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

```typescript
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
<table>
<thead>
<tr>
<th>Status</th>
<th>ID</th>
<th>Summary</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FSM-001</td>
<td>FSM shall always satisfy if (limits &amp; Isstandby &amp; isfail &amp; supported) then pulup</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-002</td>
<td>FSM shall always satisfy if (standby &amp; state = ap_transition_state) then STATE = ap_standby_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-003</td>
<td>FSM shall always satisfy if (state = ap_transition_state &amp; good &amp; supported) then STATE = ap_nominal_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-004</td>
<td>FSM shall always satisfy if (good &amp; state = ap_nominal_state) then STATE = ap_maneuver_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-005</td>
<td>FSM shall always satisfy if (state=ap_nominal_state &amp; standby) then STATE = ap_standby_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-006</td>
<td>FSM shall always satisfy if (state = ap_maneuver_state &amp; standby &amp; good) then STATE = ap_standby_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-007</td>
<td>FSM shall always satisfy if (state = ap_maneuver_state &amp; supported &amp; good) then STATE = ap_transition_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-008</td>
<td>FSM shall always satisfy if (state = ap_standby_state &amp; standby) then STATE = ap_transition_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-009</td>
<td>FSM shall always satisfy if (state = ap_standby_state &amp; ap/ail) then STATE = ap_maneuver_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-010</td>
<td>FSM shall always satisfy if (senstate = sen_nominal_state &amp; limits) then SENSTATE = sen_fault_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-011</td>
<td>FSM shall always satisfy if (senstate = sen_nominal_state &amp; request) then SENSTATE = sen_transition_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-012</td>
<td>FSM shall always satisfy if (senstate = sen_fault_state &amp; request &amp; limits) then SENSTATE = sen_transition_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>FSM-013</td>
<td>FSM shall always satisfy if (senstate = sen_transition_state &amp; request) then SENSTATE = sen_transition_state</td>
<td>Demo-FSM</td>
</tr>
<tr>
<td></td>
<td>LM-001</td>
<td>when start_button liquid_mixer shall at the next timepoint satisfy if I liquid_level_1 then valve_0</td>
<td>Liquid_mixer</td>
</tr>
<tr>
<td></td>
<td>LM-002</td>
<td>when liquid_level_1 liquid_mixer shall at the next timepoint satisfy I valve_0</td>
<td>Liquid_mixer</td>
</tr>
<tr>
<td></td>
<td>LM-003</td>
<td>if liquid_level_1 the liquid_mixer shall until emergency_button satisfy if I liquid_level_2 then valve_1</td>
<td>Liquid_mixer</td>
</tr>
<tr>
<td></td>
<td>LM-004</td>
<td>when liquid_level_2 the liquid_mixer shall at the next timepoint satisfy I valve_1</td>
<td>Liquid_mixer</td>
</tr>
<tr>
<td></td>
<td>LM-005</td>
<td>when liquid_level_2 the liquid_mixer shall at the next timepoint satisfy timer_60sec_start</td>
<td>Liquid_mixer</td>
</tr>
</tbody>
</table>
is this what I meant?

**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.

```
if altitude_hold the Autopilot shall always satisfy maintain_altitude
```

**Semantics**

**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**
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

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Parent Requirement ID</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Demo-FSM</td>
</tr>
</tbody>
</table>

**Rationale and Comments**

Rationale

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

**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.

---

No template

Choose a predefined template.

You currently have not selected predefined template.
**Rationale and Comments**

**Rationale**

LEVEL: L4  
ISSUE KEY:  
ALLOCATION: Power Subsystem  
FUNCTION: Power Management  
SUMMARY: Monitoring power switch status - description  
DESCRIPTION: The SW shall  
RATIONALE: New functionality  
TBD DESCRIPTION:  
TYPE: Functional  
VERIFICATION METHOD: Demonstration  
VERIFICATION DESCRIPTION: Demonstrate  
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.

"The SW shall ..."
behind the scenes
while cruising, the Autopilot shall always satisfy if altitude_hold then maintain_altitude

160 semantic templates / template keys!
- 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
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:** \([\text{LEFT}, \text{RIGHT}) \rightarrow [\text{FiM}, \text{LiM})\]

- \(\text{FiM} = \text{MODE and (FTP or previous (not MODE))}\)
- \(\text{LiM} = \text{not MODE and previous MODE}\)

**timing always:** \(\text{FORMULA} \rightarrow \text{RES}\)

historically (RIGHT implies previous \((\text{FORMULA} \text{ since inclusive required LEFT})\))

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
\((H (\text{MODE} \rightarrow \text{RES}))\)

instantiate
\((H (\text{cruising} \rightarrow (\text{altitude\_hold} \rightarrow \text{maintain\_altitude}))))\)
**Oracle for**  
scope: in, condition: null, timing: always, response: satisfaction

\[
\text{responseIntervals} = \{\text{RES}(1), \text{RES}(2)\}
\]

step 1: determine scope intervals  
result: \(\text{scopIntervals} = \{\text{MODE}(1), \text{MODE}(2), \text{MODE}(3)\}\)

step 2: apply timing always  
for each \(\text{scopInterval}\) in \(\text{scopIntervals}\)  
there exists a \(\text{responseInterval}\) in \(\text{responseIntervals}\) such that:  
\(\text{responseInterval}\) contains \(\text{scopInterval}\)

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

```plaintext
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

### Requirement Variables to Model Mapping: Demo-FSM

Export Language *

**Autopilot**

**FSM**

<table>
<thead>
<tr>
<th>Corresponding Model Component</th>
<th>FRET Variable Name</th>
<th>Model Variable Name</th>
<th>Variable Type</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSM_model</td>
<td>AP_MANEUVER_STATE</td>
<td>Internal</td>
<td>double</td>
<td>value 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AP_NOMINAL_STATE</td>
<td>Internal</td>
<td>double</td>
<td>value 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AP_STANDBY_STATE</td>
<td>Internal</td>
<td>double</td>
<td>value 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AP_TRANSITION_STATE</td>
<td>Internal</td>
<td>double</td>
<td>value 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>APFAIL</td>
<td>good</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GOOD</td>
<td>good</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIMITS</td>
<td>limits</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PULLUP</td>
<td>pullup</td>
<td>Output</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REQUEST</td>
<td>request</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEN_FAULT_STATE</td>
<td>Internal</td>
<td>double</td>
<td>value 2</td>
<td></td>
</tr>
</tbody>
</table>

Rows per page: 10

1-10 of 18
exporting verification code

<table>
<thead>
<tr>
<th>FRET Variable Name</th>
<th>Model Variable Name</th>
<th>Variable Type</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP_MANEUVER_STATE</td>
<td>apfail</td>
<td>Input</td>
<td>boolean</td>
<td>value 0</td>
</tr>
<tr>
<td>AP_NOMINAL_STATE</td>
<td>good</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>AP_STANDBY_STATE</td>
<td>limits</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>AP_TRANSITION_STATE</td>
<td>pullup</td>
<td>Output</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>SEN_FAULT_STATE</td>
<td>request</td>
<td>Input</td>
<td>boolean</td>
<td></td>
</tr>
</tbody>
</table>
exporting verification code

are my requirements consistent?

\[
\text{Realizable}_{AG} \overset{\text{def}}{=} \exists s. G_{I}(s) \land \text{Viable}_{AG}(s)
\]

there exists an initial state that is viable,

\[
\text{Viable}_{AG}(s) \overset{\text{def}}{=} \forall a. (A(s,a) \Rightarrow \exists s'. G_{T}(s,a,s') \land \text{Viable}_{AG}(s'))
\]

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
<table>
<thead>
<tr>
<th>ID</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSM002</td>
<td>FSM shall always satisfy (standby &amp; state = ap_transition_state) $\Rightarrow$ STATE = ap_standby_state</td>
</tr>
<tr>
<td>FSM003</td>
<td>FSM shall always satisfy (state = ap_transition_state &amp; good &amp; supported) $\Rightarrow$ STATE = ap_nominal_state</td>
</tr>
<tr>
<td>FSM004</td>
<td>FSM shall always satisfy (! good &amp; state = ap_nominal_state) $\Rightarrow$ STATE = ap_maneuver_state</td>
</tr>
<tr>
<td>FSM005</td>
<td>FSM shall always satisfy (state=ap_nominal_state &amp; standby) $\Rightarrow$ STATE = ap_standby_state</td>
</tr>
<tr>
<td>FSM006</td>
<td>FSM shall always satisfy (state = ap_maneuver_state &amp; standby &amp; good) $\Rightarrow$ STATE = ap_standby_state</td>
</tr>
<tr>
<td>FSM007</td>
<td>FSM shall always satisfy (state = ap_maneuver_state &amp; supported &amp; good) $\Rightarrow$ STATE = ap_transition_state</td>
</tr>
<tr>
<td>FSM008</td>
<td>FSM shall always satisfy (state = ap_standby_state &amp; 'standby) $\Rightarrow$ STATE = ap_transition_state</td>
</tr>
<tr>
<td>FSM009</td>
<td>FSM shall always satisfy (state = ap_standby_state &amp; apfail ) $\Rightarrow$ STATE = ap_maneuver_state</td>
</tr>
<tr>
<td>FSM010</td>
<td>FSM shall always satisfy (limits &amp; /standby &amp; iapfail &amp; supported) $\Rightarrow$ pulsep</td>
</tr>
<tr>
<td>FSM011</td>
<td>FSM shall always satisfy (semstate = sen_nominal_state &amp; limits) $\Rightarrow$ SENSTATE = sen_fault_state</td>
</tr>
</tbody>
</table>
understand conflicts
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: 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