component* shall* timing response*

Getting to the right requirement

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall set the roll hold reference to 30.

FRETish:

If abs(roll_angle) >30 & roll_hold_mode_engagement, autopilot shall always

scope condition component* shall* timing response*

Getting to the right requirement

Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall **set the roll hold reference to 30**.

FRETish:

If abs(roll_angle) >30 & roll_hold_mode_engagement, autopilot shall always satisfy roll_hold_reference = 30

scope condition component* shall* timing response*

Getting to the right requirement

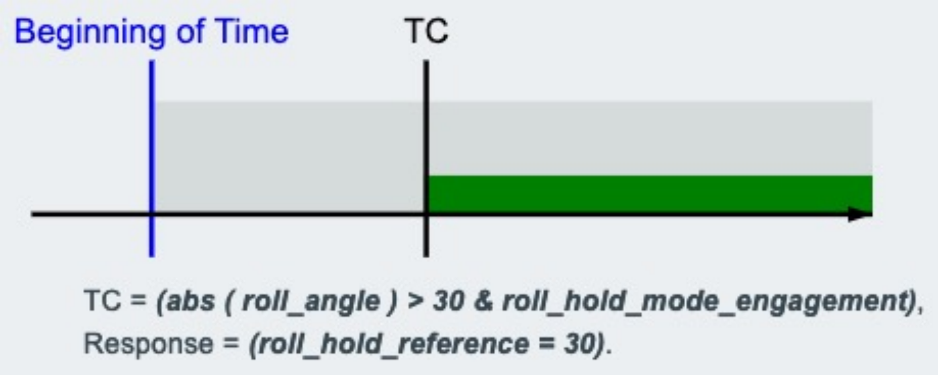
Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall set the roll hold reference to 30.

FRETish:

If abs(roll_angle) >30 & roll_hold_mode_engagement, autopilot shall always satisfy roll_hold_reference = 30



TC = *(abs (roll_angle) > 30 & roll_hold_mode_engagement)*,
Response = *(roll_hold_reference = 30)*.

Hmm, this is not what I mean..

scope condition component* shall* timing response*

Getting to the right requirement

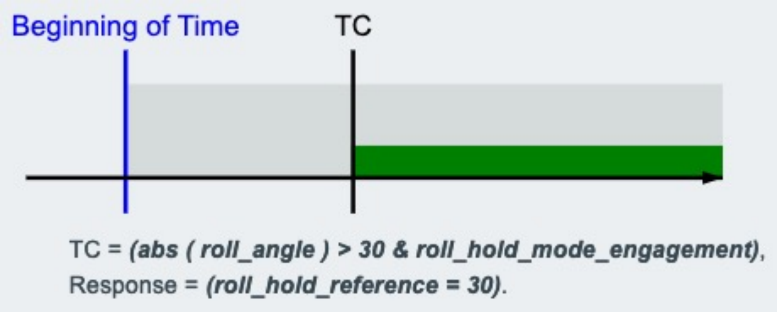
Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

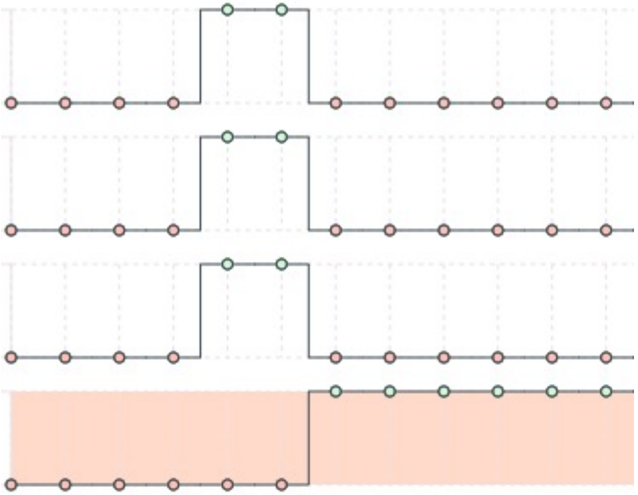
If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall set the roll hold reference to 30.

FRETish:

if abs(roll_angle) >30 & roll_hold_mode_engagement autopilot shall always satisfy roll_hold_reference = 30



TC = *(abs (roll_angle) > 30 & roll_hold_mode_engagement)*,
Response = *(roll_hold_reference = 30)*.



scope condition component* shall* timing response*

Getting to the right requirement

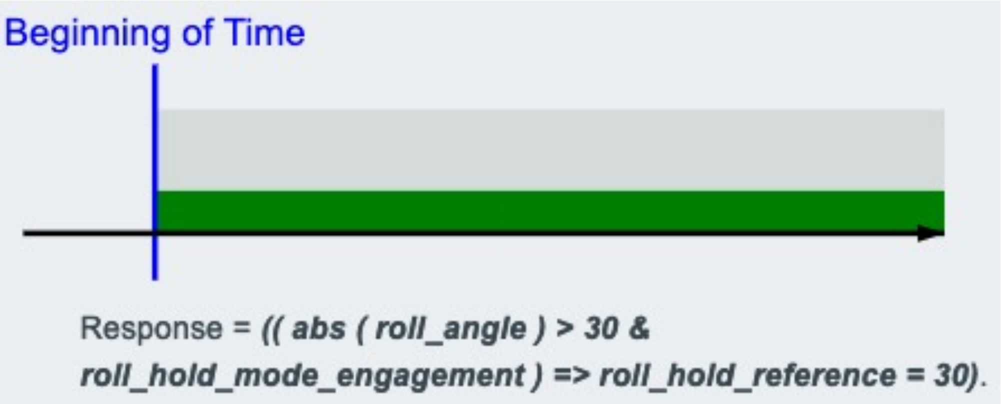
Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall set the roll hold reference to 30.

FRETish:

Autopilot shall always satisfy if (abs(roll_angle) >30 & roll_hold_mode_engagement) then roll_hold_reference = 30



Response = *((abs (roll_angle) > 30 & roll_hold_mode_engagement) => roll_hold_reference = 30)*.

scope condition component* shall* timing response*

Getting to the right requirement

Lockheed Martin Cyber-Physical System Challenge:

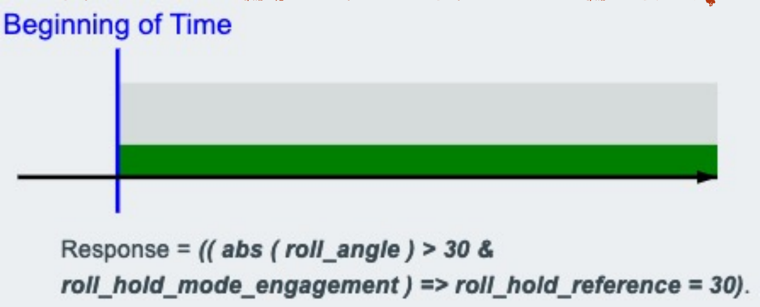
Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall set the roll hold reference to 30.

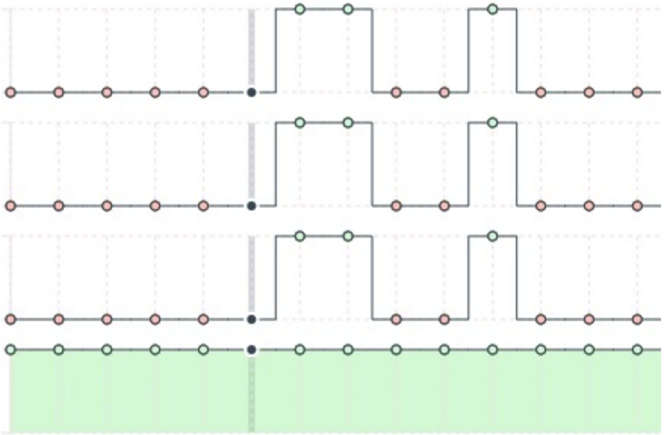
FRETish:

Autopilot shall always satisfy if (abs(roll_angle) >30 & roll_hold_mode_engagement) then roll_hold_reference = 30

what does that mean?



Response = *((abs (roll_angle) > 30 & roll_hold_mode_engagement) => roll_hold_reference = 30)*.



scope condition component* shall* timing response*

Getting to the right requirement

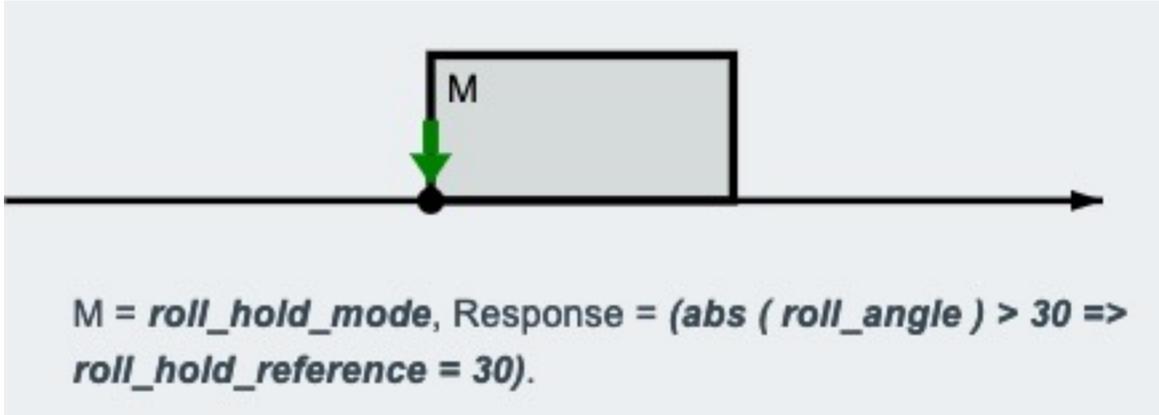
Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall set the roll hold reference to 30.

FRETish:

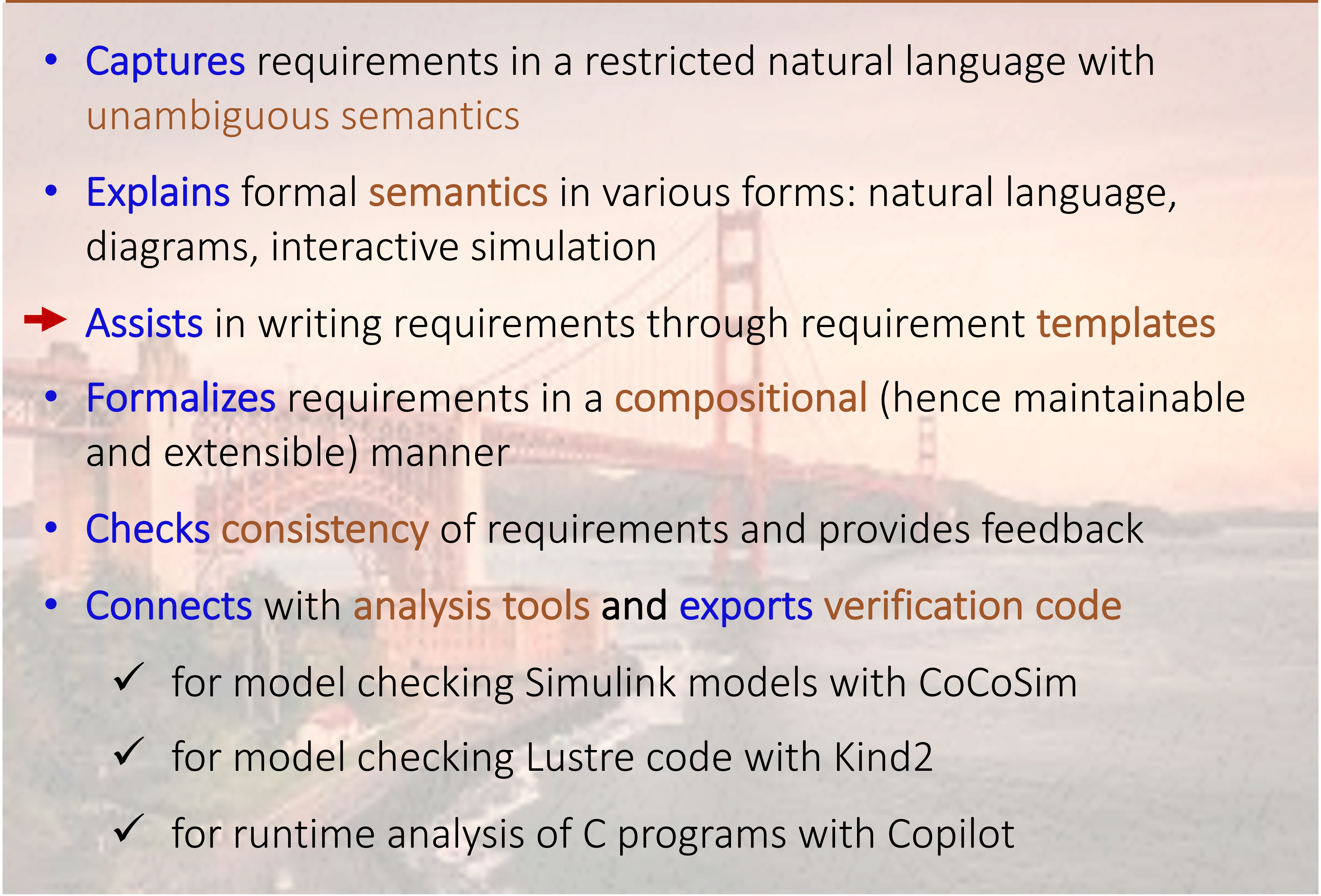
When in roll_hold_mode autopilot shall immediately satisfy if abs(roll_angle) >30 then roll_hold_reference = 30



M = ***roll_hold_mode***, Response = ***(abs (roll_angle) > 30 => roll_hold_reference = 30)***.

scope condition component* shall* timing response*

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 runtime analysis of C programs with Copilot

Requirement templates

Lockheed Martin Cyber-Physical System Challenge, component FSM:

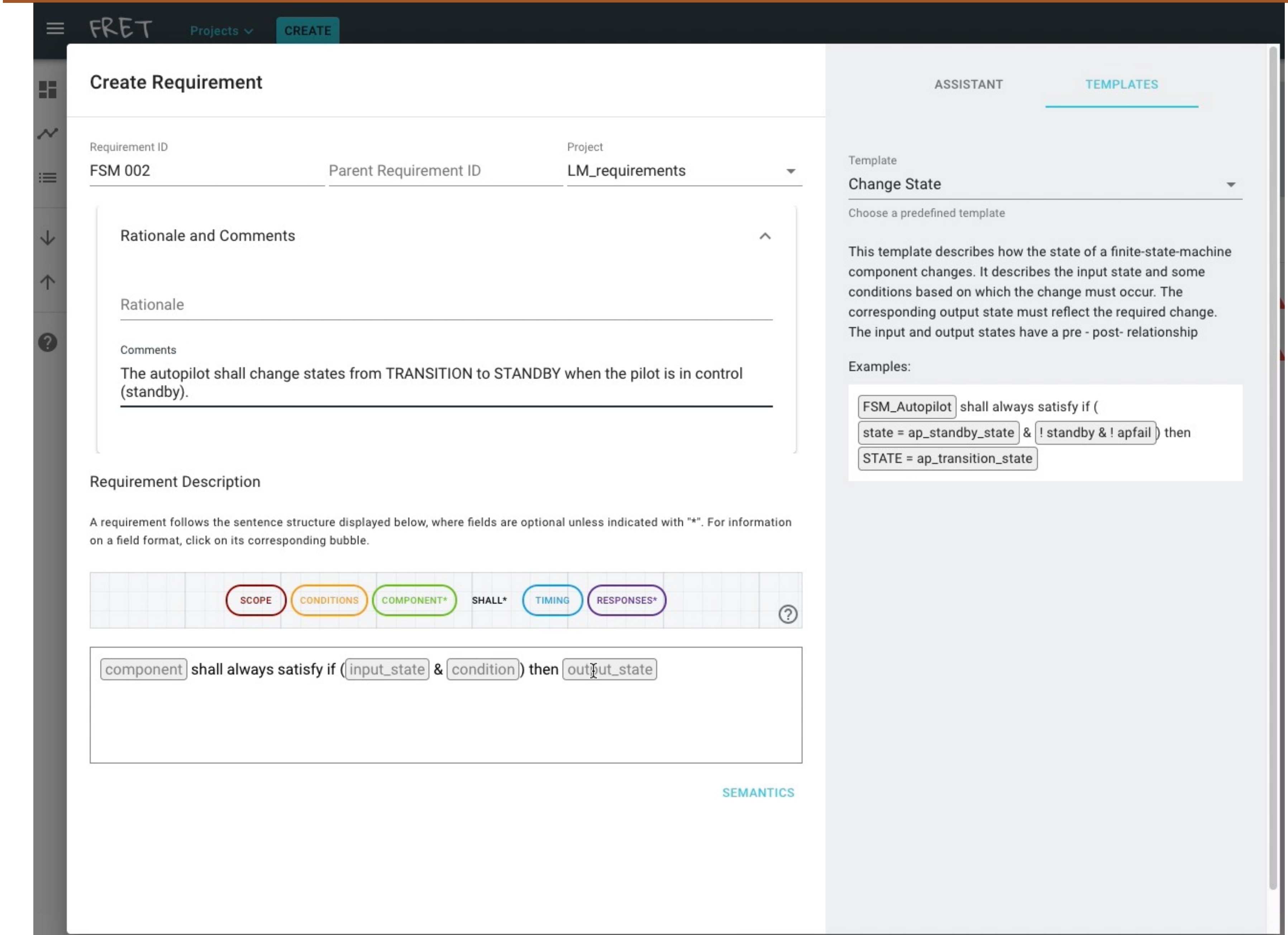
- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data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



Create Requirement

Requirement ID: FSM 002

Parent Requirement ID

Project: LM_requirements

Rationale and Comments

Rationale

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

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.

component shall always satisfy if (input_state & condition) then output_state

Template: Change State

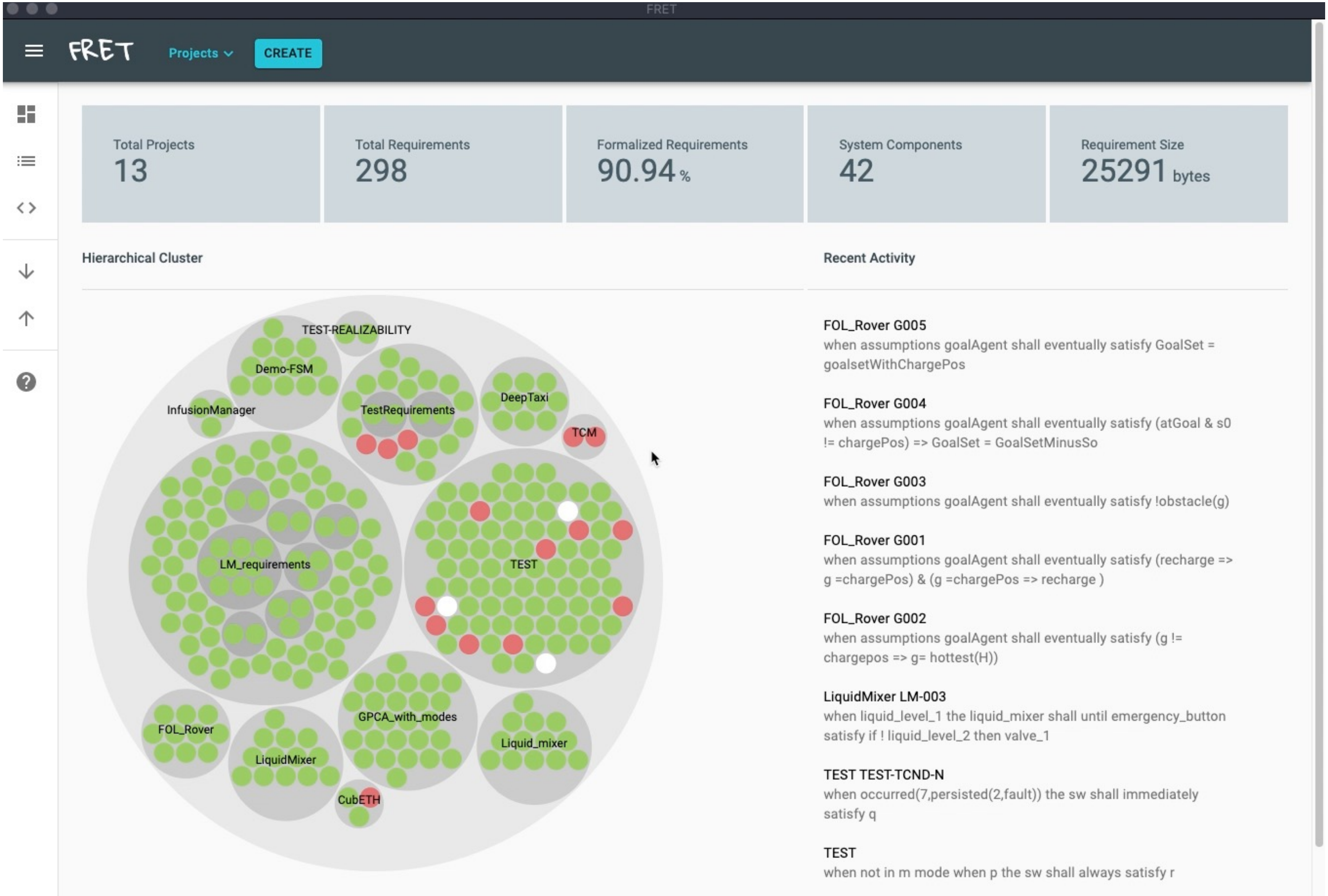
Choose a predefined template

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

Examples:

FSM_Autopilot shall always satisfy if (state = ap_standby_state & ! standby & ! apfail) then STATE = ap_transition_state

Exporting Requirements



Total Projects	Total Requirements	Formalized Requirements	System Components	Requirement Size
13	298	90.94 %	42	25291 bytes

Recent Activity

FOL_Rover G005
when assumptions goalAgent shall eventually satisfy GoalSet = goalsetWithChargePos

FOL_Rover G004
when assumptions goalAgent shall eventually satisfy (atGoal & s0 != chargePos) => GoalSet = GoalSetMinusSo

FOL_Rover G003
when assumptions goalAgent shall eventually satisfy !obstacle(g)

FOL_Rover G001
when assumptions goalAgent shall eventually satisfy (recharge => g =chargePos) & (g =chargePos => recharge)

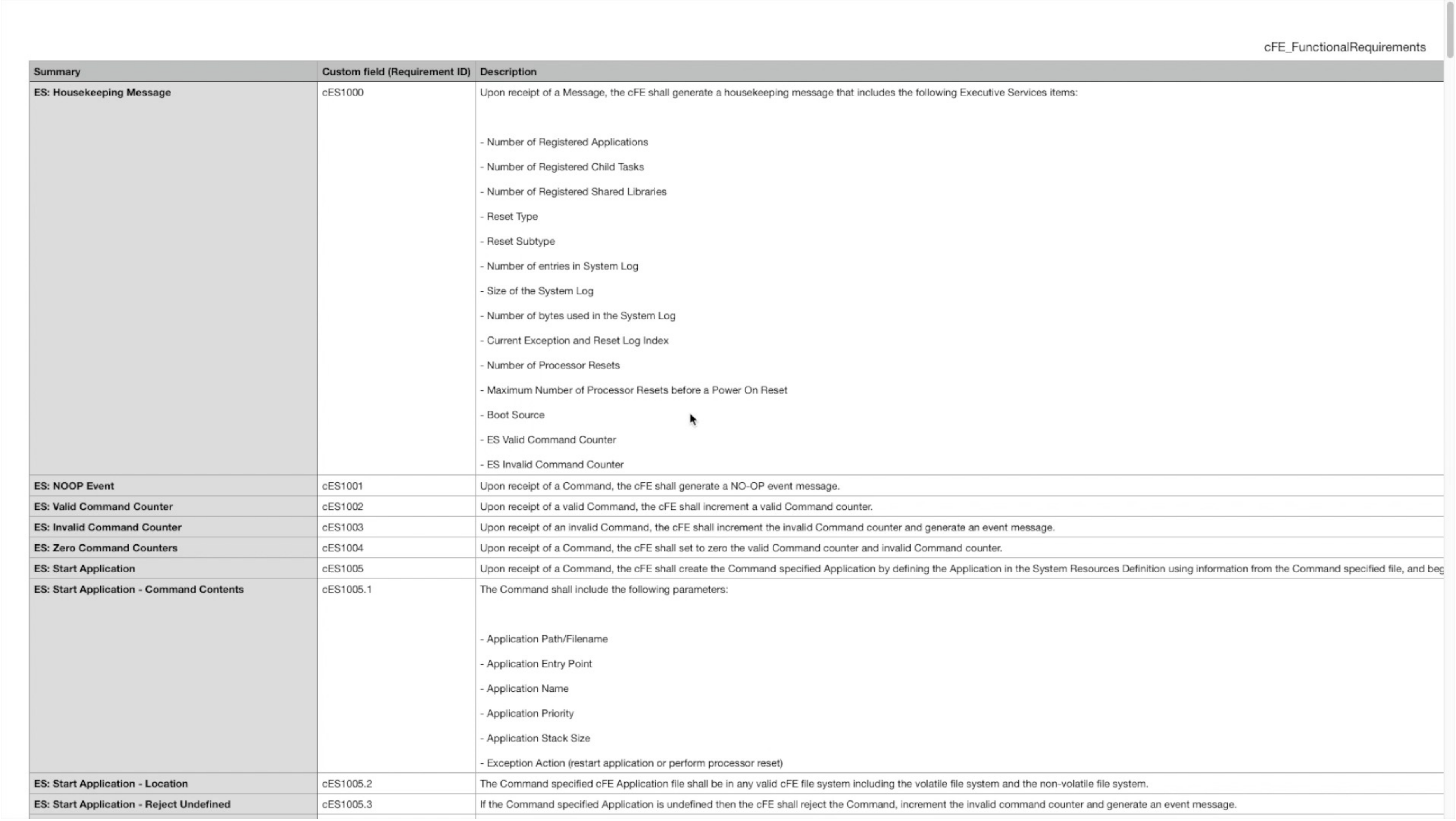
FOL_Rover G002
when assumptions goalAgent shall eventually satisfy (g != chargepos => g= hottest(H))

LiquidMixer LM-003
when liquid_level_1 the liquid_mixer shall until emergency_button satisfy if ! liquid_level_2 then valve_1

TEST TEST-TCND-N
when occurred(7,persisted(2,fault)) the sw shall immediately satisfy q

TEST
when not in m mode when p the sw shall always satisfy r

Importing Requirements

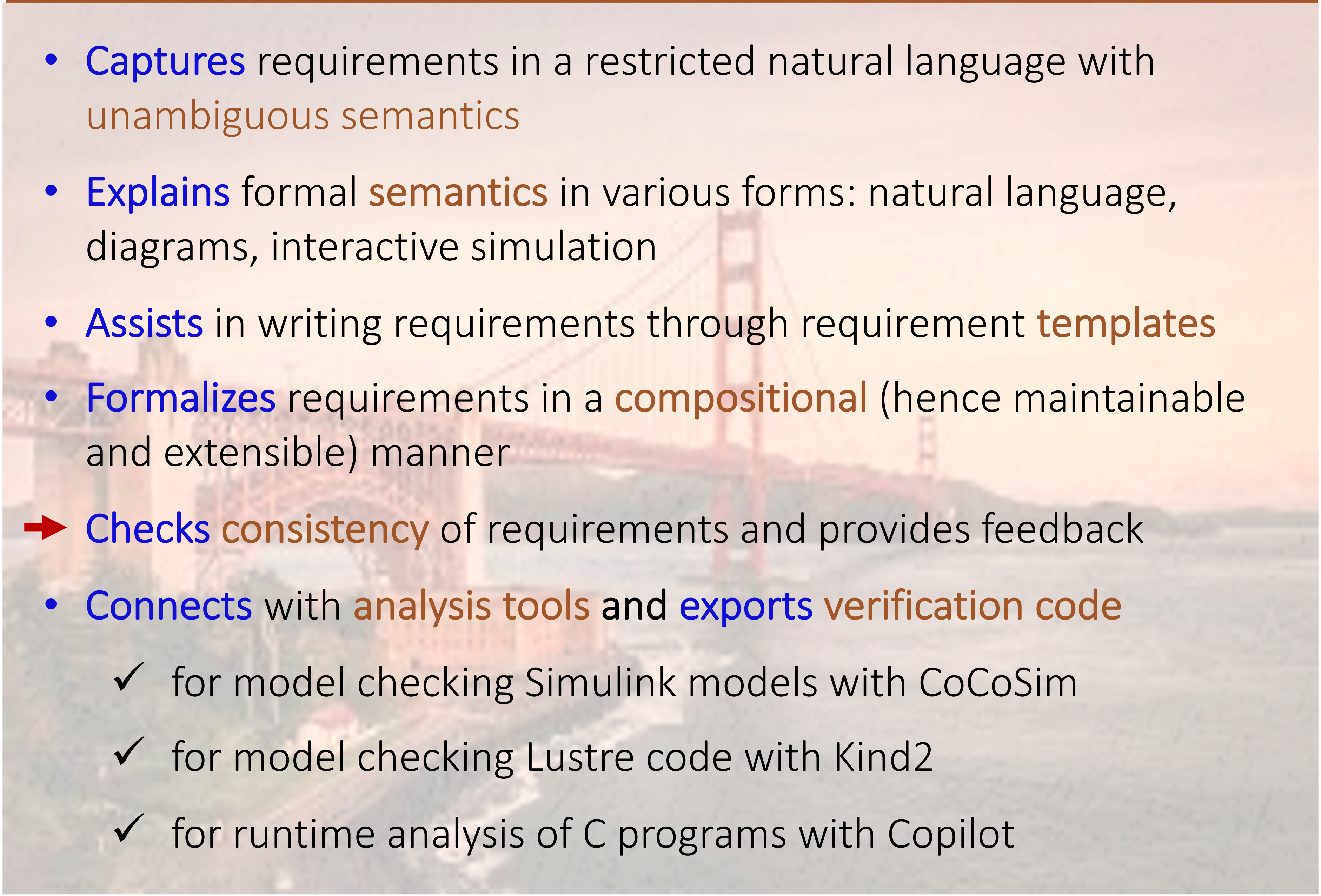


cFE_FunctionalRequirements

Summary	Custom field (Requirement ID)	Description
ES: Housekeeping Message	cES1000	Upon receipt of a Message, the cFE shall generate a housekeeping message that includes the following Executive Services items: - Number of Registered Applications - Number of Registered Child Tasks - Number of Registered Shared Libraries - Reset Type - Reset Subtype - Number of entries in System Log - Size of the System Log - Number of bytes used in the System Log - Current Exception and Reset Log Index - Number of Processor Resets - Maximum Number of Processor Resets before a Power On Reset - Boot Source - ES Valid Command Counter - ES Invalid Command Counter
ES: NOOP Event	cES1001	Upon receipt of a Command, the cFE shall generate a NO-OP event message.
ES: Valid Command Counter	cES1002	Upon receipt of a valid Command, the cFE shall increment a valid Command counter.
ES: Invalid Command Counter	cES1003	Upon receipt of an invalid Command, the cFE shall increment the invalid Command counter and generate an event message.
ES: Zero Command Counters	cES1004	Upon receipt of a Command, the cFE shall set to zero the valid Command counter and invalid Command counter.
ES: Start Application	cES1005	Upon receipt of a Command, the cFE shall create the Command specified Application by defining the Application in the System Resources Definition using information from the Command specified file, and beg
ES: Start Application - Command Contents	cES1005.1	The Command shall include the following parameters: - Application Path/Filename - Application Entry Point - Application Name - Application Priority - Application Stack Size - Exception Action (restart application or perform processor reset)
ES: Start Application - Location	cES1005.2	The Command specified cFE Application file shall be in any valid cFE file system including the volatile file system and the non-volatile file system.
ES: Start Application - Reject Undefined	cES1005.3	If the Command specified Application is undefined then the cFE shall reject the Command, increment the invalid command counter and generate an event message.

cFE requirements publicly available:
https://github.com/nasa/cFE/blob/main/docs/cFE_FunctionalRequirements.csv

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 runtime analysis of C programs with Copilot

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.

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 ✓

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.

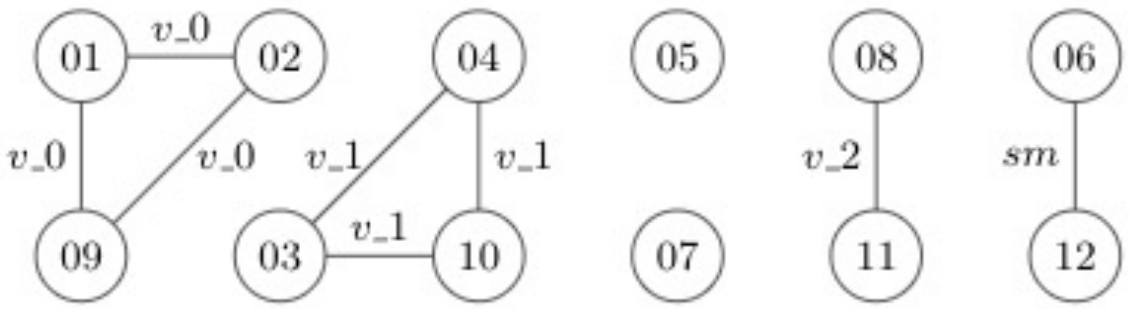
Input state: TRANSITION ✓
Condition 1: pilot is in control ✓
Condition 2: system is supported ✓
sensor data is good ✓
Output state 1: STANDBY
Output state 2: NOMINAL 🚫

FRET takes it a step further

- Realizability checking can be challenging
  - Nested quantifiers for solvers
  - Infinite-state problems are undecidable
  - Non-linear expressions (not entirely supported by SMT solvers)
- A novel approach for compositional realizability checking
  - Smaller, more tractable parts: partial specifications
- Automatically partitions a global specification into partial ones
- We proved that
  - Checking that a global spec is realizable reduces to checking partial specs
- Implementation and diagnostic analysis within FRET
  - Visualization of conflicts
  - Simulation of conflicting requirements through counterexamples

Compositional realizability checking

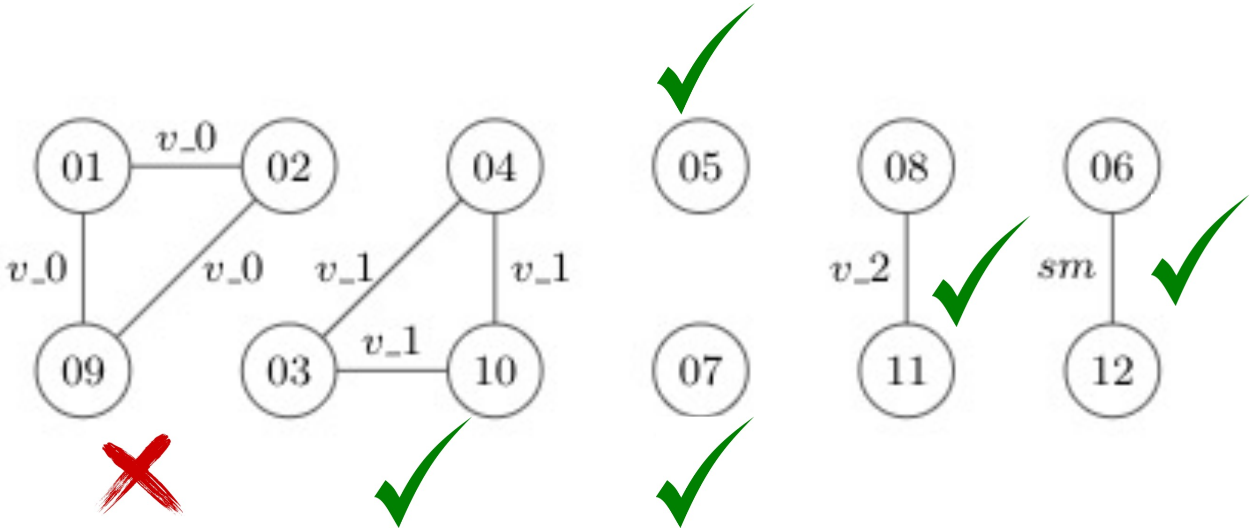
- Requirements graph
  - Each vertex corresponds to a requirement
  - If variables referenced by two requirements, their vertices are connected
  - Connected components represent partial specifications:
    - Sets of requirements that can be analyzed independently



6 connected components from 12 requirements

Compositional realizability checking

- Successful decomposition
  - Effectively reduces problem complexity
  - Surpasses challenges
  - Leads to significant performance benefits



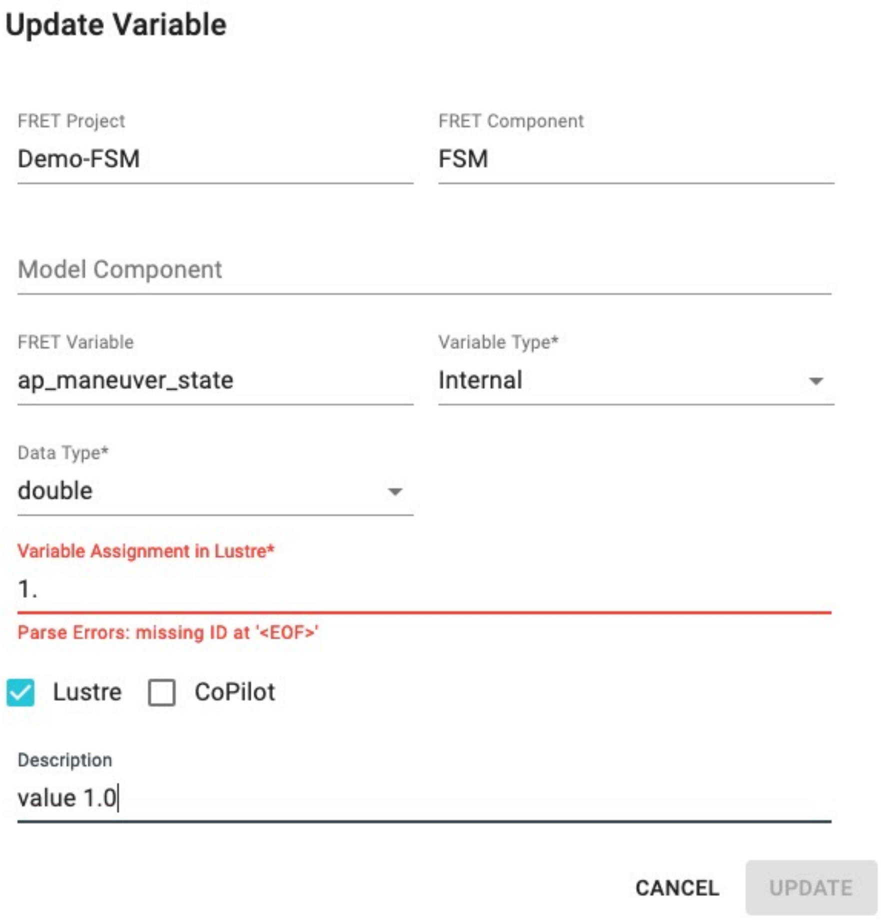
Variable declaration

Variable Type:

- Input: the system monitors the variable
- Output: the system controls the variable
- Internal: a macro for a Lustre expression

Datatype

Boolean, integer, double, unsigned integer, single



Update Variable

FRET Project: Demo-FSM

FRET Component: FSM

Model Component

FRET Variable: ap_maneuver_state

Variable Type*: Internal

Data Type*: double

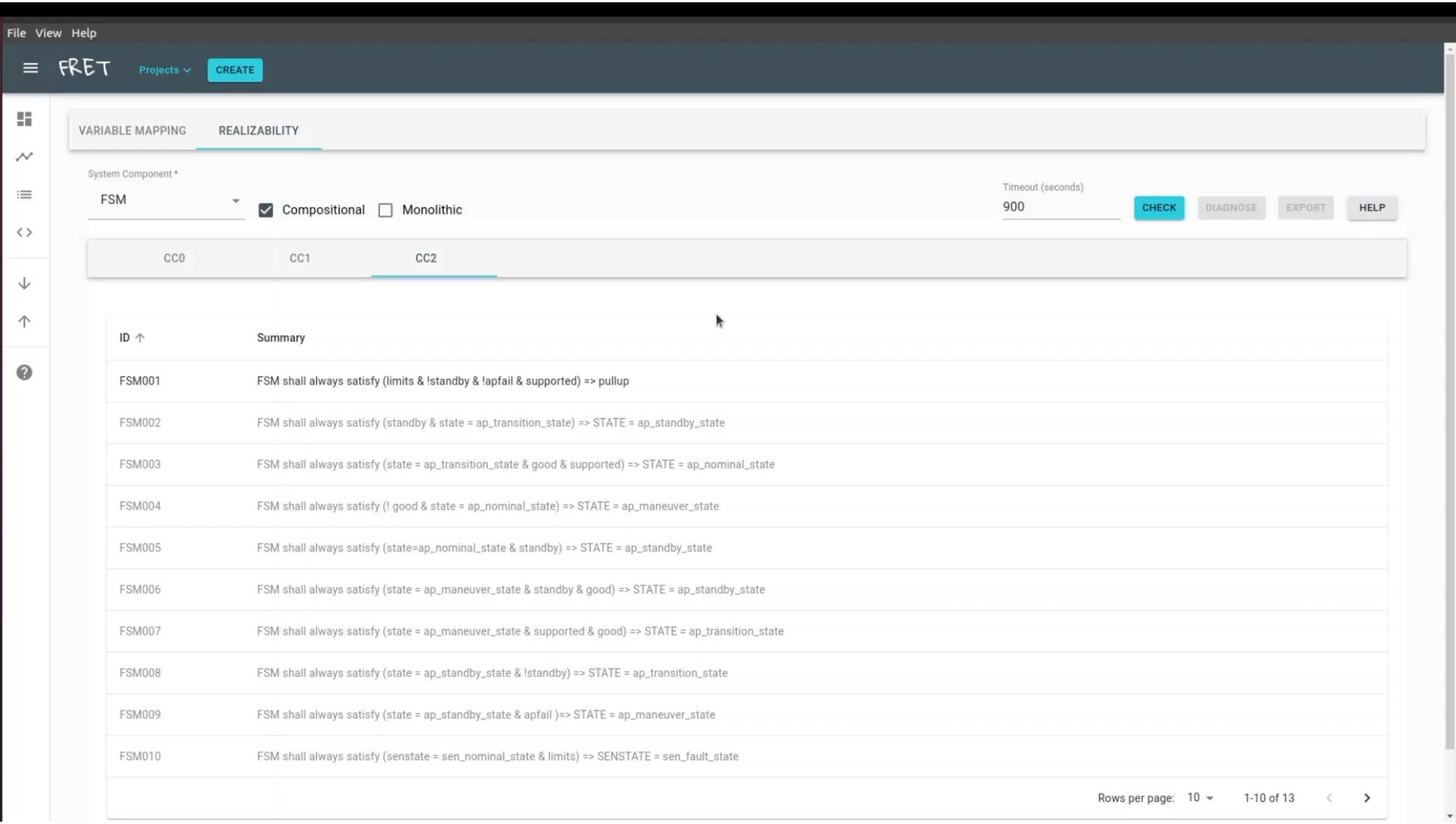
Variable Assignment in Lustre*: 1.

Parse Errors: missing ID at '<EOF>'

☑ Lustre ☐ CoPilot

Description: value 1.0

Checking realizability



System Component * FSM

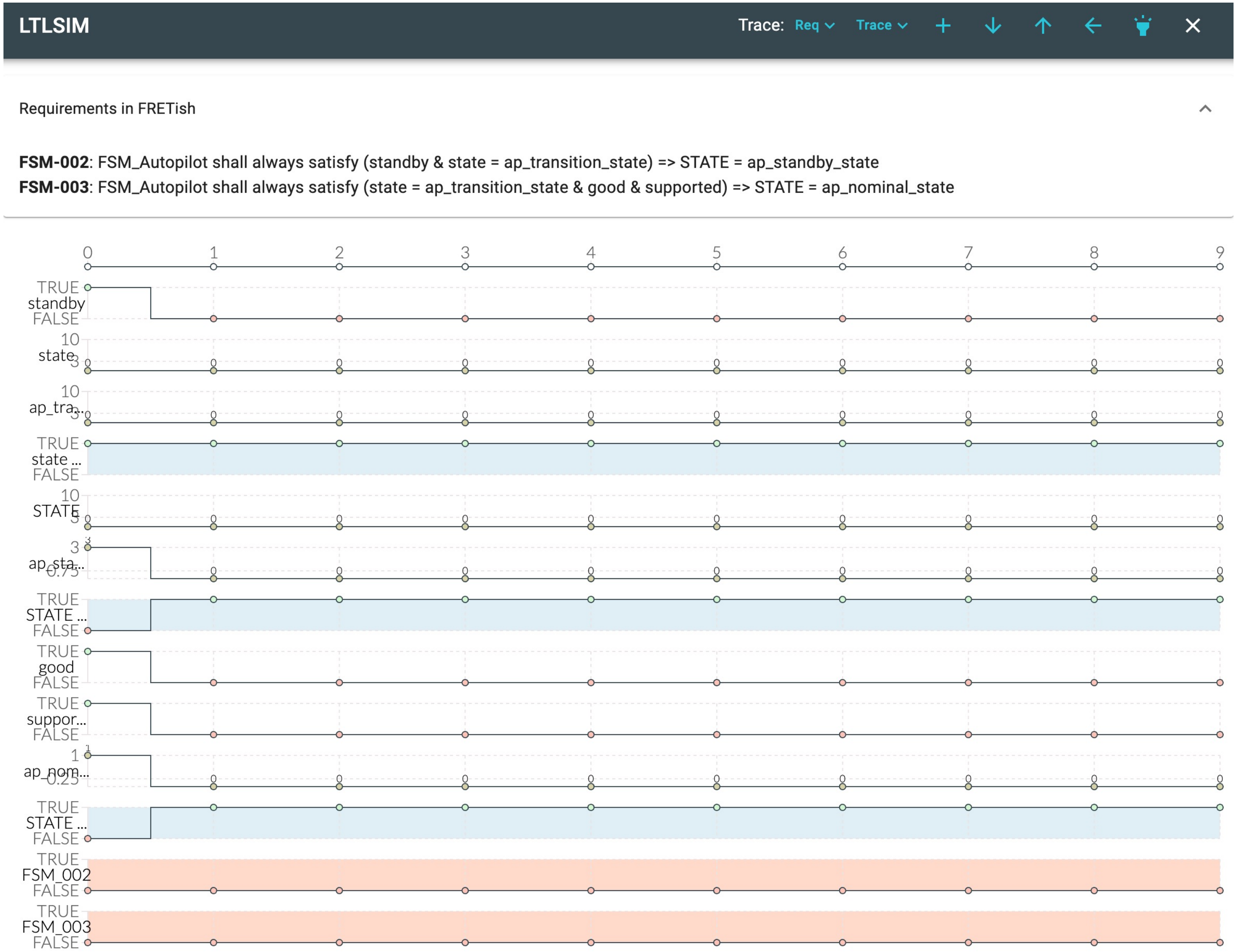
☑ Compositional ☐ Monolithic

Timeout (seconds) 900

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

Simulation of conflicting requirements



FSM-002: FSM_Autopilot shall always satisfy (standby & state = ap_transition_state) => STATE = ap_standby_state
FSM-003: FSM_Autopilot shall always satisfy (state = ap_transition_state & good & supported) => STATE = ap_nominal_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

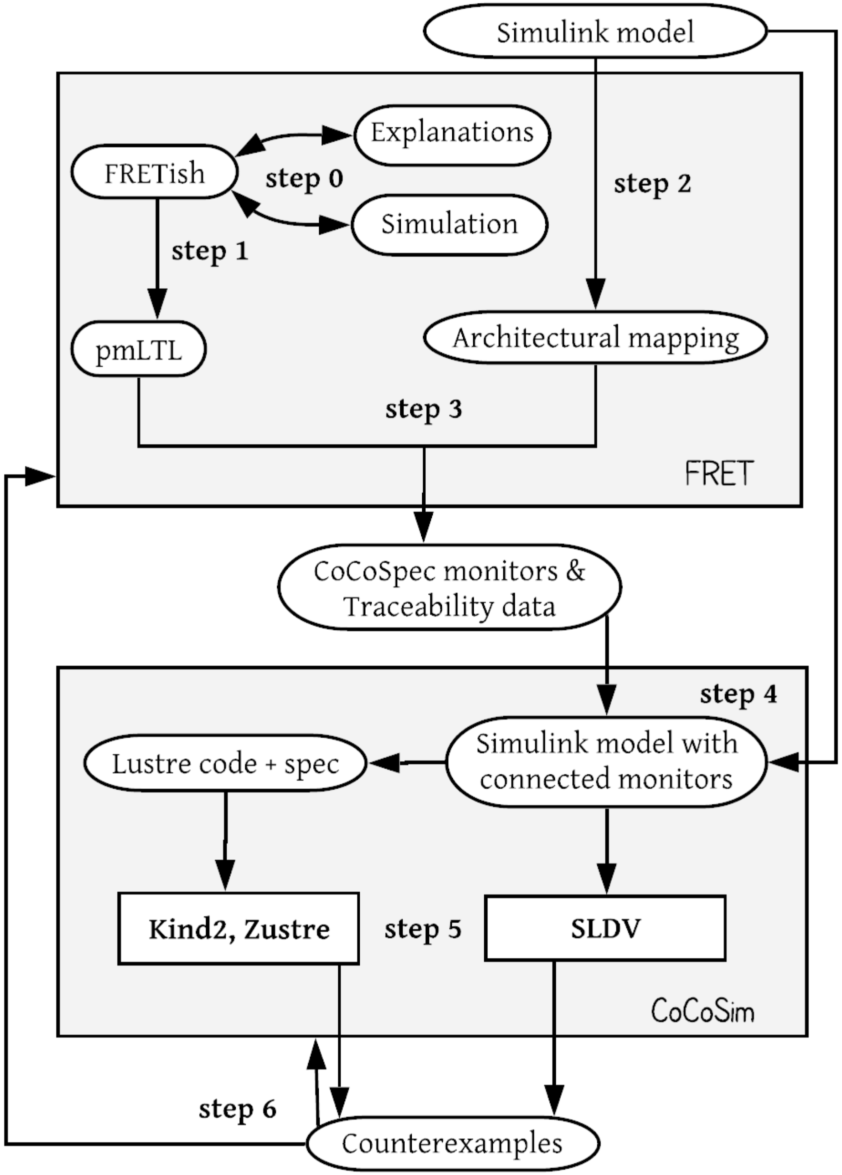
  - ✓ for model checking Simulink models with CoCoSim
  - ✓ for model checking Lustre code with Kind2
  - ✓ for runtime analysis of C programs with Copilot

The FRET-CoCoSim Integrated Framework

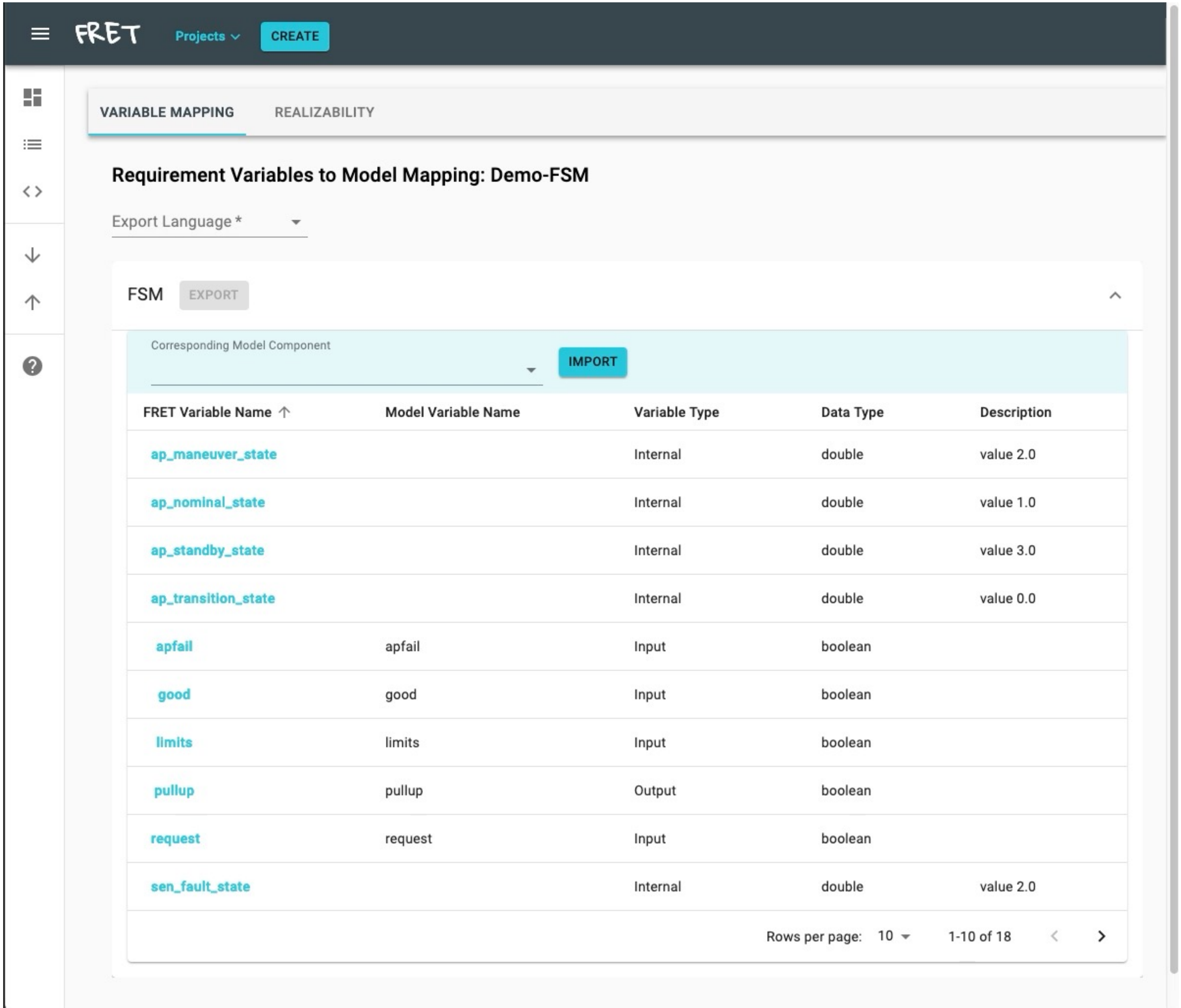
- Elicit, explain, and formalize the semantics of the given natural language requirements
  (Steps: 0, 1)

- Generate verification code and monitors that can be automatically attached to the Simulink models
  (Steps: 2, 3, 4)

- Perform verification by using Lustre-based model checkers or SLDV
  (Steps: 5, 6)



FRET-CoCoSim



Requirement Variables to Model Mapping: Demo-FSM

Export Language *

FSM EXPORT

Corresponding Model Component

IMPORT

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

Rows per page: 10 1-10 of 18

Generation of Simulink Monitors

Lockheed Martin Cyber-Physical System Challenge:

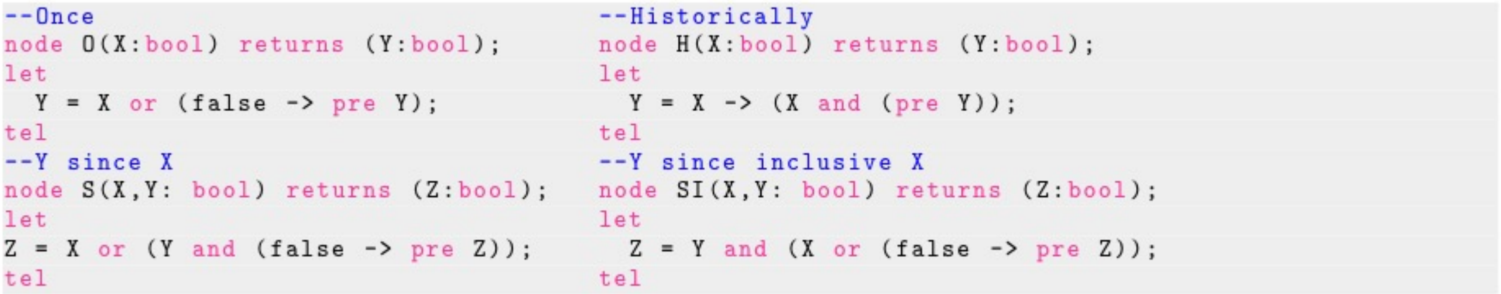
Natural language requirement:

If the roll angle is greater than 30 degrees at the time of roll hold mode engagement, the autopilot shall set the roll hold reference to 30.

FRETish:

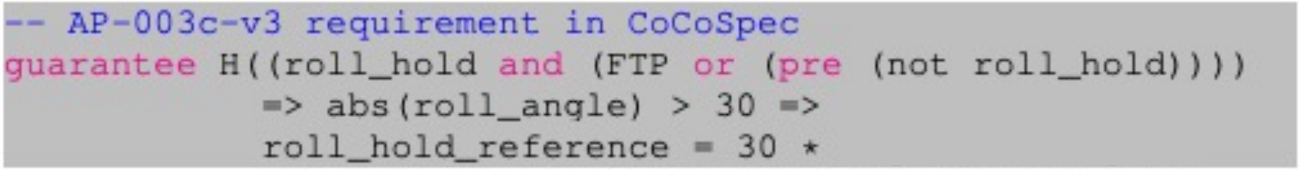
When in roll_hold_mode autopilot shall immediately satisfy if abs(roll_angle) >30 then roll_hold_reference = 30

CoCoSpec specification:



```
--Once
node O(X:bool) returns (Y:bool);
let
  Y = X or (false -> pre Y);
tel
--Y since X
node S(X,Y: bool) returns (Z:bool);
let
Z = X or (Y and (false -> pre Z));
tel
```

```
--Historically
node H(X:bool) returns (Y:bool);
let
  Y = X -> (X and (pre Y));
tel
--Y since inclusive X
node SI(X,Y: bool) returns (Z:bool);
let
  Z = Y and (X or (false -> pre Z));
tel
```



```
-- AP-003c-v3 requirement in CoCoSpec
guarantee H((roll_hold and (FTP or (pre (not roll_hold)))
            => abs(roll_angle) > 30 =>
            roll_hold_reference = 30 *
```

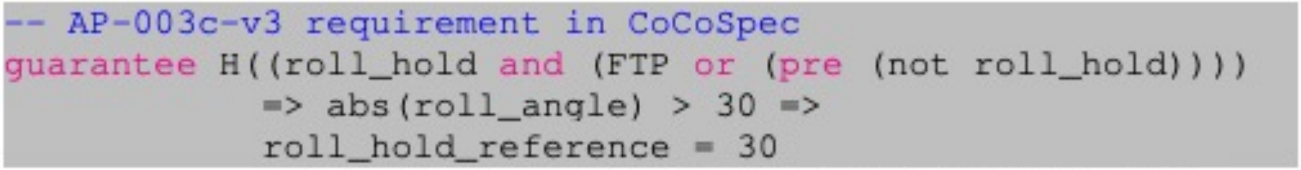
Generation of Simulink Monitors

Lockheed Martin Cyber-Physical System Challenge:

FRETish:

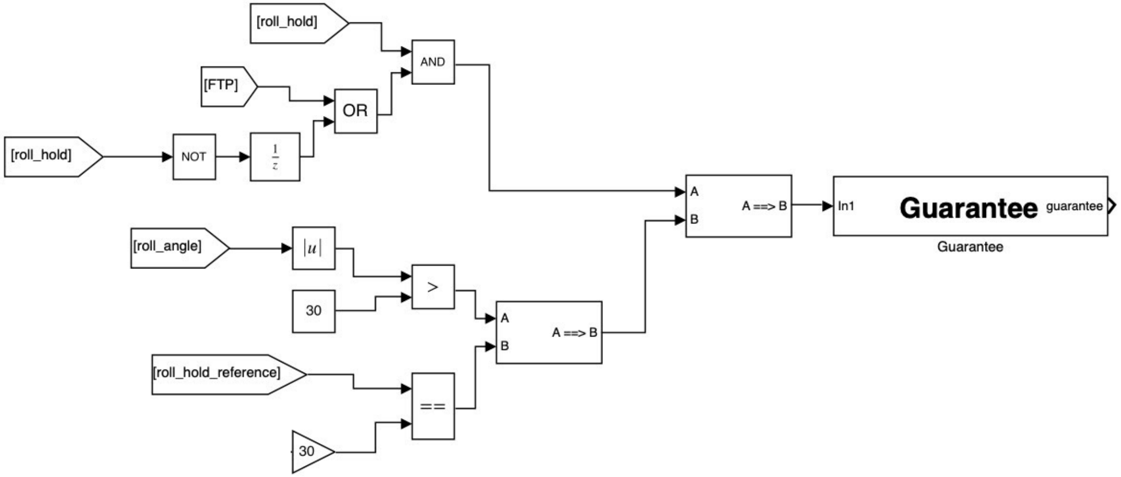
When in roll_hold_mode autopilot shall immediately satisfy if abs(roll_angle) >30 then roll_hold_reference = 30

CoCoSpec specification:



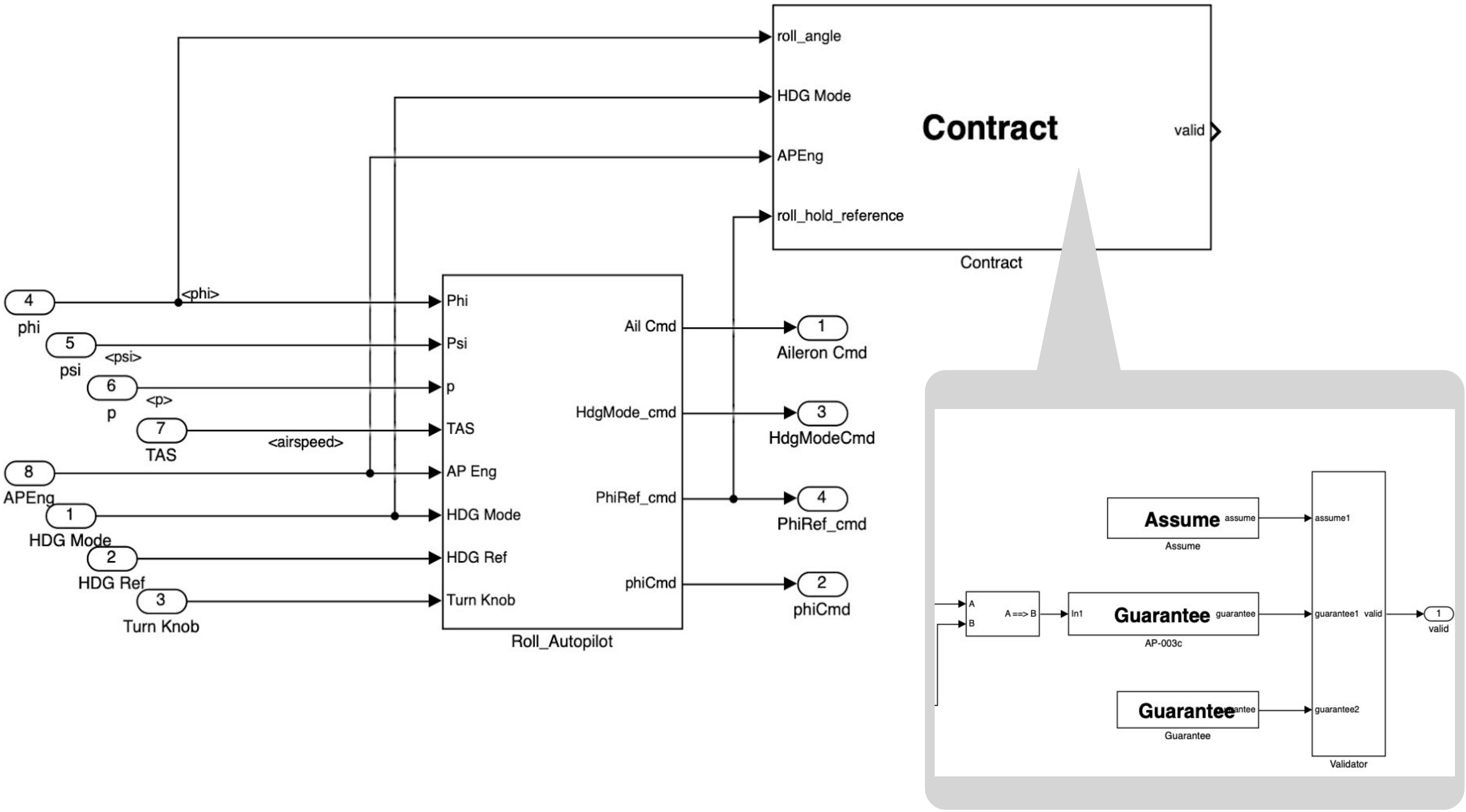
```
-- AP-003c-v3 requirement in CoCoSpec
guarantee H((roll_hold and (FTP or (pre (not roll_hold)))
            => abs(roll_angle) > 30 =>
            roll_hold_reference = 30
```

Simulink monitor

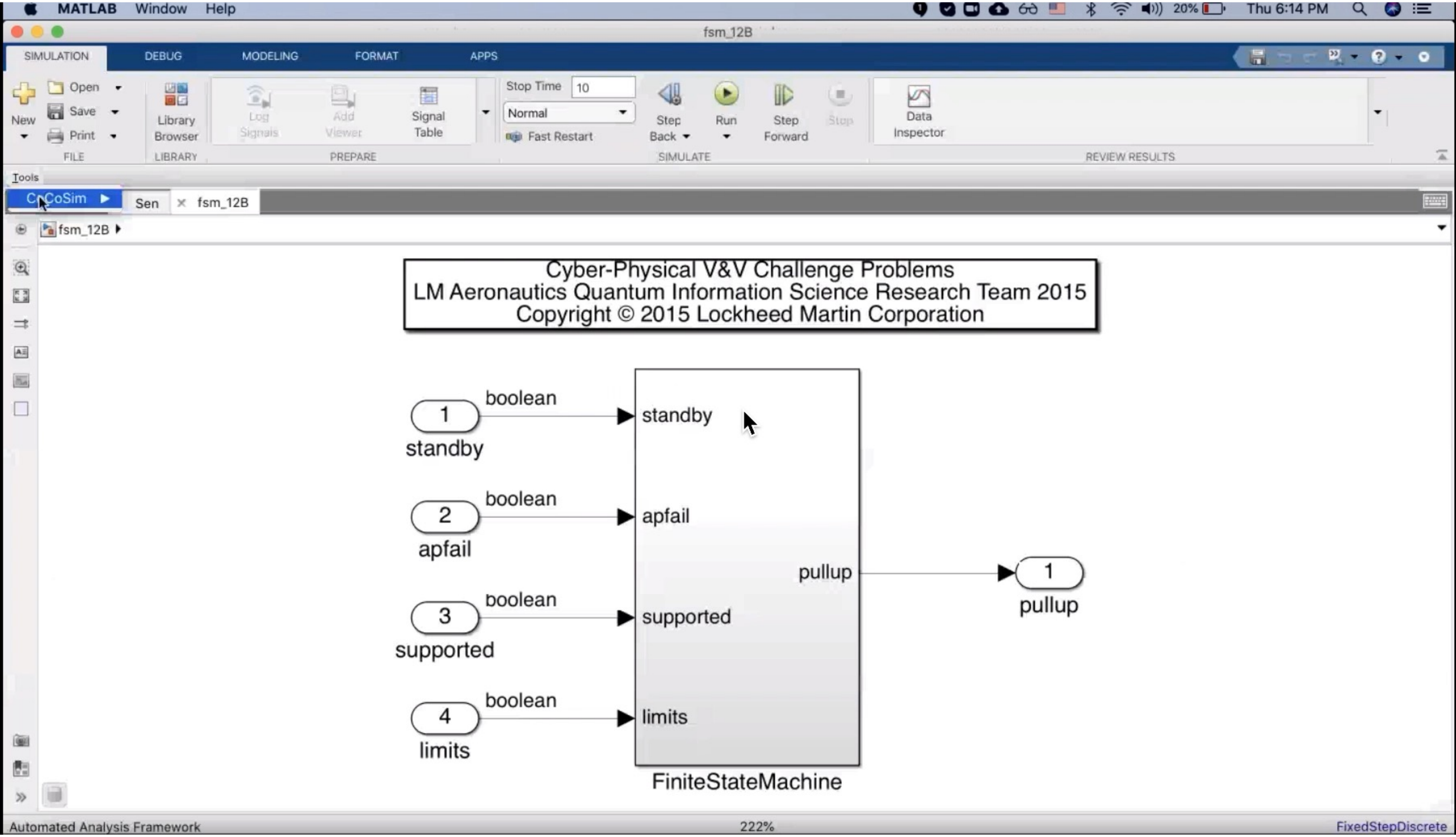


Generation of Simulink Monitors

Simulink monitor automatically attached on the model:

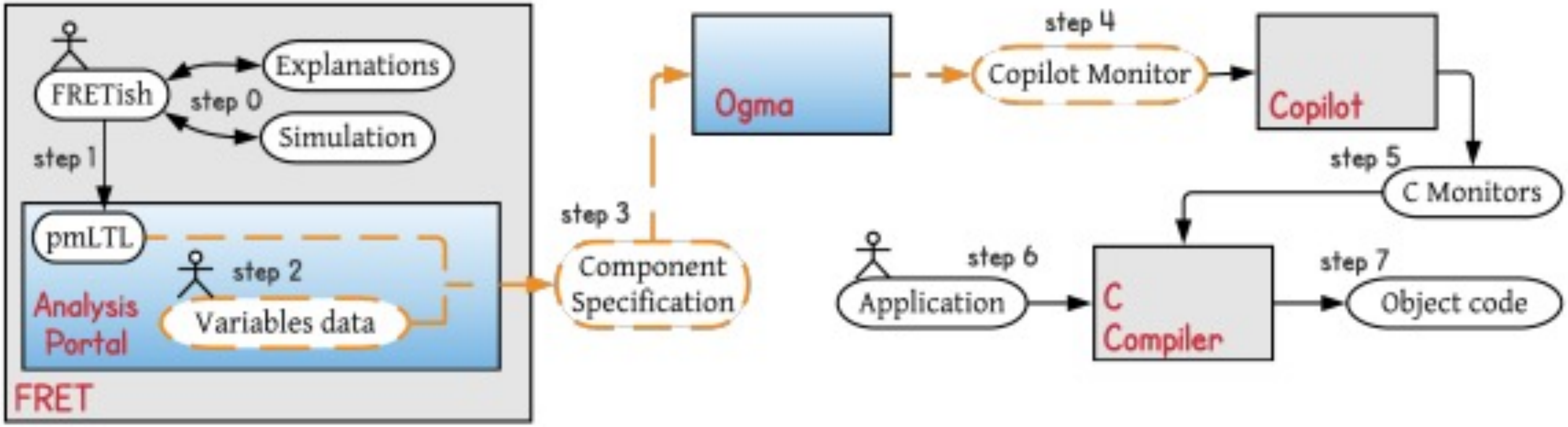


Connection with CoCoSim



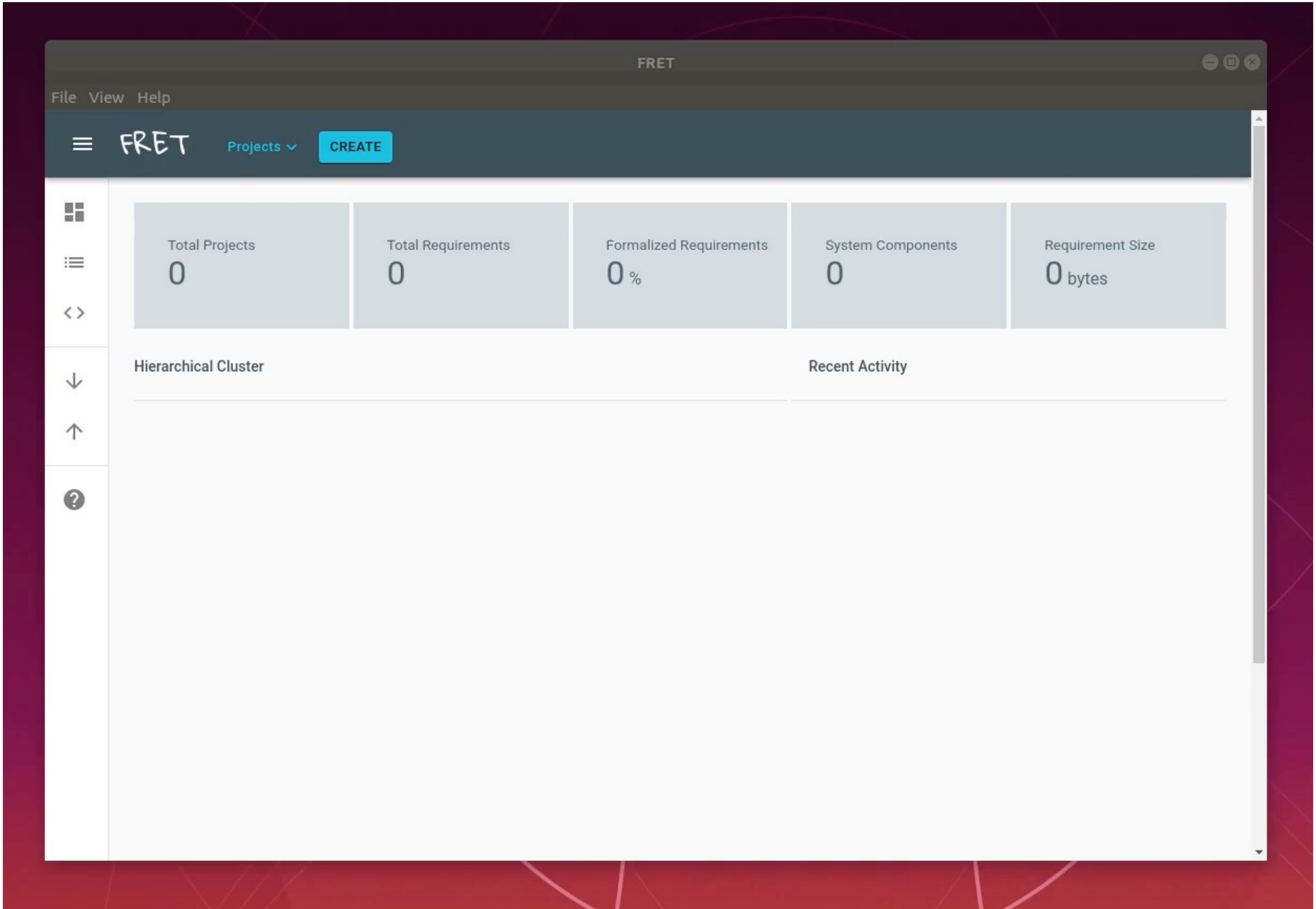
Cyber-Physical V&V Challenge Problems
LM Aeronautics Quantum Information Science Research Team 2015
Copyright © 2015 Lockheed Martin Corporation

The FRET-Copilot Integrated Framework



Copilot is a high-level runtime verification framework that generates hard real-time C99 code. **Ogma** takes the FRET generated specifications and translates them into Copilot monitors.

Connection with Copilot

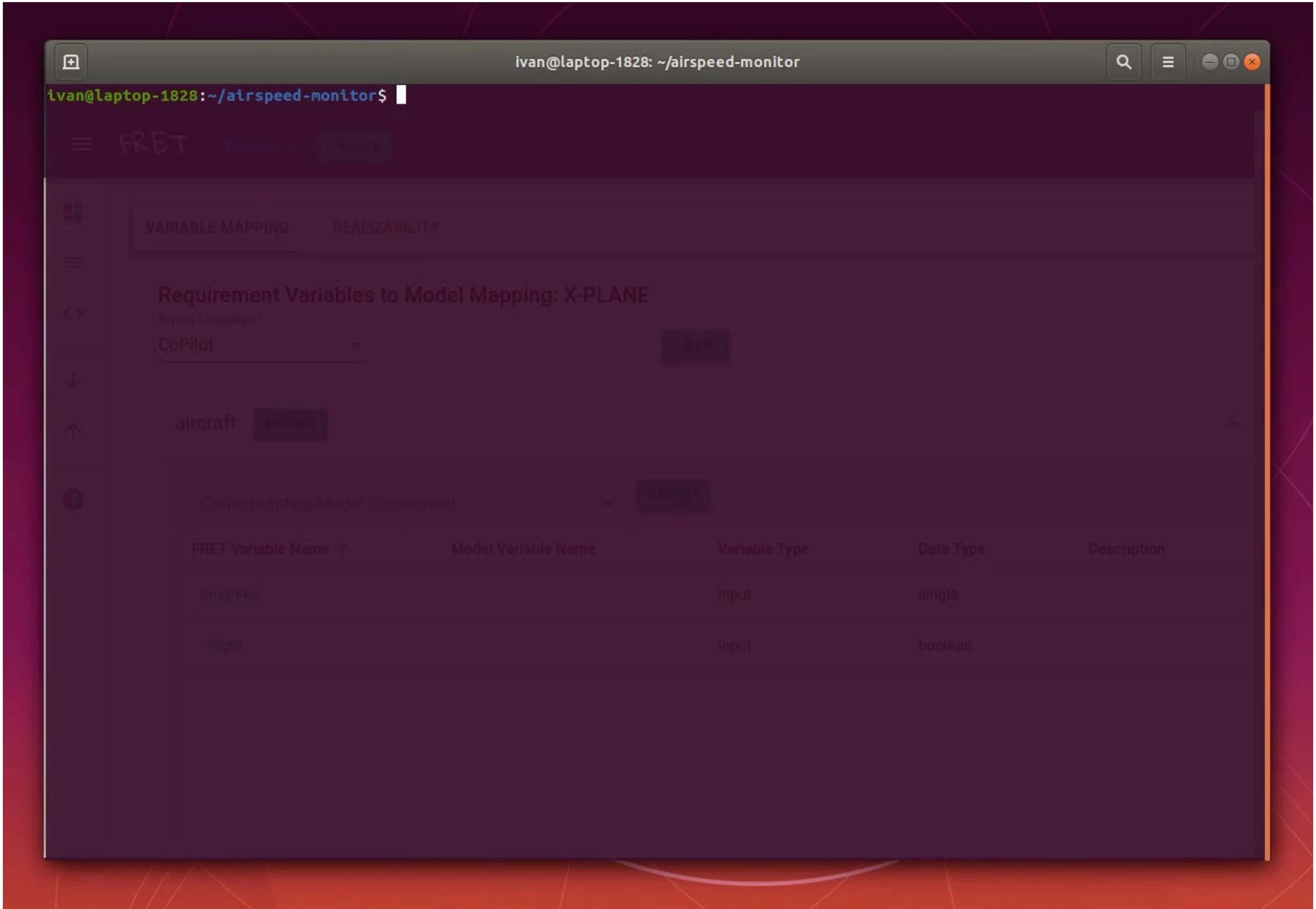


Total Projects	Total Requirements	Formalized Requirements	System Components	Requirement Size
0	0 %	0	0	0 bytes

Hierarchical Cluster

Recent Activity

Connection With Copilot



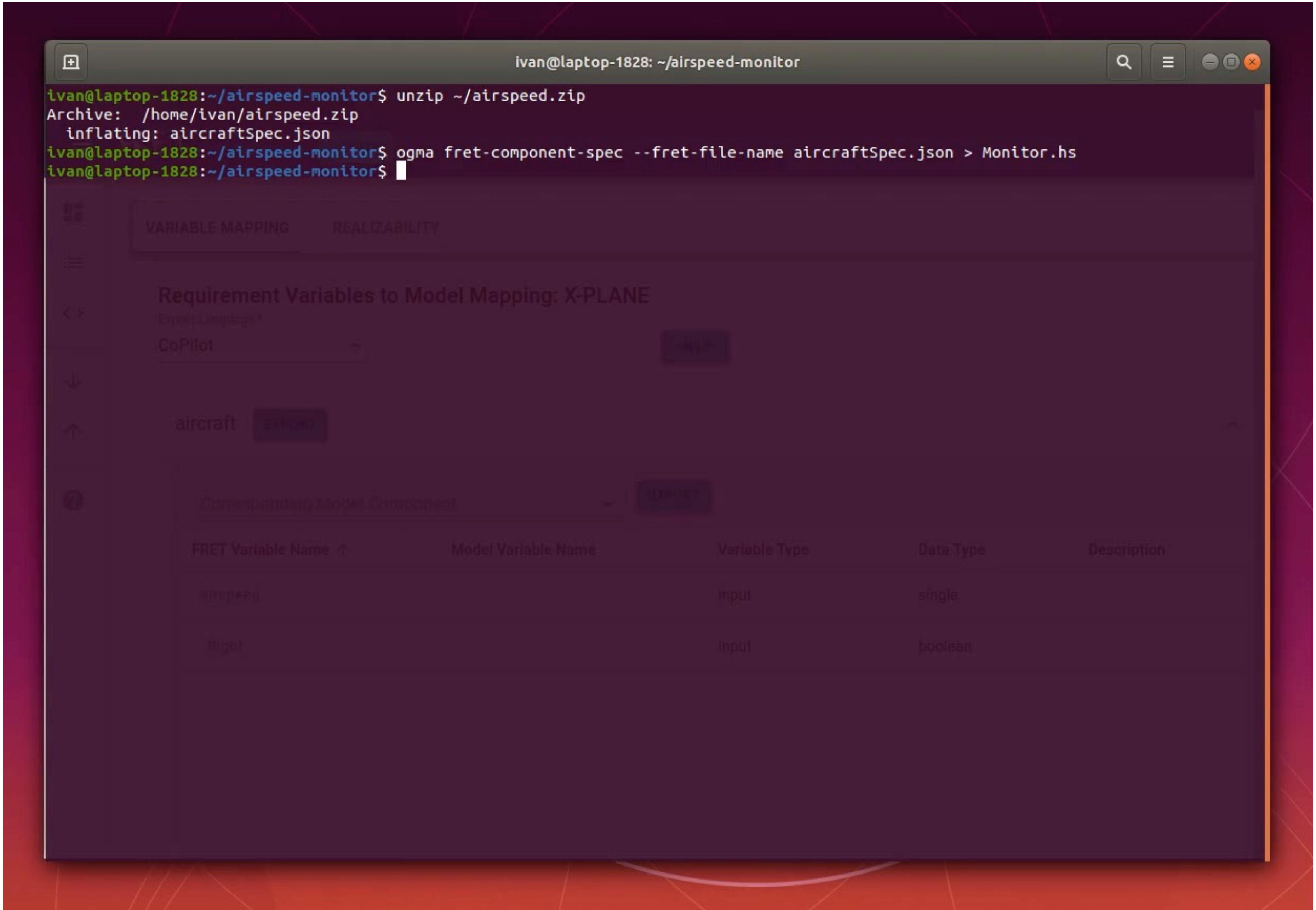
ivan@laptop-1828: ~/airspeed-monitor
FRET
VARIABLE MAPPING
REALIZABILITY
CoPilot
aircraft
EXPORT
Corresponding Model Component

```
ivan@laptop-1828:~/airspeed-monitor$ 
```

Requirement Variables to Model Mapping: X-PLANE

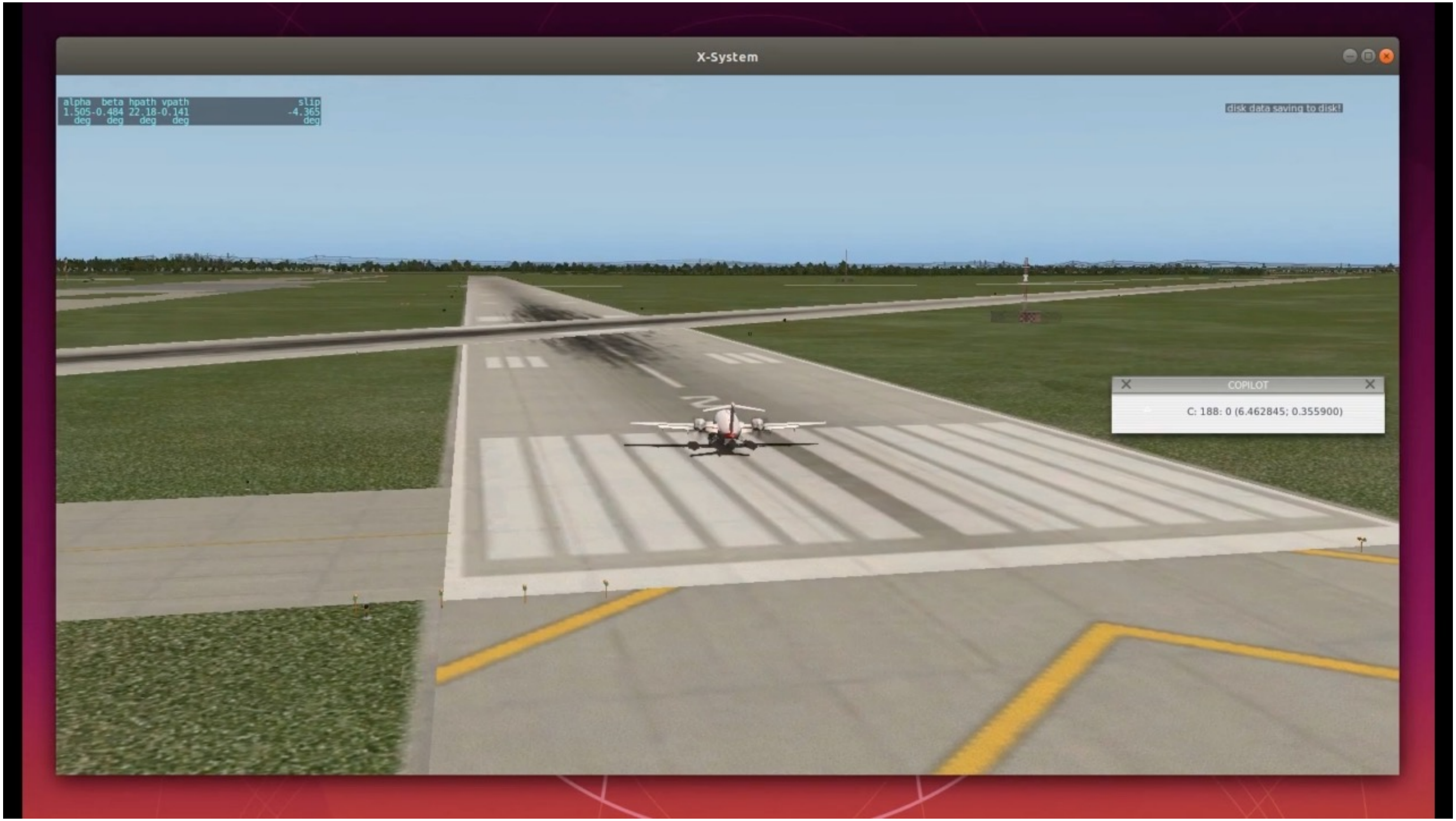
FRET Variable Name ↑	Model Variable Name	Variable Type	Data Type	Description
airspeed		input	single	
flight		input	boolean	

Connection With Copilot



```
ivan@laptop-1828:~/airspeed-monitor$ unzip ~/airspeed.zip
Archive:  /home/ivan/airspeed.zip
  inflating: aircraftSpec.json
ivan@laptop-1828:~/airspeed-monitor$ ogma fret-component-spec --fret-file-name aircraftSpec.json > Monitor.hs
ivan@laptop-1828:~/airspeed-monitor$ 
```

Connection With Copilot

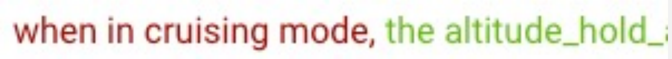


```
alpha  beta hpath vpath                     slip
1.505-0.484 22.18-0.141                   -4.365
  deg   deg   deg   deg                      deg
```

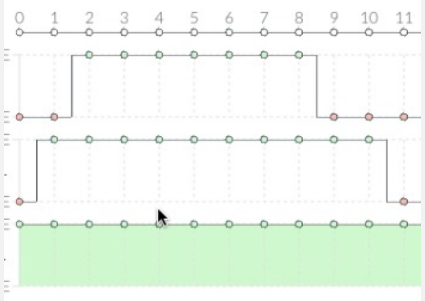
```
disk data saving to disk!
```

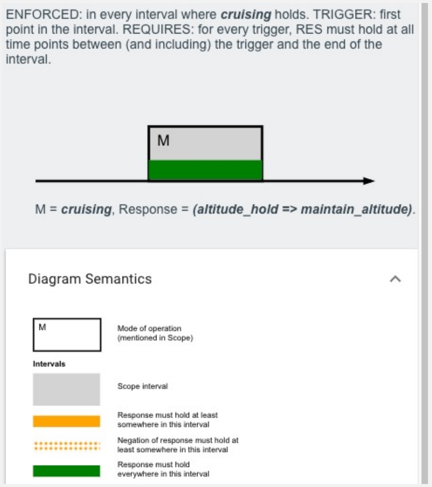
```
C: 188: 0 (6.462845; 0.355900)
```

captures + assists


when in cruising mode, the altitude_hold_

explains



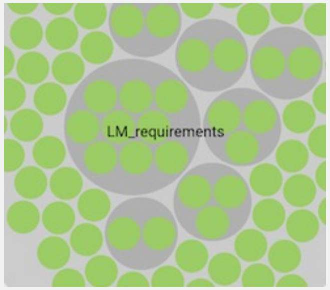

ENFORCED: in every interval where ***cruising*** holds. TRIGGER: first point in the interval. REQUIRES: for every trigger, RES must hold at all time points between (and including) the trigger and the end of the interval.

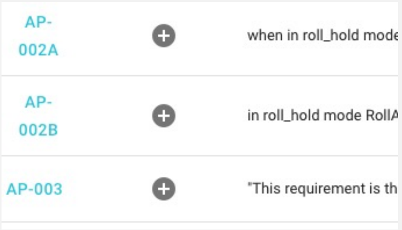
M = ***cruising***, Response = ***(altitude_hold => maintain_altitude)***.

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

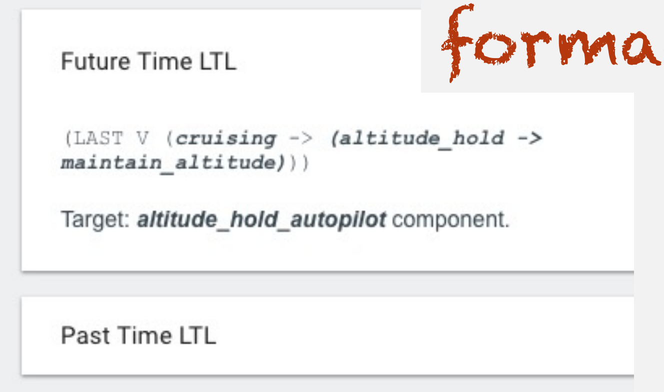
stores + displays





AP-002A	⊕	when in roll_hold mode
AP-002B	⊕	in roll_hold mode RollA
AP-003	⊕	"This requirement is th

formalizes

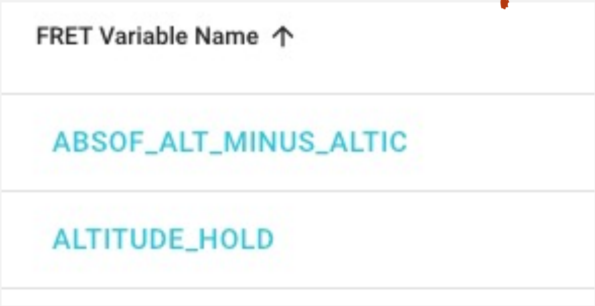

Future Time LTL

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

Target: ***altitude_hold_autopilot*** component.

Past Time LTL

connects + exports



FRET Variable Name ↑
ABSOF_ALT_MINUS_ALTIC
ALTITUDE_HOLD

checks + diagnoses

			Step
2	3	4	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.

FRET is open source: https://github.com/NASA-SW-VnV/fret

Collaborators: Hamza Bourbouh, Esther Conrad, Aaron Dutle, Marie Farrell, Pierre-Loic Garoche, Alwyn Goodloe, Mohammed Hejase, Ivan Perez, Irfan Sljivo, Laura Titolo, Tim Wang

Connection with open-source analysis tools:

CoCoSim: https://github.com/NASA-SW-VnV/CoCoSim

Copilot (through Ogma): https://github.com/NASA/ogma
https://github.com/Copilot-Language/copilot

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

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, 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.

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

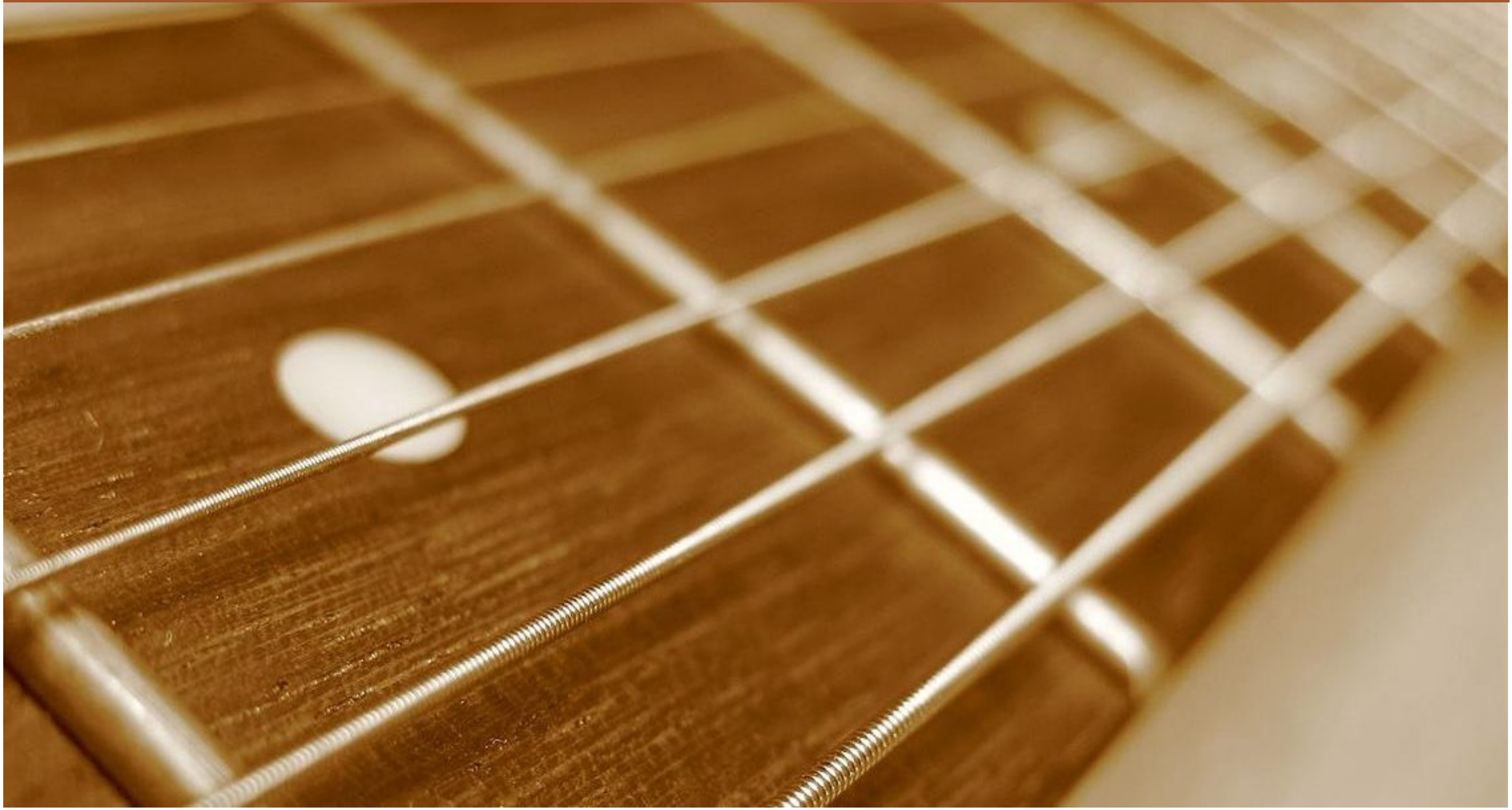
Aaron Dutle, César A. Muñoz, Esther Conrad, Alwyn Goodloe, Laura Titolo, Iván Pérez, Swee Balachandran, Dimitra Giannakopoulou, Anastasia Mavridou, Thomas Pressburger: *From Requirements to Autonomous Flight: An Overview of the Monitoring ICAROUS Project*. FMAS 2020.

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.

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

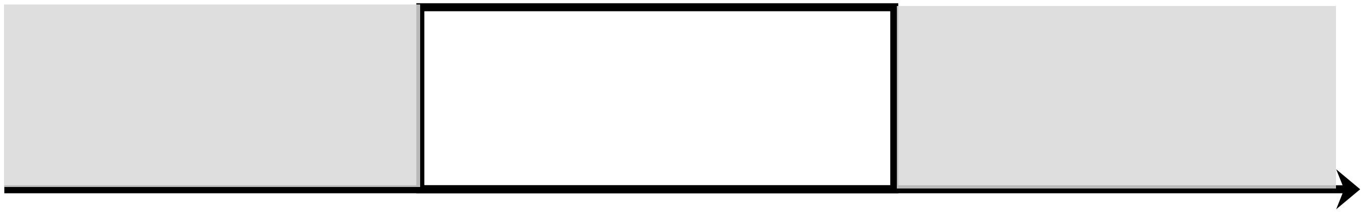
Thank you

Back up slides



Capturing requirements

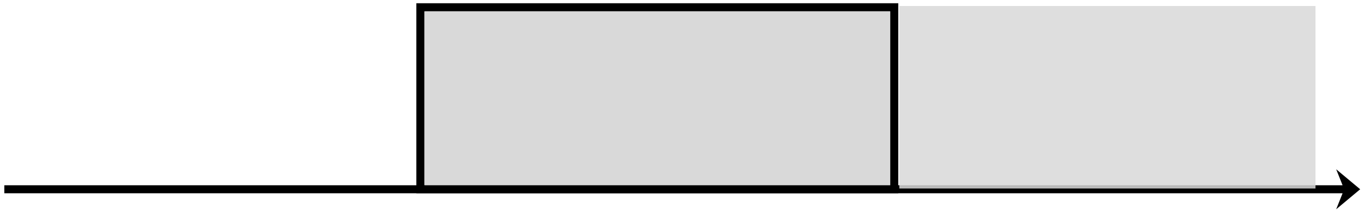
SCOPE null (global), in, before, after, notin, **onlyIn**, onlyBefore, onlyAfter



- (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

Capturing requirements

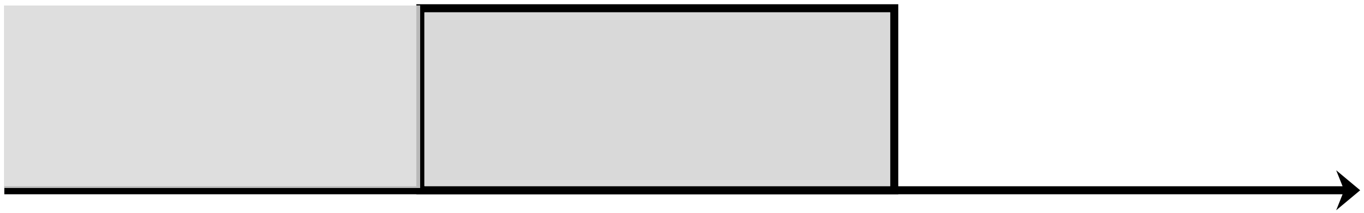
SCOPE null (global), in, before, after, notin, onlyIn, **onlyBefore**, onlyAfter



- (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

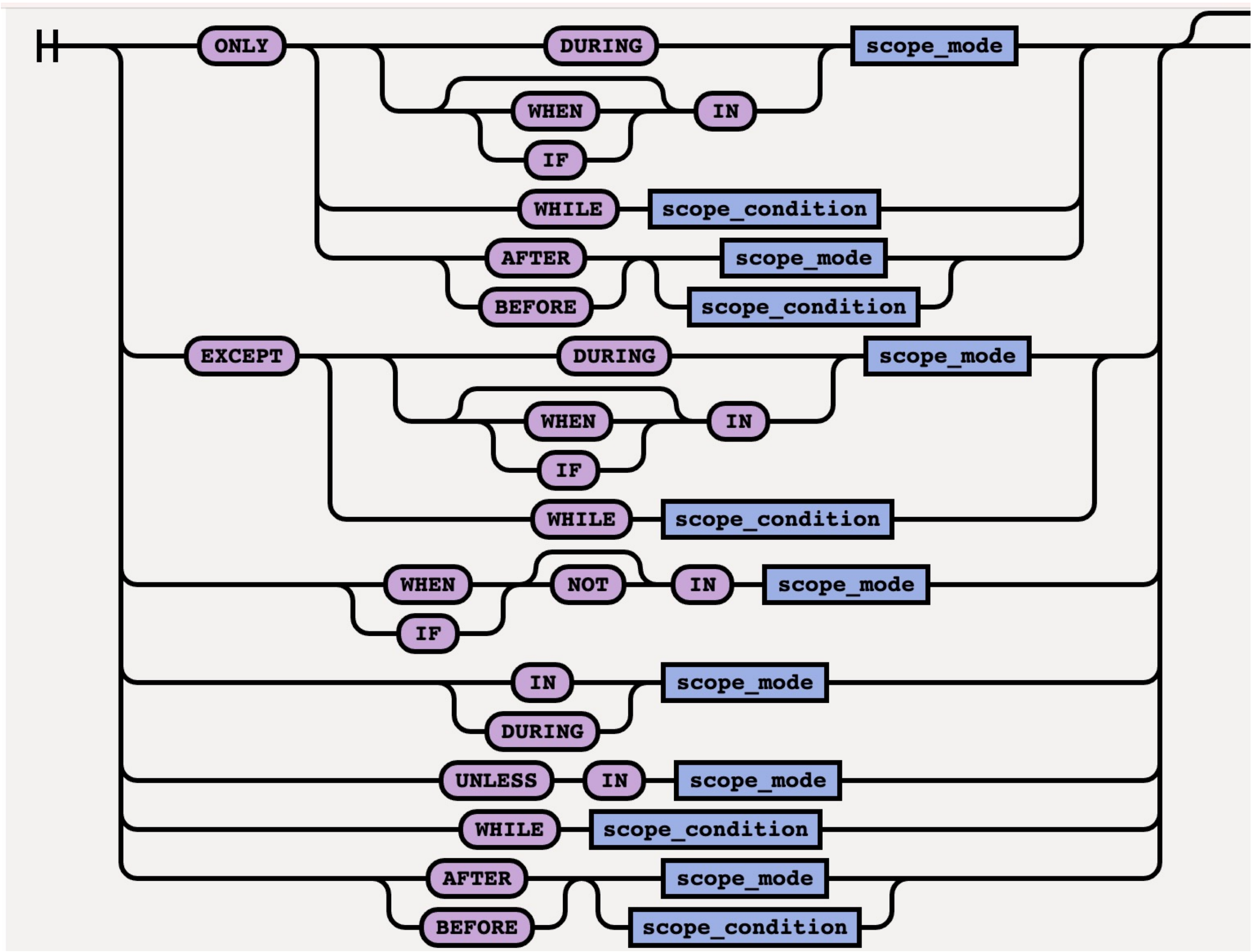
Capturing requirements

SCOPE null (global), in, before, after, notin, onlyIn, onlyBefore, **onlyAfter**

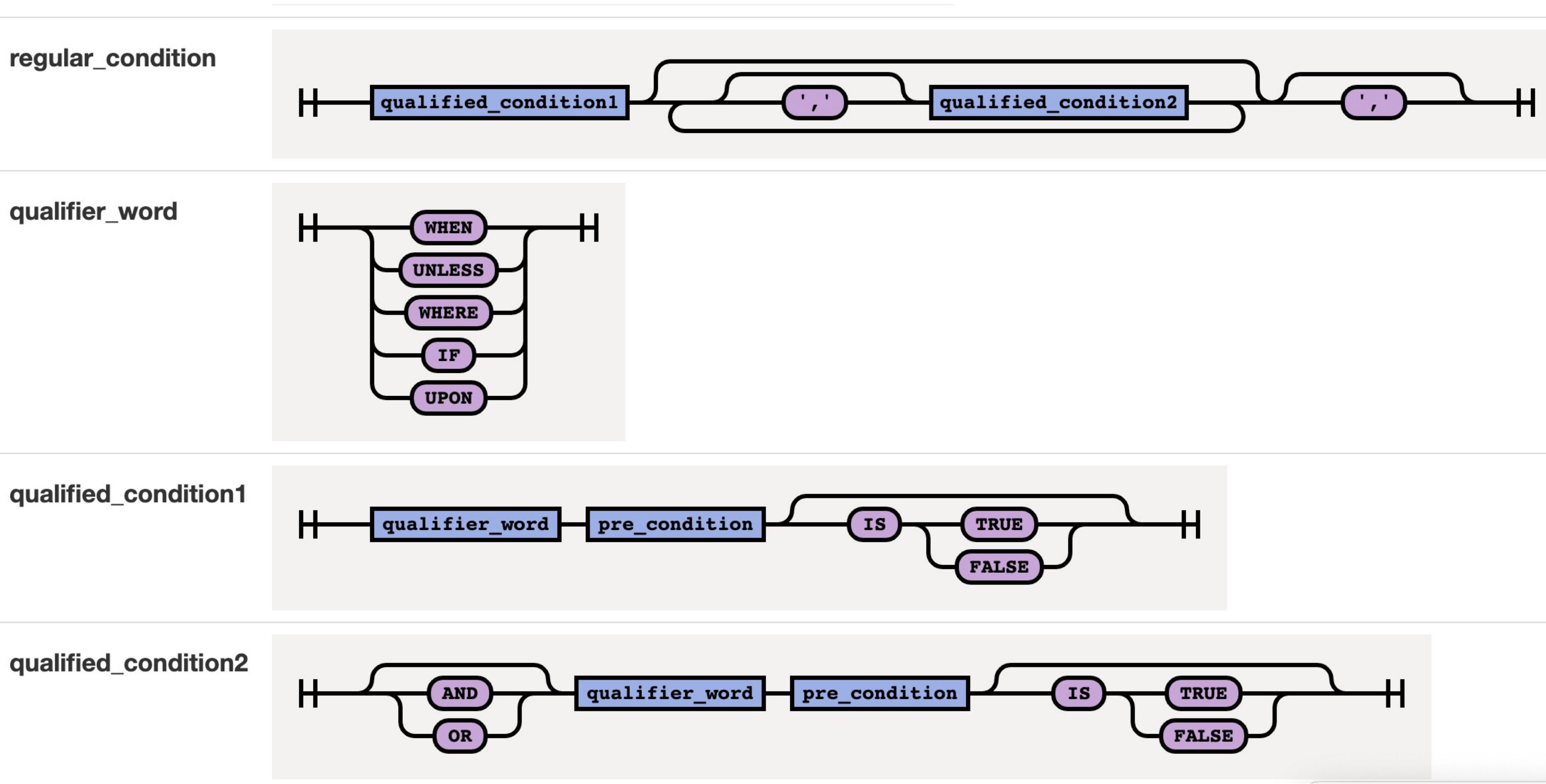


- **(global)** The system shall always satisfy count >= 0
- **In** landing mode the system shall eventually satisfy decrease_speed
- **Before** energized mode the system shall always satisfy energized_indicator_off
- **After** boot mode the system shall immediately satisfy prompt_for_password
- When **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 grammar



Condition grammar



regular_condition

qualifier_word

qualified_condition1

qualified_condition2

Explaining the semantics

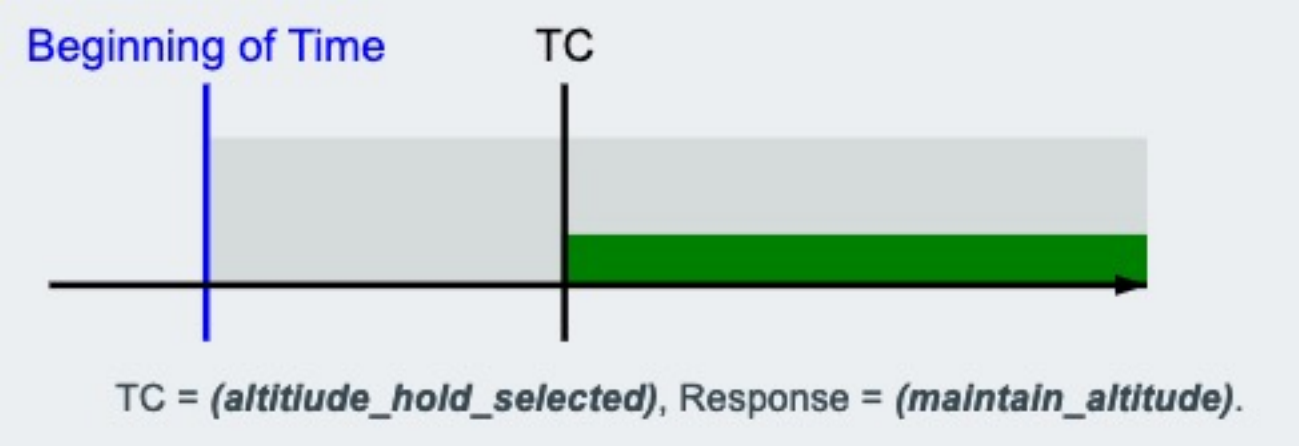
Lockheed Martin Cyber-Physical System Challenge:

Natural language requirement:

The altitude hold autopilot shall maintain altitude **whenever altitude hold is selected**

FRETish:

	if altitude_hold_selected	the altitude_hold_autopilot	**shall**	always	satisfy maintain_altitude
scope	condition	component*		timing	response*



TC = ***(altitiude_hold_selected)***, Response = ***(maintain_altitude)***.

Getting to the right requirement

Lockheed Martin Cyber-Physical System Challenge:

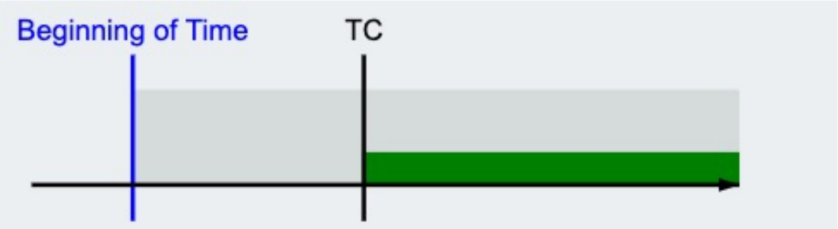
Natural language requirement:

The altitude hold autopilot shall maintain altitude **whenever altitude hold is selected**

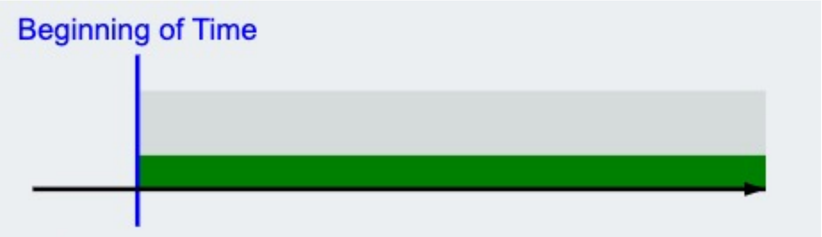
FRETish:

if altitude_hold_selected the altitude_hold_autopilot **shall** always satisfy maintain_altitude

the altitude_hold_autopilot **shall** always satisfy if altitude_hold_selected then maintain_altitude



TC = ***(altitiude_hold_selected)***, Response = ***(maintain_altitude)***.

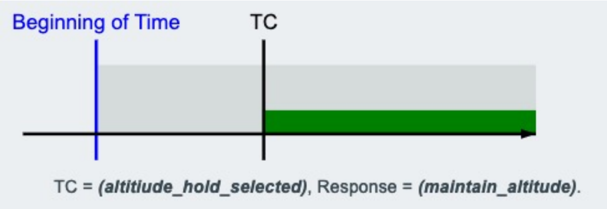


Response = ***(altitiude_hold_selected => maintain_altitude)***.

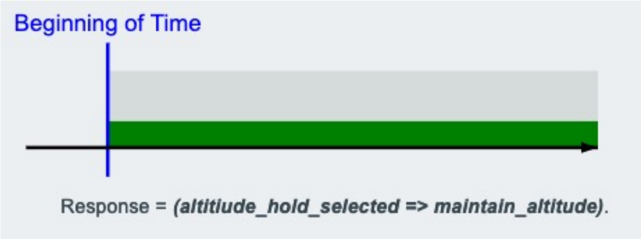
Getting to the right requirement

if altitude_hold_selected the altitude_hold_autopilot **shall** always satisfy maintain_altitude

the altitude_hold_autopilot**shall** always satisfy if altitude_hold_selected then maintain_altitude

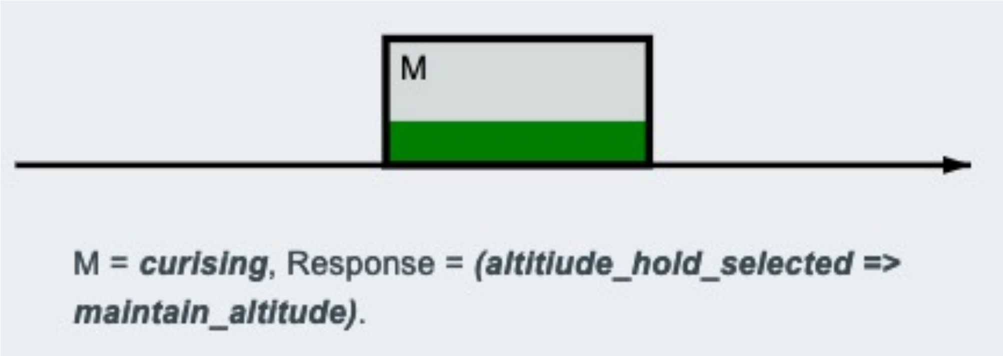


TC = *(altitiude_hold_selected)*, Response = *(maintain_altitude)*.



Response = *(altitiude_hold_selected => maintain_altitude)*.

When in cruising mode, the altitude_hold_autopilot shall always satisfy if altitude_hold_selected then maintain_altitude



M = ***curising***, Response = ***(altitiude_hold_selected => maintain_altitude)***.