



Formal Requirement Elicitation with FRET

Anastasia Mavridou

KBR at NASA Ames Research Center

anastasia.mavridou@nasa.gov

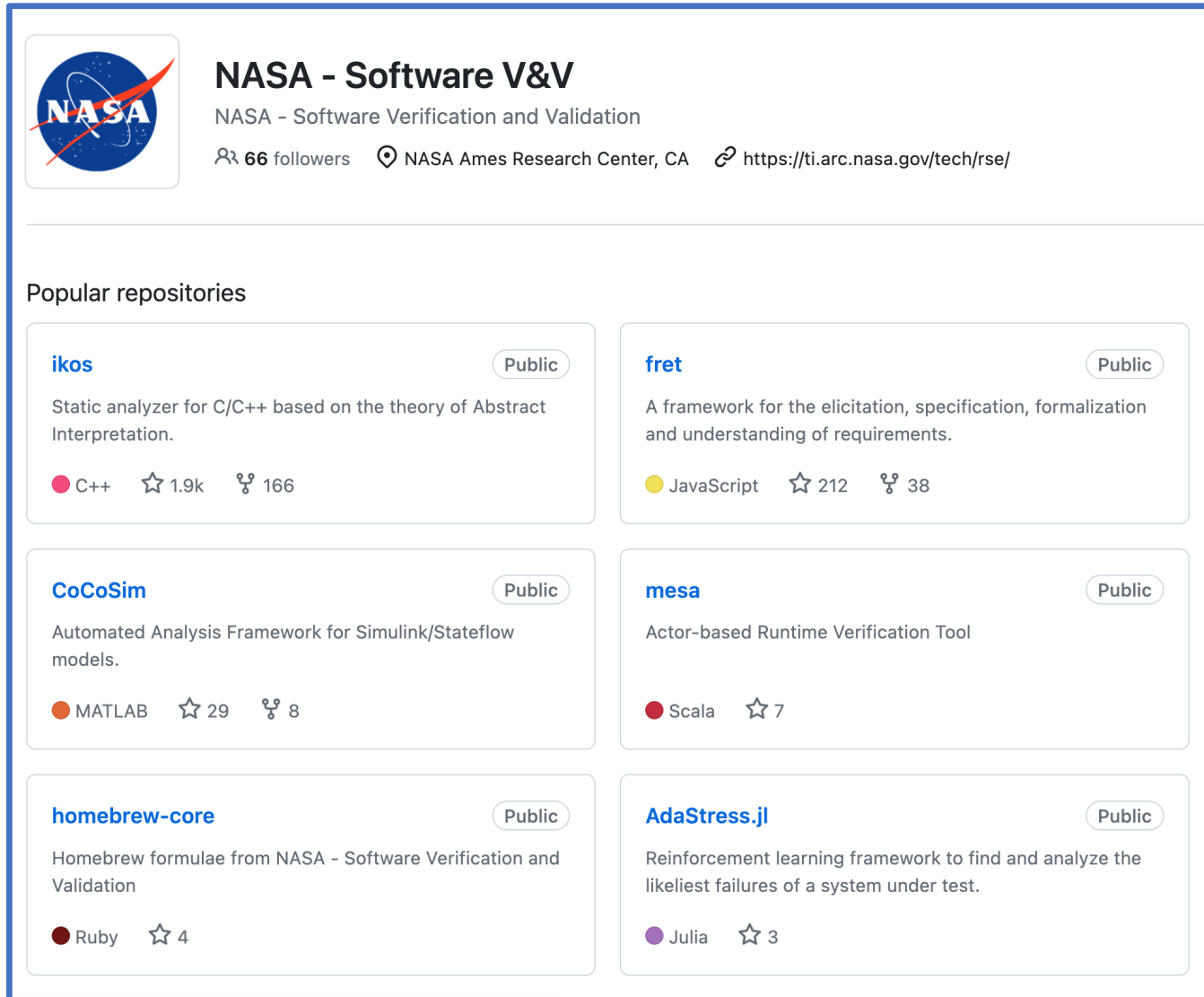
Invited talk at DTU

NASA's center in Silicon Valley



Aerial image of NASA Ames Research Center
Credits: NASA

robust software engineering technical area



The screenshot shows the GitHub profile for NASA - Software V&V. It includes the NASA logo, the repository name, a description, follower count, location, and website URL. Below this, there is a section for popular repositories with six items listed in a grid. Each item shows the repository name, a 'Public' badge, a description, and statistics for stars and forks.

NASA - Software V&V
NASA - Software Verification and Validation
66 followers NASA Ames Research Center, CA <https://ti.arc.nasa.gov/tech/rse/>

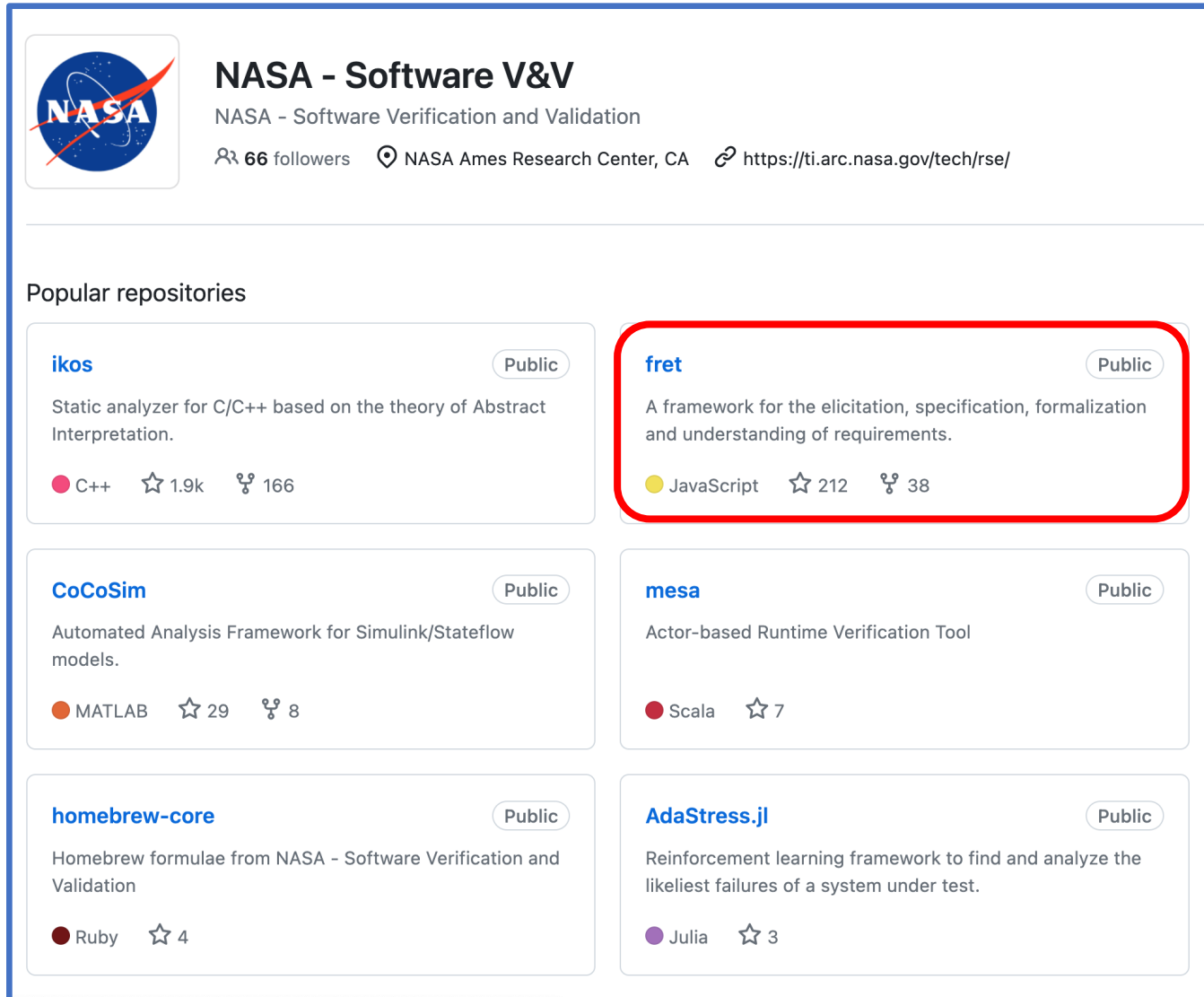
Popular repositories

Repository Name	Language	Stars	Forks
ikos	C++	1.9k	166
fret	JavaScript	212	38
CoCoSim	MATLAB	29	8
mesa	Scala	7	
homebrew-core	Ruby	4	
AdaStress.jl	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 stars | 166 forks
- fret** (Public) A framework for the elicitation, specification, formalization and understanding of requirements.
JavaScript | 212 stars | 38 forks
- CoCoSim** (Public)
Automated Analysis Framework for Simulink/Stateflow models.
MATLAB | 29 stars | 8 forks
- mesa** (Public)
Actor-based Runtime Verification Tool
Scala | 7 stars
- homebrew-core** (Public)
Homebrew formulae from NASA - Software Verification and Validation
Ruby | 4 stars
- AdaStress.jl** (Public)
Reinforcement learning framework to find and analyze the likeliest failures of a system under test.
Julia | 3 stars

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). Instantly, or within a time limit?
- 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.
- ...

what analysis tools understand

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

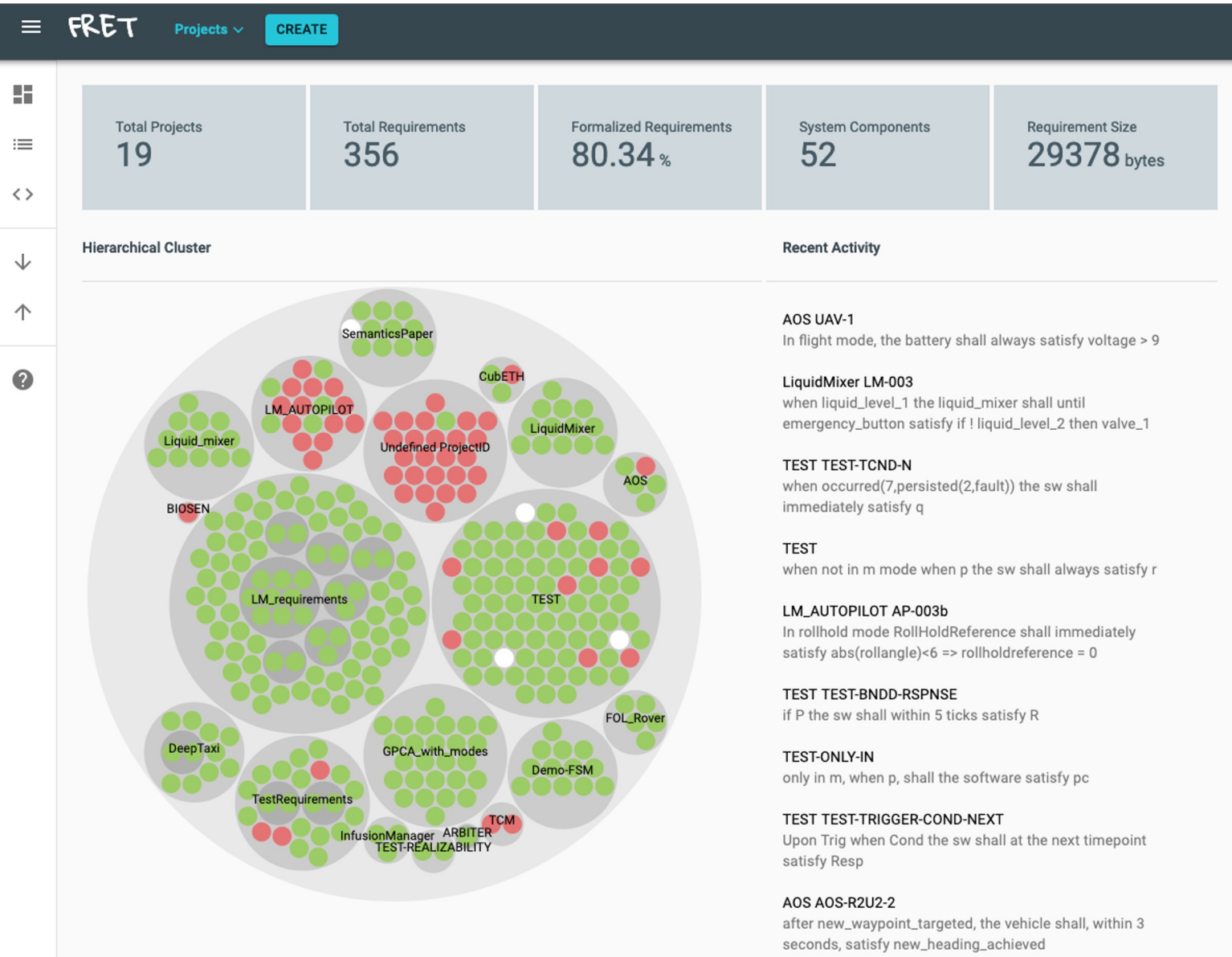
```
var autopilot: bool = (not standby) and supported and (not
  apfail);
var pre_autopilot: bool = false -> pre autopilot;
var pre_limits: bool = = false -> pre limits;
guarantee "FSM-001v2" S((((autopilot and pre_autopilot and
  pre_limits) and (pre (not (autopilot and pre_autopilot and
  pre_limits)))) 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.
- **Checks** realizability of requirements compositionally.
- **Connects** with analysis tools:
 - Exports formalizations in SMV language.
 - Exports Lustre code.
 - Exports specifications for runtime monitoring.



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

And many other collaborators..

capturing, explaining, and formalizing requirements

let's speak FRETish

The screenshot shows the FRET web application interface. At the top, there is a navigation bar with the FRET logo, a 'Projects' dropdown menu, and a 'CREATE' button. The main content area is titled 'Create Requirement'. It features a form with the following fields:

- Requirement ID:** AP-Test
- Parent Requirement ID:** (empty)
- Project:** HAMLET_SW

Below the form, there are two text input areas:

- Rationale and Comments:** A large text area containing the text: "When in cruising mode, the altitude hold autopilot shall maintain altitude whenever altitude_hold is selected."
- Requirement Description:** A text area that is currently empty.

Under the 'Requirement Description' section, there is a sentence structure guide: "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." Below this text is a row of colored bubbles representing the sentence structure: SCOPE (red), CONDITIONS (orange), COMPONENT* (green), SHALL* (grey), TIMING (blue), and RESPONSES* (purple). A question mark icon is located to the right of these bubbles.

At the bottom right of the main content area, the word 'SEMANTICS' is visible.

On the right side of the interface, there is a sidebar with two tabs: 'ASSISTANT' (selected) and 'TEMPLATES'. The 'ASSISTANT' tab contains the text: "Ready to speak FRETish? Please use the editor on your left to write your requirement or pick a predefined template from the TEMPLATES tab."

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

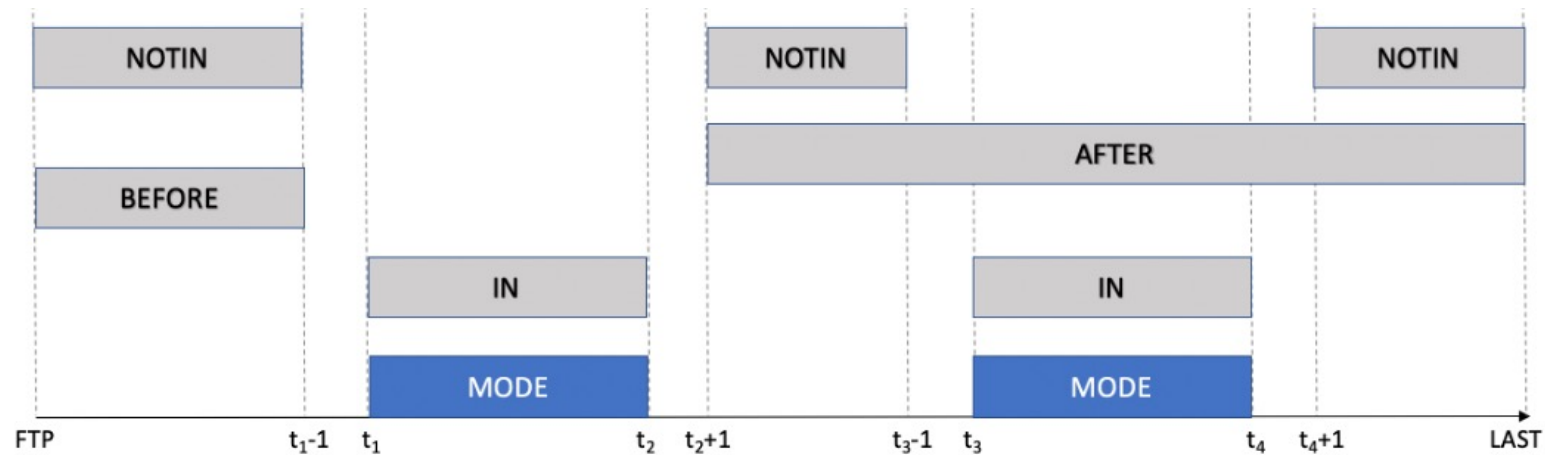
compositional generation of LTL formulas

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

scope in: [LEFT, RIGHT) \rightarrow [FiM, LiM)

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

LiM = not MODE and previous MODE



compositional generation of LTL formulas

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

scope in: [LEFT, RIGHT) \rightarrow [FiM, LiM)

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

LiM = not MODE and previous MODE

timing always: BASEFORM \rightarrow 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 \rightarrow RES))

instantiate (H (cruising \rightarrow (altitude_hold \rightarrow maintain_altitude)))

related papers

Automated Formalization of Structured Natural Language Requirements

Dimitra Giannakopoulou^{a,*}, Thomas Pressburger^a, Anastasia Mavridou^b,
Johann Schumann^b

^aNASA Ames Research Center, CA, USA

^bKBR, 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.

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?

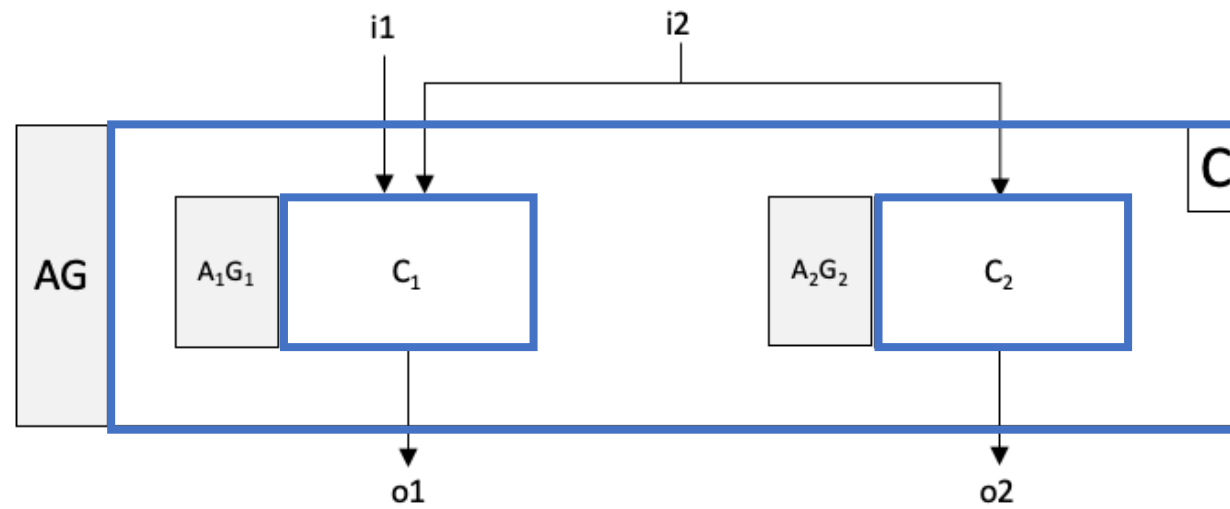
- Defining requirements is a challenging, error prone task
- Realizability checking >> consistency checking
- We want to ensure requirement consistency for **all inputs**
- And we want to do it **efficiently**

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.

We proposed a novel approach for **compositional realizability checking**.

compositional realizability

Partial AG contracts:



checking realizability within FRET

The screenshot shows the FRET web application interface. At the top, there is a navigation bar with 'File View Help' and the 'FRET' logo. Below the logo, there are 'Projects' and 'CREATE' buttons. The main content area is divided into two tabs: 'VARIABLE MAPPING' and 'REALIZABILITY', with 'REALIZABILITY' being the active tab. Under the 'REALIZABILITY' tab, there is a 'System Component *' dropdown menu set to 'FSM'. To the right of this, there are checkboxes for 'Compositional' (checked) and 'Monolithic' (unchecked). Further right, there is a 'Timeout (seconds)' field set to '900'. Below these settings are four buttons: 'CHECK' (highlighted in blue), 'DIAGNOSE', 'EXPORT', and 'HELP'. Below the buttons, there are three tabs for context: 'CC0', 'CC1', and 'CC2', with 'CC2' being the active tab. The main area displays a table with 10 rows of realizability checks. The table has two columns: 'ID' and 'Summary'. The rows contain the following data:

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

At the bottom right of the table, there is a pagination control showing 'Rows per page: 10' and '1-10 of 13'.

related papers

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

Anastasia Mavridou¹, Andreas Katis¹, Dimitra Giannakopoulou², David Kooi³, Thomas Pressburger², and Michael W. Whalen⁴

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 Katis¹[0000-0001-7013-1100], Anastasia Mavridou¹, Dimitra Giannakopoulou^{2*}, Thomas Pressburger², and Johann Schumann¹

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

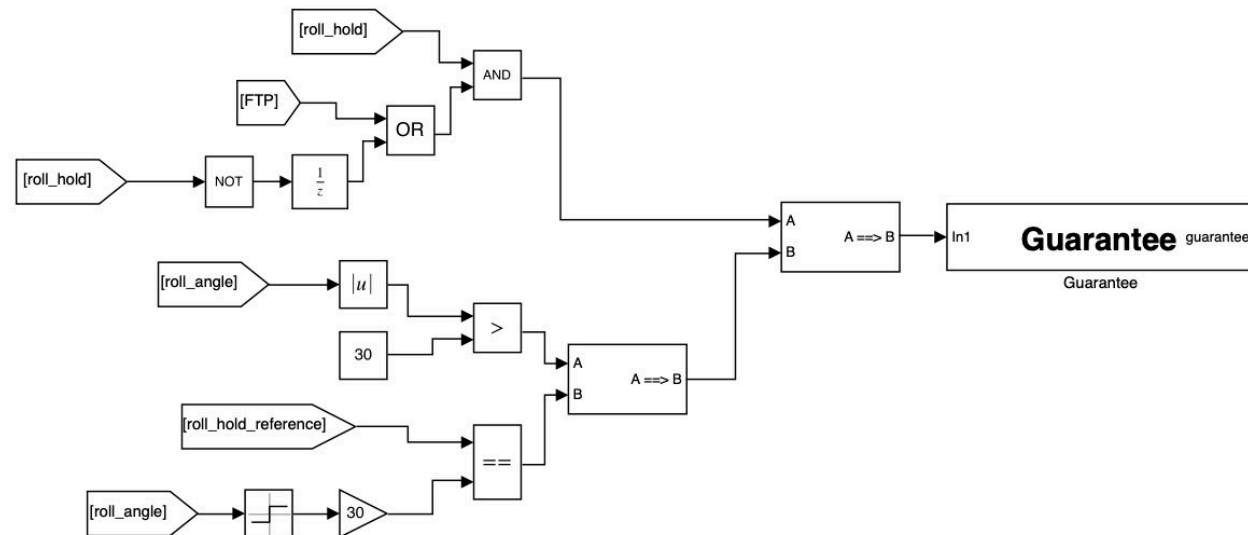
FRETish:

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

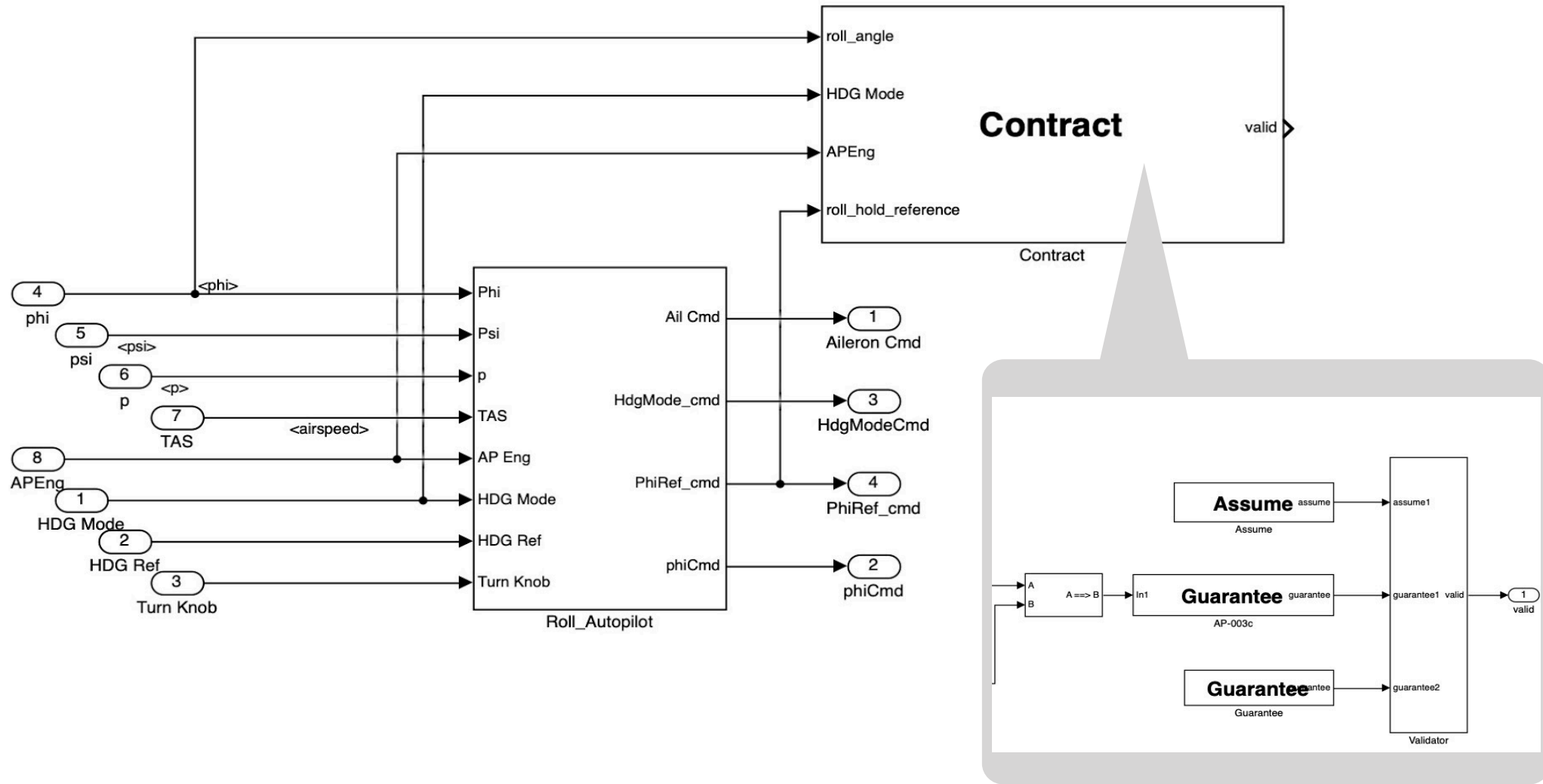
Lustre specification:

```
-- 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

The screenshot displays the PLVerif GUI application interface. The main window is titled "FRET_Requirement.vc3 (verification case)". The interface is divided into several sections:

- Project Explorer:** Shows a tree view of the project files, including "src-gen", "builtin.scl", "CPC_BASE_Unicos.scl", "CPC_FB_ONOFF.scl", "CPC_GLOBAL_VARS.scl", and "FRET_Requirement.vc3".
- Verification case:**
 - Metadata:** Contains fields for "ID" (FRET_Requirement), "Name" (if the ONOFF object is in "Manual mode" and the condition "Auto Auto Mode Request" is TRUE, the ONOFF object shall eventually...), and "Description" (Checking transition between Manual Mode and Auto Mode).
 - Source files:** A list of source files with checkboxes. The checked files are "builtin.scl" and "*scl (all scl files in this project's root)". A "Reload source files" button is present.
 - Language frontend:** Set to "STEP 7".
 - Entry block:** Set to "CPC_FB_ONOFF".
 - Verification backend:** Set to "NuSMV" with the algorithm "IC3 (nuXmv only)".
 - Advanced settings:** A section for further configuration.
- Requirement:** A section for defining the requirement to be verified. It includes a "Requirement type" dropdown set to "FRET requirement", an "Edit in FRET" button, and a text area containing the requirement description: "When (instance.MMoSt & instance.AuAuMoR) the CPC_FB_OnOff shall eventually satisfy instance.AuMoSt & PLC_END". Below this, a complex LTL formula is shown: $((G (((!(instance.MMoSt \& instance.AuAuMoR)) \& (X (instance.MMoSt \& instance.AuAuMoR)))) \rightarrow (X (F (instance.AuMoSt \& \{PLC_END\})))))) \& (((instance.MMoSt \& instance.AuAuMoR) \rightarrow (F (instance.AuMoSt \& \{PLC_END\}))))$. Labels "Fretish requirement:" and "TL requirement:" are visible to the right of the formula.
- Requirement - advanced:** A section for advanced configuration.
- Reporters:** A section for selecting report generators.
- Advanced settings (0):** A section for advanced settings.
- Verify:** A section with a "Verify!" button and status information: "Last result: N/A", "Last execution: N/A", and "Last duration: N/A". An "Open report" button is also present.
- Diagnostics:** A section for diagnostic information.

model checking PLC code

The screenshot displays the PLCverif GUI application with the 'Update Requirement' dialog box open. The dialog is titled 'Update Requirement' and contains the following sections:

- Metadata:** Fields for ID (FRET_Requirement), Name, and Description.
- Rationale and Comments:** A text area for 'Rationale' containing the text: "if the ONOFF object is in 'Manual mode' and the condition 'Auto Auto Mode Request' is TRUE, the ONOFF object shall eventually be in 'Auto Mode' at the end of the PLC cycle". A 'Comments' field contains: "Checking transition between Manual Mode and Auto Mode".
- Requirement Description:** A section explaining the sentence structure with a template: "When (instance.MMoSt & instance.AuAuMoR) the CPC_FB_OnOff shall eventually satisfy instance.AuMoSt & PLC_END".
- SEMANTICS:** A list of variables and their types, including CPC_DB_VERSION, CPC_GLOBAL_VARS, instance, and various AI and AIB variables.

The background shows the 'Verification case' configuration for 'FRET_Requirement.vc3', including source files and verification backend settings.

runtime monitoring



examples of case studies/projects that use FRET



|galois|

High Assurance Rigorous Digital Engineering for Nuclear Safety (HARDENS)

Joe Kiniry, Galois (kiniry@galois.com)

May 2022

Theme: Driving FM to Practice

Keywords: digital engineering, model-based engineering, software engineering, hardware engineering, safety engineering, requirements engineering, formal verification, rigorous runtime verification, Cryptol, SAW, ACSL, SysML, FRET, RISC-V

This work is supported by the U.S. Nuclear Regulatory Commission (NRC), Office of Nuclear Regulatory Research, under contract/order number 31310021C0014.



related papers

Zsófia Ádám, Ignacio D. Lopez-Miguel, Anastasia Mavridou, Thomas Pressburger, Marcin Beś, 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 Slijivo, 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>

source code

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

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

Ogma: <https://github.com/nasa/ogma>

PLCverif: <https://gitlab.com/plcverif-oss>

acknowledgements

Zsófia Ádám, Alexanders Bakst, Swee Balachandran, Milan Bhandari, Marcin Beś, 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.

Thank you!