

Requirements elicitation, analysis and verification using FRET and CoCoSim

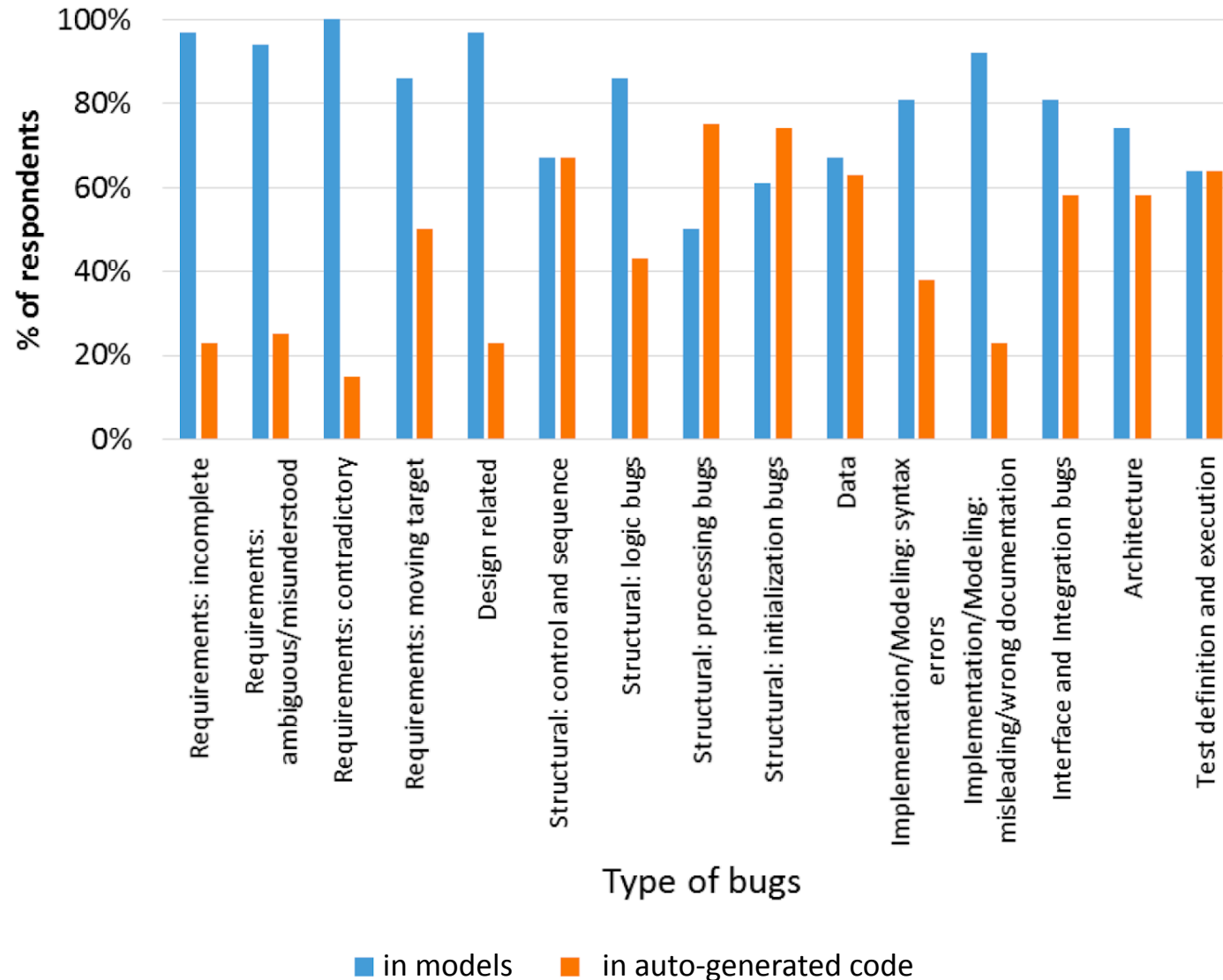
Andreas Katis^{1,2}

FEANICES 2022, December 6, Toulouse, France

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

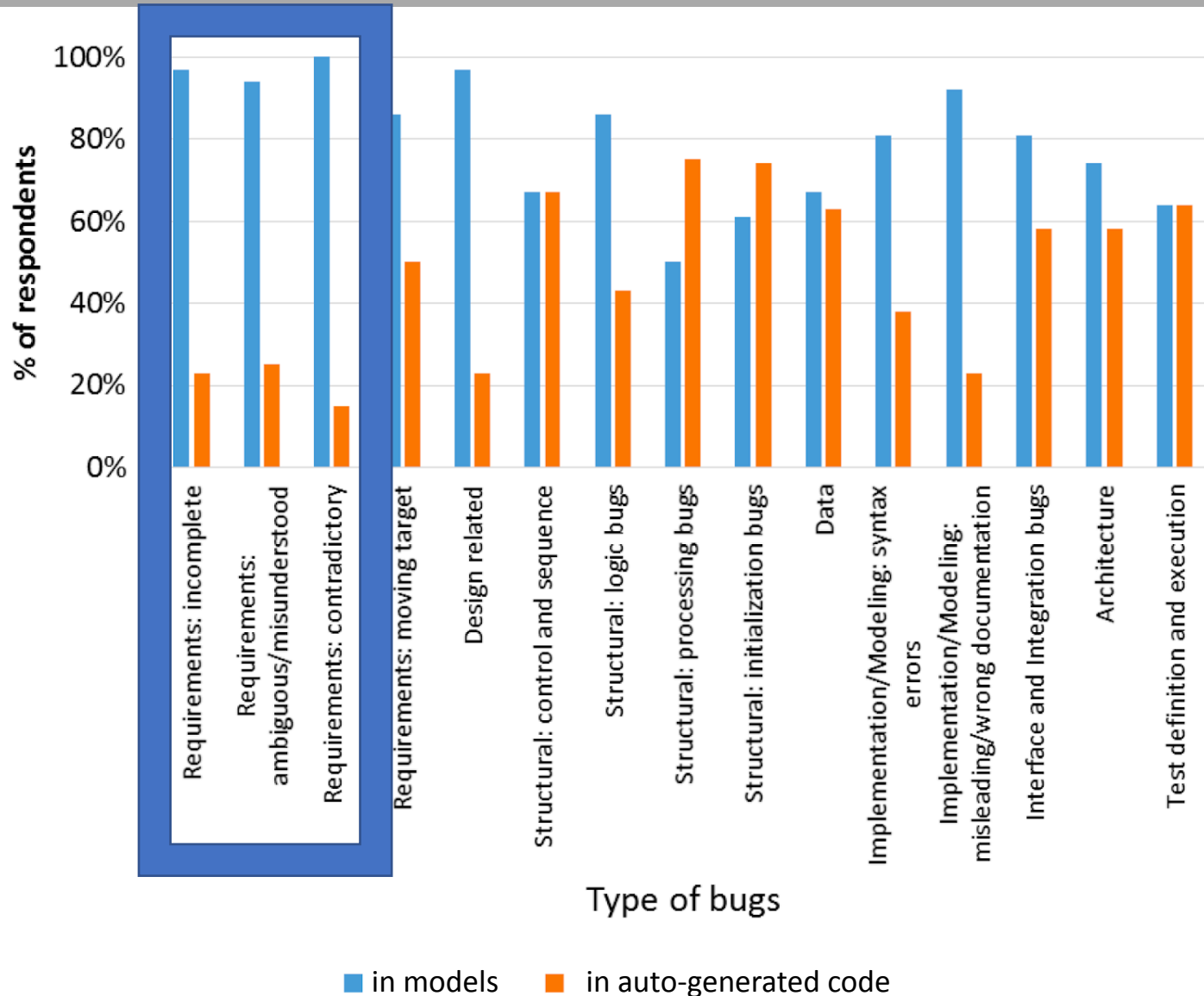
² NASA Ames Research Center, CA, USA

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 a fail).

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));
```



Total Projects

19

Total Requirements

356

Formalized Requirements

80.34%

System Components

52

Requirement Size

29378 bytes

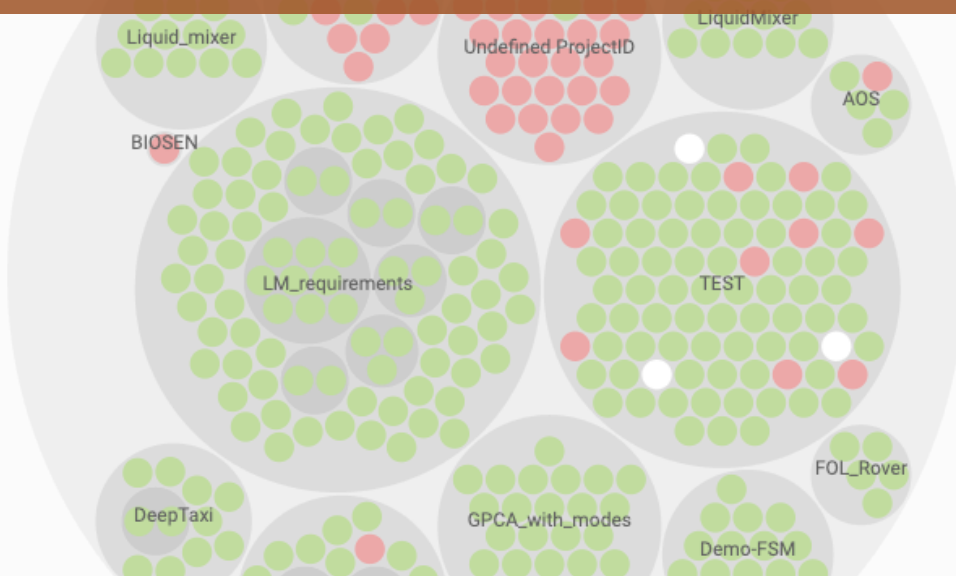


Hierarchical Cluster

Recent Activity

Welcome to FRET

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



AOS UAV-1

when not in m mode when p the sw shall always satisfy voltage > 9

LiquidMixer LM-003

when liquid_level_1 the liquid_mixer shall until emergency_button satisfy if ! liquid_level_2 then valve_1

TEST TEST-TCND-N

when occurred(7,persisted(2,fault)) the sw shall immediately satisfy q

TEST

when not in m mode when p the sw shall always satisfy r

LM_AUTOPILOT AP-003b

In rollhold mode RollHoldReference shall immediately satisfy $abs(rollangle) < 6 \Rightarrow rollholdreference = 0$

TEST TEST-BNDD-RSPNSE

if P the sw shall within 5 ticks satisfy R

TEST-ONLY-IN

only in m, when p, shall the software satisfy pc

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

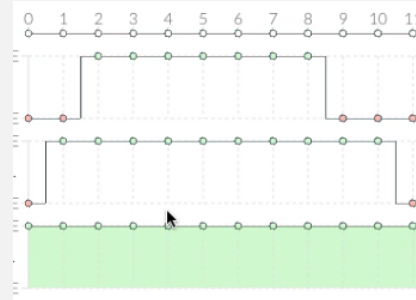
captures + assists



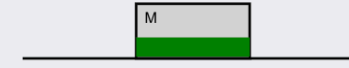
FRETish

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

M Mode of operation (mentioned in Scope)

Intervals

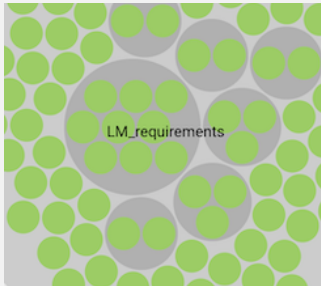
Scope interval

Response must hold at least somewhere in this interval

Negation of response must hold at least somewhere in this interval

Response must hold everywhere in this interval

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

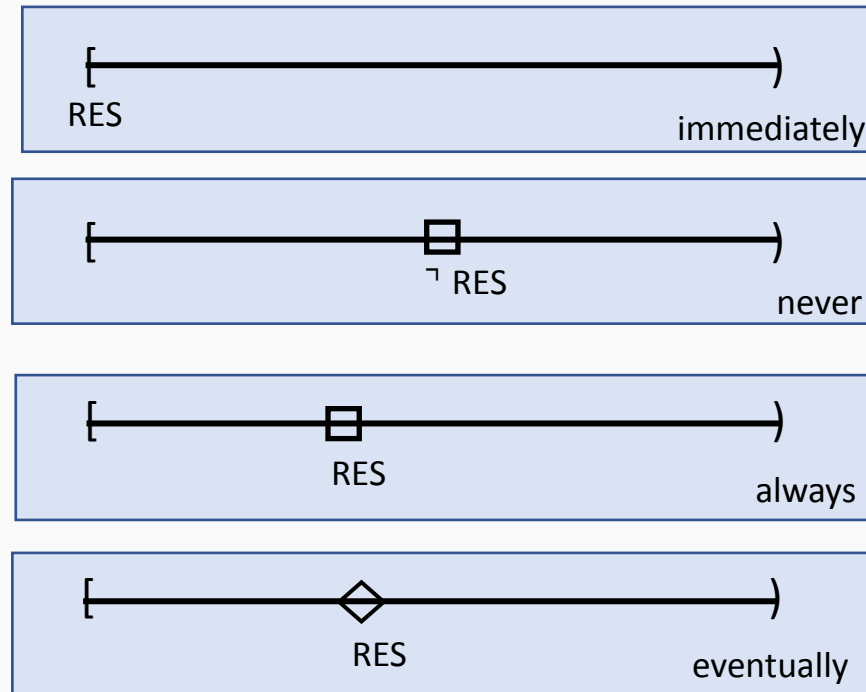
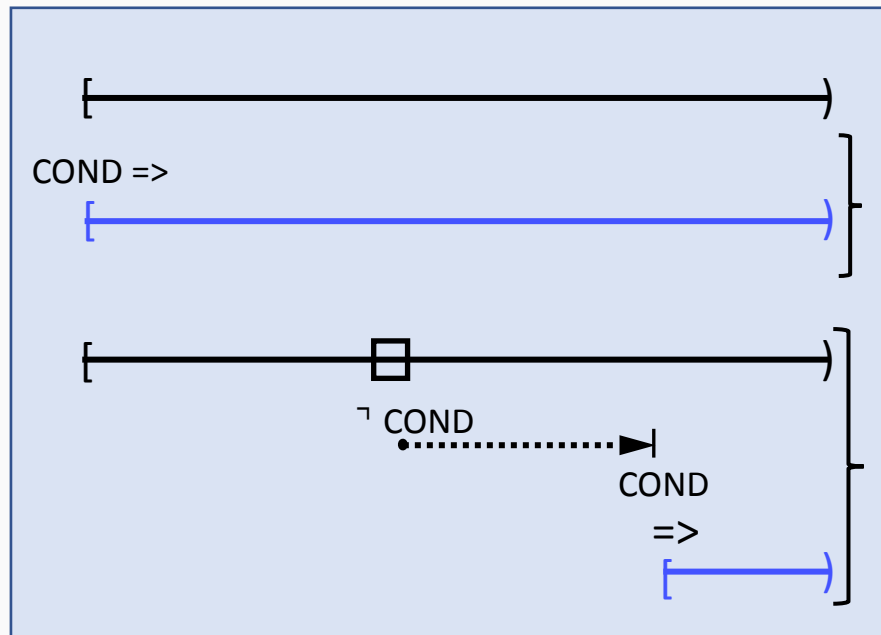
	Step 3	Step 4	Step 5
true	true	true	true
true	true	true	true
true	true	true	true

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.



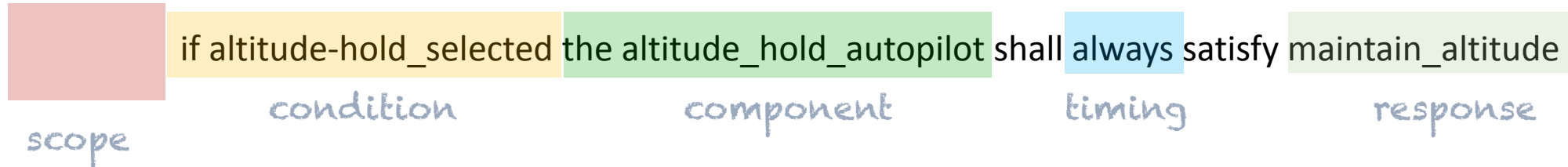
Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Johann Schumann:
“Automated Formalization of Structured Natural Language”, *Information and Software Technology*, 2021

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 in, before, after, notin, onlyIn, onlyBefore, onlyAfter; when omitted, global

CONDITION null, regular

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

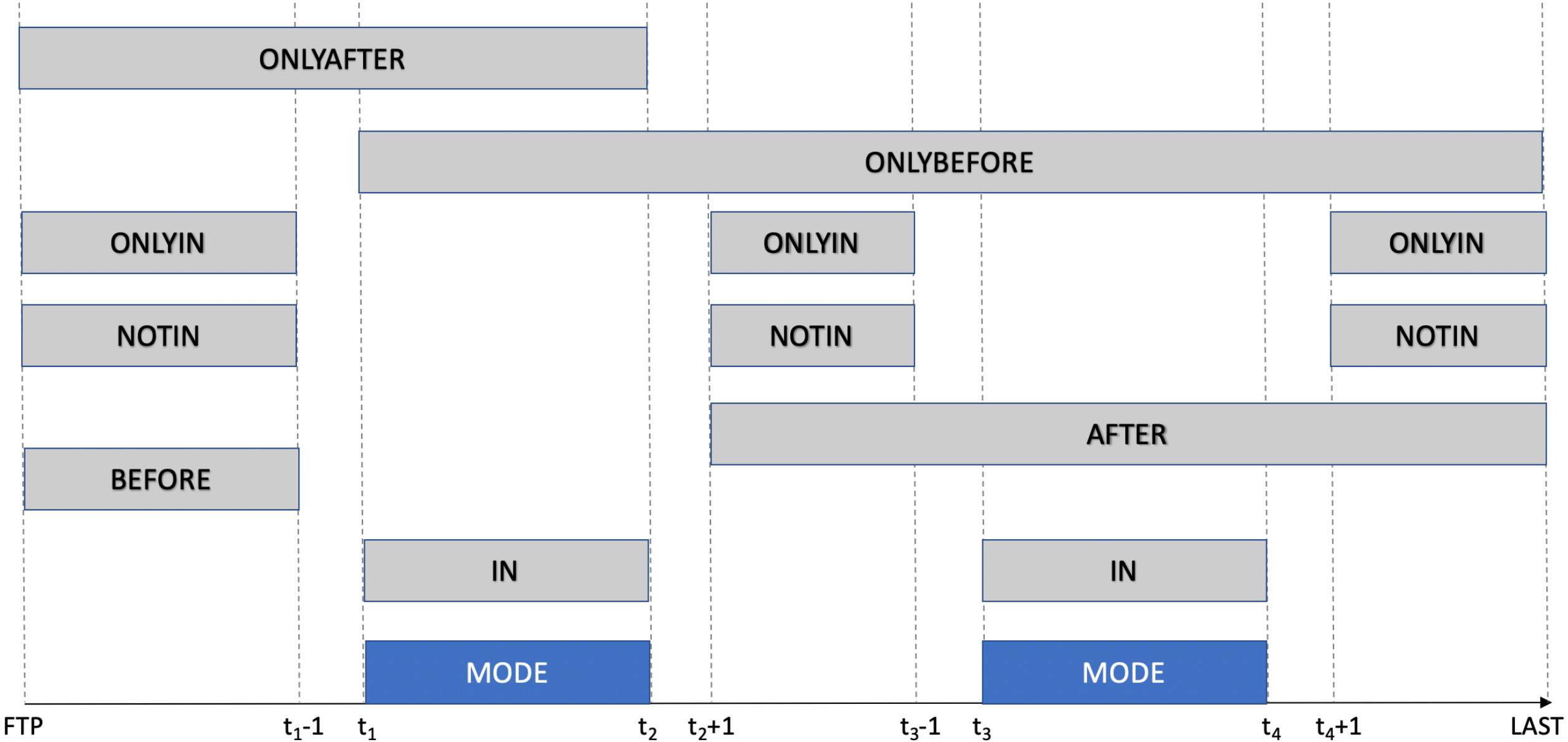
RESPONSE satisfaction

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`

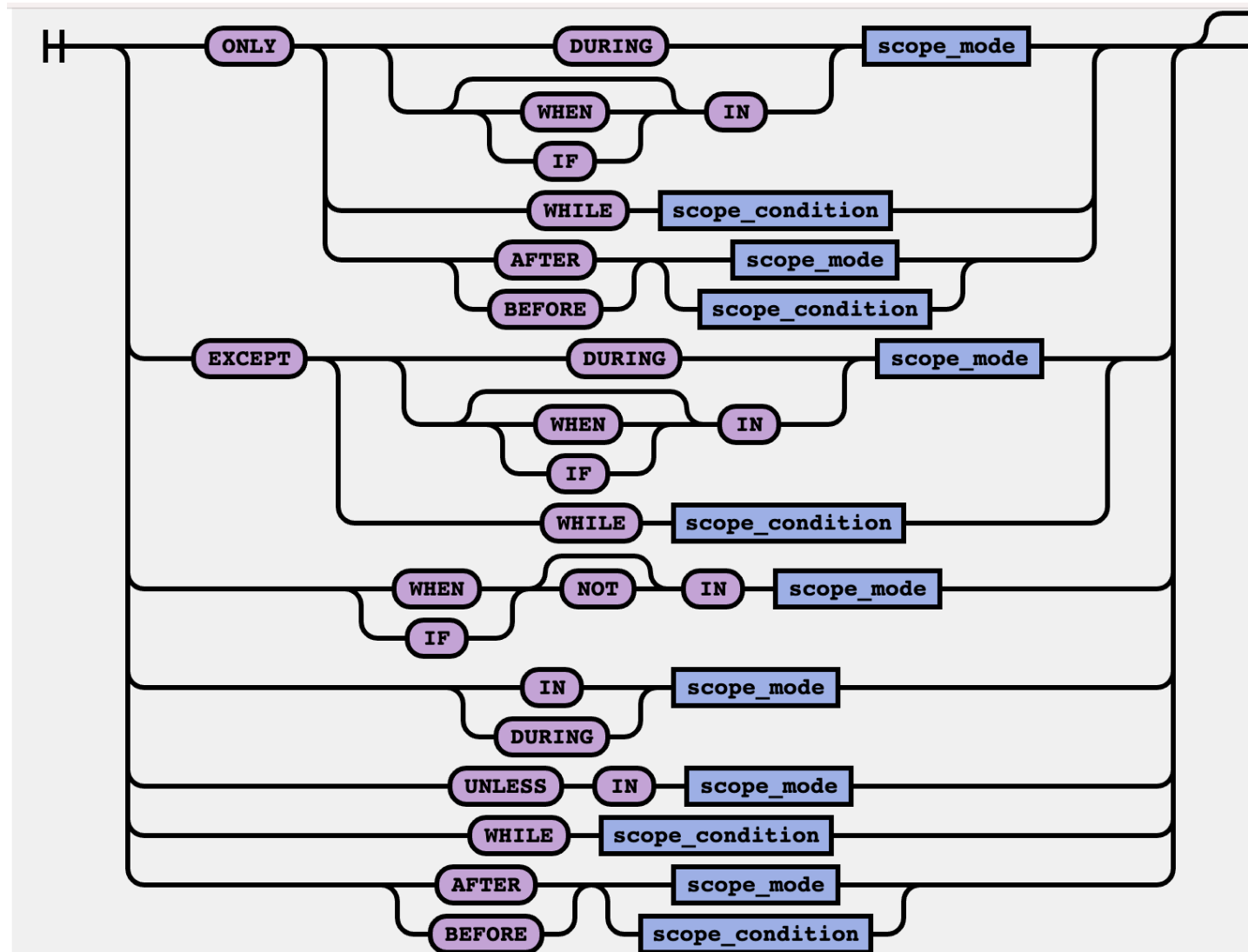
Scope Intervals



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

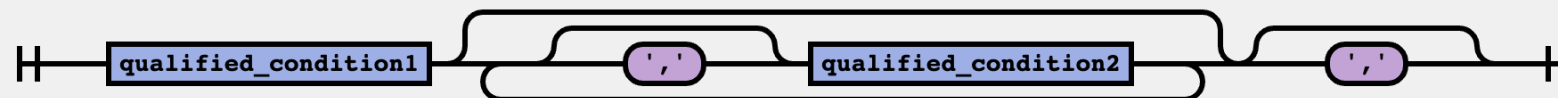


Conditions

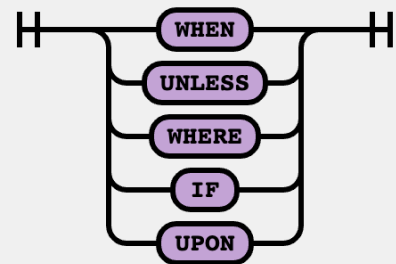
- *upon, if, when, where; unless*
- Boolean expression
- Trigger: **upon** the Boolean expression becoming true from being false in the scope, or being true at the beginning of the scope.

Condition grammar

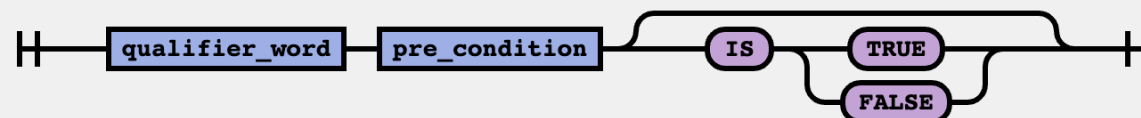
regular_condition



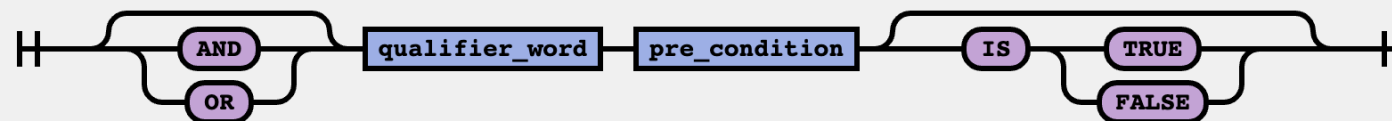
qualifier_word



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
- Formalizes requirements in a **compositional** (hence maintainable and extensible) manner: past, future linear temporal logic, Lustre
- Assists in writing requirements through requirement **templates**
- 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

CREATE

Create Requirement

Project: Demo-FSM

Requirement ID: _____ Parent Requirement ID: _____

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*

SEMANTICS

CANCEL **CREATE**

ASSISTANT TEMPLATES GLOSSARY

Ready to speak FRETish?

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

LM_AUTOPILOT REG_YAW_ACC_REQ
when not and for de2: 50. Repeated shall with

Update Requirement

Requirement ID	Parent Requirement ID	Project
Test-ALTHOLD		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.

SCOPED CONDITIONS COMPONENT* SHALL* TIMING RESPONSES*

if altitude_hold_selected the altitude_hold_autopilot shall always satisfy maintain_altitude



but this is not what I mean...

SEMANTICS

Semantics

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



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

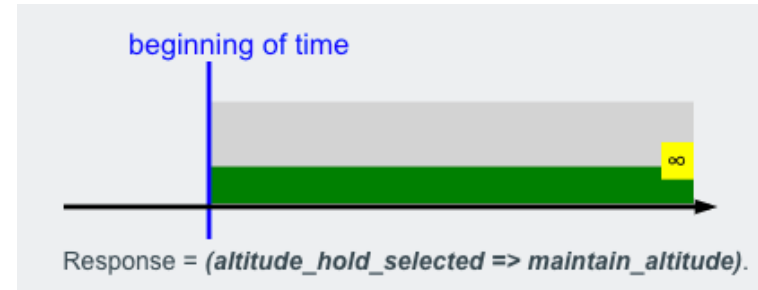
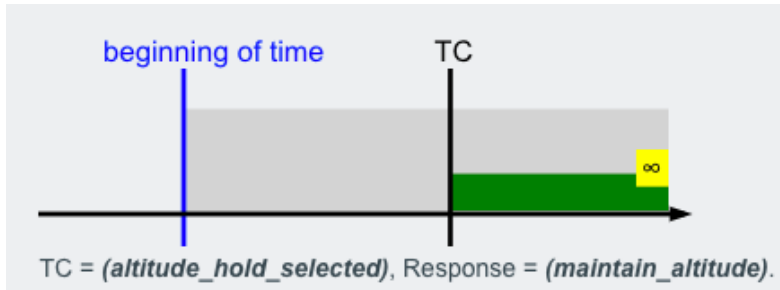
```
(H ((H (! (altitude_hold_selected)) | (maintain_altitude))))
```

Target: *altitude_hold_autopilot* component.

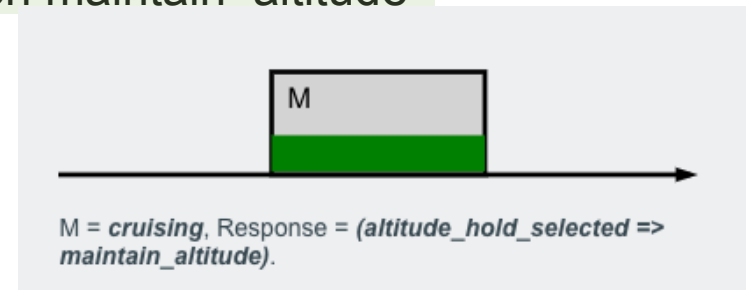
getting to the right requirement

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
- Formalizes requirements in a compositional (hence maintainable and extensible) manner
- Assists in writing requirements through requirement templates
- 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

The screenshot displays the FRET software interface. The main window is titled 'Create Requirement' and shows a form for creating a new requirement. The form includes fields for 'Requirement ID' (FSM 002) and 'Project' (LM_requirements). Below these fields is a 'Rationale and Comments' section with a 'Rationale' field containing the text 'The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).' and a 'Comments' field. Below the form is a 'Requirement Description' section with a text box containing the sentence structure 'component shall always satisfy if (input_state & condition) then output_state'. A 'SEMANTICS' section below the text box shows a grid of bubbles for 'SCOPE', 'CONDITIONS', 'COMPONENT*', 'SHALL*', 'TIMING', and 'RESPONSES*'. The sidebar on the right is titled 'TEMPLATES' and shows a 'Change State' template. The template description states: 'This template describes how the state of a finite-state-machine component changes. It describes the input state and some conditions based on which the change must occur. The corresponding output state must reflect the required change. The input and output states have a pre - post- relationship'. Examples of the template are shown in a code block: 'FSM_Autopilot shall always satisfy if (state = ap_standby_state & ! standby & ! apfail) then STATE = ap_transition_state'.

Create Requirement

Requirement ID: FSM 002 | Project: LM_requirements

Rationale and Comments

Rationale: The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).

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.

SEMANTICS: SCOPE, CONDITIONS, COMPONENT*, SHALL*, TIMING, RESPONSES*

component shall always satisfy if (input_state & condition) then output_state

TEMPLATES

Template: Change State

Choose a predefined template

This template describes how the state of a finite-state-machine component changes. It describes the input state and some conditions based on which the change must occur. The corresponding output state must reflect the required change. The input and output states have a pre - post- relationship

Examples:

```
FSM_Autopilot shall always satisfy if (
state = ap_standby_state & ! standby & ! apfail ) then
STATE = ap_transition_state
```

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

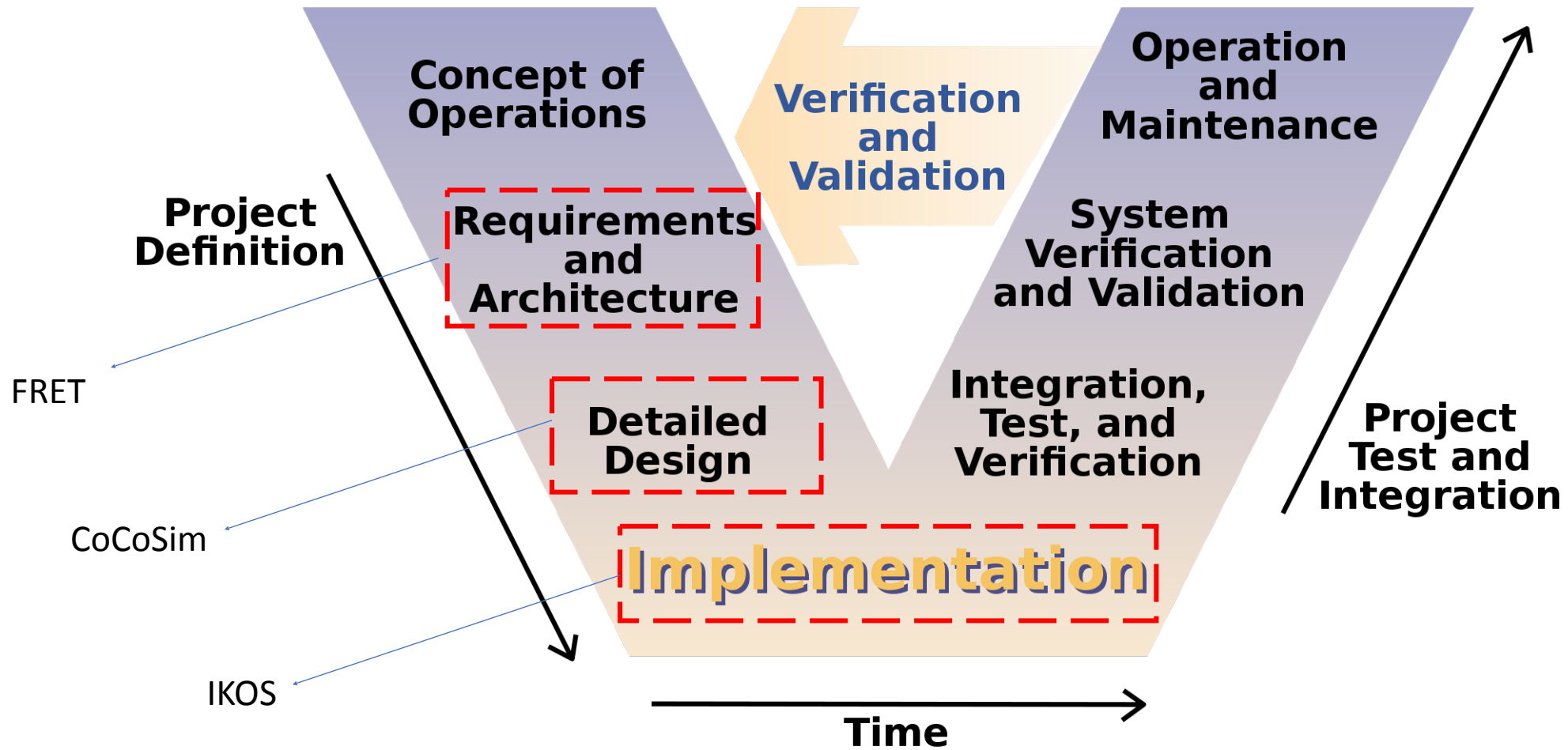
The screenshot shows the FRET web application interface. At the top, there is a menu with 'File', 'View', and 'Help'. Below the menu, the 'FRET' logo is visible, along with a 'Projects' dropdown and a 'CREATE' button. The main content area is divided into two tabs: 'VARIABLE MAPPING' and 'REALIZABILITY', with 'REALIZABILITY' being the active tab. Under the 'REALIZABILITY' tab, there is a 'System Component*' dropdown set to 'FSM'. To the right of this dropdown, 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 is a horizontal bar with three tabs: 'CC0', 'CC1', and 'CC2', with 'CC2' being the active tab. Below this bar is a table with two columns: 'ID ↑' and 'Summary'. The table contains 10 rows of data, each representing a consistency check result. At the bottom right of the table, there is a pagination control showing 'Rows per page: 10' and '1-10 of 13'.

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

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

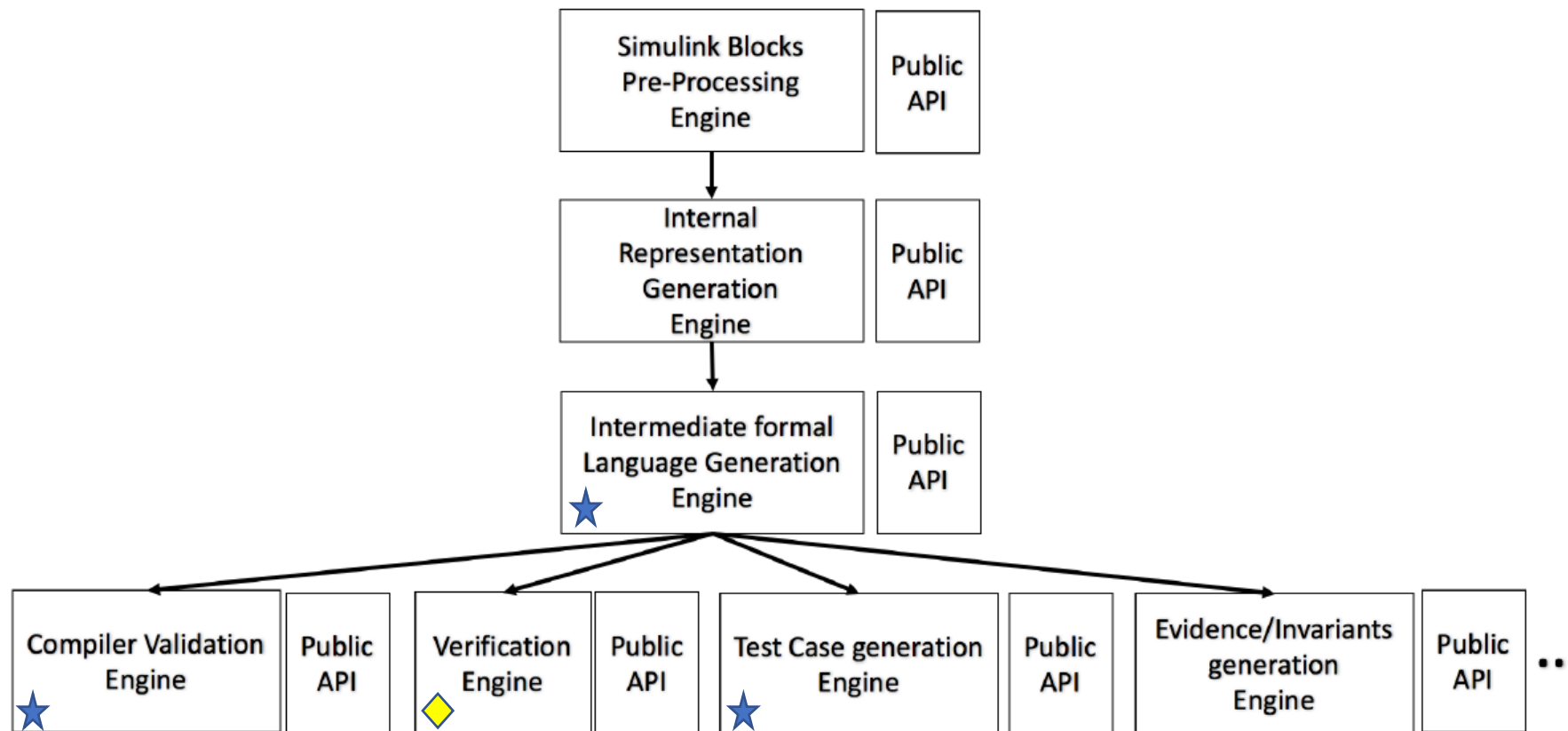
- Captures requirements in a restricted natural language with unambiguous semantics
- Explains formal semantics in various forms: natural language, diagrams, interactive simulation
- Assists in writing requirements through requirement templates
- Formalizes requirements in a compositional (hence maintainable 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



CoCoSim: an open-source MATLAB plugin for

- Contract-based Compositional Verification of Simulink/Stateflow Models
 - Simulink/Stateflow translation to Verimag Lustre / C
 - Work in progress: Link with requirements specification tools (FRET), Test Case Generation
-
- Bourbough, Hamza, Marie Farrell, Anastasia Mavridou, Irfan Slijivo, Guillaume Brat, Louise A. Dennis, and Michael Fisher. "Integrating formal verification and assurance: an inspection rover case study." In *NASA Formal Methods Symposium*, pp. 53-71. Springer, Cham, 2021.
 - Mavridou, Anastasia, Hamza Bourbough, Dimitra Giannakopoulou, Thomas Pressburger, Mohammad Hejase, Pierre-Loic Garoche, and Johann Schumann. "The ten lockheed martin cyber-physical challenges: formalized, analyzed, and explained." In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pp. 300-310. IEEE, 2020.
 - Mavridou, Anastasia, Hamza Bourbough, Pierre Loic Garoche, Dimitra Giannakopoulou, Thomas Pessburger, and Johann Schumann. "Bridging the gap between requirements and Simulink model analysis." In *Joint 26th International Conference on Requirements Engineering: Foundation for Software Quality Workshops, Doctoral Symposium, Live Studies Track, and Poster Track*. 2020.
 - Bourbough, Hamza, Pierre-Loic Garoche, Thomas Loquen, Éric Noulard, and Claire Pagetti. "CoCoSim, a code generation framework for control/command applications An overview of CoCoSim for multi-periodic discrete Simulink models." In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*. 2020.
 - Bourbough, Hamza, Guillaume Brat, and Pierre-Loic Garoche. "CoCoSim: an automated analysis framework for Simulink/Stateflow." In *Model Based Space Systems and Software Engineering-European Space Agency Workshop (MBSE 2020)*. 2020.

CoCoSim Architecture

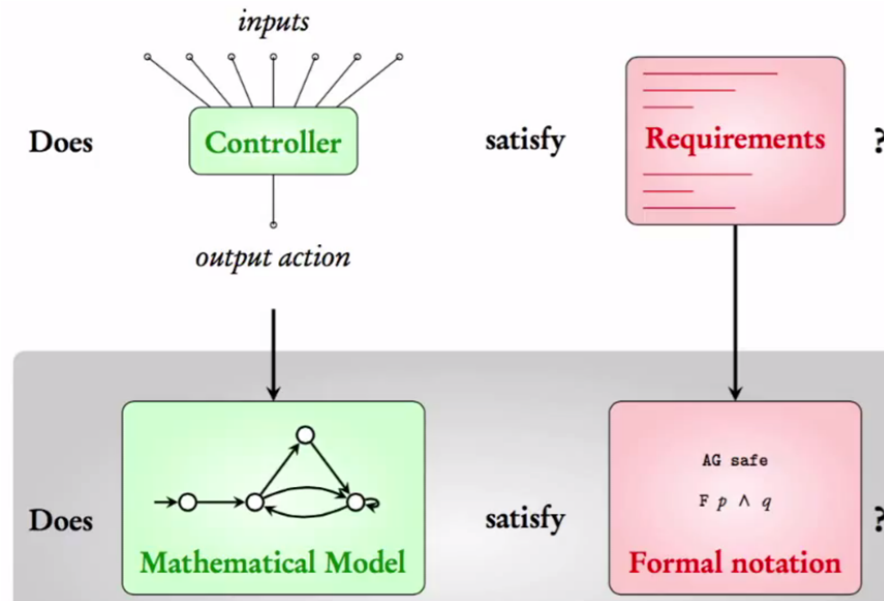


★ LustreC

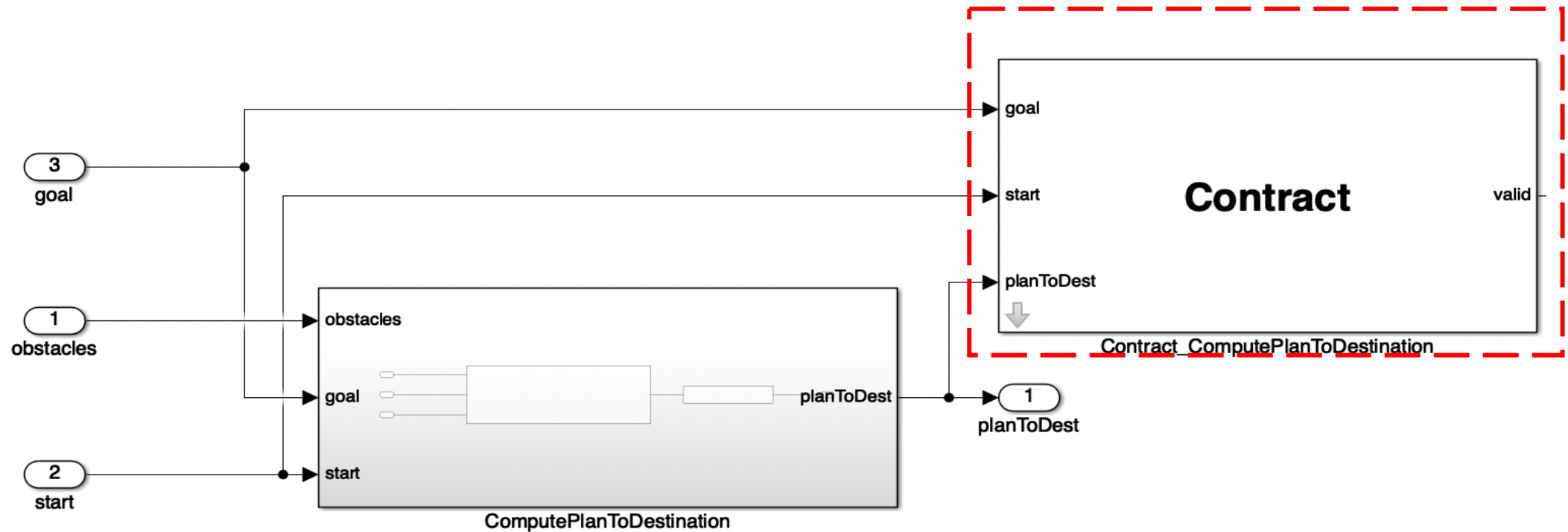
◆ Kind 2

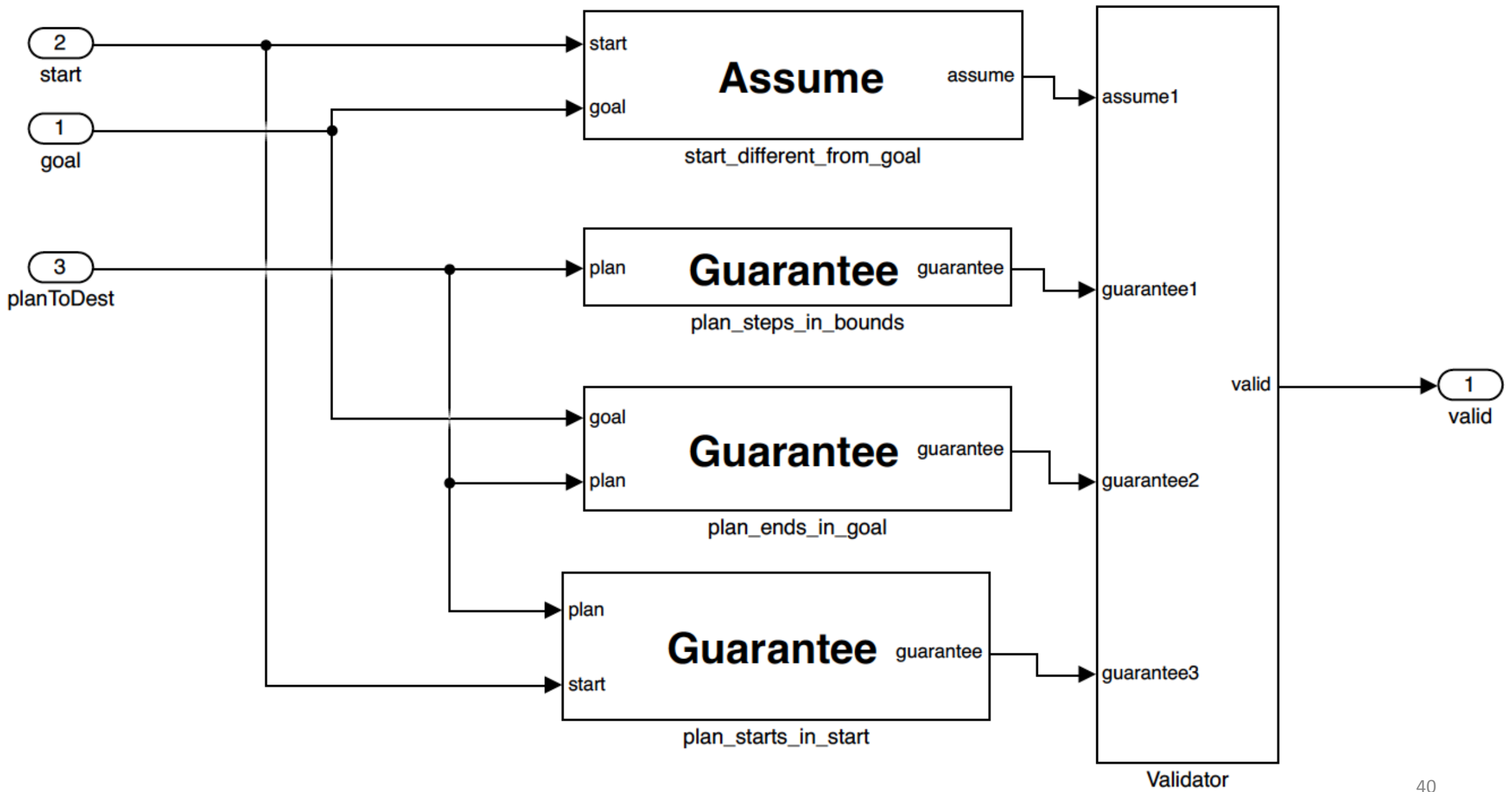
Verification

- Formal verification: Prove that a system **always** conforms to our requirements
- By constructing a mathematical proof!
- Model checking



Requirements are expressed using the CoCoSim library.





HOME PLOTS APPS

New Script New Live Script New Open Compare Import Data Save Workspace New Variable Open Variable Clear Workspace Favorites Analyze Code Run and Time Clear Commands Simulink Layout Preferences Set Path Add-Ons Help Community Request Support Learn MATLAB

FILE VARIABLE CODE SIMULINK ENVIRONMENT RESOURCES

Current Folder

/ home akatis Documents CoCoSim demo

Name
absolute.slx

Details

Command Window

New to MATLAB? See resources for [Getting Started](#).

```

>> start_cocosim
*****
WELCOME TO COCOSIM (NASA Ames)
*****
... Starting cocoSim configuration
From https://github.com/NASA-SW-VnV/CoCoSim
* branch      installation_fix -> FETCH_HEAD
Already up to date.
Already on 'cocosim_nasa'
Your branch is up to date with 'origin/cocosim_nasa'.
From https://github.com/coco-team/cocoSim2
* branch      cocosim_nasa -> FETCH_HEAD
Already up to date.
Already on 'master'
From https://github.com/hbourbouh/cocosim-external-libs
* branch      master -> FETCH_HEAD
Already up to date.
[warning][tools_config] YICES2 is not found in '/home/akatis/git/CoCoSim/tools/verifiers/linux/bin/yices-
[warning][tools_config] You can ignore the previous warning if you are not going to use YICES2 with Kind:
Tools config is already run and will be ignored.
To force it run "tools_config" in your Matlab Command Window.

Click here to change pre-processing configuration.
... Configuration is Done

Click here to start with a simple verification example.
fg >>

```

Workspace

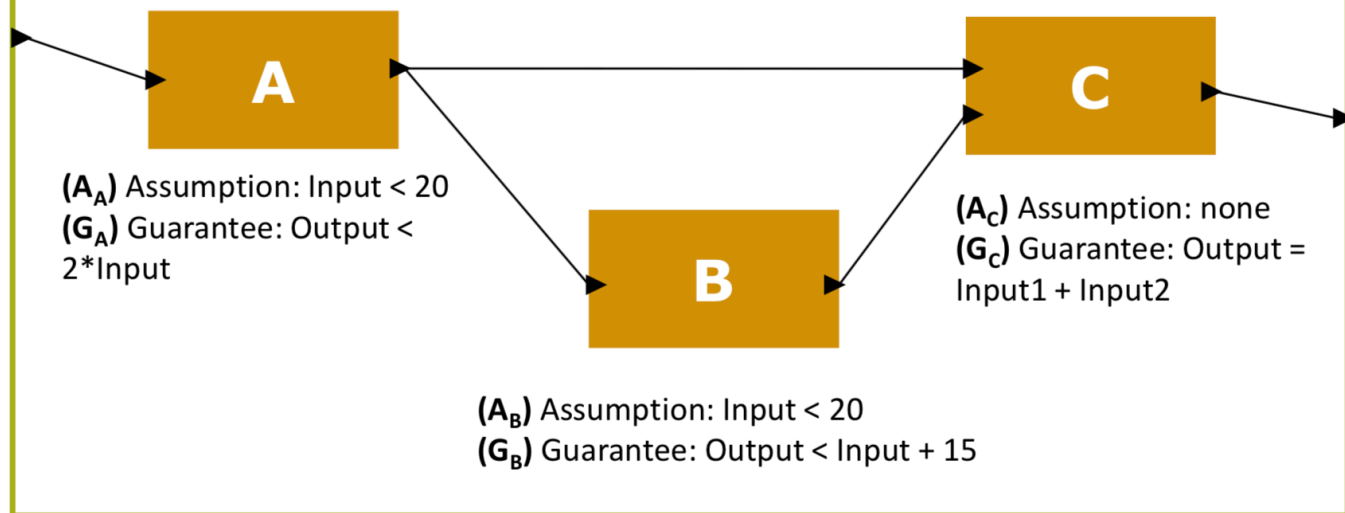
Name	Value
cocosim_verb...	0

Compositional Verification

Example (to prove)

$A_S \rightarrow A_A$
 $A_S \wedge G_A \rightarrow A_B$
 $A_S \wedge G_A \wedge G_B \rightarrow A_C$
 $A_S \wedge G_A \wedge G_B \wedge G_C \rightarrow G_S$

(A_S) Assumption: Input < 10
 (G_S) Guarantee: Output < 50



$Input_A = 9$, $Output_A = 17$, $Output_B = 31$, $Output_C = 48$

New Script New Live Script New Open Compare Import Data Save Workspace Clear Workspace
 FILE VARIABLE

Analyze Code Run and Time Clear Commands
 CODE

Simulink Layout Set Path Add-Ons Help
 SIMULINK ENVIRONMENT RESOURCES

Community Request Support Learn MATLAB

home > akatis > Documents > CoCoSim > demo >

Current Folder

Name
cocosim_output
ABC.slx
ABC_PP.slx
absolute.slx
absolute_PP.slx

ABC.slx (Simulink Model)

Command Window

```

New to MATLAB? See resources for Getting Started.

>> start_cocosim
*****
WELCOME TO COCOSIM (NASA Ames)
*****
... Starting cocoSim configuration
From https://github.com/NASA-SW-VnV/CoCoSim
 * branch      installation_fix -> FETCH_HEAD
Already up to date.
Already on 'cocosim_nasa'
Your branch is up to date with 'origin/cocosim_nasa'.
From https://github.com/coco-team/cocoSim2
 * branch      cocosim_nasa -> FETCH_HEAD
Already up to date.
Already on 'master'
From https://github.com/hbourbouh/cocosim-external-libs
 * branch      master -> FETCH_HEAD
Already up to date.
[warning][tools_config] YICES2 is not found in '/home/akatis/git/CoCoSim/tools/verifiers/linux/bin/yices-
[warning][tools_config] You can ignore the previous warning if you are not going to use YICES2 with Kind:
Tools config is already run and will be ignored.
To force it run "tools_config" in your Matlab Command Window.

Click here to change pre-processing configuration.
... Configuration is Done

Click here to start with a simple verification example.

f >>
  
```

Workspace

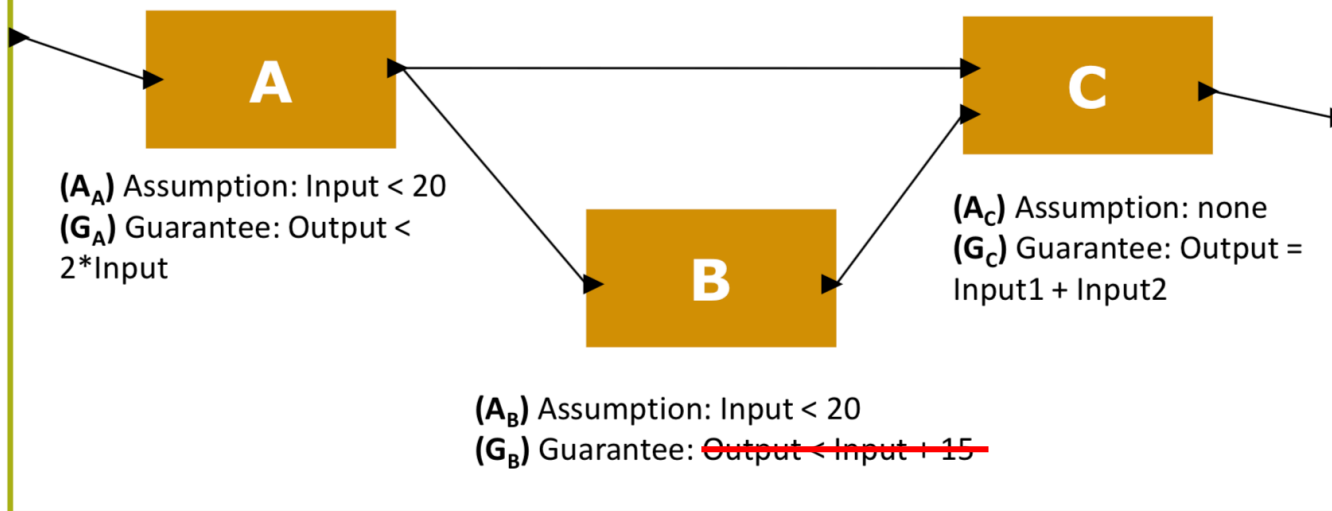
Name	Value
cocosim_verb...	0

Compositional Verification

Example (to prove)

$A_S \rightarrow A_A$
 $A_S \wedge G_A \rightarrow A_B$
 $A_S \wedge G_A \wedge G_B \rightarrow A_C$
 $A_S \wedge G_A \wedge G_B \wedge G_C \rightarrow G_S$

(A_S) Assumption: $Input < 10$
 (G_S) Guarantee: $Output < 50$



$$G_B = Output < Input + 50$$

Verification Summary

ABC_PP/S/C__abstracted/C

Result



ABC_PP/S/B__abstracted/B

Result



ABC_PP/S/A__abstracted/A

Result



ABC_PP/S

Abstract C

Abstract B

Abstract A

Abstract C

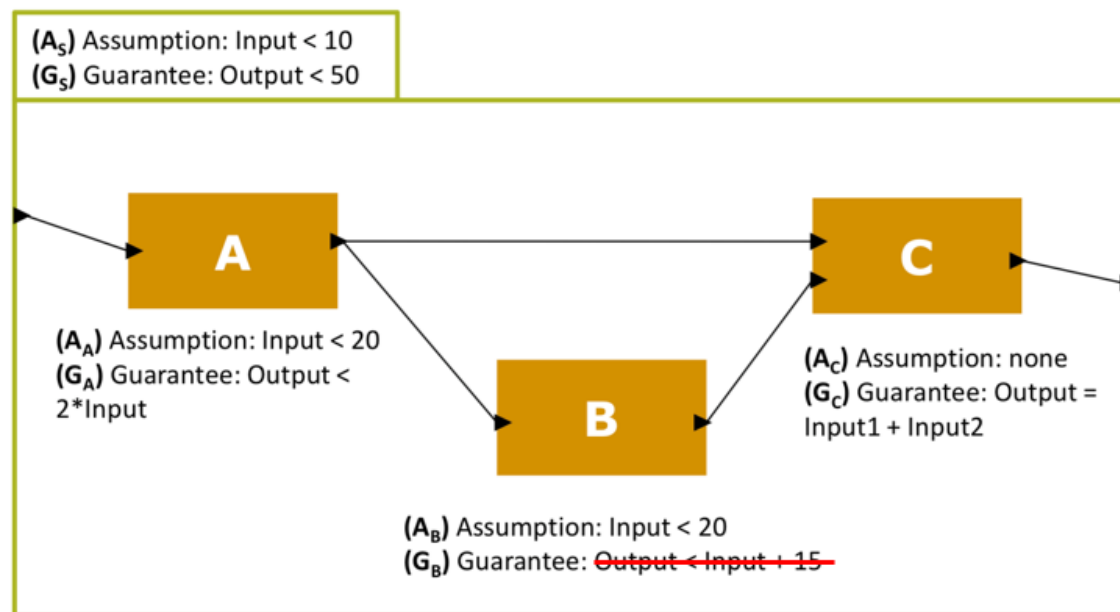
Result



The new contract for B is not sufficient to prove the system-level contract!

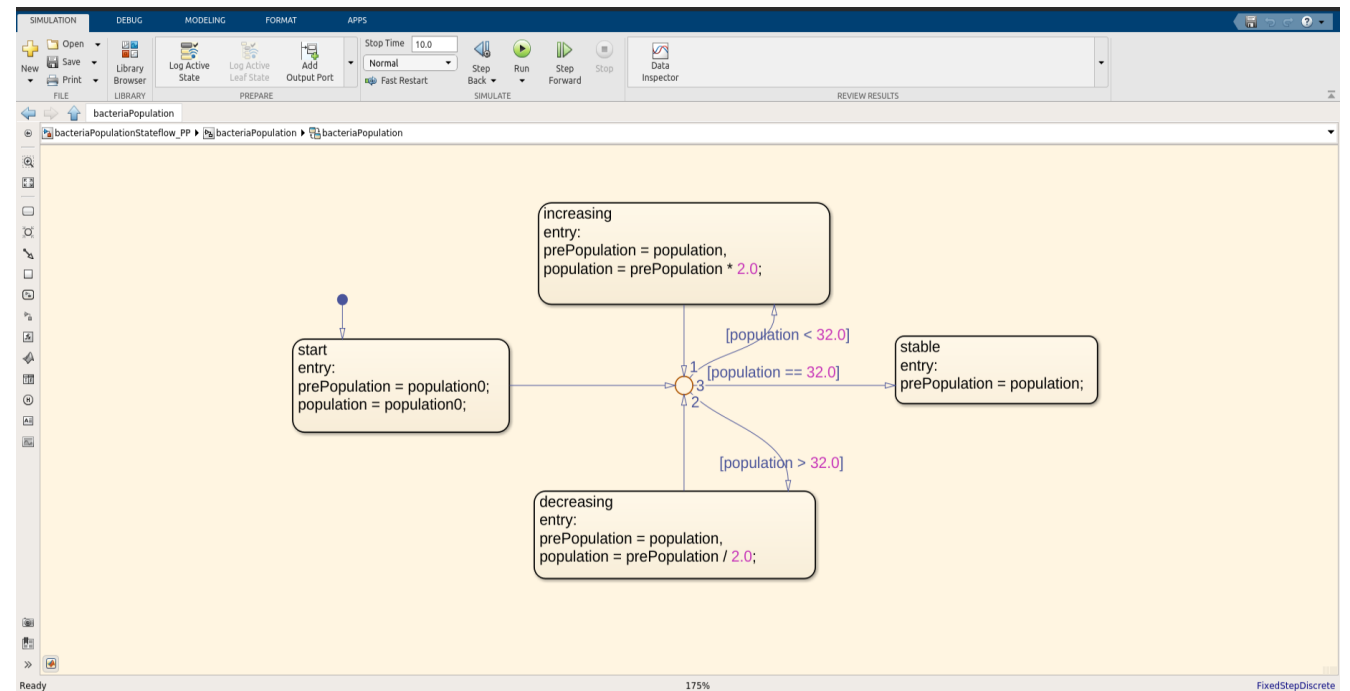
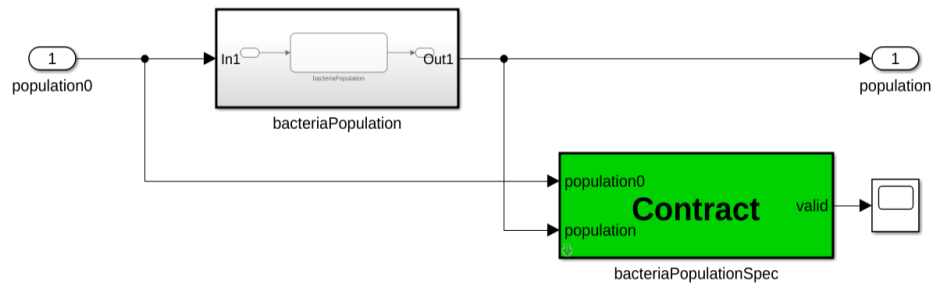
ABC_PP/S

Time	0
Input (input)	9
Output (output)	50
C__abstracted (local)	50
A__abstracted (local)	17
B__abstracted (local)	33



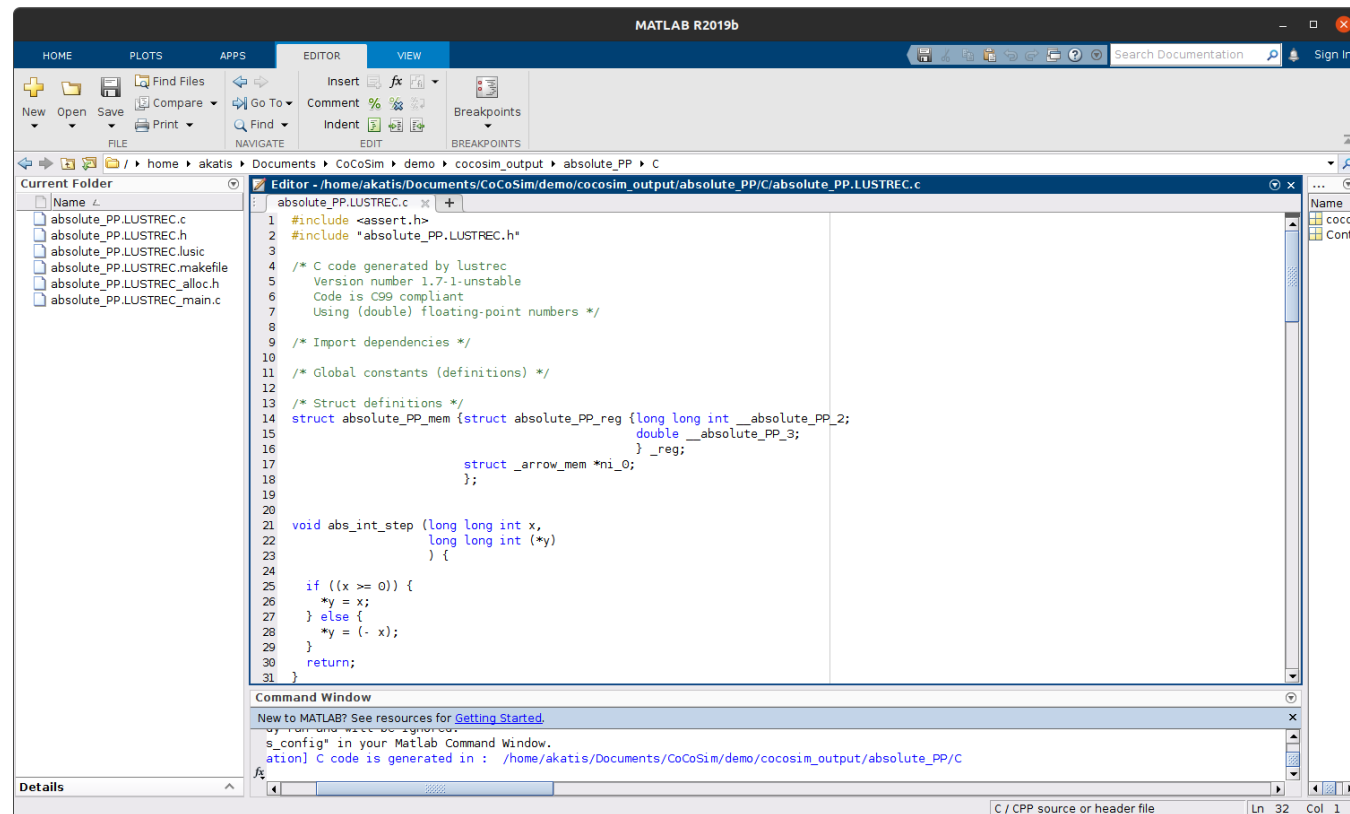
$$G_B = Output < Input + 50$$

Stateflow verification



Code Generation

- Verimag Lustre & C (through LustreC)
- Rust (through Kind 2)
- Compiler validation through equivalence testing and equivalence checking



```
absolute_PP.LUSTREC.c
1  #include <assert.h>
2  #include "absolute_PP.LUSTREC.h"
3
4  /* C code generated by lustrec
5  Version number 1.7.1-unstable
6  Code is C99 compliant
7  Using (double) floating-point numbers */
8
9  /* Import dependencies */
10
11 /* Global constants (definitions) */
12
13 /* Struct definitions */
14 struct absolute_PP_mem {struct absolute_PP_reg {long long int __absolute_PP_2;
15                                     double __absolute_PP_3;
16                                     } _reg;
17                                     struct _arrow_mem *_ni_0;
18 };
19
20
21 void abs_int_step (long long int x,
22                  long long int (*y)
23                  ) {
24
25     if ((x >= 0)) {
26         *y = x;
27     } else {
28         *y = (- x);
29     }
30     return;
31 }
```

Command Window

New to MATLAB? See resources for [Getting Started](#).

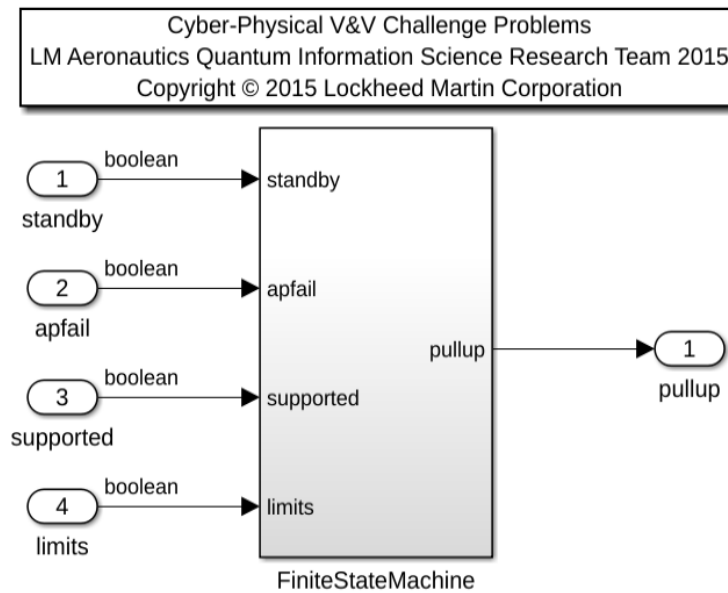
...
s_config" in your Matlab Command Window.
ation] C code is generated in : /home/akatis/Documents/CoCoSim/demo/cocosim_output/absolute_PP/C

Details

C / CPP source or header file Ln 32 Col 1

Connection with FRET

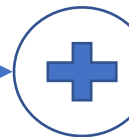
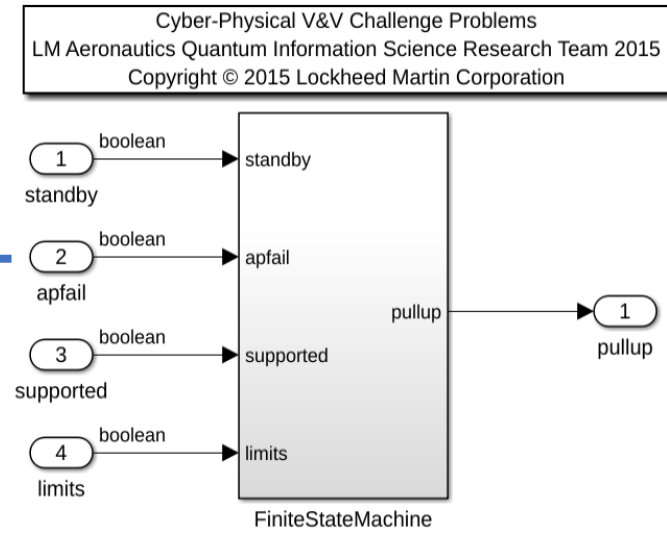
- Import FRET requirements in CoCoSim
- Automated generation of Simulink monitors based on traceability data
- Analysis results can be traced back to original FRET requirements



FRET Requirements

Status	ID	Summary	Project
Green	FSM-001V1	FSM shall always satisfy (limits & lstandby & lpfail & supported) => pullup	CoCoSim_FSM
Green	FSM-001V2	if autopilot & pre_autopilot & pre_limits FSM shall immediately satisfy pullup	CoCoSim_FSM
Green	FSM-001V3	if htlore3_autopilot & htlore3_notpreprelimits & pre_limits FSM shall immediately satisfy pullup	CoCoSim_FSM
Green	FSM-002	FSM_Autopilot shall always satisfy (standby & state = ap_transition_state) => STATE = ap_standby_state	CoCoSim_FSM
Green	FSM-003	FSM_Autopilot shall always satisfy (state = ap_transition_state & good & supported) => STATE = ap_nominal_state	CoCoSim_FSM
Green	FSM-004	FSM_Autopilot shall always satisfy (! good & state = ap_nominal_state) => STATE = ap_maneuver_state	CoCoSim_FSM
Green	FSM-004V2	FSM_Autopilot shall always satisfy (state = ap_nominal_state & ! good & ! standby) => STATE = ap_maneuver_state	CoCoSim_FSM
Green	FSM-005	FSM_Autopilot shall always satisfy (state=ap_nominal_state & standby) => STATE = ap_standby_state	CoCoSim_FSM
Green	FSM-006	FSM_Autopilot shall always satisfy (state = ap_maneuver_state & standby & good) => STATE = ap_standby_state	CoCoSim_FSM
Green	FSM-007	FSM_Autopilot shall always satisfy (state = ap_maneuver_state & supported & good) => STATE = ap_transition_state	CoCoSim_FSM

Model Information (JSON)



Export CoCoSpec (Lustre) + Traceability data

FRET Variable Name	Model Variable Name	Variable Type	Data Type	Description
APFAIL	apfail	Input	boolean	
AUTOPILOT		Internal	boolean	
HTLORE3_AUTOPILOT		Internal	boolean	
HTLORE3_NOTPREPRELIMITS		Internal	boolean	
LIMITS	supported	Input	boolean	

Requirement Variables to Model Mapping: LM_requirements

Export Language * ▼

Autopilot EXPORT

FSM EXPORT

Corresponding Model Component fsm_12B/FiniteStateMachine IMPORT

FRET Variable Name	Model Variable Name	Variable Type	Data Type	Description
APFAIL	apfail	Input	boolean	
AUTOPILOT		Internal	boolean	
HTLORE3_AUTOPILOT		Internal	boolean	
HTLORE3_NOTPREPRELIMITS		Internal	boolean	
LIMITS	supported	Input	boolean	

MATLAB Window Help fsm_12B

SIMULATION DEBUG MODELING FORMAT APPS

New Open Save Print Library Browser Log Signals Add Viewer Signal Table Stop Time 10 Normal Fast Restart Step Back Run Step Forward Stop Data Inspector

Tools CoSim Sen x fsm_12B

fsm_12B

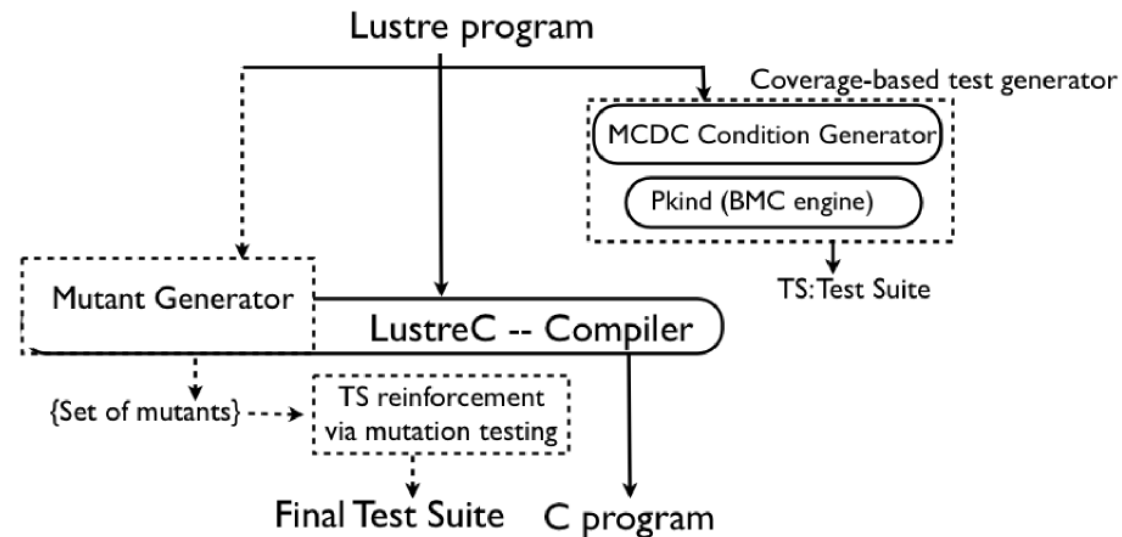
Cyber-Physical V&V Challenge Problems
LM Aeronautics Quantum Information Science Research Team 2015
Copyright © 2015 Lockheed Martin Corporation

```
graph LR; I1((1)) -- boolean --> S[standby]; I2((2)) -- boolean --> A[apfail]; I3((3)) -- boolean --> Su[supported]; I4((4)) -- boolean --> L[limits]; S --> FSM[FiniteStateMachine]; A --> FSM; Su --> FSM; L --> FSM; FSM -- pullup --> O1((1))
```

Automated Analysis Framework 222% FixedStepDiscrete

Test-case generation

- Random test generation (Simulink)
 - MC/DC
 - Mutation-based
- } via LustreC (Work in Progress)



HOME PLOTS APPS

New Script New Live Script New Open Find Files Compare Import Data Save Workspace New Variable Open Variable Clear Workspace Favorites Analyze Code Run and Time Clear Commands Simulink Layout Preferences Set Path Add-Ons Help Community Request Support Learn MATLAB

FILE VARIABLE CODE SIMULINK ENVIRONMENT RESOURCES

Search Documentation Sign In

Current Folder

/ > home > akatis > Documents > CoCoSim > demo

Name
cocosim_output
ABC.slx
ABC_PP.slx
absolute.slx
absolute_PP.slx
mcdc_test.slx

Details

Command Window

New to MATLAB? See resources for [Getting Started](#).

```
>> clear  
fx >>
```

Workspace

Name	Value
------	-------

Library	# supp. Blocks	% supp. Blocks	Unsupported blocks
Discontinuities	11/12	91%	Backlash
Discrete	19/21	90%	Discrete PID Controller, Discrete PID Controller (2DOF)
Logic & Bit Operations.	18/19	95%	Extract Bits
Lookup Tables.	9/9	100%	
Math Operations.	31/37	83%	Algebraic Constraint, Complex to Magnitude-Angle, Complex to Real-Imag, Find, Magnitude-Angle to Complex, Real-Imag to Complex
Model Verification	11/11	100%	
Ports & Subsystems.	29/31	93%	While Iterator Subsystem, While Iterator
Signal Attributes.	13/14	93%	Unit Conversion
Signal Routing.	13/25	52%	Data Store Memory, Data Store Read, Data Store Write, Environment Controller, Goto Tag Visibility, Index Vector, State Reader, State Writer, Variant Source, Variant Sink, Manual Variant Source, Manual Variant Sink
Sinks.	9/9	100%	Visualization blocks are ignored
Sources.	15/26	57%	Band-Limited White Noise, Counter Free-Running, Counter Limited, From File, From Spreadsheet, Repeating Sequence, Repeating Sequence Interpolated, Repeating Sequence Stair, Signal Editor, Signal Generator, Waveform Generator
User-Defined Functions.	2/15	13%	Argument Inport, Argument Outport, Event Listener, Function Caller, Initialize Function, Interpreted MATLAB Function, Level-2 MATLAB S-Function, MATLAB System, Reset Function, S-Function, S-Function Builder, Simulink Function, Terminate Function

Block Name	Unsupported Options
Trigonometry	Operator 'cos + jsin' is not supported.
Switch	Allow different data input sizes option is not supported.
Relational Operator	Operator "isInf", "isNaN", "isFinite" are not supported.
Merge	- Allow unequal port widths is not supported. - We support only Merge blocks that are connected to conditionally-executed subsystem.
Function Call Generator	Number of iterations > 1 is not supported.
For Iterator	- External iteration limit source is not supported. - External increment is not supported. - Action Ports inside a For Iterator block should have "States when execution is resumed" option set to "reset". - Outports in a conditionally executed Subsystem inside a For Iterator block should have "Output when disabled" set to "reset". - Memory blocks are only allowed in the first level of the For Iterator Subsystem.
Discrete Pulse Generator	Option "Use external signal" is not supported.
Demux	Bus selection mode should be off.
Selector	"Starting and ending indices (port)" option is not supported.
Multiport Switch	- "Specify indices" option is not supported. - Allow different data input sizes is not supported.
Lookup Table blocks	- More than 7 dimensions interpolation is not supported. - "Intermediate results Data Type" option should be set to double or single.
From Workspace	"Cyclic repetition" option is not supported.
Delay	When Delay length > 1, the initial condition should be scalar.
Concatenate	Concatenate dimension > 2 is not supported.
Assignment	- OutputInitialize set to 'Specify size for each dimension in table' is not supported, - IndexOptionArray set to 'Starting and ending indices (port)' is not supported.

Thank you!

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

Anastasia Mavridou (anastasia.mavridou@nasa.gov)

Thomas Pressburger (tom.pressburger@nasa.gov)

Johann Schumann (johann.schumann@nasa.gov)

Andreas Katis (andreas.katis@nasa.gov)

Khanh Trinh (khanh.v.trinh@nasa.gov)

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

Andreas Katis (andreas.katis@nasa.gov)

Guillaume Brat (guillaume.p.brat@nasa.gov)

Pierre-Loïc Garoche (pierre-loic.garoche@enac.fr)

