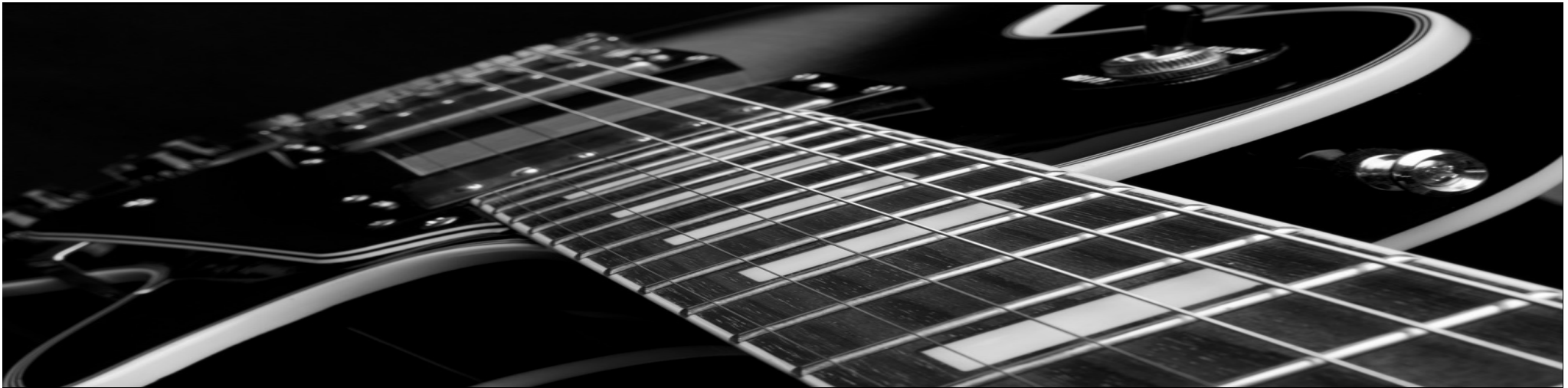


FRET Requirements?



Dimitra Giannakopoulou • NASA Ames

how developers write requirements

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?

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.
- ...

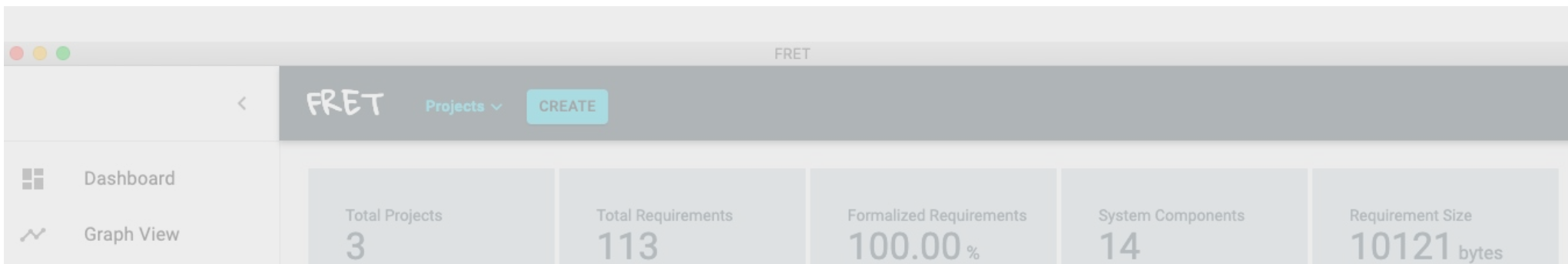
are these requirements consistent? does my model/code satisfy them?

languages formal analysis tools understand

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

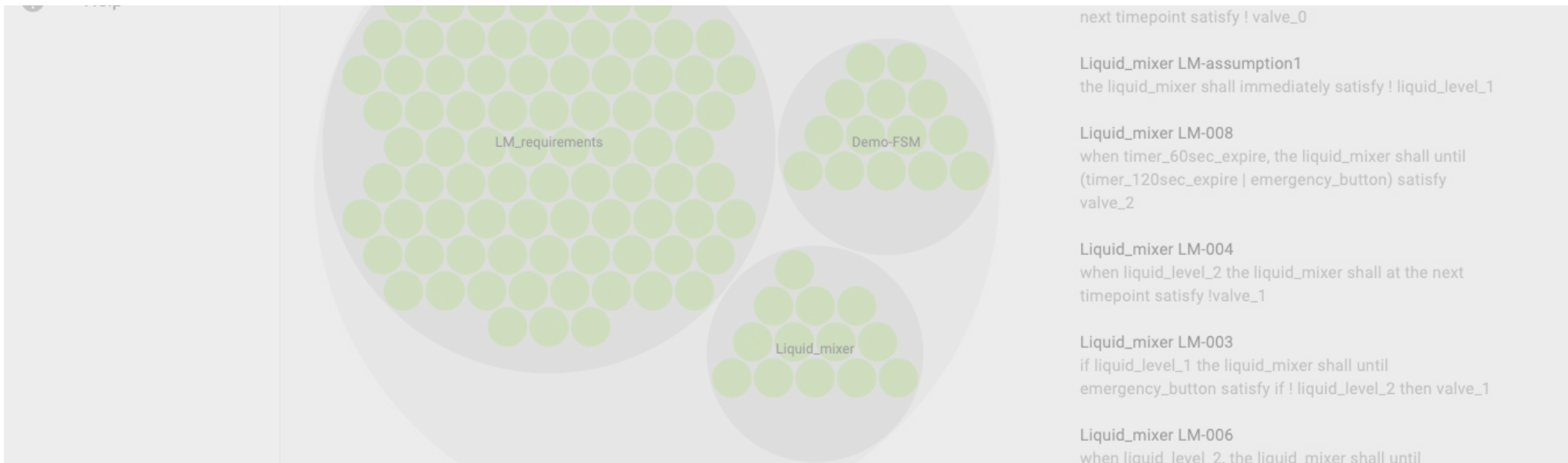
FRET bridges the gap

- **do you speak FRETish?**: an extensible grammar defines a restricted natural language with **unambiguous** semantics
- **explanations** of the formal semantics in various forms: natural language, diagrams, interactive simulation
- compositional (hence maintainable and extensible) generation of **formulas** from requirement fields for analysis tools
- checks **consistency** of requirements and provides feedback
- **exports** verification code
 - for model checking of Simulink models with Cocosim
 - for runtime analysis of C programs with Copilot



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, George Karamanolis, David Kooi, Julian Rhein, Nija Shi

creating requirements

FRET Projects CREATE				
Requirements: All Projects				
Status	ID ↑		Summary	Project
▼	FSM-001	+	FSM shall always satisfy if (limits & !standby & lapfail & supported) then pullup	Demo-FSM
▼	FSM-002	+	FSM shall always satisfy if (standby & state = ap_transition_state) then STATE = ap_standby_state	Demo-FSM
▼	FSM-003	+	FSM shall always satisfy if (state = ap_transition_state & good & supported) then STATE = ap_nominal_state	Demo-FSM
▼	FSM-004	+	FSM shall always satisfy if (! good & state = ap_nominal_state) then STATE = ap_maneuver_state	Demo-FSM
▼	FSM-005	+	FSM shall always satisfy if (state=ap_nominal_state & standby) then STATE = ap_standby_state	Demo-FSM
▼	FSM-006	+	FSM shall always satisfy if (state = ap_maneuver_state & standby & good) then STATE = ap_standby_state	Demo-FSM
▼	FSM-007	+	FSM shall always satisfy if (state = ap_maneuver_state & supported & good) then STATE = ap_transition_state	Demo-FSM
▼	FSM-008	+	FSM shall always satisfy if (state = ap_standby_state & !standby) then STATE = ap_transition_state	Demo-FSM
▼	FSM-009	+	FSM shall always satisfy if (state = ap_standby_state & apfail) then STATE = ap_maneuver_state	Demo-FSM
▼	FSM-010	+	FSM shall always satisfy if (senstate = sen_nominal_state & limits) then SENSTATE = sen_fault_state	Demo-FSM
▼	FSM-011	+	FSM shall always satisfy if (senstate = sen_nominal_state & !request) then SENSTATE = sen_transition_state	Demo-FSM
▼	FSM-012	+	FSM shall always satisfy if (senstate = sen_fault_state & !request & !limits) then SENSTATE = sen_transition_state	Demo-FSM
▼	FSM-013	+	FSM shall always satisfy if (senstate = sen_transition_state & request) then SENSTATE = sen_nominal_state	Demo-FSM
▼	LM-001	+	when start_button liquid_mixer shall at the next timepoint satisfy if ! liquid_level_1 then valve_0	Liquid_mixer
▼	LM-002	+	when liquid_level_1 liquid_mixer shall at the next timepoint satisfy ! valve_0	Liquid_mixer
▼	LM-003	+	if liquid_level_1 the liquid_mixer shall until emergency_button satisfy if ! liquid_level_2 then valve_1	Liquid_mixer
▼	LM-004	+	when liquid_level_2 the liquid_mixer shall at the next timepoint satisfy !valve_1	Liquid_mixer
▼	LM-005	+	when liquid_level_2 the liquid_mixer shall at the next timepoint satisfy timer_60sec_start	Liquid_mixer

is this what I meant?

FRET

Projects

CREATE

Requirements: All Projects

Update Requirement

Requirement ID
AP-001

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.

SCOPE

CONDITIONS

COMPONENT*

SHALL*

TIMING

RESPONSES*

if altitude_hold the Autopilot shall always satisfy maintain_altitude

SEMANTICS

CANCEL UPDATE

ASSISTANT

TEMPLATES

GLOSSARY

ENFORCED: in the interval defined by the entire execution.
TRIGGER: first point in the interval if (*altitude_hold*) is true and any point in the interval where (*altitude_hold*) 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.

Beginning of Time TC

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

Diagram Semantics

Formalizations

Future Time LTL

Past Time LTL

SIMULATE

but it's all FRETish to me!

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.

Create Requirement

Requirement ID	Parent Requirement ID	Project
		Demo-FSM

Rationale and Comments

Rationale

The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.

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.

SCOPE CONDITIONS COMPONENT* SHALL* TIMING RESPONSES*

--

SEMANTICS

CANCEL

CREATE

ASSISTANT

TEMPLATES

GLOSSARY

Template

No template

Choose a predefined template

You currently have not selected predefined template.

importing requirements

Requirement ID
L3-██████████2

Parent Requirement ID

Project
VIPER_All

Rationale and Comments

Rationale

LEVEL: L4
ISSUE KEY: ██████████62
ALLOCATION: Power Subsystem
FUNCTION: Power Management
SUMMARY: Monitoring power switch status - de ██████████
S ██████████
DESCRIPTION: The SW shall r ██████████
██████████
RATIONALE: Ned ██████████, at
way.
TBD DESCRIPTION:
TYPE: Functional
VERIFICATION METHOD: Demonstration
VERIFICATION DESCRIPTION: Demonstrate f ██████████
██████████
VERIFICATION UNIT: Full Rover
KDR:
LINKED ISSUE:

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.

SCOPE

CONDITIONS

COMPONENT*

SHALL*

TIMING

RESPONSES*

?

"The SW shall r ██████████ps."

Ready to speak FRETish?

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

behind the scenes

semantic templates

while cruising, **the Autopilot** shall **always** satisfy if altitude_hold then maintain_altitude

scope timing response

SCOPE in, before, after, notin, onlyIn, onlyBefore, onlyAfter, null

CONDITION null, regular

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

RESPONSE satisfaction

160 semantic templates / template keys!

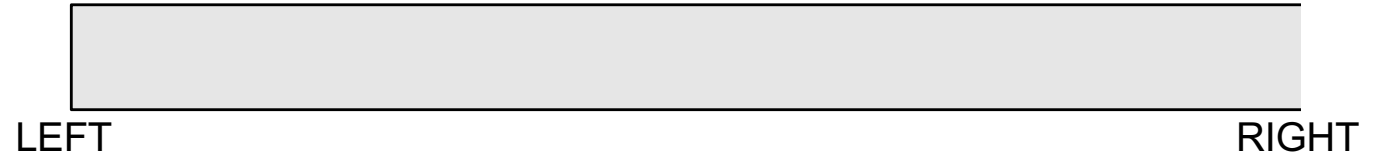
pre-processed cache

```
{,
  "in,null,always,satisfaction": {
    "ft": "(LAST V ($scope_mode$ -> $post_condition$))",
    "ftExpanded": "(LAST V ($scope_mode$ -> $post_condition$))",
    "pt": "(H ($scope_mode$ -> $post_condition$))",
    "ptExpanded": "(H ($scope_mode$ -> $post_condition$))",
    "CoCoSpecCode": "(H($scope_mode$ => $post_condition$))",
    "R2U2Code": "Under construction.",
    "ftInfAU": "(G ($scope_mode$ -> $post_condition$))",
    "ftInfAUEExpanded": "(G ($scope_mode$ -> $post_condition$))",
    "ftInfBtw": "((G (!! Fin_$scope_mode$) | (X ((F Lin_$scope_mode$) -> (Lin_$scope_mode$ V $post_condition$)))) & ($scope_mode$ -> ((F Lin_$s
    "ftInfBtwExpanded": "((G (!! (!! $scope_mode$) & (! LAST)) & (X $scope_mode$)) | (X ((F (($scope_mode$ & (! LAST)) & (X (! $scope_mode$)))
    "ftFinBtw": "((LAST V (!! (Fin_$scope_mode$ & (! LAST))) | (X (!! LAST) U Lin_$scope_mode$) -> (Lin_$scope_mode$ V $post_condition$)))) &
    "ftFinBtwExpanded": "((LAST V (!! (!! (!! $scope_mode$) & (! LAST)) & (X $scope_mode$)) & (! LAST)) | (X (!! LAST) U (($scope_mode$ & (! LAS
    "ptFinBtw": "(H ((Lin_$scope_mode$ & (! FTP)) -> (Y ($post_condition$ S ($post_condition$ & Fin_$scope_mode$)))))",
    "ptFinBtwExpanded": "(H (!! (!! $scope_mode$) & (Y $scope_mode$)) & (Y TRUE)) -> (Y ($post_condition$ S ($post_condition$ & ($scope_mode$ & (
    "CoCoSpecCodeFinBtw": "(H((( not $scope_mode$) and (pre ($scope_mode$))) and ( not FTP)) => (pre (SI( ($scope_mode$ and (FTP or (pre ( not
    "description": "ENFORCED: in every interval where $scope_mode$ holds.\nTRIGGER: first point in the interval.\nREQUIRES: for every trigger, R
    "diagram": "_media/user-interface/examples/svgDiagrams/in_null_always_satisfaction.svg"
  },
```

- semantic templates have RTGIL semantics
 - **ALL** aspects of our approach are compositional therefore extensible
-
- Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Johann Schumann: **Generation of Formal Requirements from Structured Natural Language**. REFSQ 2020
 - Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Johann Schumann: **Automated Formalization of Structured Natural Language Requirements**. IST Journal, 2021

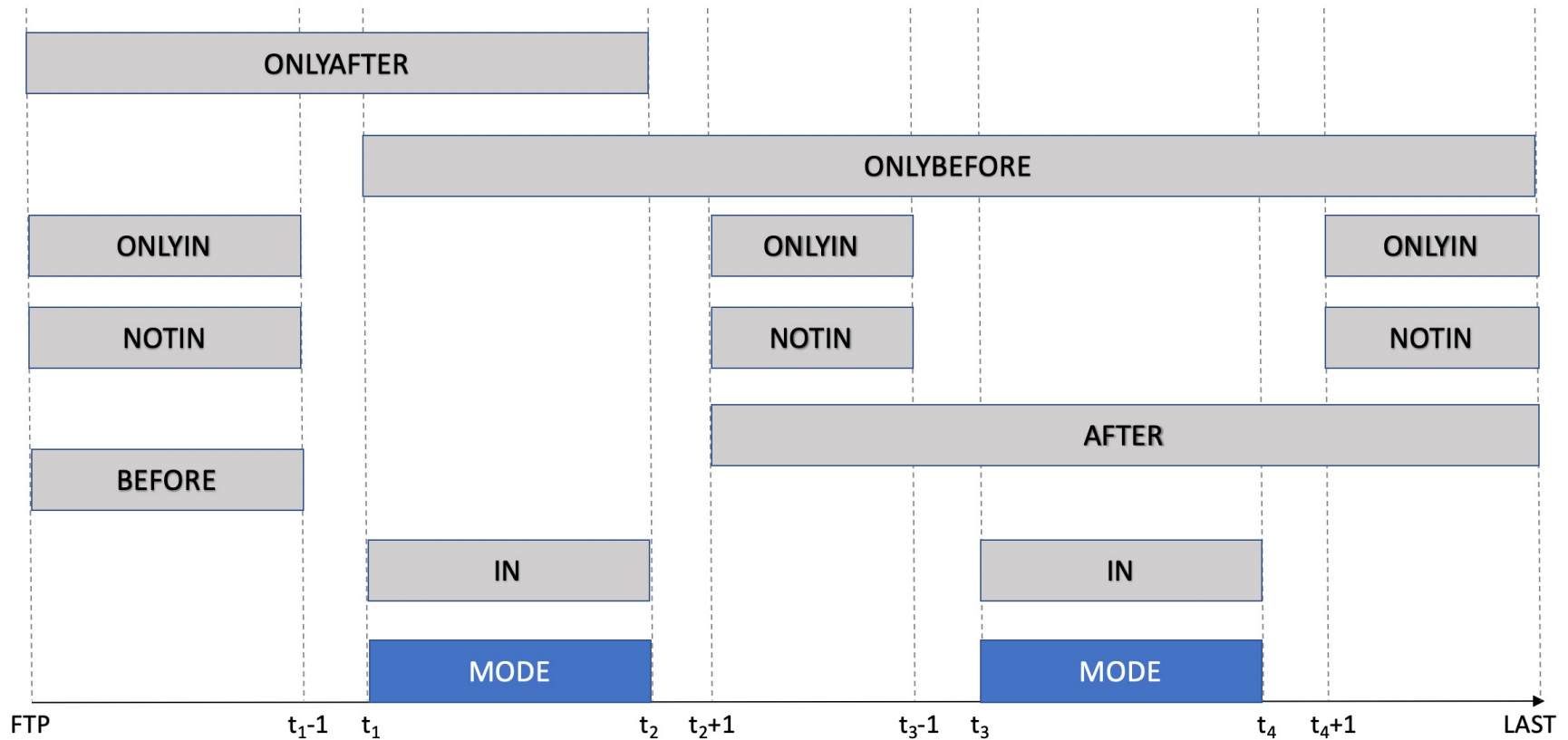
past-time formulas

[LEFT, RIGHT):



historically (RIGHT implies previous

(FORMULA since inclusive required LEFT))



constructing formulas from fields

while cruising, 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: FORMULA $\rightarrow$ RES

historically (RIGHT implies previous

(FORMULA 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)

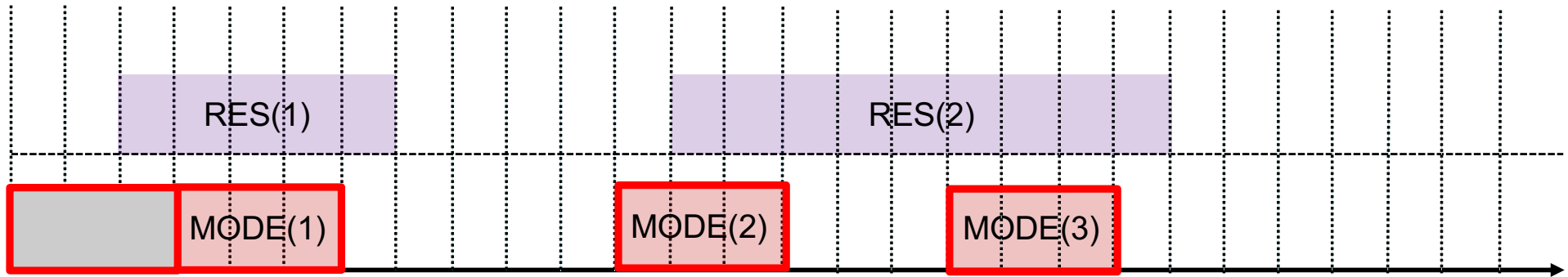
translate to SMV

(H (MODE $\rightarrow$ RES))

instantiate

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

oracles



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

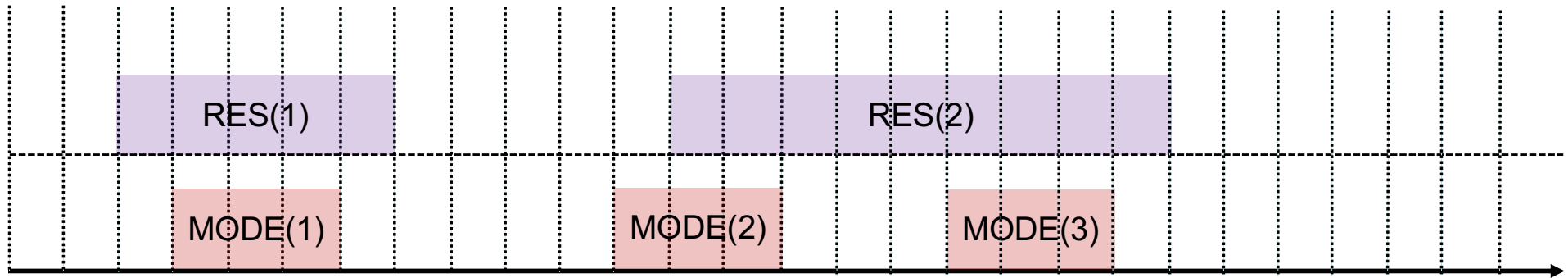
for each `scopeInterval` in `scopeIntervals`

there exists a `responseInterval` in `responseIntervals` such that:

`responseInterval` contains `scopeInterval`

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

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:
targeted scenarios; random scenarios

we also use SMV to check equivalence between:

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

constructing proofs in PVS with NASA Langley

< > analysis portal

mapping requirement variables

FRET

Projects

CREATE

Requirement Variables to Model Mapping: Demo-FSM

Export Language *

Autopilot

EXPORT

FSM

EXPORT

Corresponding Model Component

FSM_model

IMPORT

FRET Variable Name ↑	Model Variable Name	Variable Type	Data Type	Description
AP_MANEUVER_STATE		Internal	double	value 2
AP_NOMINAL_STATE		Internal	double	value 1
AP_STANDBY_STATE		Internal	double	value 3
AP_TRANSITION_STATE		Internal	double	value 0
APFAIL				
GOOD	good	Input	boolean	
LIMITS	limits	Input	boolean	
PULLUP	pullup	Output	boolean	
REQUEST	request	Input	boolean	
SEN_FAULT_STATE		Internal	double	value 2

Rows per page: 10 1-10 of 18

FRET
Projects ▾ CREATE

Requirement Variables to Model Mapping: Demo-FSM

Export Language * ▾

Autopilot
EXPORT
▾

FSM
EXPORT
▴

Corresponding Model Component				
FSM_model ▾		IMPORT		
FRET Variable Name ↑	Model Variable Name	Variable Type	Data Type	Description
AP_MANEUVER_STATE		Internal	double	value 2
AP_NOMINAL_STATE		Internal	double	value 1
AP_STANDBY_STATE		Internal	double	value 3
AP_TRANSITION_STATE		Internal	double	value 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

Rows per page: 10 ▾ 1-10 of 18 < >

exporting verification code

```
FSMSpec.lus
1  --Historically node H(X:bool) returns (Y:bool); let Y = X -> (X and (pre Y));
2  --tel
3
4  --Y since inclusive X node SI(X,Y: bool) returns (Z:bool); let Z = Y and (X or
5  --(false -> pre Z)); tel
6
7  --Y since X node S(X,Y: bool) returns (Z:bool); let Z = X or (Y and (false ->
8  --pre Z)); tel
9
10 --Once node O(X:bool) returns (Y:bool); let Y = X or (false -> pre Y); tel
11
12 --Timed Once: less than or equal to N node OTlore(const N: int; X: bool;)
13 --returns (Y: bool); var C:int; let C = if X then 0 else (-1 -> pre C + (if pre
14 --C < 0 then 0 else 1));
15
16     Y = 0 <= C and C <= N; tel
17
18 --Timed Once: general case node OT(const L: int; const R: int; X: bool;) returns
19 --(Y: bool); var D:bool; let D=delay(X, R); Y=OTlore(L-R, D); tel
20
21 -- Timed Historically: general case node HT(const L: int; const R: int; X:
22 -- bool;) returns (Y: bool); let Y = not OT(L, R, not X); tel
23
24 -- Timed Since: general case node ST(const L: int; const R: int; X: bool; Y:
25 -- bool;) returns (Z: bool); let Z = S(X, Y) and OT(L, R, X); tel
26
27 -- Timed Since: general case node SIT(const L: int; const R: int; X: bool; Y:
28 -- bool;) returns (Z: bool); let Z = SI(X, Y) and OT(L, R, X); tel
29
```

are my requirements consistent?

$$\text{Realizable}_{AG} \stackrel{\text{def}}{=} \exists s. G_I(s) \wedge \text{Viable}_{AG}(s)$$

there exists an initial state that is viable,

$$\text{Viable}_{AG}(s) \stackrel{\text{def}}{=} \forall a. (A(s, a) \Rightarrow \exists s'. G_T(s, a, s') \wedge \text{Viable}_{AG}(s'))$$

meaning that for **each** input that satisfies the **assumptions** of the contract, it is **possible to transition** to a viable state, **without violating** the contract

realizability in FRET

- uses variable mapping from analysis portal
- automatically decomposes realizability of a set of requirements into an equivalent set of smaller realizability problems
- computes “minimal conflicts”
- visualizes results

CREATE

VARIABLE MAPPING	REALIZABILITY
------------------	---------------

System Component *

FSM ☒ Compositional☐ Monolithic

Timeout (seconds)

900

CHECK

DIAGNOSE

EXPORT

HELP

CC0 CC1 CC2 

ID ↑	Summary
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
FSM001	FSM shall always satisfy (limits & !standby & !apfail & supported) => pullup
FSM010	FSM shall always satisfy (senstate = sen_nominal_state & limits) => SENSTATE = sen_fault_state

understand conflicts

VARIABLE MAPPING

REALIZABILITY

System Component *

FSM

☒ Compositional ☐ Monolithic

Timeout (seconds)
900

CHECK

DIAGNOSE

EXPORT

HELP

CC0

CC1

CC2

Diagram illustrating the relationship between FSM components (FSM002 to FSM009) and conflicts (Conflict 1 to Conflict 4). The components are arranged in a semi-circle, and the conflicts are arranged in a semi-circle. A green shaded region highlights the area around Conflict 1 and Conflict 2.

Counterexample For
Conflict 1

Variable name	Variable type	Step 0
good	bool	true
standby	bool	true
state	real	0
supported	bool	true
STATE	real	2

ID ↑	Summary
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
CCM001	CCM shall always satisfy (limits & standby & failed & connected) => null

FRET Requirements!



solid foundations, extensible, sleek
focus on usability
FRET v2.0 coming soon

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

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

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

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

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

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