Let’s speak FRETish

Anastasia Mavridou

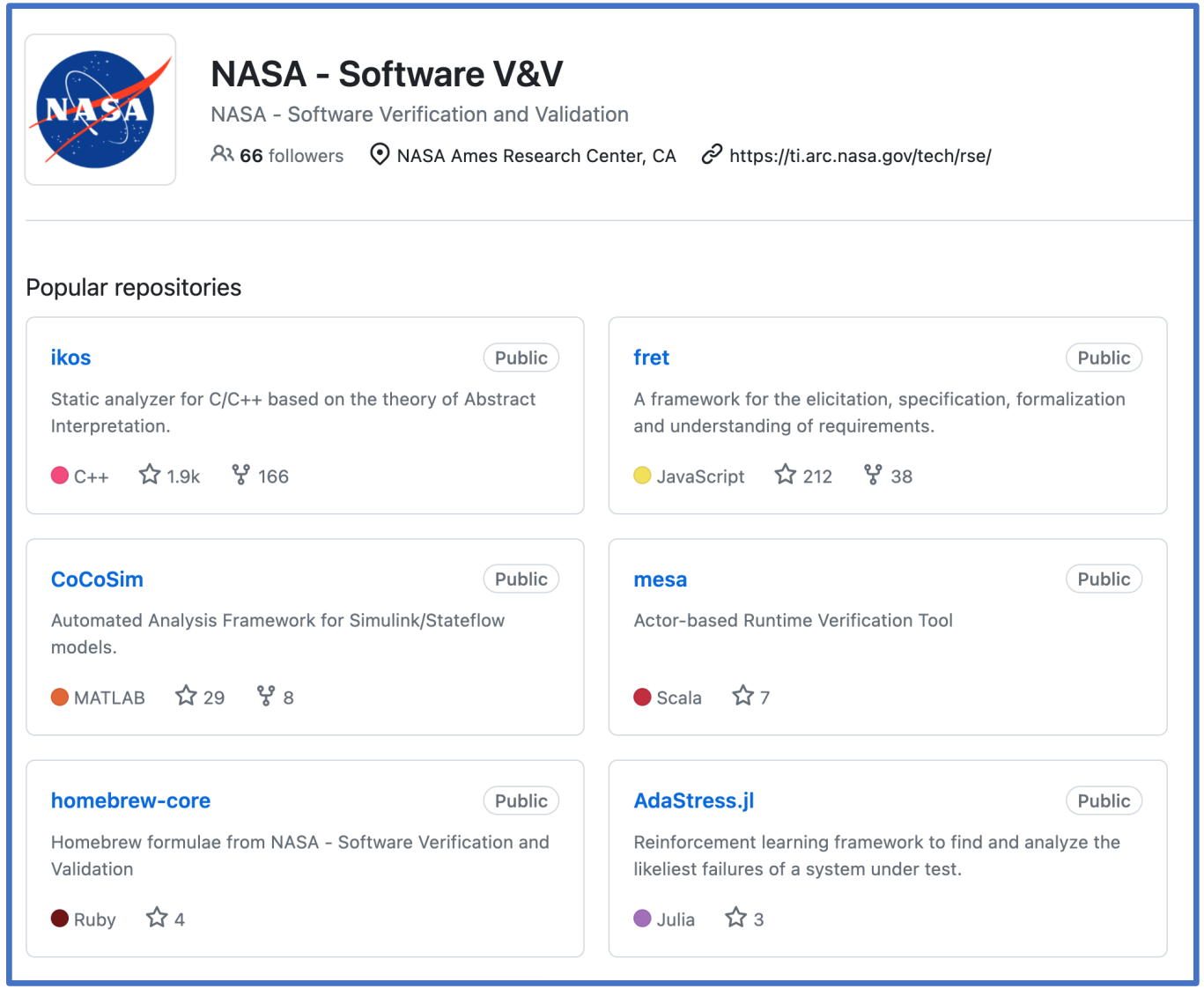
KBR at NASA Ames Research Center

anastasia.mavridou@nasa.gov

NASA’s center in Silicon Valley

Aerial image of NASA Ames Research Center
Credits: NASA

robust software engineering technical area



NASA - Software V&V

NASA - Software Verification and Validation

66 followers · NASA Ames Research Center, CA · https://ti.arc.nasa.gov/tech/rse/

Popular repositories

ikos — Public

Static analyzer for C/C++ based on the theory of Abstract Interpretation.

C++ · 1.9k · 166

fret — Public

A framework for the elicitation, specification, formalization and understanding of requirements.

JavaScript · 212 · 38

CoCoSim — Public

Automated Analysis Framework for Simulink/Stateflow models.

MATLAB · 29 · 8

mesa — Public

Actor-based Runtime Verification Tool

Scala · 7

homebrew-core — Public

Homebrew formulae from NASA - Software Verification and Validation

Ruby · 4

AdaStress.jl — Public

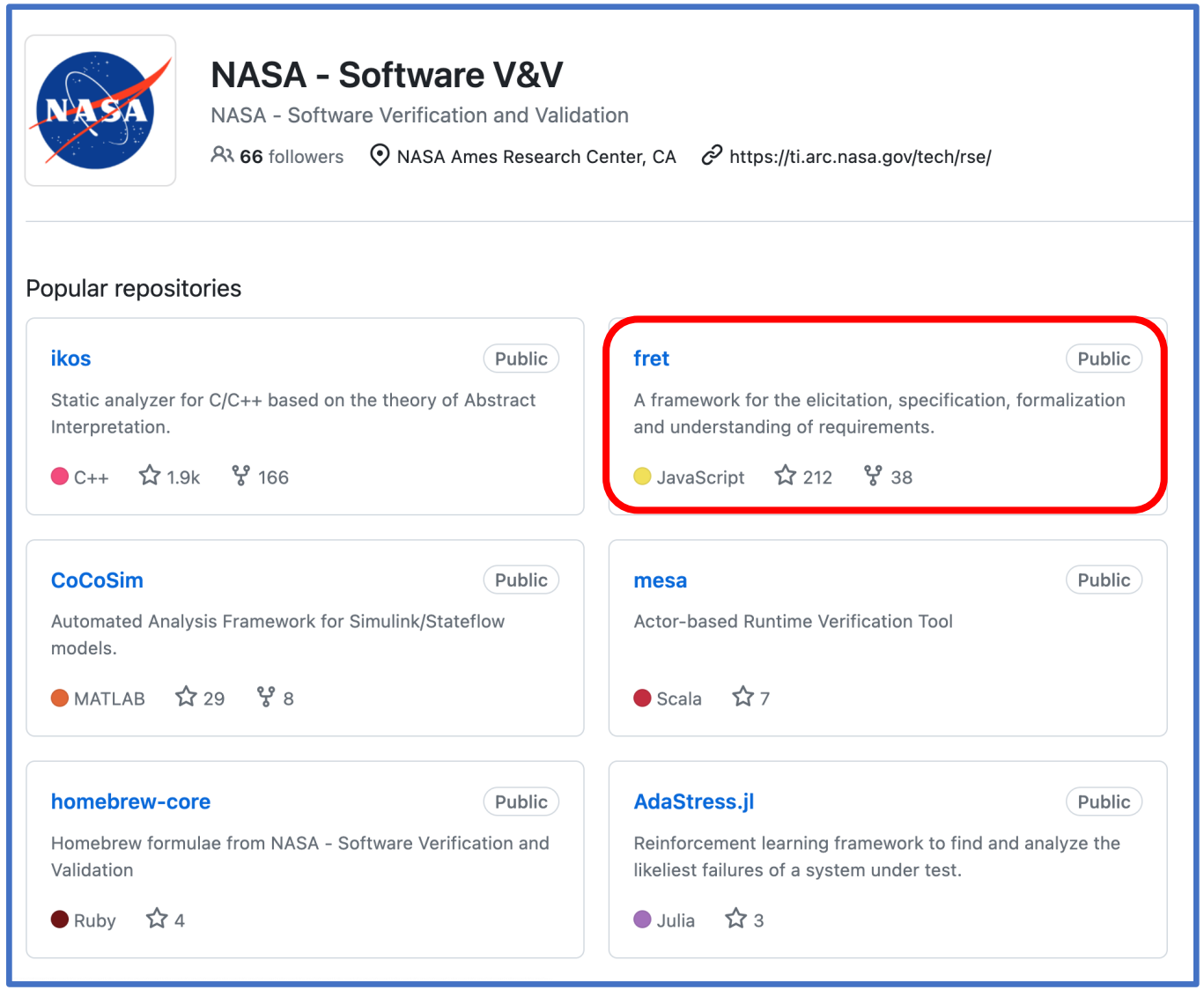
Reinforcement learning framework to find and analyze the likeliest failures of a system under test.

Julia · 3

github.com/NASA-SW-VnV

www.nasa.gov/isd-robust-software-engineering

robust software engineering technical area



NASA - Software V&V

NASA - Software Verification and Validation

66 followers · NASA Ames Research Center, CA · https://ti.arc.nasa.gov/tech/rse/

Popular repositories

ikos — Public

Static analyzer for C/C++ based on the theory of Abstract Interpretation.

C++ · 1.9k · 166

fret — Public

A framework for the elicitation, specification, formalization and understanding of requirements.

JavaScript · 212 · 38

CoCoSim — Public

Automated Analysis Framework for Simulink/Stateflow models.

MATLAB · 29 · 8

mesa — Public

Actor-based Runtime Verification Tool

Scala · 7

homebrew-core — Public

Homebrew formulae from NASA - Software Verification and Validation

Ruby · 4

AdaStress.jl — Public

Reinforcement learning framework to find and analyze the likeliest failures of a system under test.

Julia · 3

github.com/NASA-SW-VnV

www.nasa.gov/isd-robust-software-engineering

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).
- 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.

how developers write requirements

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

Every time these conditions hold or only when they become true?

- 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.

how developers write requirements

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

- Exceeding sensor limits shall latch an autopilot pullup when 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.
- …

Instantly, or within a time limit?

what analysis tools understand

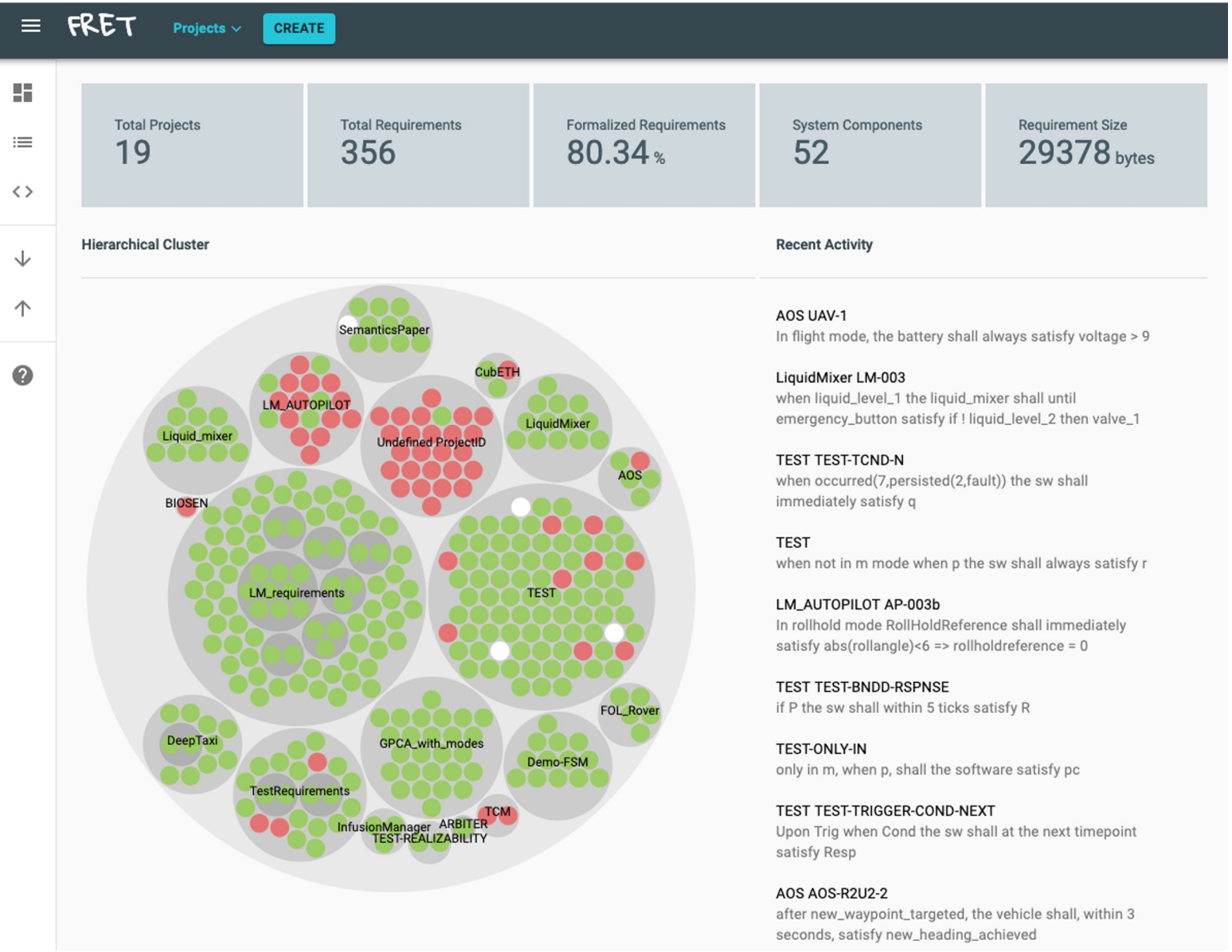
10 Lockheed Martin Cyber-Physical System Challenge, component FSM:

```
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 structured natural language with unambiguous semantics.
- Explains formal semantics in various forms.
- Formalizes requirements in a compositional (hence extensible) manner.
- Analyzes consistency of requirements and provides feedback.
- Connects with analysis tools:
  - Runtime monitoring tools.
  - Simulink model checkers.
  - Lustre code analysis tools.
  - Programmable Logic Controller analysis tools.

welcome to FRET



`github.com/NASA-SW-VnV/fret`

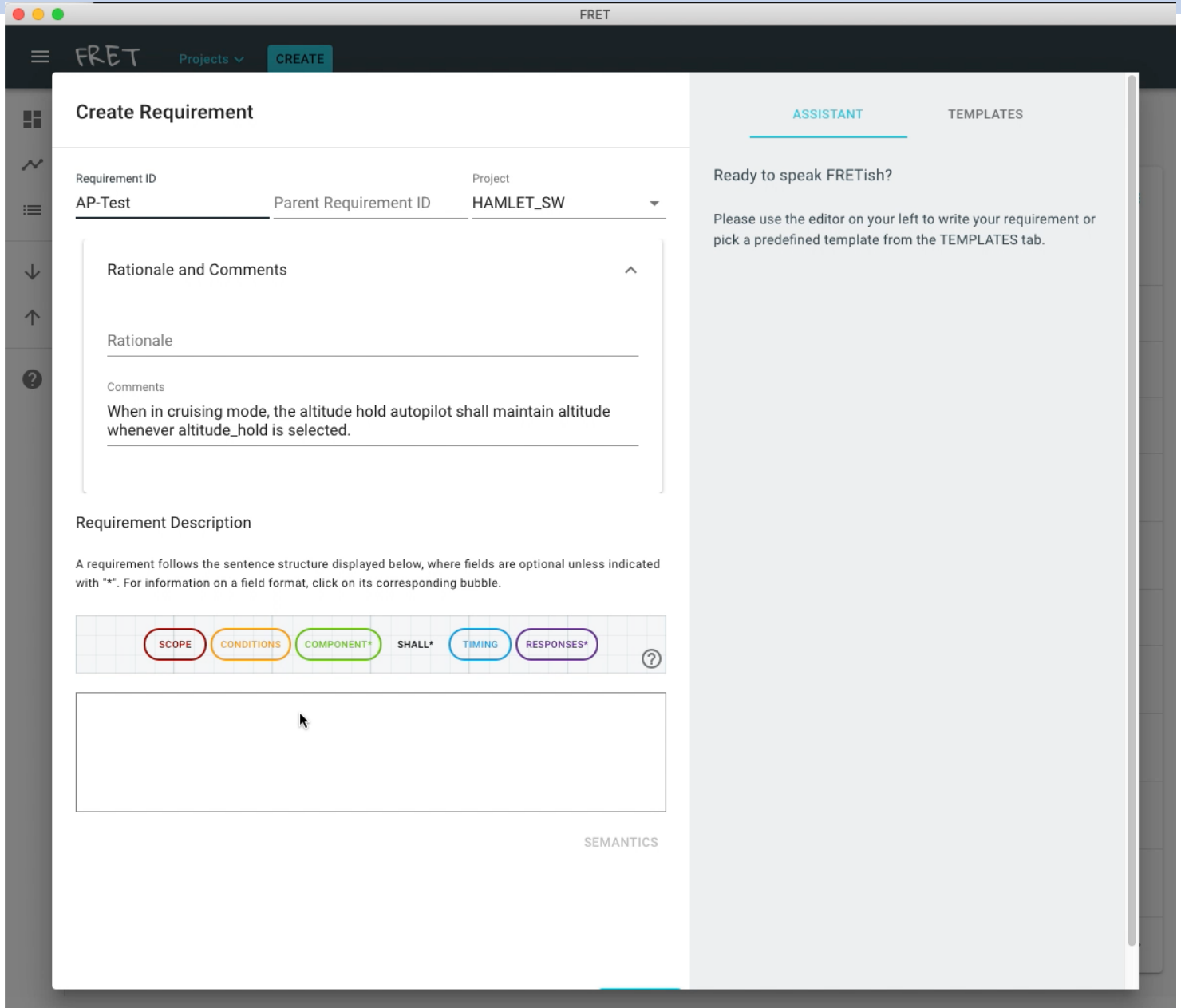
Team: Andreas Katis, Anastasia Mavridou, Tom Pressburger, Johann Schumann, Khanh Trinh

Alumni: David Bushnell, Dimitra Giannakopoulou, Nija Shi

Interns: Milan Bhandari, Tanja DeJong, Kelly Ho, George Karamanolis, David Kooi, Jessica Phelan, Julian Rhein, Daniel Riley, Gricel Vazquez

capturing, explaining, and formalizing requirements

let’s speak FRETish



FRETish fields

In cruising mode, the autopilot shall always satisfy if altitude_hold then maintain_altitude

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

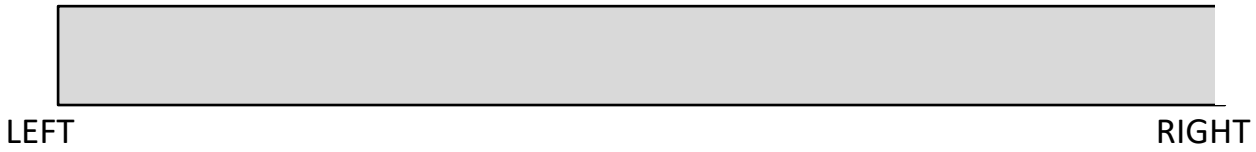
CONDITION null, regular

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

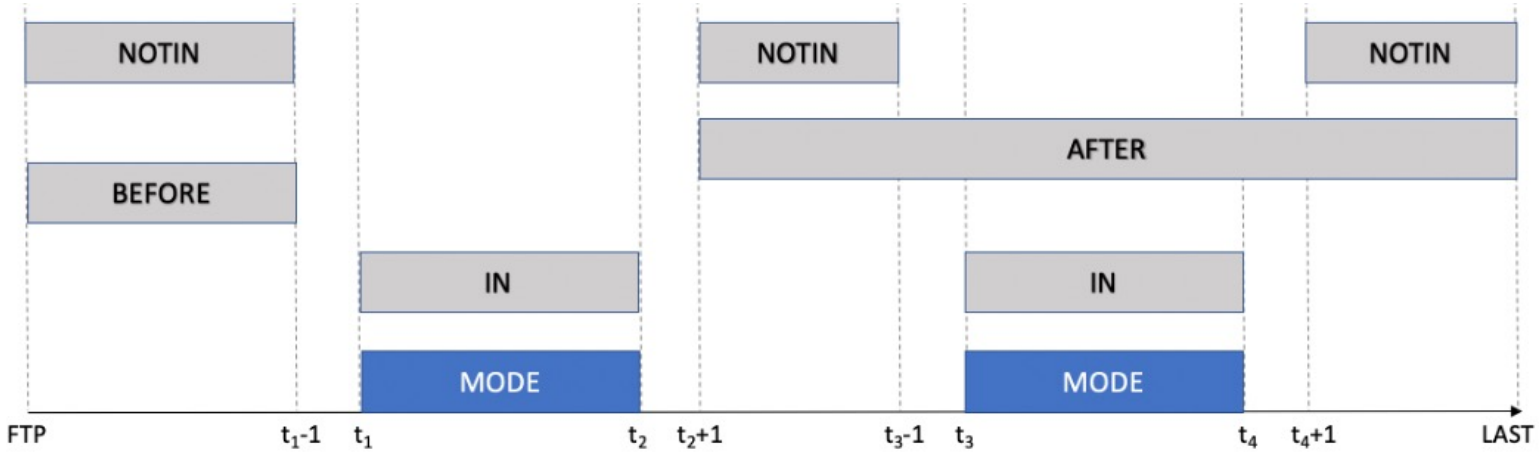
RESPONSE satisfaction

scope abstraction

[LEFT, RIGHT):




```
historically (RIGHT implies previous
                                (BASEFORM since inclusive required LEFT))
```



constructing formulas from fields

In cruising mode, the autopilot shall always satisfy if altitude_hold then maintain_altitude

scope in: [LEFT, RIGHT) → [FiM, LiM)

```
FiM = MODE and (FTP or previous (not MODE))
LiM = not MODE and previous MODE
```

timing always: `BASEFORM → RES`

```
historically (RIGHT implies previous (BASEFORM since inclusive required LEFT))
```

scope: in, condition: null, timing: always, response: satisfaction

```
historically (LiM implies previous (RES since inclusive required FiM))
```

optimize

```
historically (MODE implies RES)
```

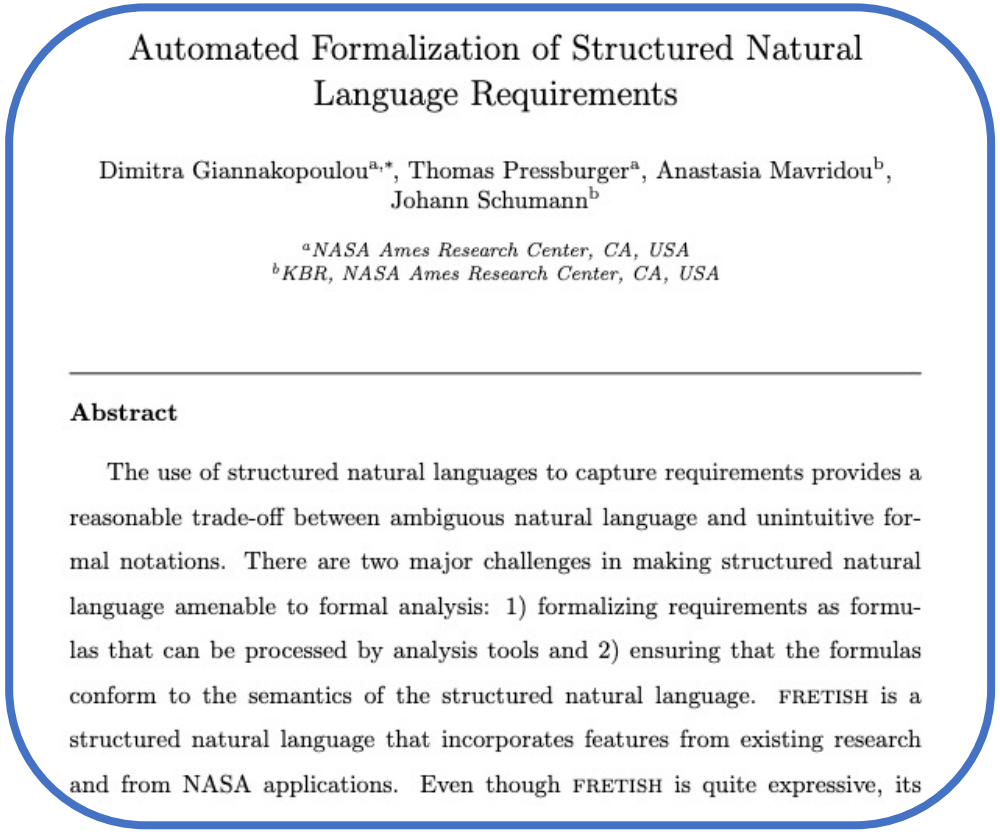
translate to SMV

```
(H (MODE → RES))
```

instantiate

```
(H (cruising → (altitude_hold → maintain_altitude)))
```

related papers



Automated Formalization of Structured Natural Language Requirements

Dimitra Giannakopoulou[a,*], Thomas Pressburger[a], Anastasia Mavridou[b], Johann Schumann[b]

[a]NASA Ames Research Center, CA, USA
[b]KBR, NASA Ames Research Center, CA, USA

Abstract

The use of structured natural languages to capture requirements provides a reasonable trade-off between ambiguous natural language and unintuitive formal notations. There are two major challenges in making structured natural language amenable to formal analysis: 1) formalizing requirements as formulas that can be processed by analysis tools and 2) ensuring that the formulas conform to the semantics of the structured natural language. FRETISH is a structured natural language that incorporates features from existing research and from NASA applications. Even though FRETISH is quite expressive, its

Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Johann Schumann (2021). Automated formalization of structured natural language requirements, Information and Software Technology (IST) Journal, 137, 106590, Special Section on REFSQ'20, 2021.

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

requirement templates

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

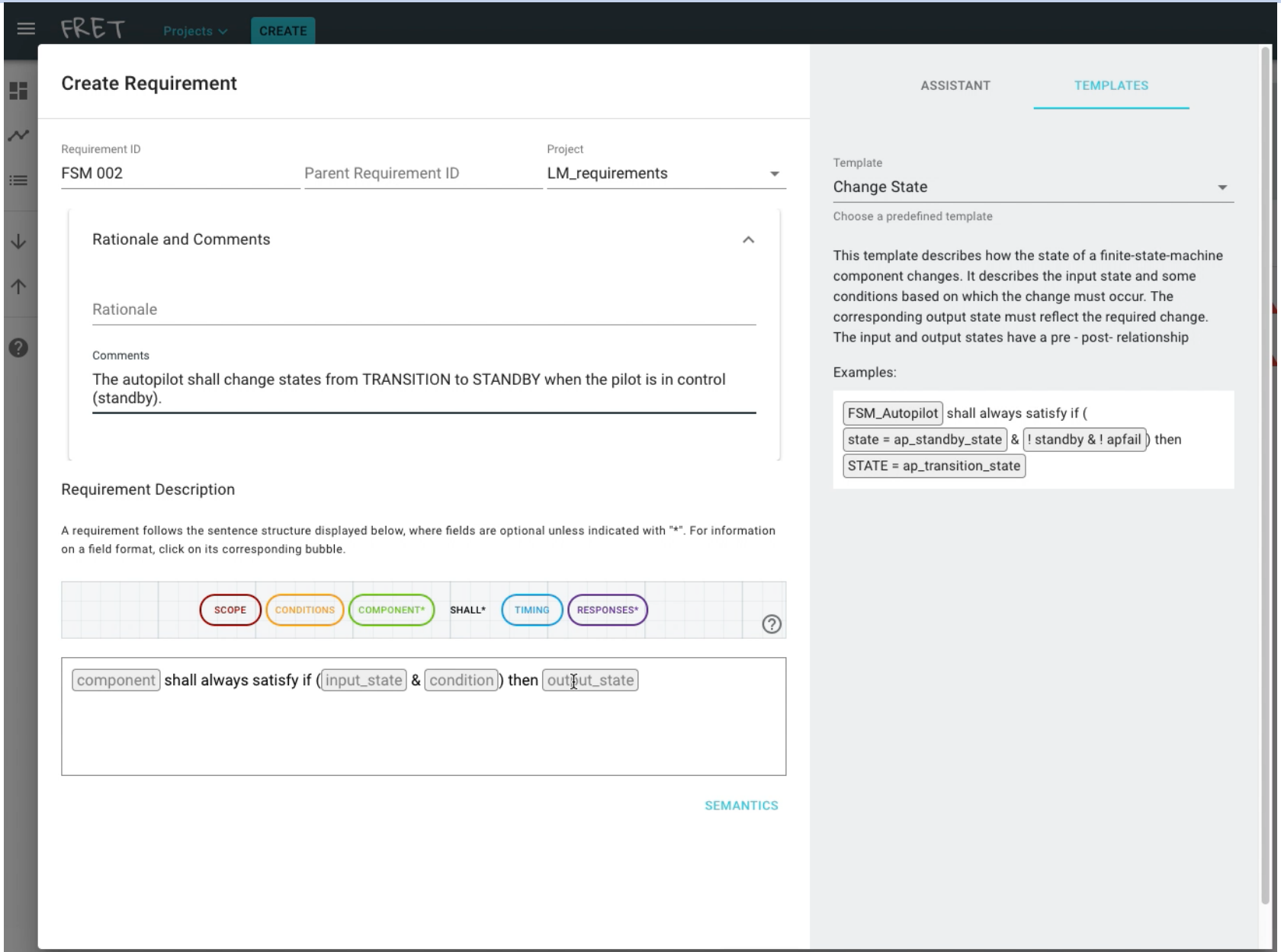
- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data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

10 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



checking realizability of requirements

even simple requirements can be conflicting

10 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.

even simple requirements can be conflicting

10 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

even simple requirements can be conflicting

10 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: standby
- Condition 2: supported & good_sensor_data

even simple requirements can be conflicting

10 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: standby ✓
- Condition 2: supported & good_sensor_data ✓

even simple requirements can be conflicting

10 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: standby ✓
- Condition 2: supported & good_sensor_data ✓
- Output state 1: STANDBY
- Output state 2: NOMINAL 

why realizability?

- Defining requirements is a challenging, error prone task
- Realizability checking >> consistency checking
- We want to ensure requirement consistency for all inputs
- And we want to do it efficiently

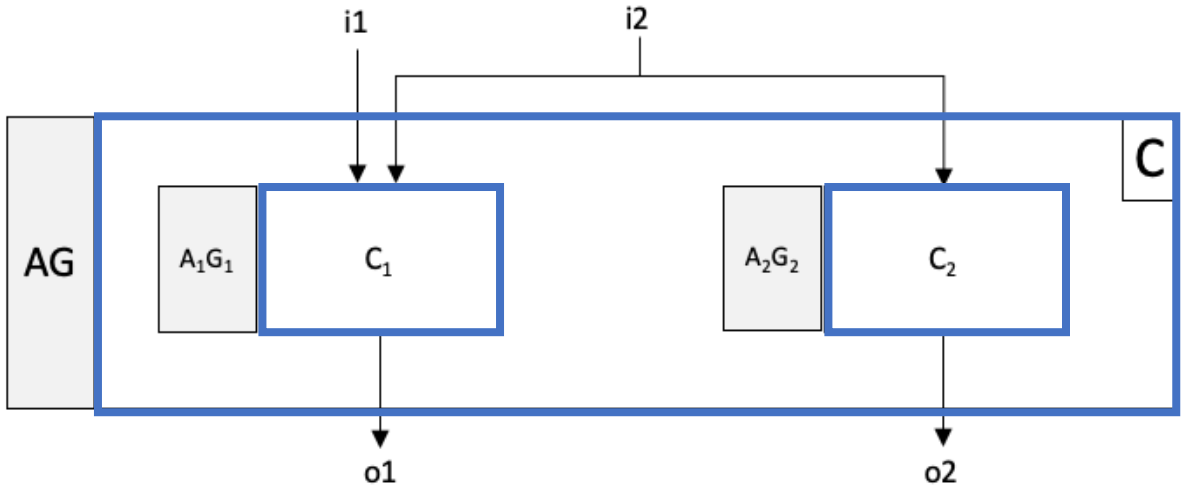
An AG contract is realizable if there exists a system implementation that satisfies the contract guarantees for all assumption-complying stimuli provided by the environment.

compositional realizability

- Realizability checking can be challenging
  - Nested quantifiers for solvers
  - 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

Partial AG contracts: contracts that observe only part of the state of a system



Discover conditions under which partial contracts (A_1, G_1) and (A_2, G_2) can be equivalently represented by (A,G)

We identified challenges:

- Guarantee interference – may return false positives

decomposition approach

Decompose an AG contract by splitting the guarantees

- Notion of connected components for undirected graphs

Requirements graph

- Each vertex corresponds to a requirement
- If state variables referenced by two requirements, their vertices are connected
- Connected components represent partial contracts

decomposition approach

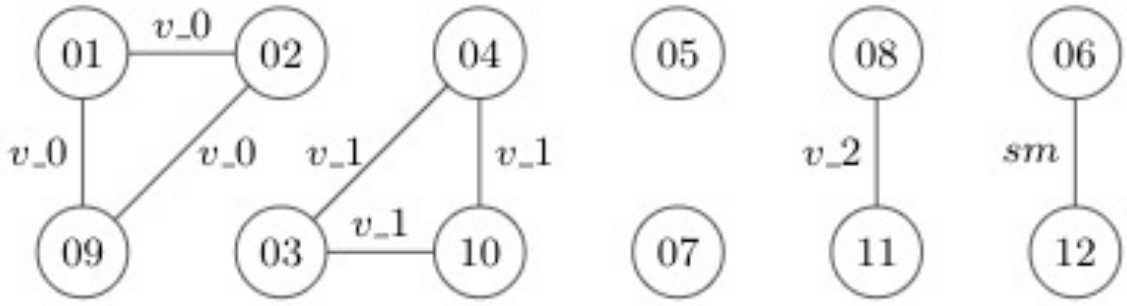
Decompose an AG contract by splitting the guarantees

- Notion of connected components for undirected graphs

Requirements graph

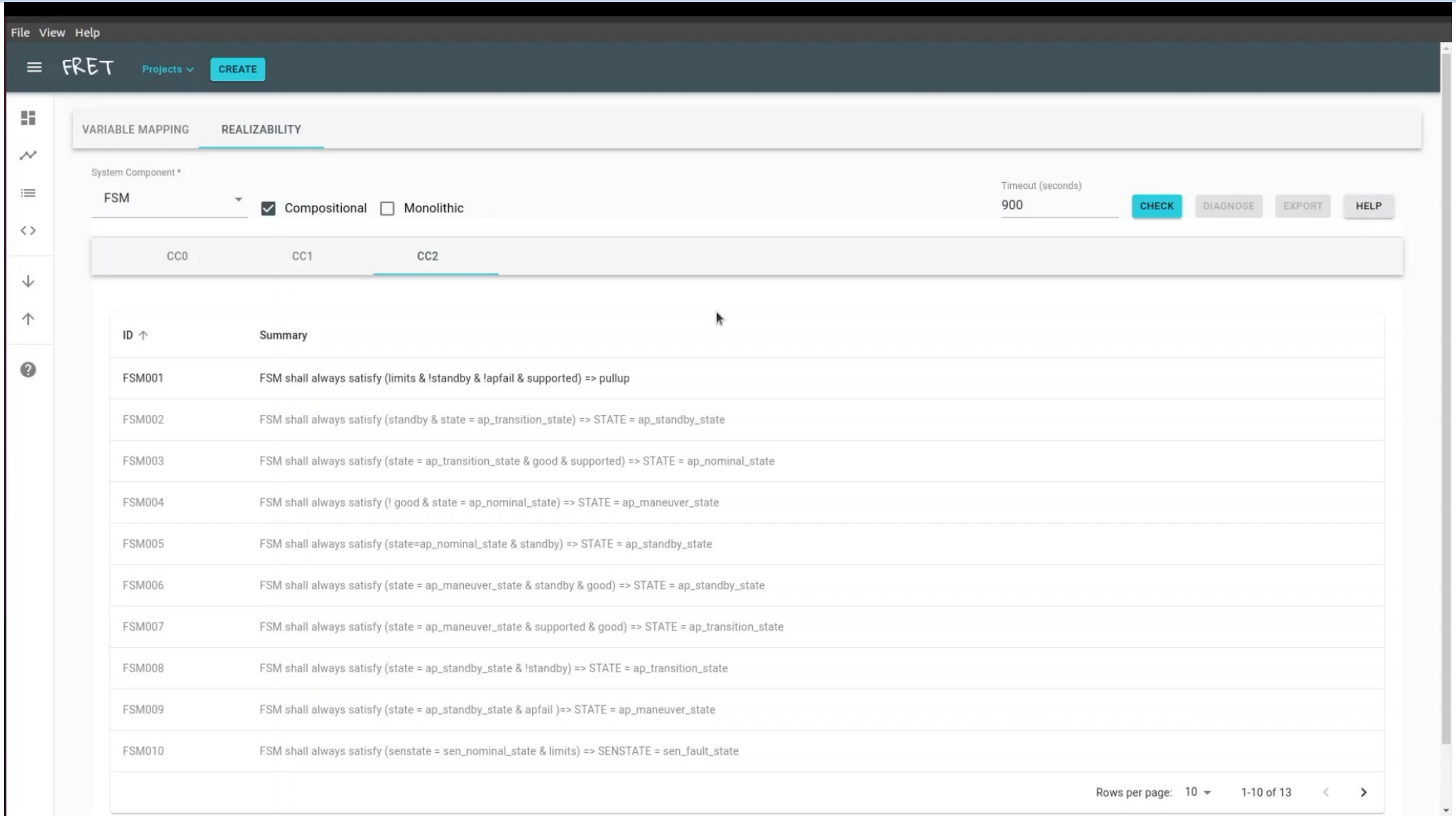
- Each vertex corresponds to a requirement
- If state variables referenced by two requirements, their vertices are connected
- Connected components represent partial contracts

Effectively reduces problem complexity!



6 connected components from 12 requirements

checking realizability within FRET



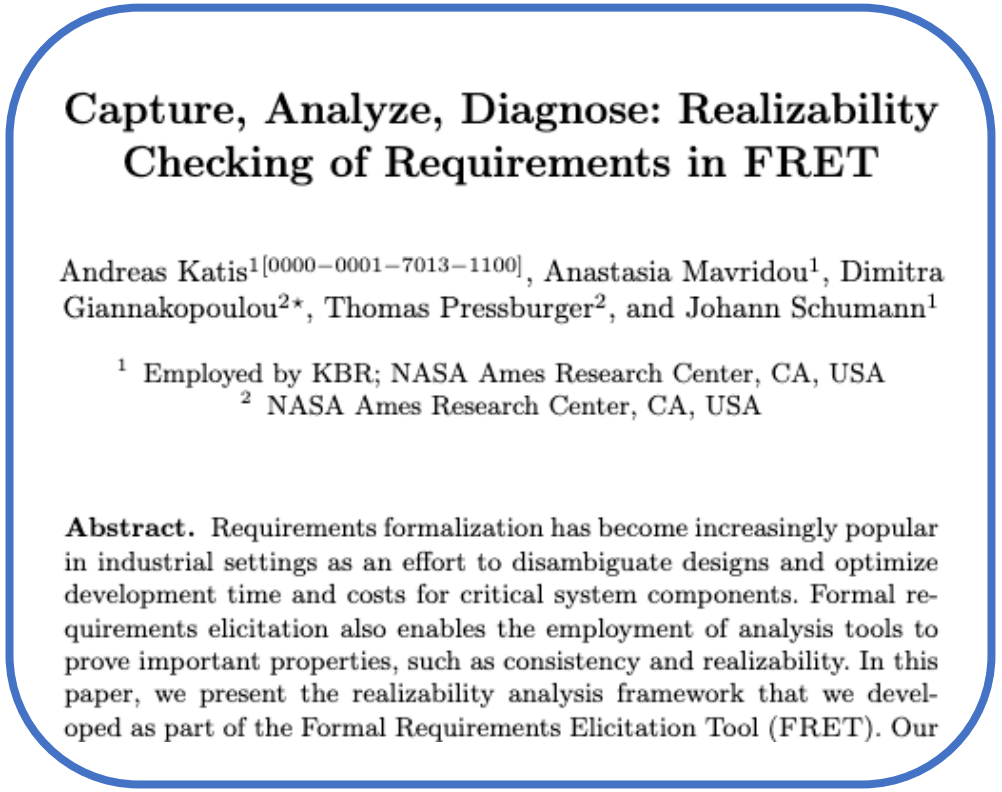
related papers

From Partial to Global Assume-Guarantee Contracts: Compositional Realizability Analysis in FRET

Anastasia Mavridou[1], Andreas Katis[1], Dimitra Giannakopoulou[2], David Kooi[3], Thomas Pressburger[2], and Michael W. Whalen[4]

[1] KBR, NASA Ames Research Center, CA, USA
[2] NASA Ames Research Center, CA, USA
{anastasia.mavridou, andreas.katis, dimitra.giannakopoulou, tom.pressburger}@nasa.gov
[3] University of California, Santa Cruz, CA, USA dkooi@ucsc.edu
[4] University of Minnesota, MN, USA whalen@cs.umn.edu

Abstract. Realizability checking refers to the formal procedure that aims to determine whether an implementation exists, always complying to a set of requirements, regardless of the stimuli provided by the system's environment. Such a check is essential to ensure that the specification does not allow behavior that can force the system to violate safety constraints. In this paper, we present an approach that decomposes realizability checking into smaller, more tractable problems. More



Capture, Analyze, Diagnose: Realizability Checking of Requirements in FRET

Andreas Katis[1][0000−0001−7013−1100], Anastasia Mavridou[1], Dimitra Giannakopoulou[2⋆], Thomas Pressburger[2], and Johann Schumann[1]

[1] Employed by KBR; NASA Ames Research Center, CA, USA
[2] NASA Ames Research Center, CA, USA

Abstract. Requirements formalization has become increasingly popular in industrial settings as an effort to disambiguate designs and optimize development time and costs for critical system components. Formal requirements elicitation also enables the employment of analysis tools to prove important properties, such as consistency and realizability. In this paper, we present the realizability analysis framework that we developed as part of the Formal Requirements Elicitation Tool (FRET). Our

Andreas Katis, Anastasia Mavridou, Dimitra Giannakopoulou, Thomas Pressburger, Johann Schumann. Capture, Analyze, Diagnose: Realizability Checking of Requirements in FRET, CAV 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.

connection with analysis tools

generation of Simulink monitors

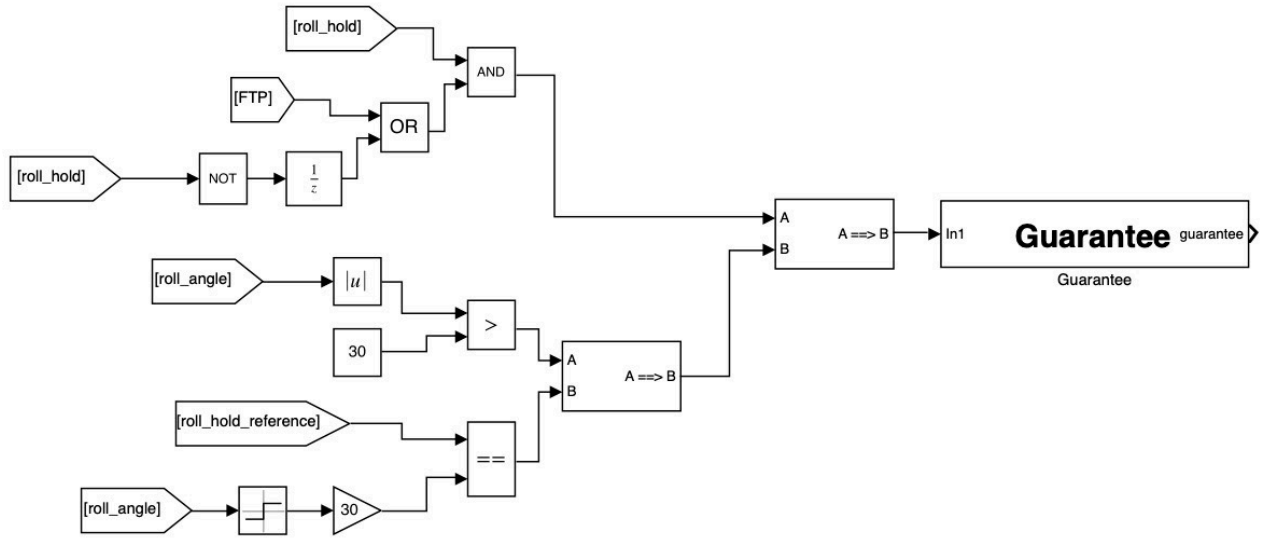
FRETish:

when in roll_hold_mode autopilot shall immediately satisfy if roll_angle > 3 then roll_hold_reference = 3

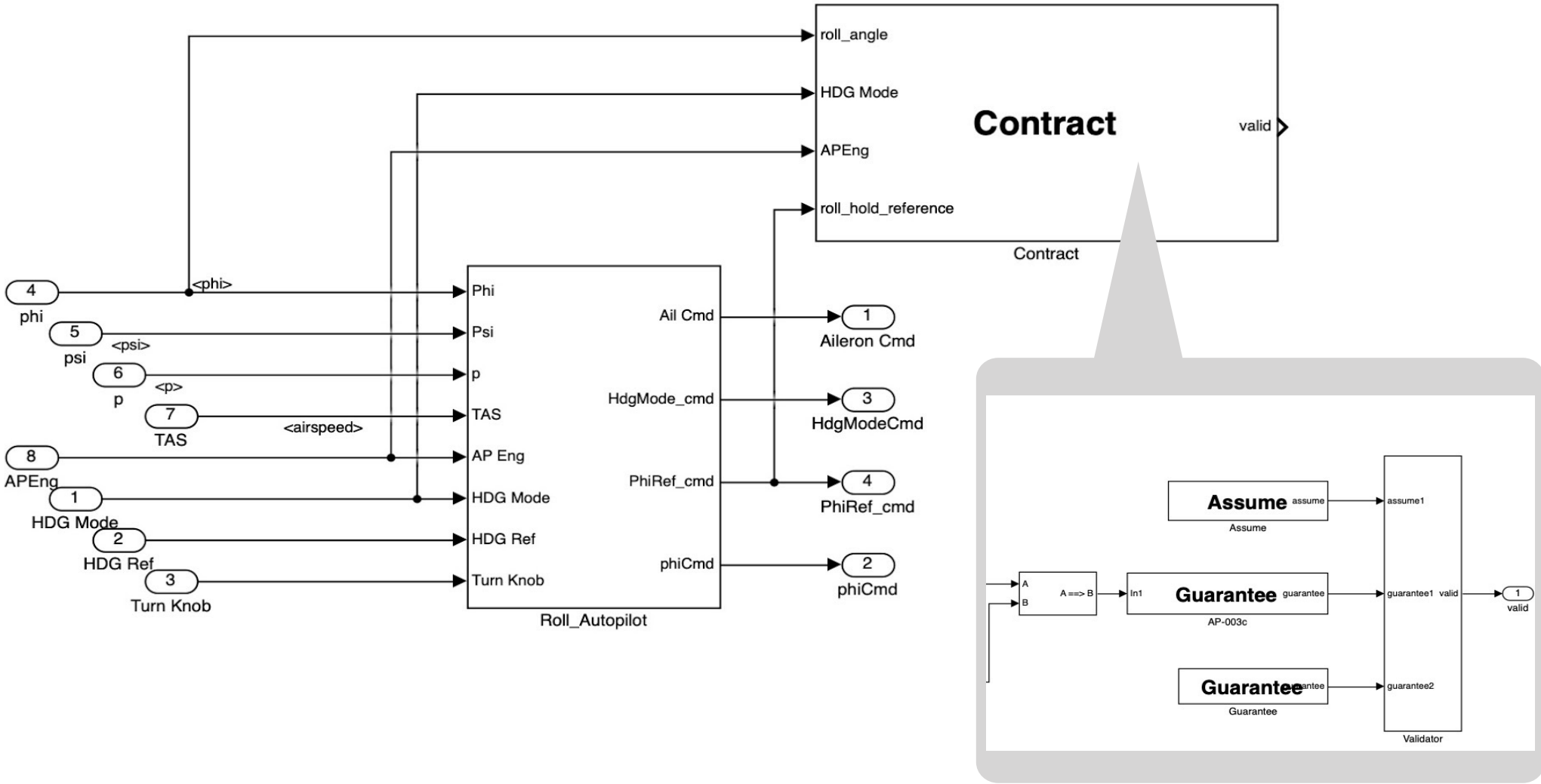
Lustre specification:

```
-- 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 * sign(roll_angle))
```

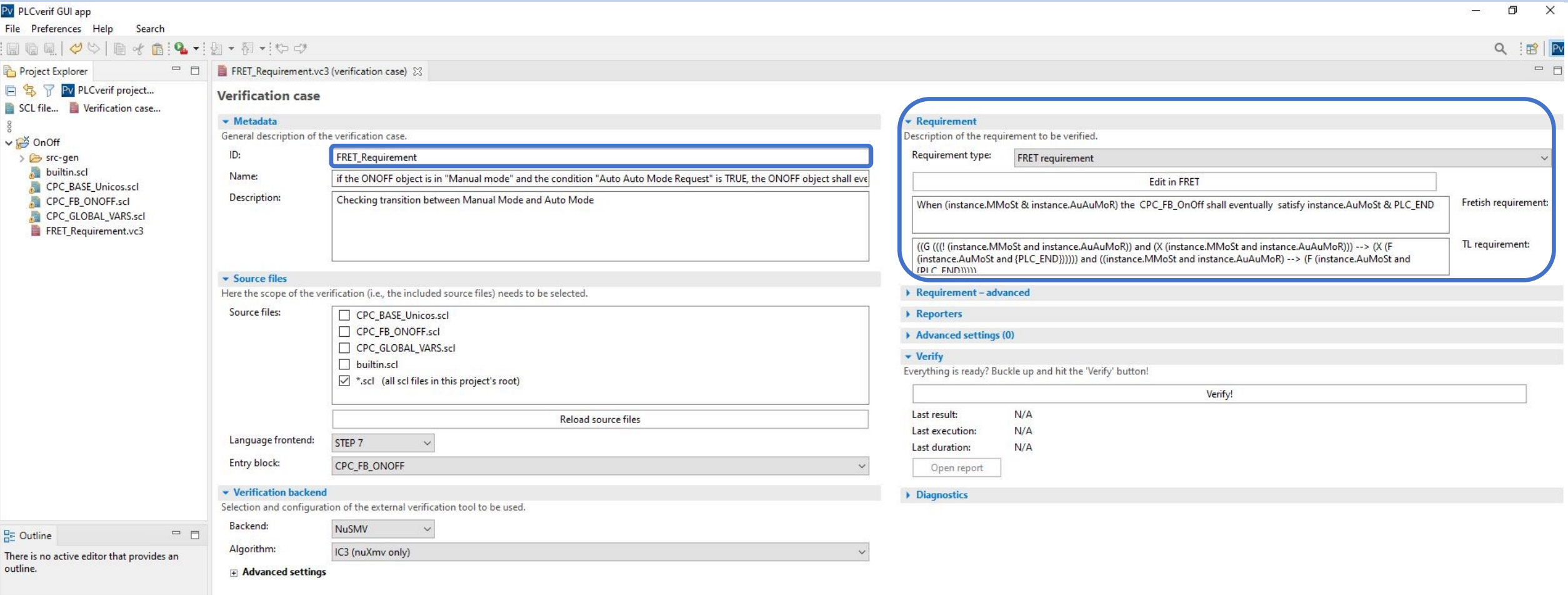
Simulink monitor



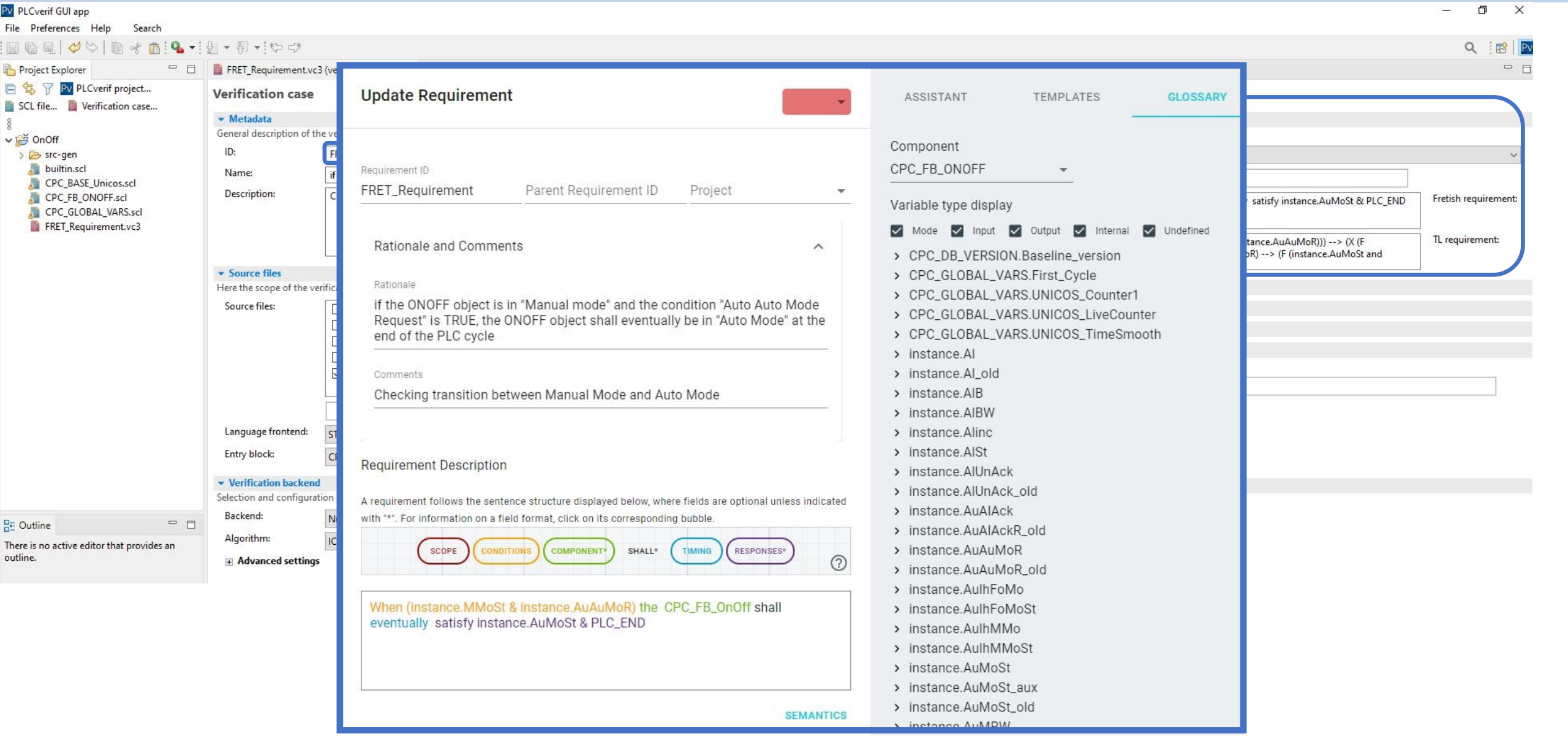
model checking Simulink models



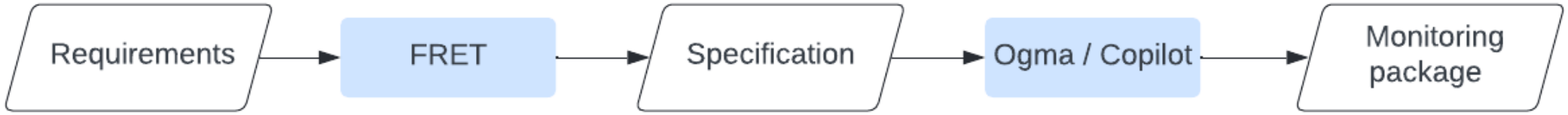
model checking PLC code



model checking PLC code



runtime monitoring



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.

related papers

Zsófia Ádám, Ignacio D. Lopez-Miguel, Anastasia Mavridou, Thomas Pressburger, Marcin Bęś, Enrique Blanco Viñuela, Andreas Katis, Jean-Charles Tournier, Khanh V. Trinh, Borja Fernandez Adiego. From Natural Language Requirements to the Verification of Programmable Logic Controllers: Integrating FRET into PLCverif, NFM 2023.

Joseph Kiniry, Alexanders Bakst, Simon Hansen, Michal Podhradsky, and Andrew Bivin. The HARDENS Final Report, Galois Inc Technical Report.

Thomas Pressburger, Andreas Katis, Aaron Dutle, Anastasia Mavridou. Authoring, Analyzing, and Monitoring Requirements for a Lift-Plus-Cruise Aircraft, REFSQ 2023.

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

Hamza Bourbouh, Marie Farrell, Anastasia Mavridou, Irfan Sljivo, Guillaume Brat, Louise A. Dennis, Michael Fisher. Integrating Formal Verification and Assurance: An Inspection Rover Case Study, NFM 2021.

Anastasia Mavridou, Hamza Bourbouh, Dimitra Giannakopoulou, Tom Pressburger, Pierre-Loic Garoche, Johann Schumann. The Ten Lockheed Martin Cyber-Physical Challenges: Formalized, Analyzed, and Explained, RE 2020, Industry track.

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

Full list: https://github.com/NASA-SW-VnV/fret/blob/master/PUBLICATIONS.md

case studies

examples of projects that used FRET

NASA Lift Plus Cruise aircraft

Features of FRET used:

- ✓ Requirement specification
- ✓ Interactive simulator
- ✓ Requirement templates
- ✓ Requirement formalization
- ✓ Realizability checking
- ✓ Exporting requirement specs for monitor synthesis

Feedback:

"Crucial to the requirement formulation process were the interactive simulator of FRET and the realizability checking mechanism that guided the discussion to corner cases, important sanity checks, and complete requirement sets"

"FRETish allows for the easy specification of many complex and time-based interactions inside a system"

UTRC Ireland Aircraft Engine Controller

Features of FRET used:

- ✓ Requirement specification
- ✓ Requirement templates
- ✓ Requirement formalization
- ✓ Parent-child relations between requirement levels

Feedback:

"FRETish provides a useful conduit for conversation"

"FRETish requirements much more clear than natural-language requirements"

"Using FRET was useful because it forces you to think about the actual meaning behind the natural-language requirements"

examples of projects that used FRET

PLCverif at CERN (European Council for Nuclear Research): industrial control systems

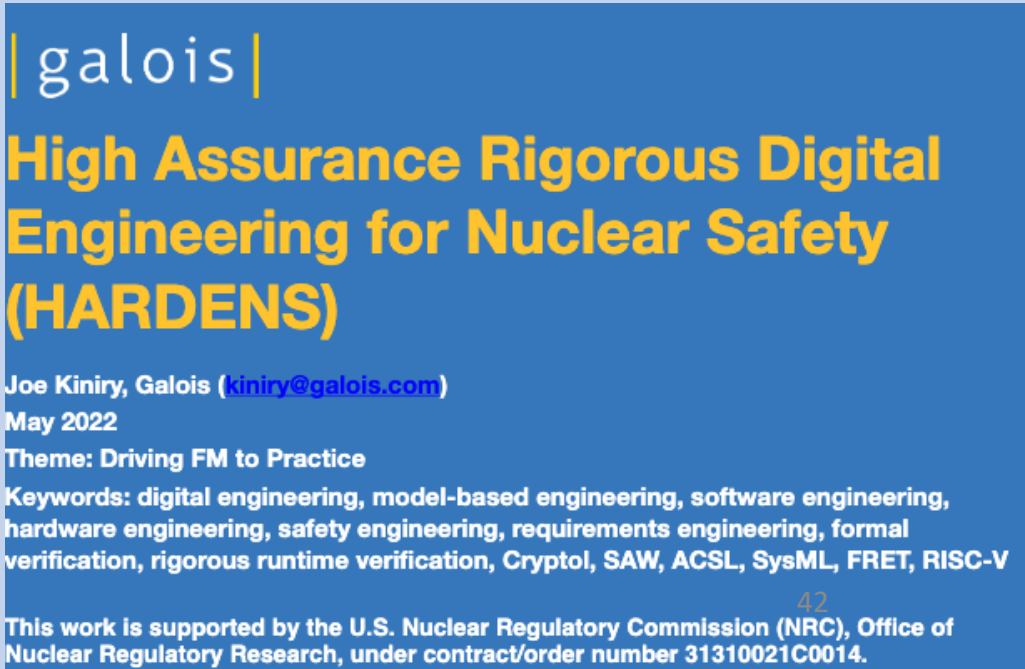
- ✓ Requirement specification
- ✓ Interactive simulator
- ✓ Requirement templates
- ✓ Requirement formalization
- ✓ Exporting requirement specifications for model checking and runtime monitoring industrial control code

Feedback:
"The creation of requirements in FRET fits well into the verification workflow and improves both usability and expressiveness"

Safety Demonstrator System for NRC

- ✓ Requirement specification
- ✓ Interactive simulator
- ✓ Requirement formalization
- ✓ Realizability checking

Feedback:
"FRET allows the user to use diagrams, simulators, and model checkers to explore requirement semantics. The user can use these tools to gain confidence that the formal representation accurately captures the intent of the corresponding natural language requirement"



examples of projects that used FRET

NASA Ames Research Center project Troupe: rovers that autonomously map their environment

FRET was used to capture and formalize requirements for model checking the Decision Making application

- ✓ Requirement specification
- ✓ Requirement templates
- ✓ Requirement formalization
- ✓ Exporting requirement specifications for model checking Simulink models

FRET was used to capture and formalize requirements for runtime monitoring CFS applications

- ✓ Requirement specification
- ✓ Interactive simulator
- ✓ Requirement formalization
- ✓ Exporting requirement specifications for runtime monitors

requirements for robotics

Integrating Formal Verification and Assurance: An Inspection Rover Case Study* **

Hamza Bourbouh[1], Marie Farrell[3***], Anastasia Mavridou[1], Irfan Sljivo[1], Guillaume Brat[2], Louise A. Dennis[4], and Michael Fisher[4]

[1] KBR/NASA Ames Research Center, USA
[2] NASA Ames Research Center, USA
[3] Department of Computer Science, Maynooth University, Ireland
[4] Department of Computer Science, University of Manchester, UK

Abstract. The complexity and flexibility of autonomous robotic systems necessitates a range of distinct verification tools. This presents new challenges not only for design verification but also for assurance approaches. Combining the distinct formal verification tools, while maintaining sufficient formal coherence to provide compelling assurance evidence is difficult, often being abandoned for less formal approaches. In this paper we demonstrate, through a case study, how a variety of distinct formal techniques can be brought together in order to develop a justifiable assurance case. We use the AdvoCATE assurance case tool to guide our analyses and to integrate the artifacts from the formal methods that we use, namely: FRET, COCOSIM and Event-B. While we present our methodology as applied to a specific Inspection Rover case study, we believe that this combination provides benefits in maintaining coherent formal links across development and assurance processes for a wide range of autonomous robotic systems.

Robotics: A New Mission for FRET Requirements

Gricel Vazquez[1], Anastasia Mavridou[2], Marie Farrell[3], Tom Pressburger[4], and Radu Calinescu[1]

[1] Department of Computer Science, University of York, UK
[2] KBR Inc., NASA Ames Research Center, USA
[3] Department of Computer Science, The University of Manchester, Manchester, UK
[4] NASA Ames Research Center, USA

Abstract. Mobile robots are used to support planetary exploration and safety-critical environments such as nuclear plants. Central to the development of mobile robots is the specification of complex required behaviors known as missions. In this paper, we use NASA's Formal Requirements Elicitation Tool (FRET) to specify functional robotic mission requirements. To examine the applicability of FRET in the mobile robotics domain, we studied robotic mission patterns that were specified in Linear Temporal Logic (LTL). These patterns were originally derived from a large repository that included patterns from the literature and consultation with industrial experts. We extend this repository with those found during our own extensive literature review. Although FRET has been successfully used in the past in case studies within the aerospace domain, mobile robot requirements present new challenges in their specification. To this end, our work provides a methodological basis for the use of FRET in the specification of robotic mission requirements.

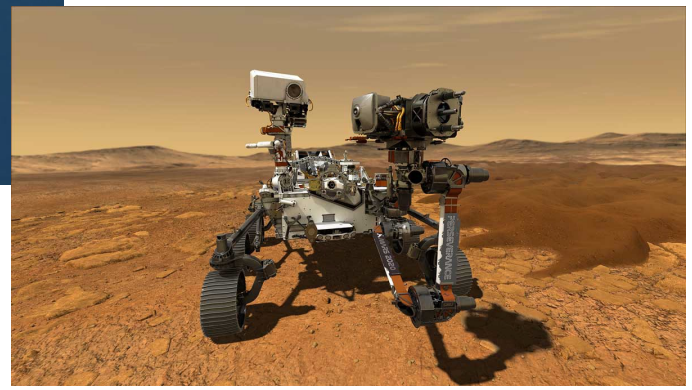
Hamza Bourbouh, Marie Farrell, Anastasia Mavridou, Irfan Sljivo, Guillaume Brat, Louise A. Dennis, Michael Fisher. Integrating Formal Verification and Assurance: An Inspection Rover Case Study, NFM 2021.

Gricel Vazquez, Anastasia Mavridou, Marie Farrell, Thomas Pressburger, Radu Calinescu. Robotics: A New Mission for FRET Requirements, NFM 2024.

new directions:
ongoing work with Marie Farrell and Michael Fisher

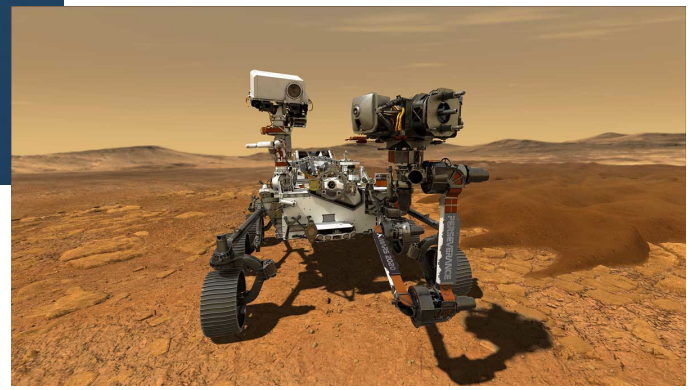
Illustration of NASA astronauts on the lunar South Pole. Credit: NA

Strong Reliability for Software that Learns



- Problem:
  - Use of **'non-traditional'** systems with **Machine Learning (ML)** components.
  - **Uncertainty** introduces a unique set of challenges related to the **specification** of **requirements.**

Strong Reliability for Software that Learns



- Our focus:
  - Identify common characteristics across sets of ML requirements.
  - Provide a framework for requirements specification, formalization, and analysis.

Examining ML requirements

Req ID	Requirement
	Literature Studies
[LR-001]	The aircraft location does not exceed a specified lateral offset from the runway centerline during taxiing.
[LR-002]	The aircraft does not veer off the sides of the runway during taxiing.
[LR-005]	The ML component shall identify the presence of any person present in the defined area with an accuracy of at least 0.93
	R-RAV Case Study
[RRAV-005]	Neural network shall output a sensible angle: the value must be between -90 and 90 degrees.
[RRAV-006]	The neural network shall achieve a minimum of X% accuracy on training and Y% accuracy on testing.
[RRAV-007]	(Local robustness) The neural network shall be robust to small perturbations in the image (pixels).
[RRAV-011]	The magnitude of the cross track heading error shall drop below X degrees within T seconds and remain there.
	Other Industrial Case Studies and Missions
[IC-001]	The s/w shall achieve an average PARAMETER value of X.
[IC-002]	The s/w shall estimate PARAMETER to within $\pm X$ with a Y% confidence.
[IC-012]	The s/w shall detect CONDITION that implies EVENT is probable.

We distilled 13 templates after analyzing 770 missions/industrial requirements.

ML requirement attributes and characteristics

- **Confidence, Criticality, and Risk Levels:**
  - Characterizes the amount of confidence that must be achieved to assure that the corresponding requirement is met.
  - *“The sw shall determine PARAMETER with a high level of confidence.”*

ML requirement attributes and characteristics

- Confidence, Criticality, and Risk Levels:
  - *“The sw shall determine PARAMETER with a high level of confidence.”*
- Accuracy as a Measure of Functional Correctness:
  - Defines the rate at which the ML must answer correctly.
  - *“The neural network shall achieve a minimum of X% accuracy on training and Y% accuracy on testing. ”*

ML requirement attributes and characteristics

- Confidence, Criticality, and Risk Levels:
  - *“The sw shall determine PARAMETER with a high level of confidence.”*
- Accuracy as a Measure of Functional Correctness:
  - *“The neural network shall achieve a minimum of X% accuracy on training and Y% accuracy on testing. ”*
- Achievement of Average Value:
  - Evaluates the consistency of the output data of an ML system.
  - “*The requirement shall be verified by measuring the average of the parameter over N repetitions.”*

ML requirement attributes and characteristics

- Confidence, Criticality, and Risk Levels:
  - *“The sw shall determine PARAMETER with a high level of confidence.”*
- Accuracy as a Measure of Functional Correctness:
  - *“The neural network shall achieve a minimum of X% accuracy on training and Y% accuracy on testing. ”*
- Achievement of Average Value:
  - *“The requirement shall be verified by measuring the average of the parameter over N repetitions.”*
- Robustness:
  - *“The neural network shall be robust to small perturbations in the image (pixels)”*

Uncertainty in ML requirements

- Not always probabilistic:
  - *"Neural network shall output a sensible angle: the value must be between -90 and 90 degrees."*

- Probabilities within requirements:
  - *"The probability of avoiding collision with an obstacle shall be greater than 99% over the course of the mission."*

Uncertainty in ML requirements

- Not always probabilistic:
  - *"Neural network shall output a sensible angle: the value must be between -90 and 90 degrees."*

- Probabilities within requirements:
  - *"The probability of avoiding collision with an obstacle shall be greater than 99% over the course of the mission."*

FRETish fields with probability

CONDITION	COMPONENT		PROBABILITY	TIMING	RESPONSE
Whenever taxing,	the aircraft	shall	with probability > 0.9999	always	satisfy turn < prescribedDegree

CONDITION	null, trigger, continual
PROBABILITY	almostsure, bound, query
TIMING	always, never, eventually, immediately, finally, for, within, after, until, before
RESPONSE	satisfaction

FRETish fields with probability

CONDITION	COMPONENT		PROBABILITY	TIMING	RESPONSE
Whenever taxing,	the aircraft	shall	with probability > 0.9999	always	satisfy turn < prescribedDegree

CONDITION	null, trigger, continual
PROBABILITY	almostsure, bound, query
TIMING	always, never, eventually, immediately, finally, for, within, after, until, before
RESPONSE	satisfaction

Probabilistic formalizations in PRISM format

- Automatically translate requirements into probabilistic temporal logics.

- Probabilistic invariance:
  - *The sw shall with probability >= 0.95 for 1000 seconds satisfy ! error*
  - P>=0.95[G<=1000 (! error)]
- Probabilistic existence:
  - *The rover shall with probability < 0.01 eventually satisfy battery_depleted*
  - P<0.01[F battery_depleted]
- Probabilistic response
  - *Whenever error, the rover shall with probability > 0.98 always satisfy ! collision*
  - P>= 1[G (error => P>0.98 [G (! collision)])]
- Conditional probabilities

FRETish for autonomy

- Requirements formalization for autonomous systems is challenging.
  - Tools need to be able to capture requirements related to uncertainty.

Our mission includes:

- Studying the characteristics of real-life requirements for autonomy.
- Providing a taxonomy of requirements for autonomy.
- Providing an *intuitive platform for capturing requirements with rigorous semantics.*

- Probably FRET?
  - github.com/NASA-SW-VnV/fret

acknowledgements

Zsófia Ádám, Alexanders Bakst, Swee Balachandran, Milan Bhandari, Marcin Bęś, Enrique Blanco Viñuela, Geoffrey Biggs, David Bushnell, Maxime Artaud, Hamza Bourbouh, Guillaume Brat, Esther Conrad, Louise A. Dennis, Tanja DeJong, Michael Dille, Aaron Dutle, Marie Farrell, Borja Fernandez Adiego, Michael Fisher, Pierre-Loic Garoche, Dimitra Giannakopoulou, Alwyn Goodloe, Simon Hansen, Kelly Ho, Michael Jeronimo, George Karamanolis, Andreas Katis, Joseph Kiniry, David Kooi, Ignacio D. Lopez-Miguel, Carlos Mao de Ferro, Patrick J. Martin, Francisco Martins, Amalaye Oyake, Ivan Perez, Jessica Phelan, Tom Pressburger, Julian Rhein, Daniel Riley, Johann Schumann, Nija Shi, Irfan Sljivo, Laura Titolo, Jean-Charles Tournier, Khanh V. Trinh, Gricel Vazquez, Tim Wang, Michael W. Whalen, Alexander Will.

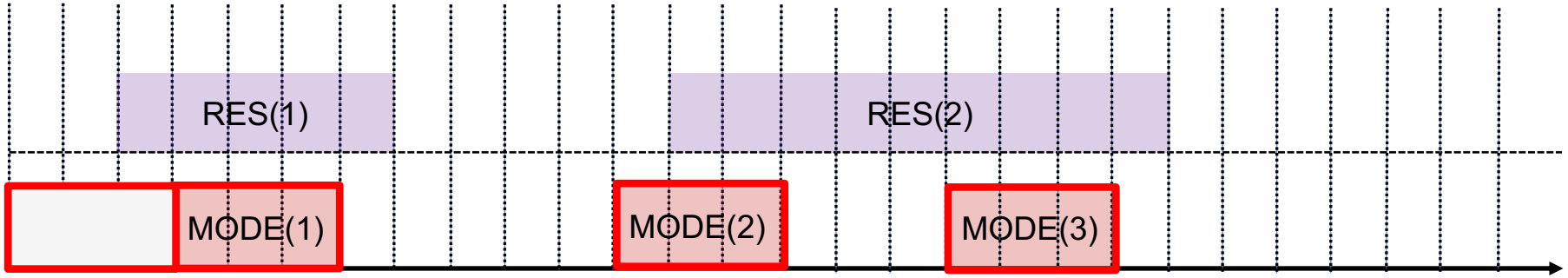
RSE software engineering tools

Tools	Description	Availability	Technical POC	POC Email
FRET	Requirement elicitation and analysis	Open Source	Anastasia Mavridou	anastasia.mavridou@nasa.gov
CoCoSim	Simulink model analyzer	Open Source	Andreas Katis	andreas.katis@nasa.gov
IKOS	Static code analysis for C/C++	Open Source	Ivan Perez	Ivan.perezdominguez@nasa.gov
AdvoCATE	Assurance case automation toolset	Open Source	Ewen Denney	ewen.w.denney@nasa.gov
MARGInS	ML/statistical libraries for system testing	Usage Agreement	Carlos Paradis	carlos.v.paradis@nasa.gov
SysAI	ML/statistical libraries for system testing	Ask us	Yuning He	yuning.he@nasa.gov
AdaStress	Adaptive stress testing	Open Source	Rory Lipkis	rory.lipkis@nasa.gov
RACE	Runtime for Airspace Concept Evaluation	Open Source	Peter Mehlitz	peter.c.mehlitz@nasa.gov
MESA	Run-time analysis of live data streams	Open Source	Nastaran Shafiei	nastaran.shafiei@nasa.gov
Prophecy	Formal analysis of Neural Networks	Ask us	Corina Pasareanu	corina.s.pasareanu@nasa.gov
R2U2	Vehicle-level run-time analysis	Usage Agreement	Johann Schumann	johann.m.schumann@nasa.gov

Thank you!

backup slides

oracles (1)



Oracle for scope: in, condition: null, timing: always, response: satisfaction

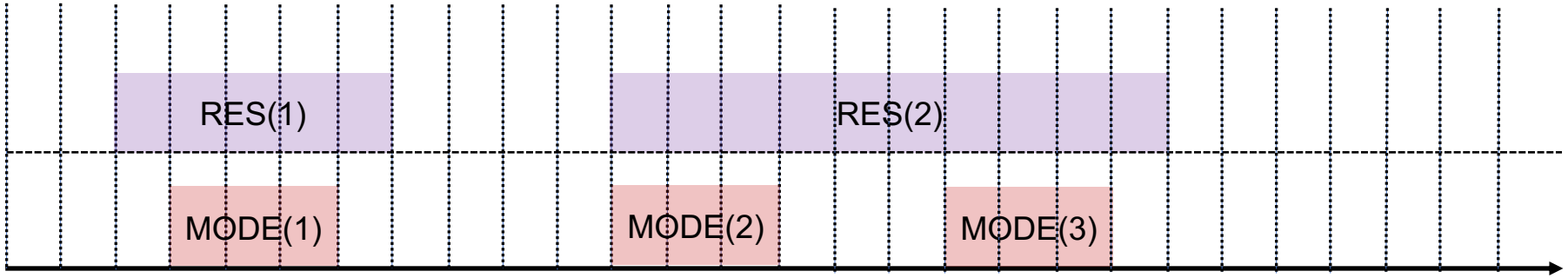
responseIntervals = {RES(1), RES(2)}

step1: determine scope intervals
result: scopeIntervals = {MODE(1), MODE(2), MODE(3)}

step 2: apply timing always
for each scopeInterval in scopeIntervals
there exists a responseInterval in responseIntervals such that:
responseInterval contains scopeInterval

Expected value: false (MODE(2) not contained in RES(1) or RES(2))

oracles (2)



Oracle for scope:in, condition: null, timing: always, response: satisfaction

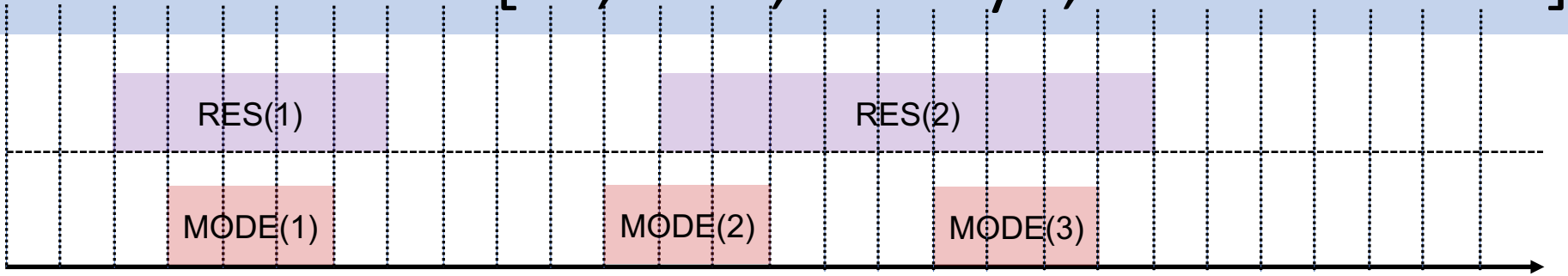
responseIntervals = {RES(1), RES(2)}

step1: determine scope intervals
result: scopeIntervals = {MODE(1), MODE(2), MODE(3)}

step 2: apply timing always
for each scopeInterval in scopeIntervals
there exists a responseInterval in responseIntervals such that:
responseInterval contains scopeInterval

Expected value: true (MODE(1) contained in RES(1); MODE(2) & MODE(3) contained in RES(2))

Verification [in,null,always,satisfaction]



trace in SMV

```
DEFINE
   LAST := (t = 27);
   RES := case
          t < 2 : FALSE ;
          t <= 7 : TRUE ;
          t < 12 : FALSE ;
          t <= 21 : TRUE ;
          TRUE : FALSE ;
esac ;
   COND := case
          TRUE : FALSE ;
esac ;
   MODE := case
          t < 3 : FALSE ;
          t <= 6 : TRUE ;
          t < 11 : FALSE ;
          t <= 14 : TRUE ;
          t < 17 : FALSE ;
          t <= 20 : TRUE ;
          TRUE : FALSE ;
esac ;
```

→ `(H (MODE → RES))` holds at timepoint LAST?

SMV returns: false expected: false ✓

example traces are generated automatically:

1) brute force; 2) targeted scenarios; 3) random scenarios

we also use SMV to check equivalence between:

- past and future time properties for each template key
- optimized and non-optimized formulas