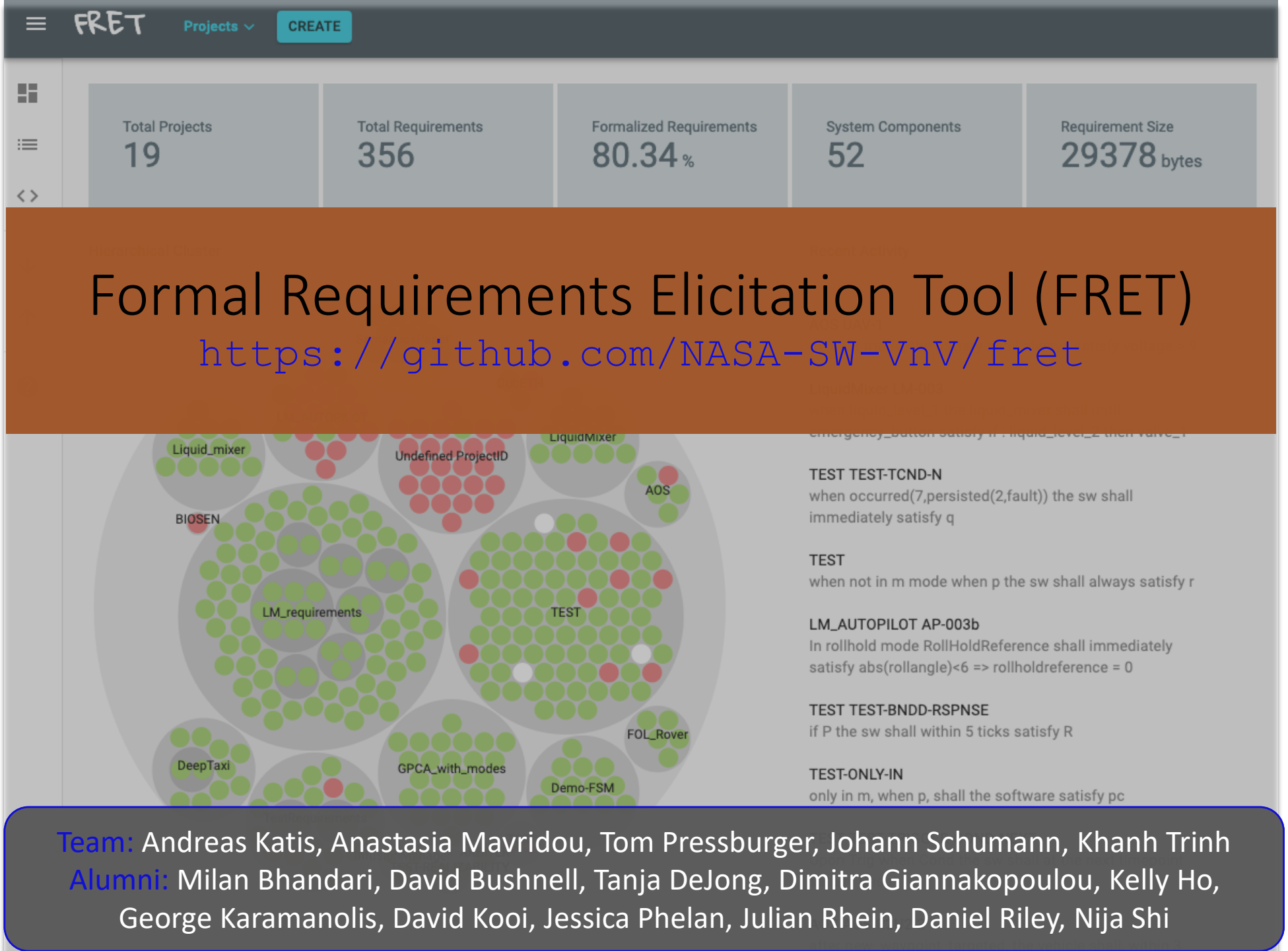
Requirements elicitation, analysis and verification using FRET and CoCoSim

Andreas Katis[1]

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



Formal Requirements Elicitation Tool (FRET)

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

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

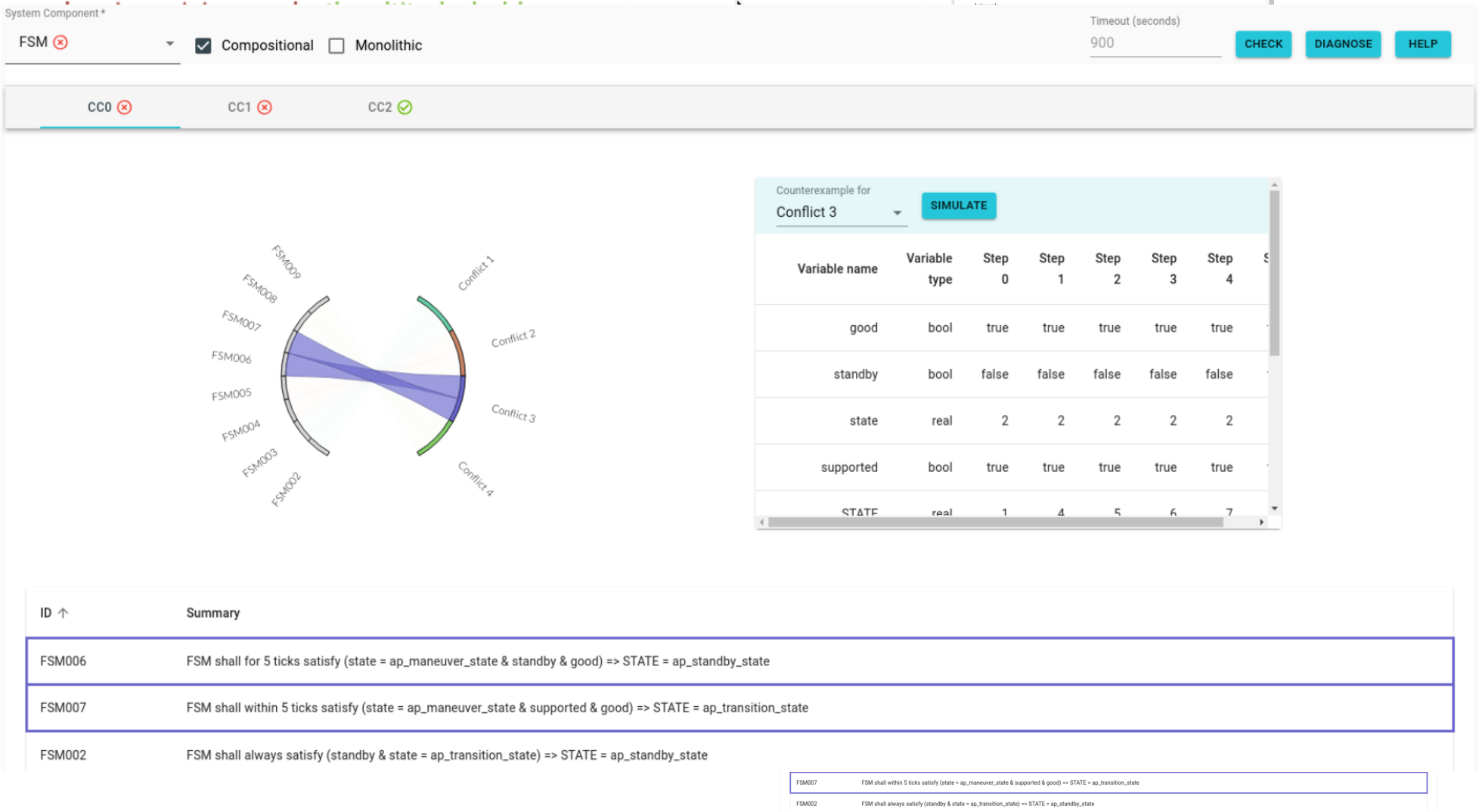
Alumni: Milan Bhandari, David Bushnell, Tanja DeJong, Dimitra Giannakopoulou, Kelly Ho, George Karamanolis, David Kooi, Jessica Phelan, Julian Rhein, Daniel Riley, Nija Shi

captures + assists

explains

Realizability Checking in FRET (Live Demo)

FRETish



Realizability Checking

[FSM-006]	*"The autopilot shall change **states** from* MANEUVER *to* STANDBY *when the pilot is in control (**standby**) and sensor data is **good** and stay in this state for the next 5 ticks."*
[FSM-007]	*"The autopilot shall, within the next 5 ticks, change **states** from* MANEUVER *to* TRANSITION *when the system is **supported** and sensor data is **good**."*

- **Reactive Systems**: *Inputs* controlled by the environment; *outputs* controlled by system.

  Inputs:

  standby, supported, good : Bool

  Outputs:

  STATE : {MANEUVER, STANDBY, TRANSITION}

- Formal Analysis

  Consistency: Are the requirements satisfiable?

  Realizability: Are the requirements satisfiable for any input provided by the environment?

Realizability in Requirements Specification Tools

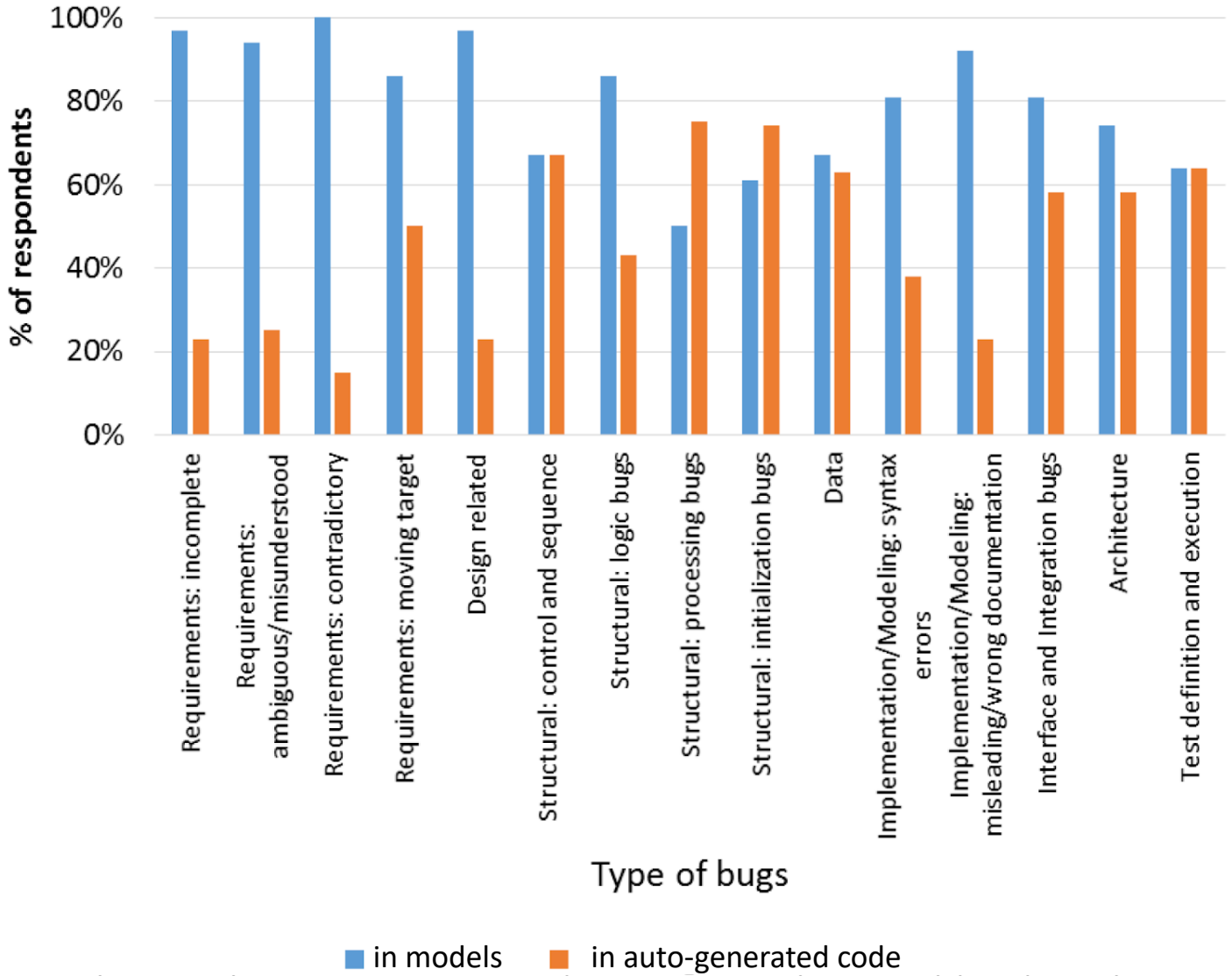
Tools	Finite State	Infinite State	Decomposition	Liveness	Unrealizable Cores
Spectra	✓	✗	✗	✓	✓
SpeAR	✓	✓	✗	✗	✗
AGREE	✓	✓	✗	✗	✗
RATSY	✓	✗	✗	✓	✓
EARS-CTRL	✓	✗	✗	✗	✗
FRET	✓	✓	✓	✗	✓

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

Tools	Finite State	Infinite State	Decomposition	Liveness	Unrealizable Cores	Algorithms	Backend	Other features
Spectra	✓	✗	✗	✓	✓	BDD-based fixpoint	CUDD + JTLV	Well-separation, Vacuity Checking, Counterstrategies
SpeaAR	✓	✓	✗	✗	✗	*k*-induction	JKind	N/A
AGREE	✓	✓	✗	✗	✗	*k*-induction	JKind	N/A
RATSY	✓	✗	✗	✓	✓	BDD-based fixpoint	CUDD + NuSMV	Counterstrategies
EARS-CTRL	✓	✗	✗	✗	✗	BDD-based fixpoint	autoCode4	N/A
FRET	✓	✓	✓	✗	✓	*k*-induction, SMT-based fixpoint	JKind, Kind2	Simulation of conflicting requirements

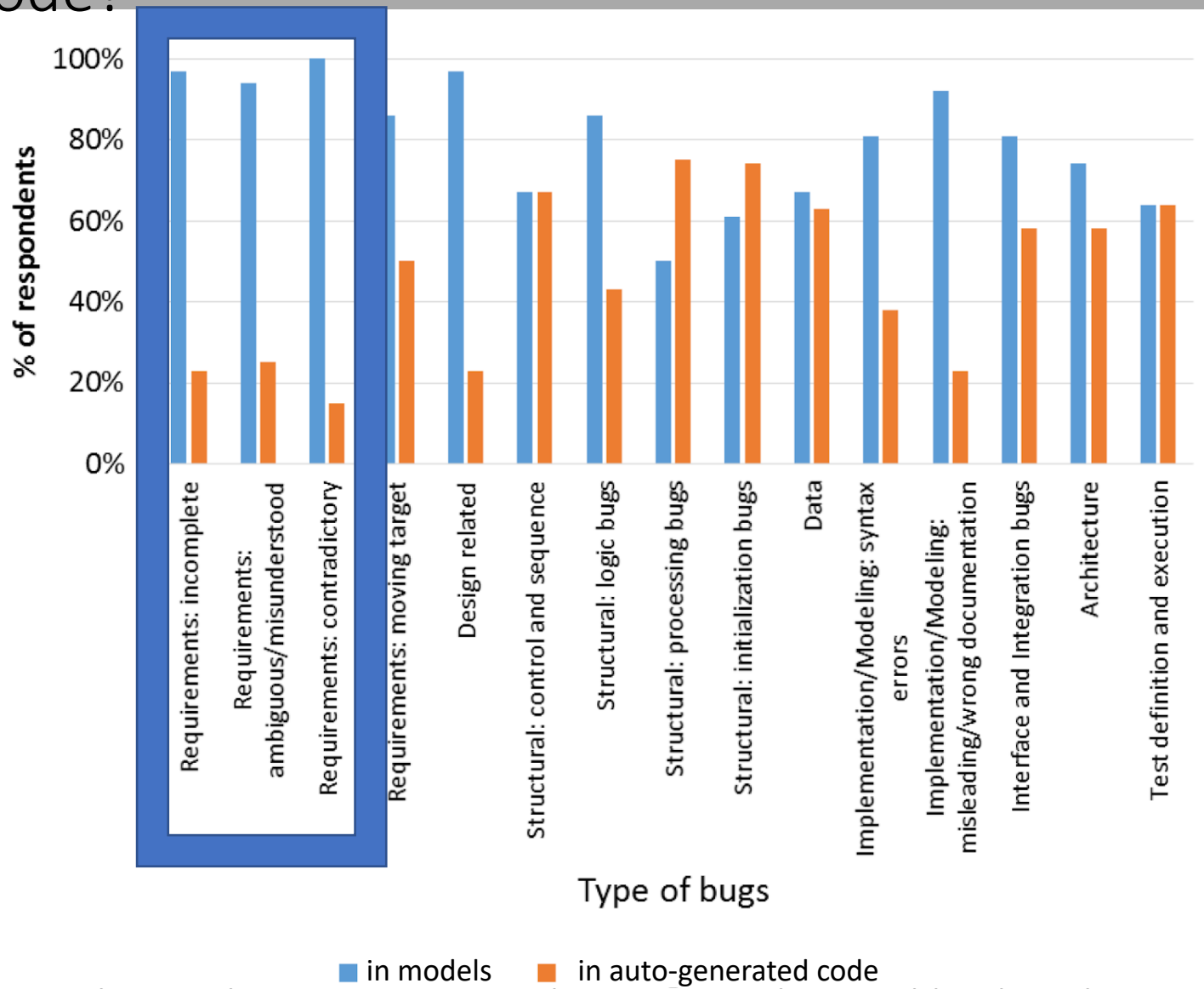
1 Employed by KBR; NASA Ames Research Center, CA, USA 2
NASA Ames Research Center, CA, USA

What types of bugs are found in models and code?



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

What types of bugs are found in models and code?



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

language of developers forced to write reqs

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).

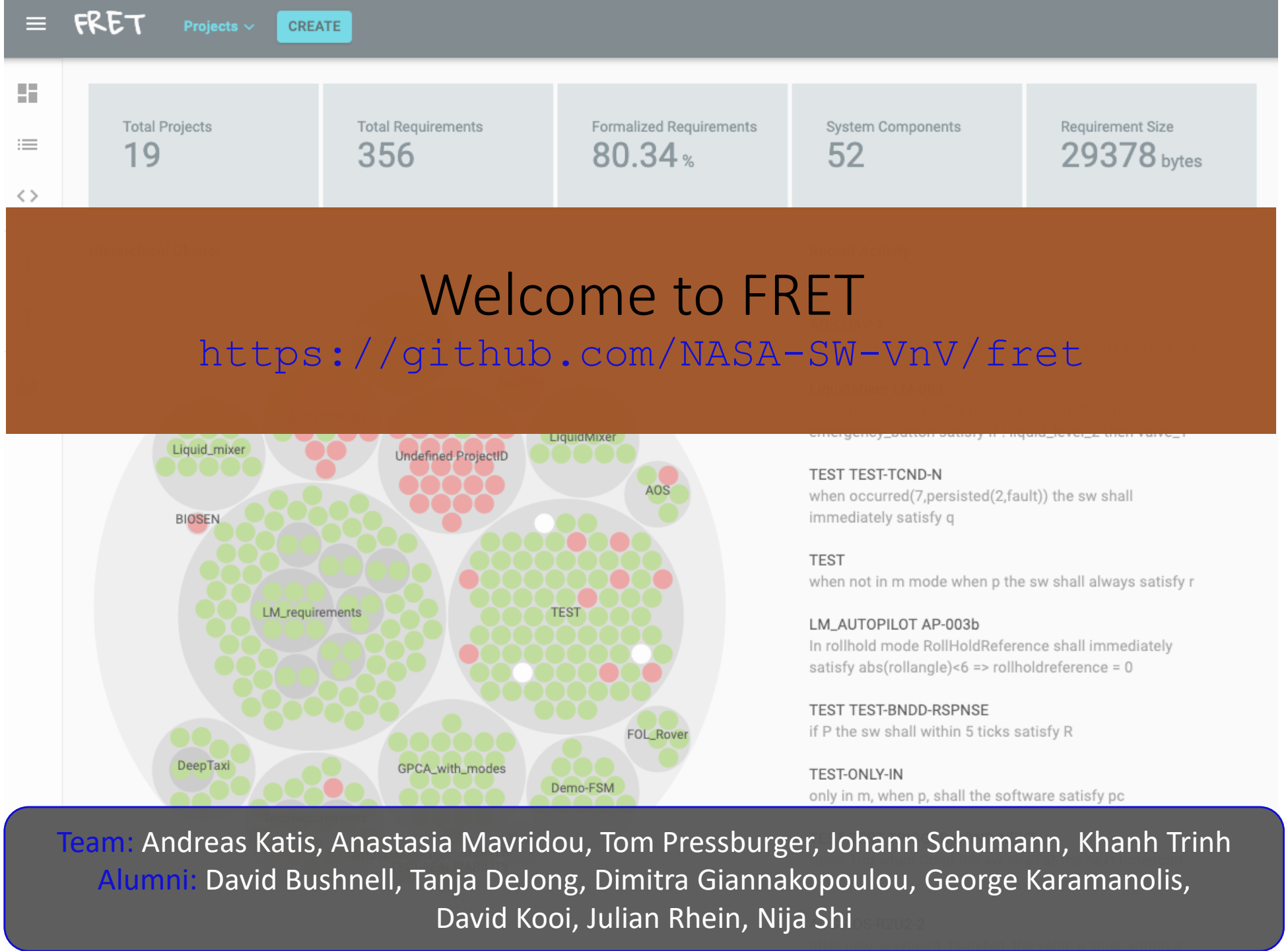
every time these conditions hold or only when they **become** true?

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

are these requirements consistent? does my model/code satisfy them?

language formal analysis tools understand

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



Welcome to FRET

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

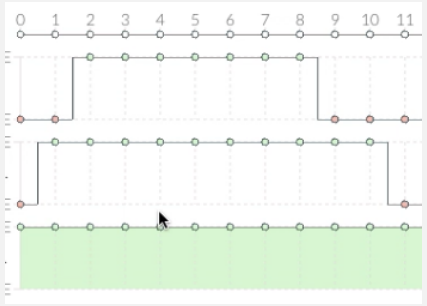
Team: Andreas Katis, Anastasia Mavridou, Tom Pressburger, Johann Schumann, Khanh Trinh
Alumni: David Bushnell, Tanja DeJong, Dimitra Giannakopoulou, George Karamanolis, David Kooi, Julian Rhein, Nija Shi

captures + assists

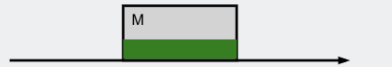
FRETish

when in cruising mode, the altitude_hold_

explains

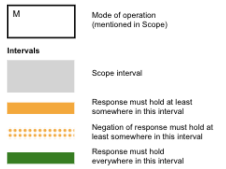


ENFORCED: in every interval where ***cruising*** holds. TRIGGER: first point in the interval. REQUIRES: for every trigger, RES must hold at all time points between (and including) the trigger and the end of the interval.

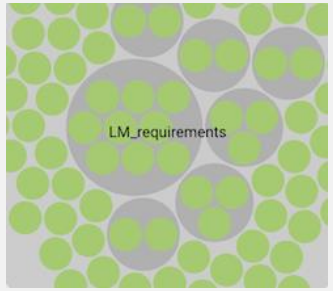


M = ***cruising***, Response = ***(altitude_hold => maintain_altitude)***.

Diagram Semantics



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

diagnoses

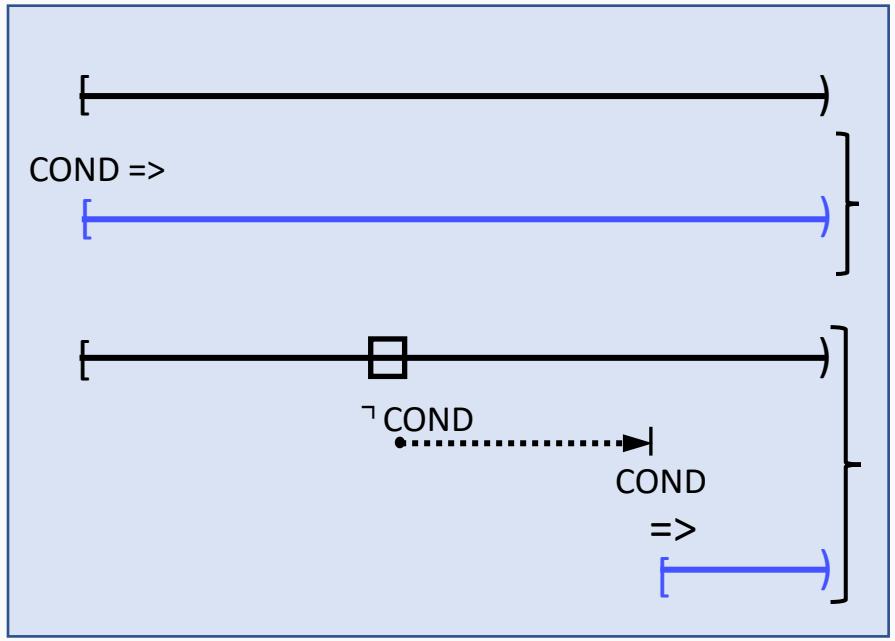
		Step 4	Step 5
rue	true	true	true
rue	true	true	true
rue	true	true	true

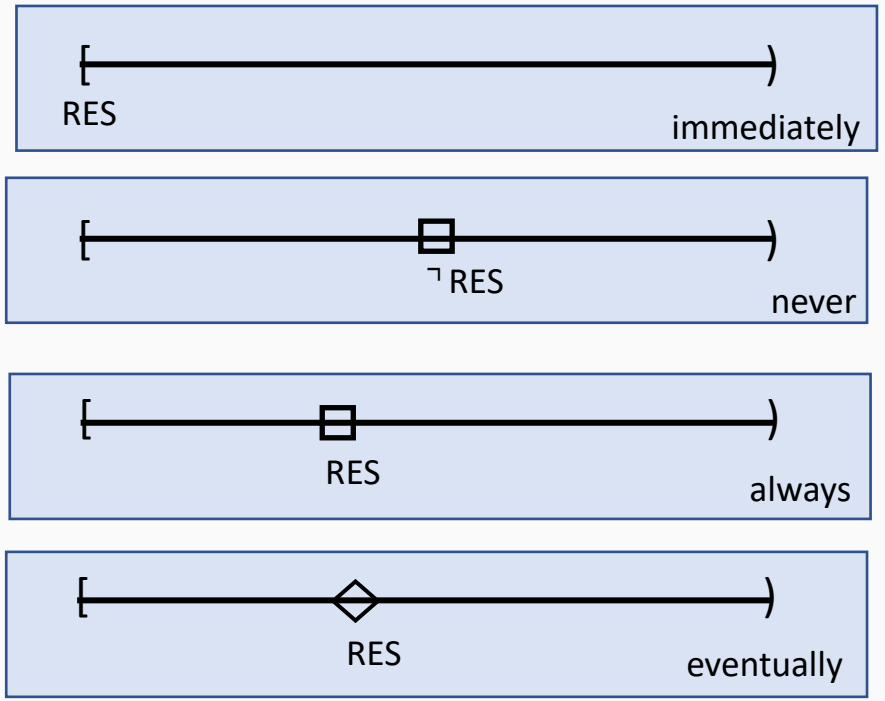
FRET bridges the gap

- 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

FRET is rigorous and extensible

- semantic templates have RTGIL semantics. FRET generates formulas in future- (finite and infinite-trace) and past-time metric temporal logics. A verification framework within FRET ensures correctness of formalization algorithms.
- all aspects of our approach are compositional – based on requirement fields.





Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Johann Schumann: "Automated Formalization of Structured Natural Language", *Information and Software Technology*, 2021

FRET bridges the gap

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

capturing requirements in FRETish

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

scope	condition	component	timing	response
	if altitude-hold_selected	the altitude_hold_autopilot shall	always satisfy	maintain_altitude

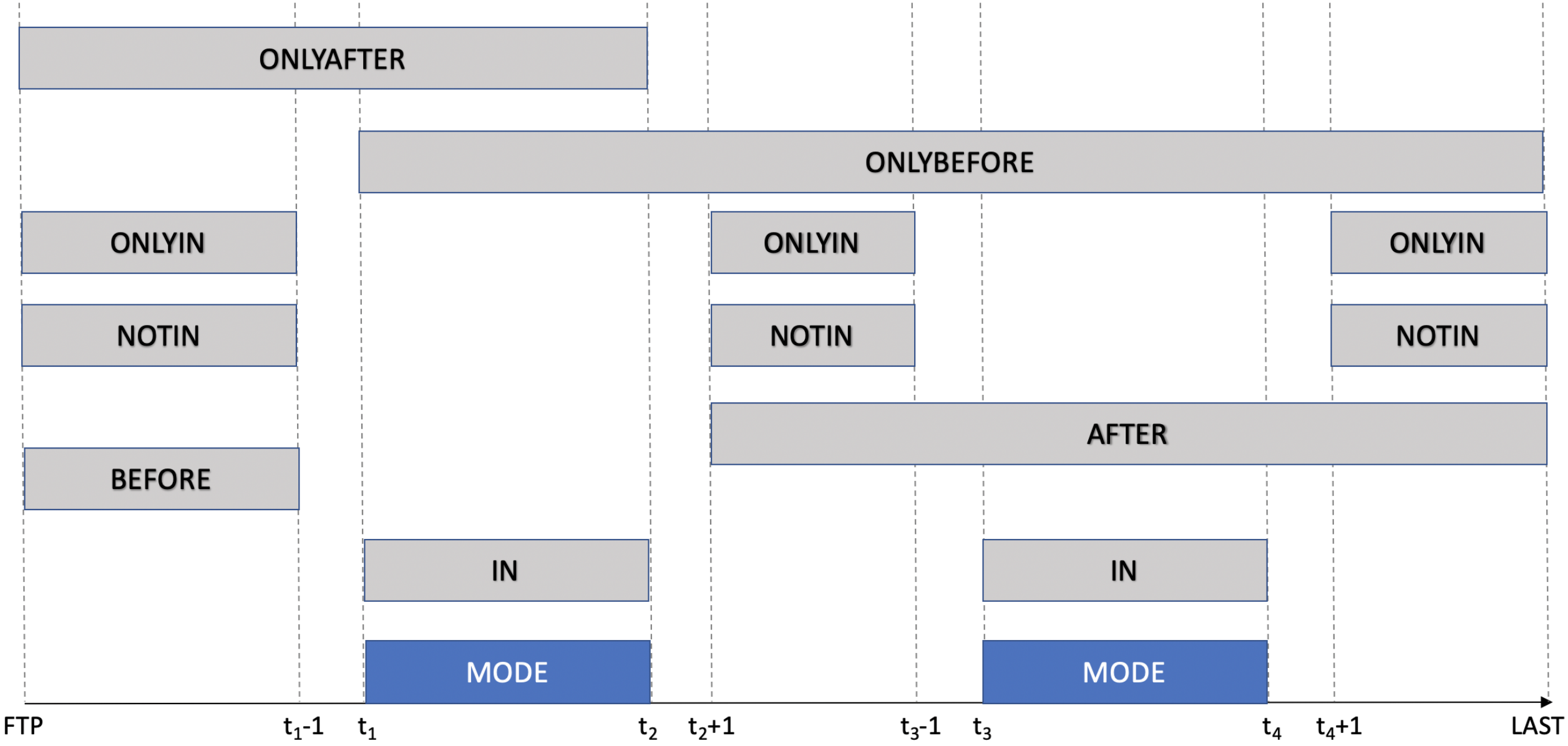
SCOPE	in, before, after, notin, onlyIn, onlyBefore, onlyAfter; when omitted, global
CONDITION	null, regular
TIMING	immediately, next, always, never, eventually, until, before, for, within, after
RESPONSE	satisfaction

160 semantic templates / template keys!

Scopes

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

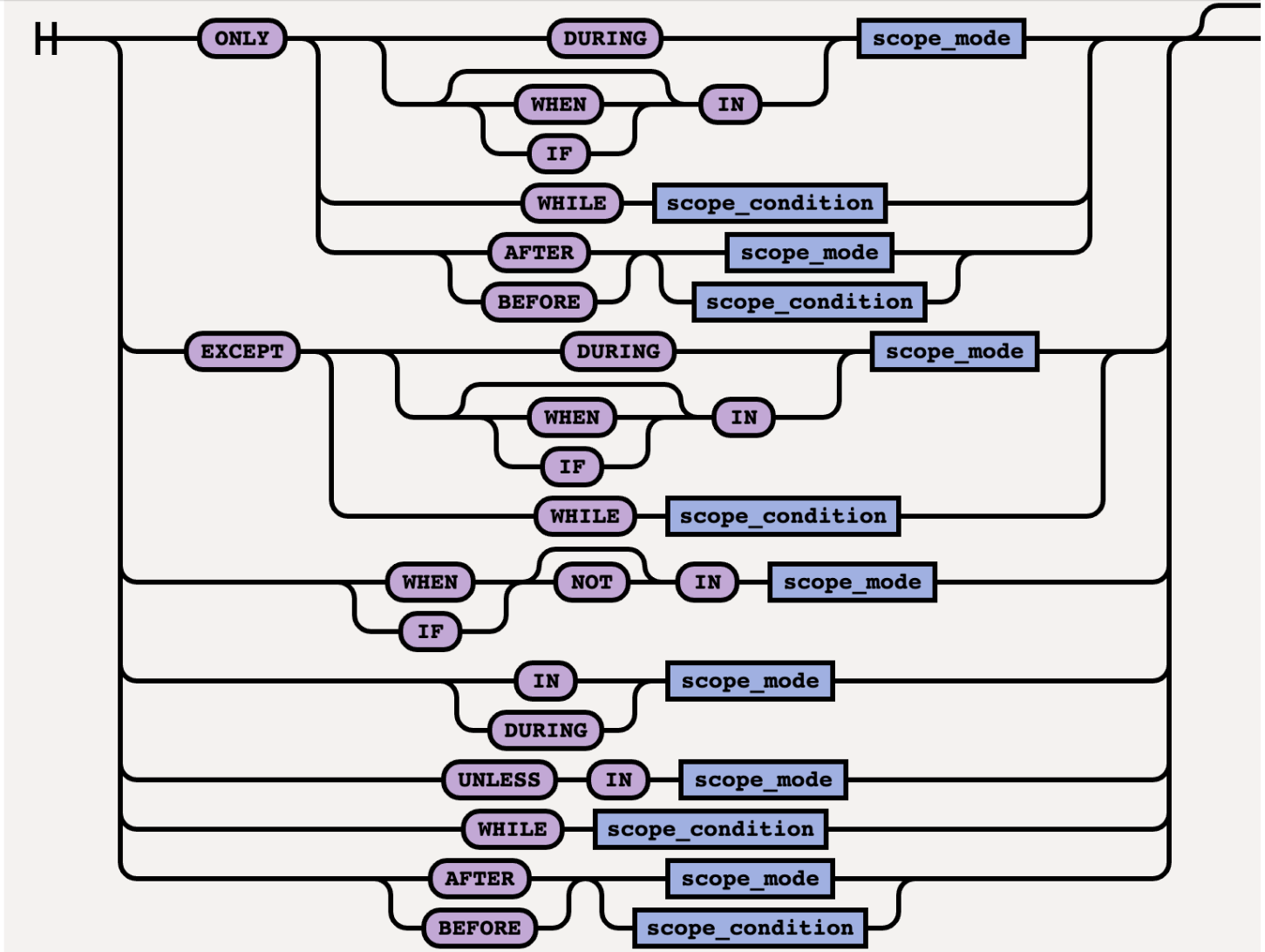
Scope Intervals



Scopes (contd)

- **While** mode = 4 the watch shall always satisfy alarm_icon_on
- **While** persisted(4,high_temperature) the monitor shall until shutoff satisfy alarm_on
- **Before** taxiing & receivedClearance the plane shall always satisfy !takeoff
- **After** landed & powerOff the doors shall within 5 seconds satisfy unlocked

Scope grammar

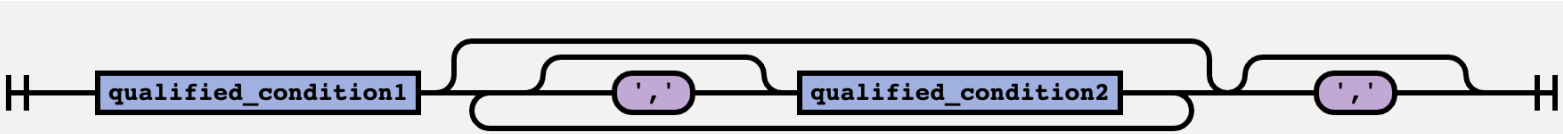


Conditions

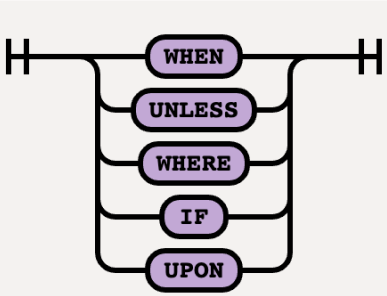
- *upon, if, when, where; unless*
- Boolean expression
- Trigger: **upon** the Boolean expression becoming true from being false in the scope, or being true at the beginning of the scope.

Condition grammar

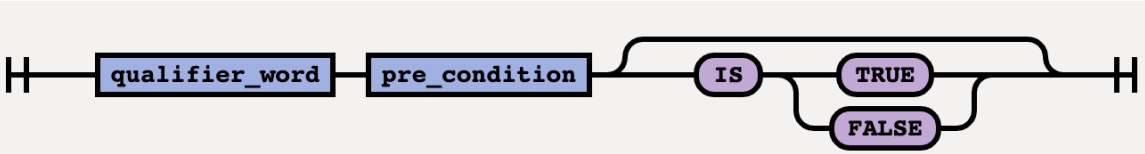
regular_condition



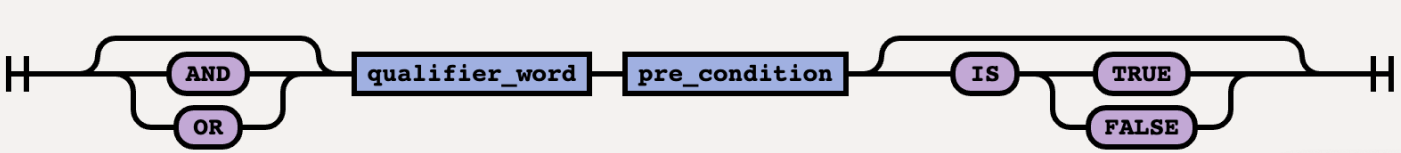
qualifier_word



qualified_condition1



qualified_condition2



Timings

- 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

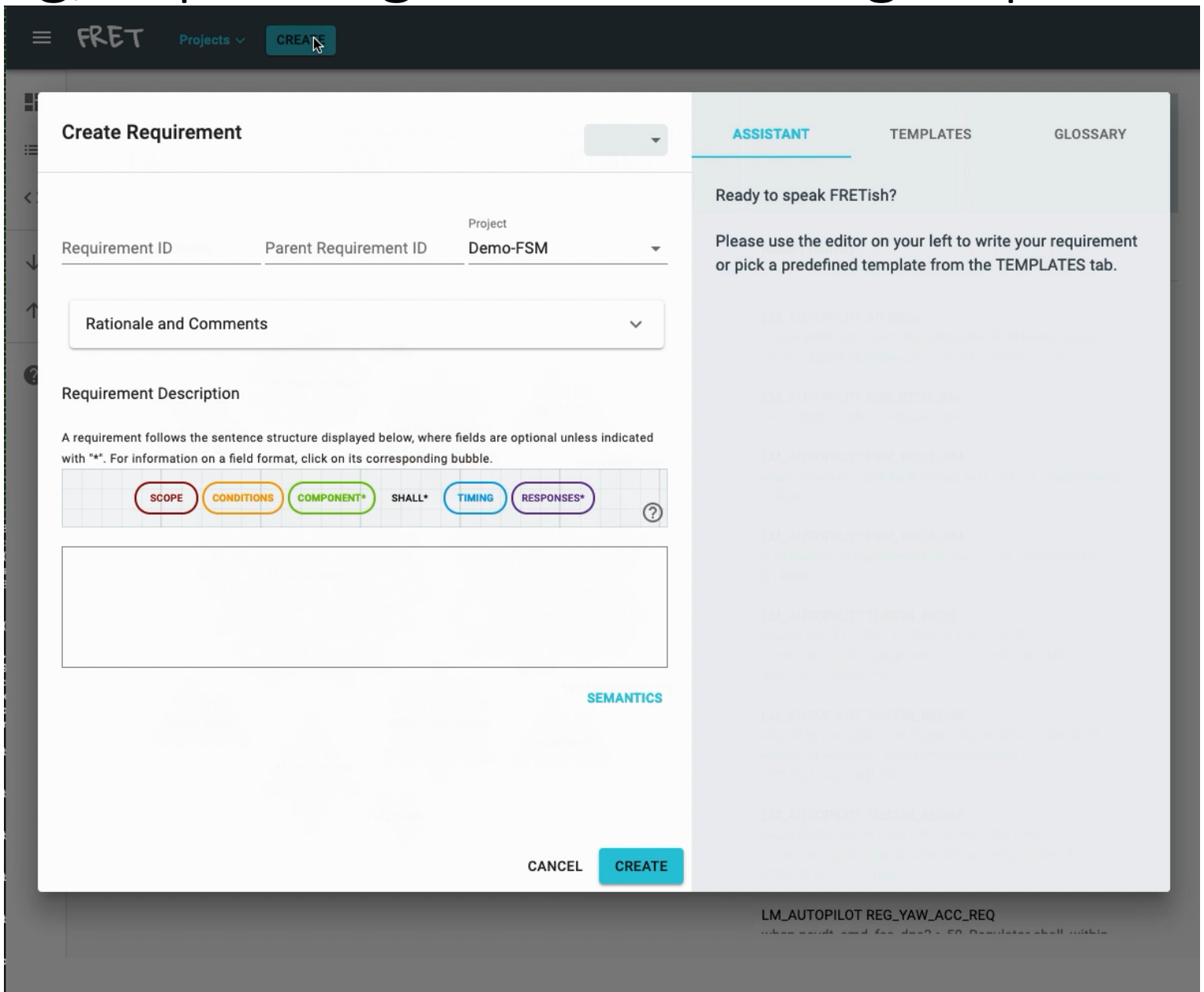
FRET bridges the gap

Captures requirements in a restricted natural language with unambiguous semantics

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

- Assists in writing requirements through requirement templates
- Formalizes requirements in a compositional (hence maintainable and extensible) manner: *past, future linear temporal logic, Lustre*
- Checks consistency of requirements and provides feedback
- Connects with analysis tools and exports verification code
  - ✓ for model checking Simulink models with CoCoSim
  - ✓ for model checking Lustre code with Kind2
  - ✓ for runtime analysis of C programs with Copilot

Capturing, explaining and formalizing requirements



Update Requirement

Requirement ID: Test-ALTHOLD

Parent Requirement ID

Project: LM_requirements

Rationale and Comments

Rationale

Comments

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

Requirement Description

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

SCOPE | CONDITIONS | COMPONENT* | SHALL* | TIMING | RESPONSES*

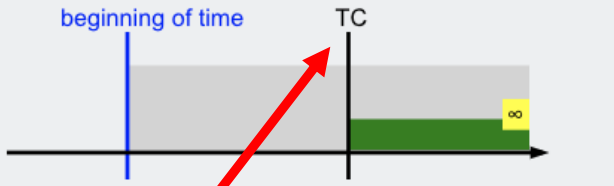
if altitude_hold_selected the altitude_hold_autopilot shall always satisfy maintain_altitude

SEMANTICS

but this is not what I mean…

Semantics

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



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

Diagram Semantics

Formalizations

Future Time LTL

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

Target: ***altitude_hold_autopilot*** component.

Past Time LTL

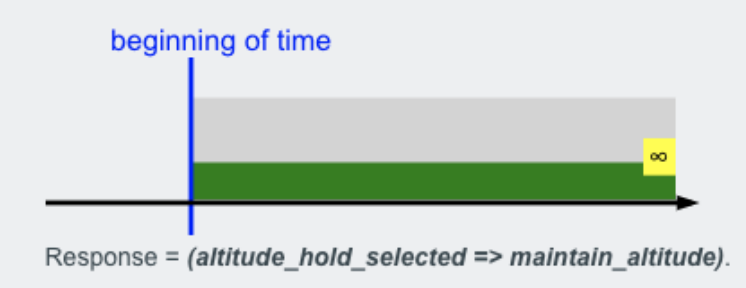
```
(H ((H (! (altitude_hold_selected))) |
(maintain_altitude)))
```

Target: ***altitude_hold_autopilot*** component.

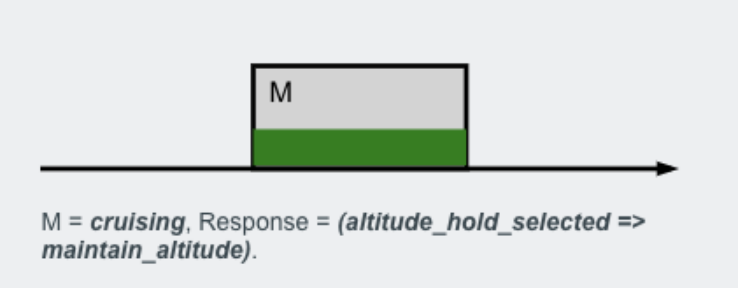
getting to the right requirement

TAKE1: if altitude_hold_selected the altitude_hold_autopilot shall always satisfy maintain_altitude

TAKE2: the altitude_hold_autopilot shall always satisfy if altitude_hold_selected then maintain_altitude



TAKE3: when in cruising mode, the altitude_hold_autopilot shall always satisfy if altitude_hold_selected then maintain_altitude



FRET bridges the gap

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

Assistance: Requirement templates

Lockheed Martin Cyber-Physical System Challenge, component FSM:

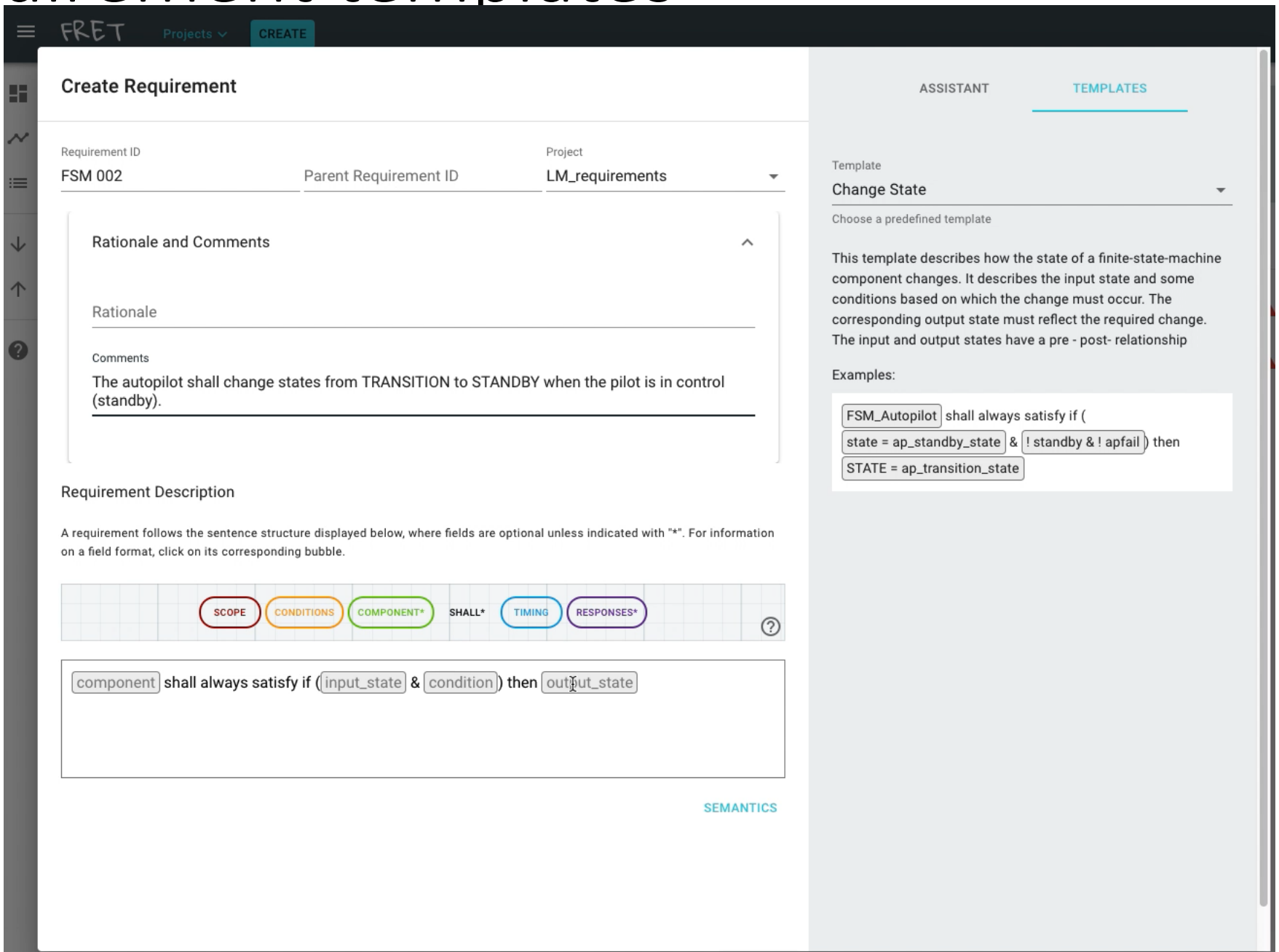
- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data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

Requirement templates

Lockheed Martin Cyber-Physical System Challenge, component FSM:

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

Requirement templates



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 consistency

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 consistency

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 consistency

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 consistency

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 

Checking Realizability

- Realizable requirements: A system exists that satisfies the requirements for *every* valid environment input

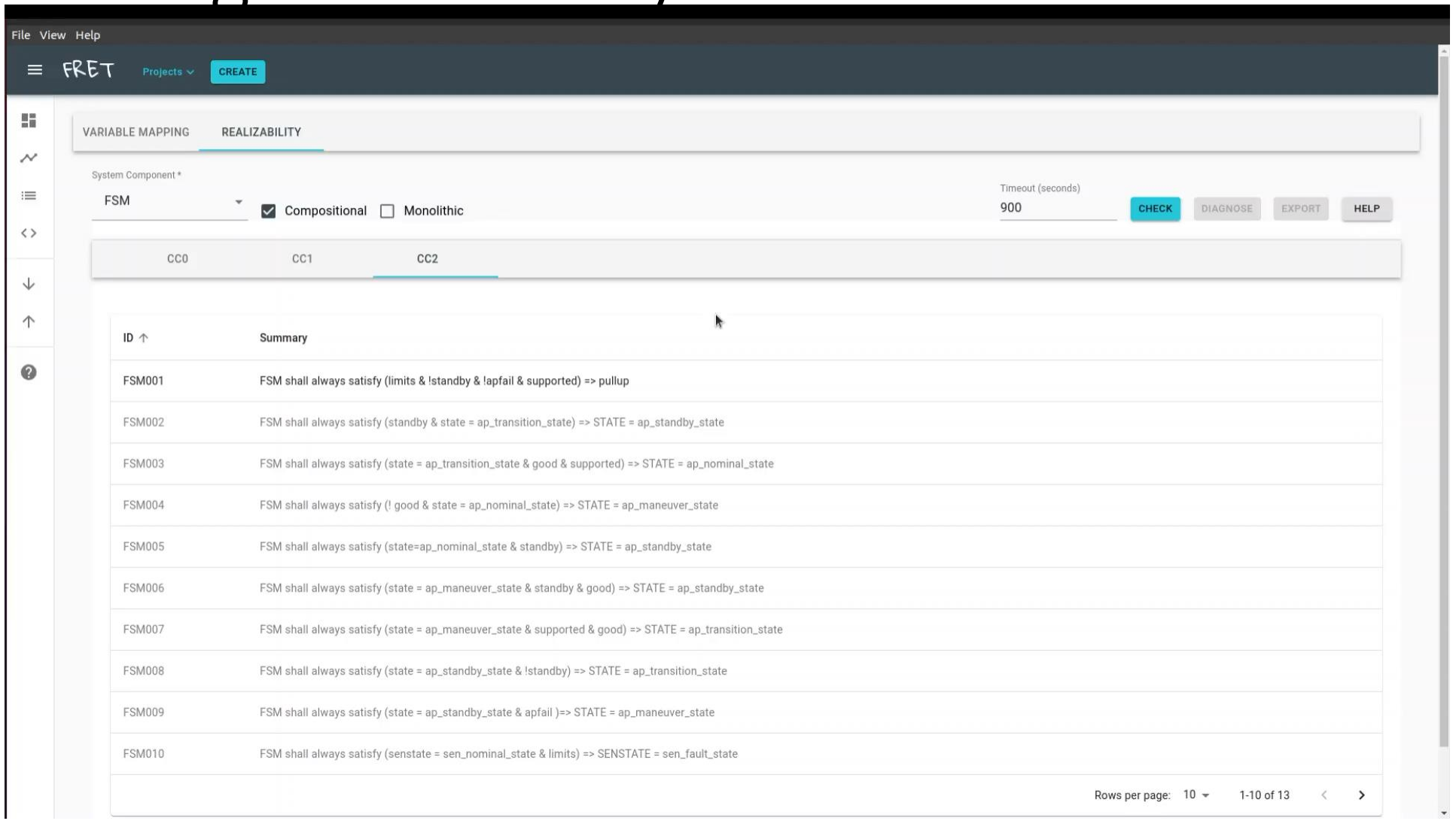
- Unrealizable requirements: Diagnostic analysis
  - Identify minimal sets of unrealizable requirements in specification
  - Counterexamples
  - Simulation of conflicting requirements

- Compositional Realizability Checking

Giannakopoulou, Dimitra, Andreas Katis, Anastasia Mavridou, and Thomas Pressburger. "Compositional realizability checking within FRET." (2021).

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

Checking consistency



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.

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

Connection with analysis tools