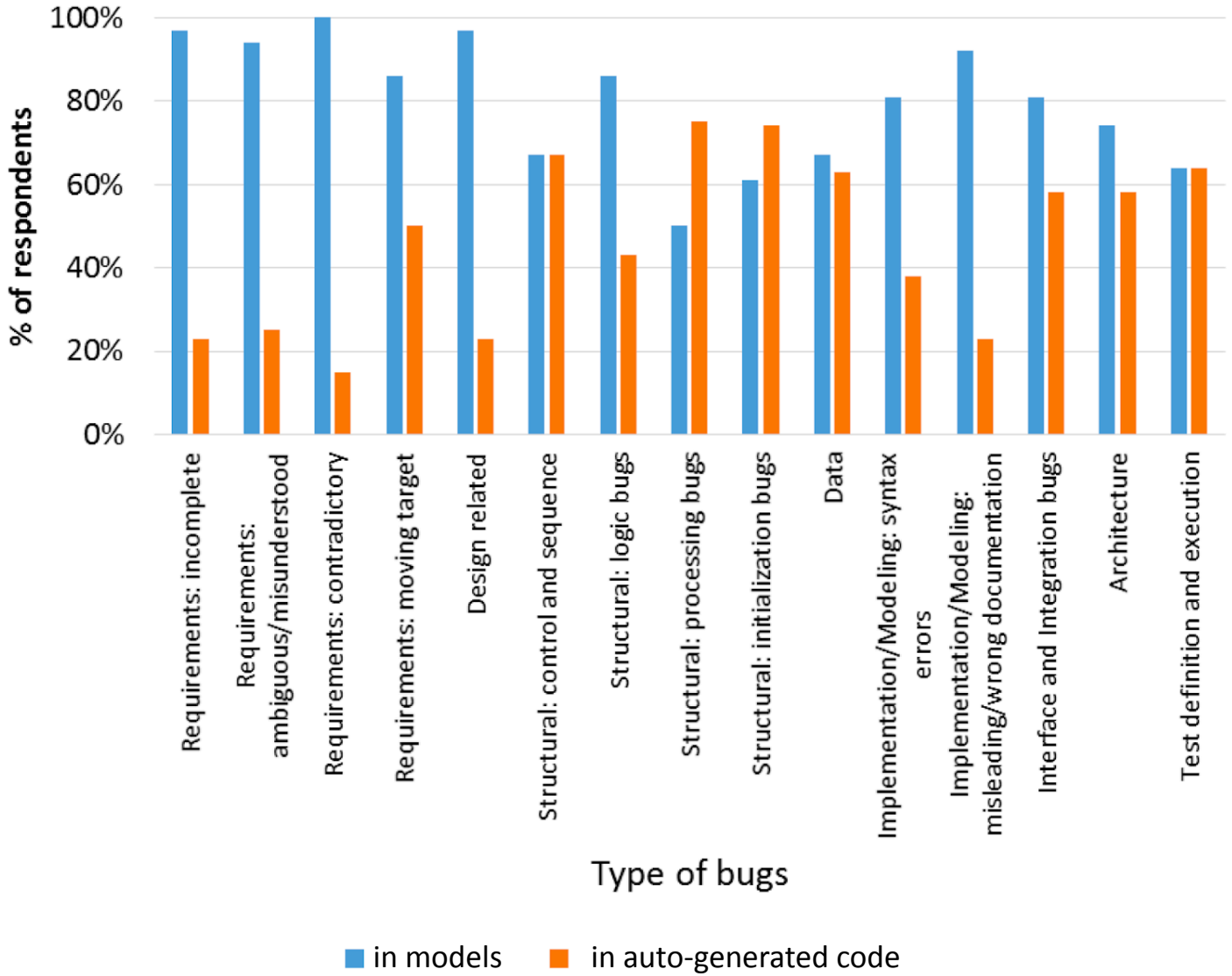
Requirements elicitation, analysis and verification using FRET and CoCoSim

Andreas Katis[1,2]

FEANICSES 2022, December 6, Toulouse, France

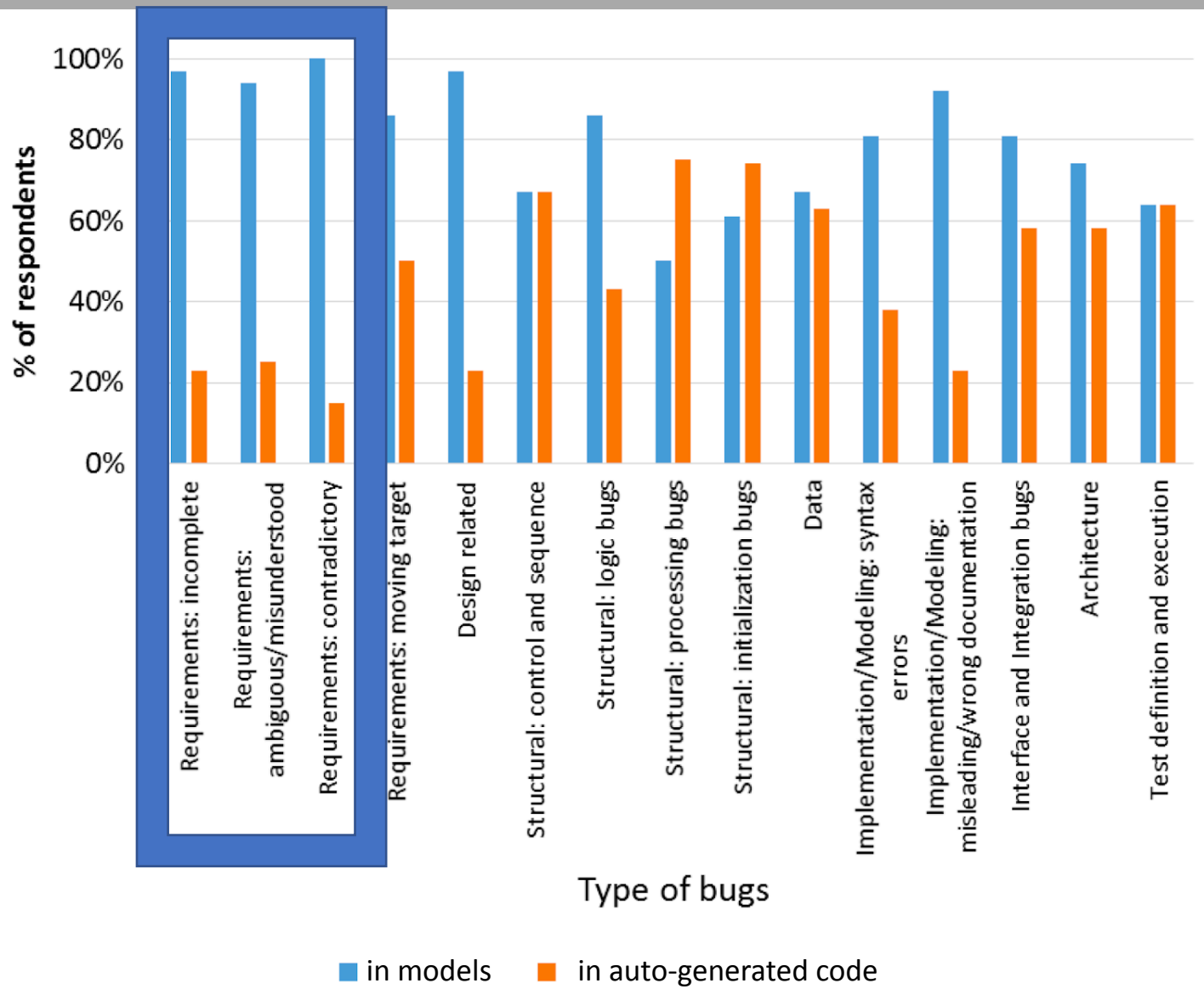
[1] Employed by KBR; NASA Ames Research Center, CA, USA
[2] NASA Ames Research Center, CA, USA

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 apfail).

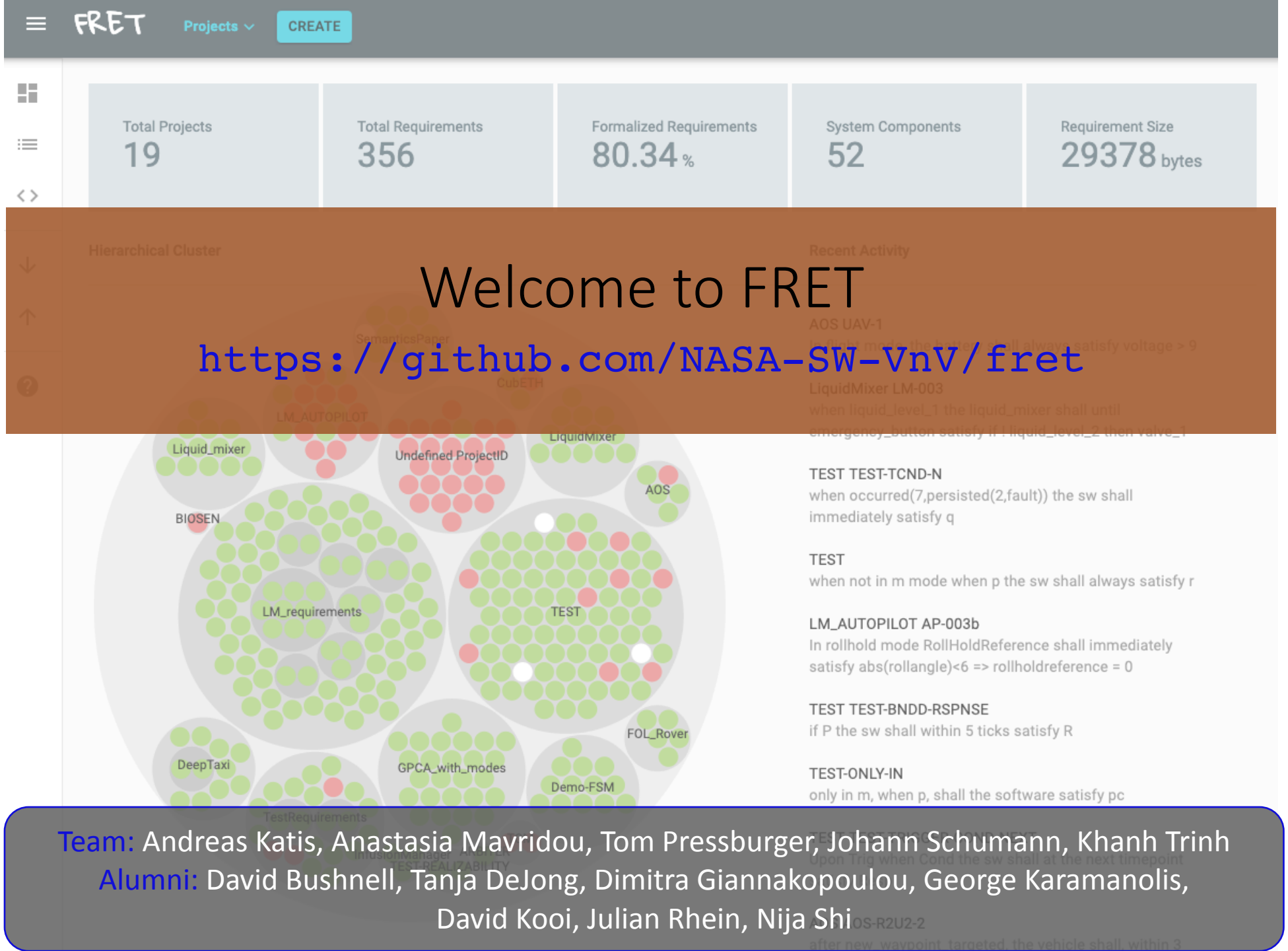
every time these conditions hold or only when they **become** true?

- 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

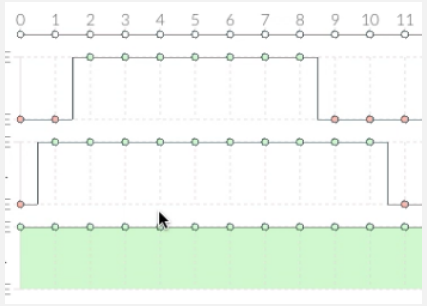
Alumni: David Bushnell, Tanja DeJong, Dimitra Giannakopoulou, George Karamanolis, David Kooi, Julian Rhein, Nija Shi

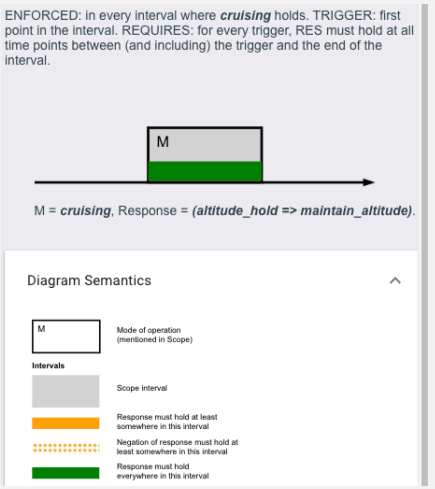
captures + assists

FRETish

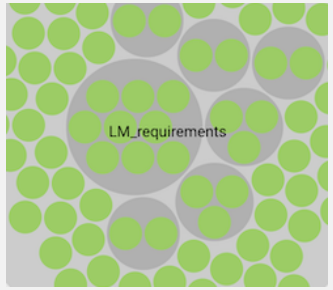
when in cruising mode, the altitude_hold_

explains





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

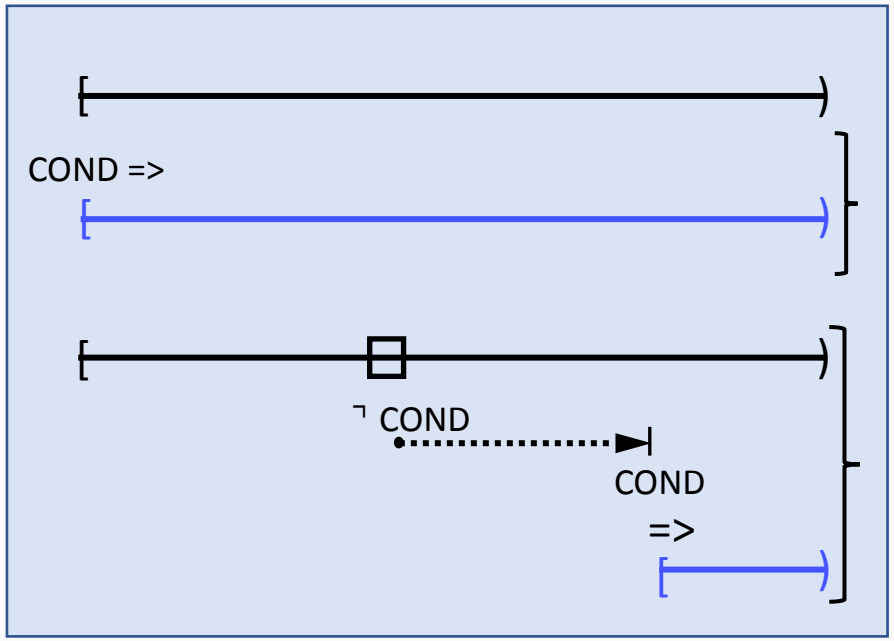
ep 3	Step 4	Step 5
rue true	true	true
rue true	true	true
rue true	true	true

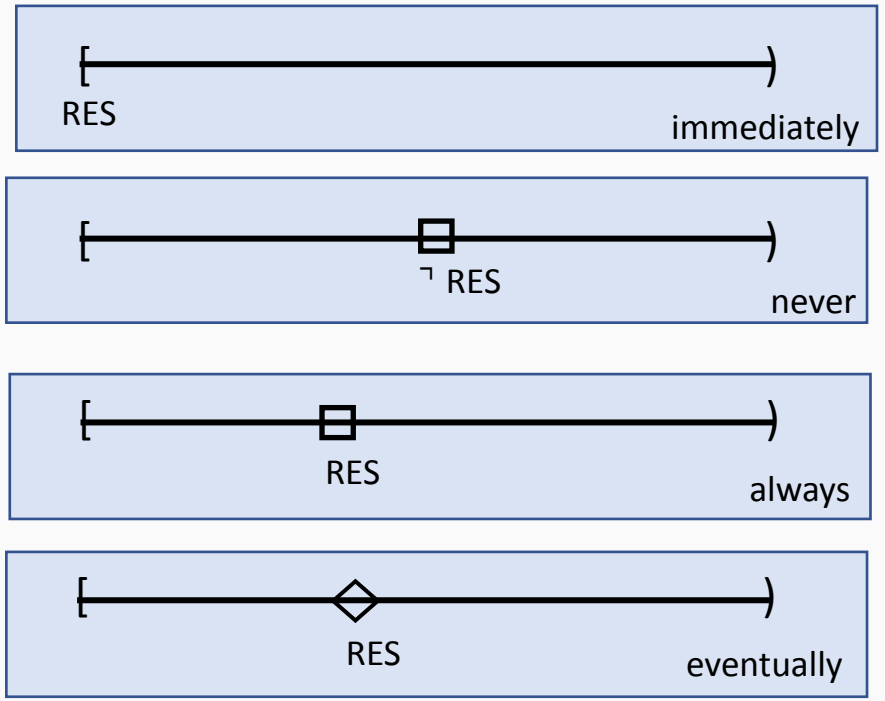
FRET bridges the gap

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

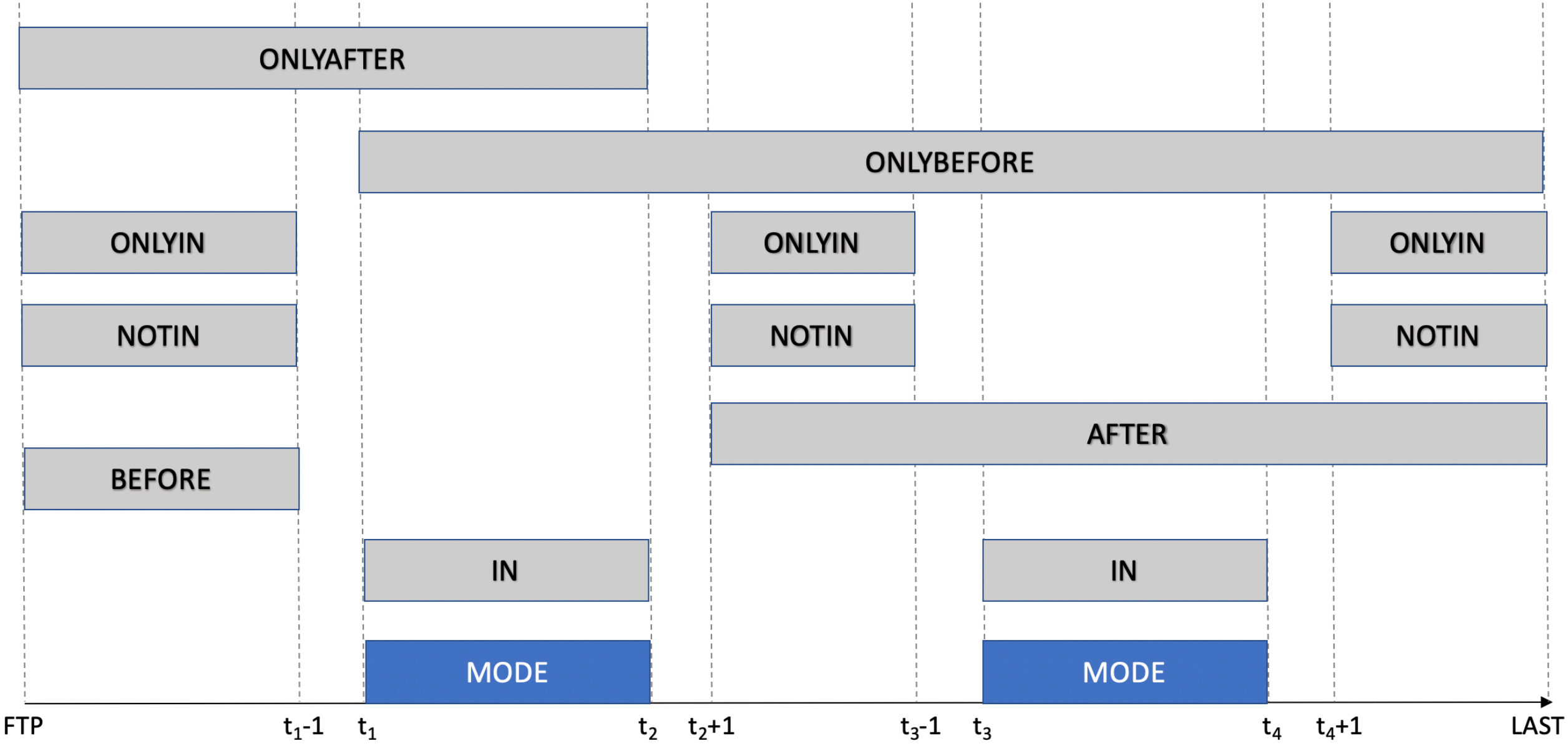
SCOPE	in, before, after, notin, onlyIn, onlyBefore, onlyAfter; when omitted, global
CONDITION	null, regular
TIMING	immediately, next, always, never, eventually, until, before, for, within, after
RESPONSE	satisfaction

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

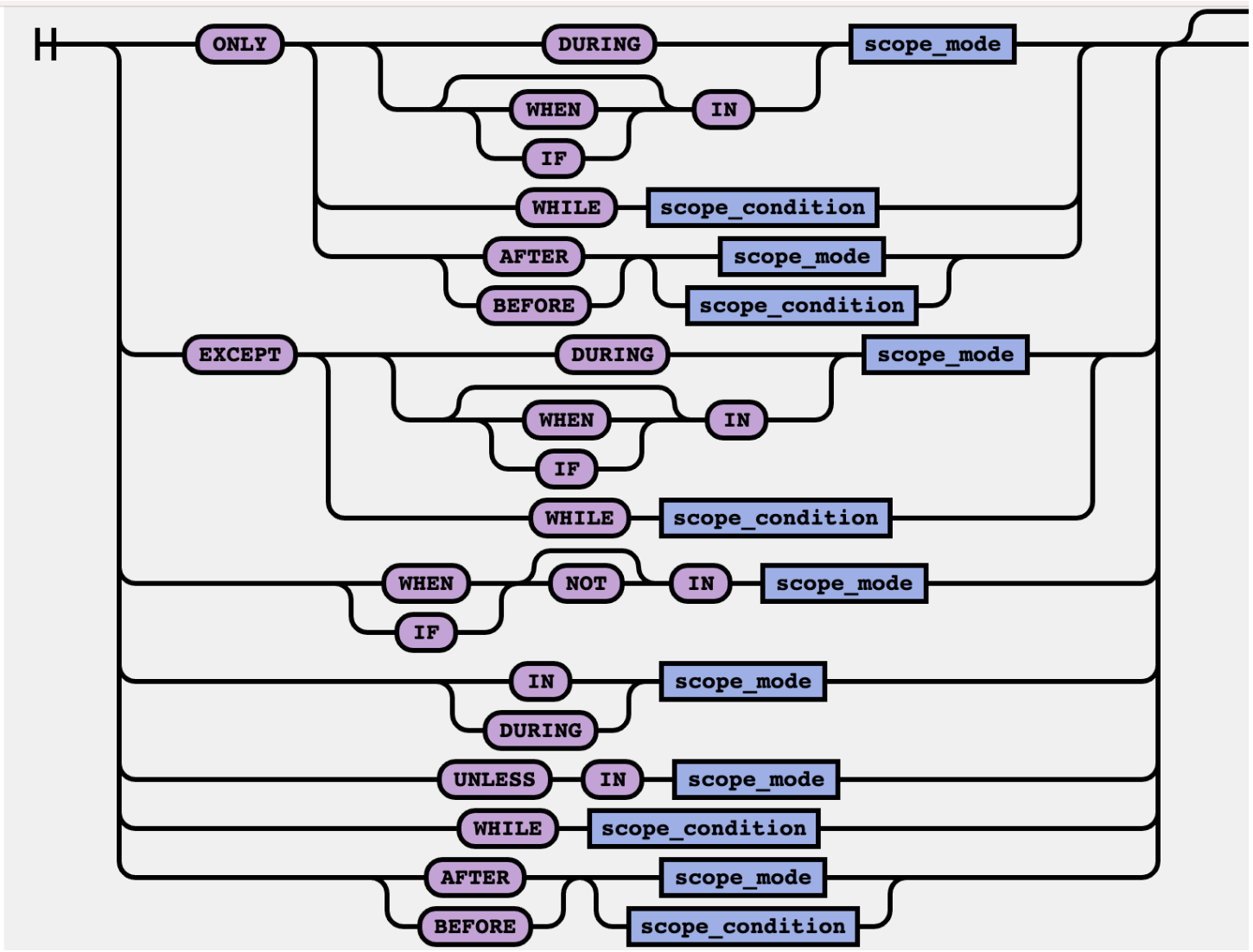
Scope Intervals



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

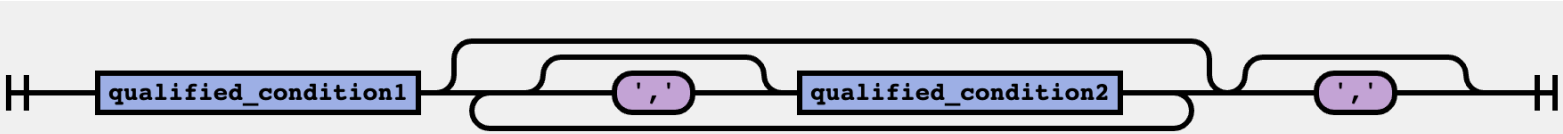


Conditions

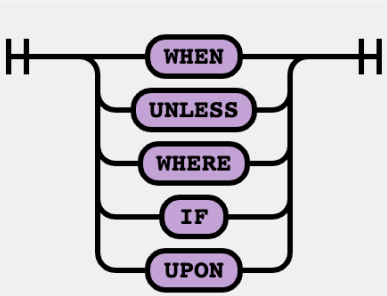
- *upon, if, when, where; unless*
- Boolean expression
- Trigger: **upon** the Boolean expression becoming true from being false in the scope, or being true at the beginning of the scope.

Condition grammar

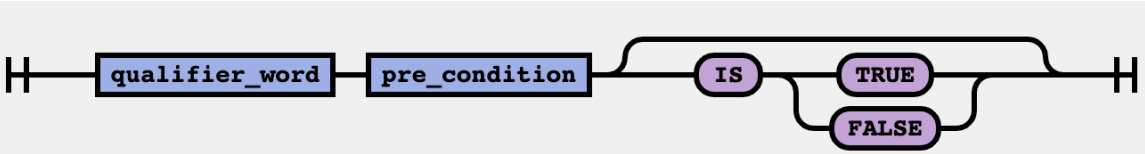
regular_condition



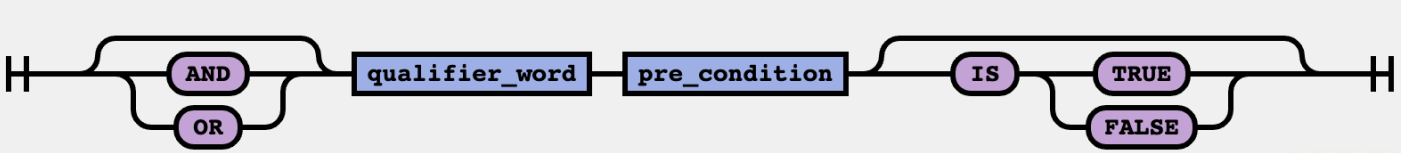
qualifier_word



qualified_condition1



qualified_condition2



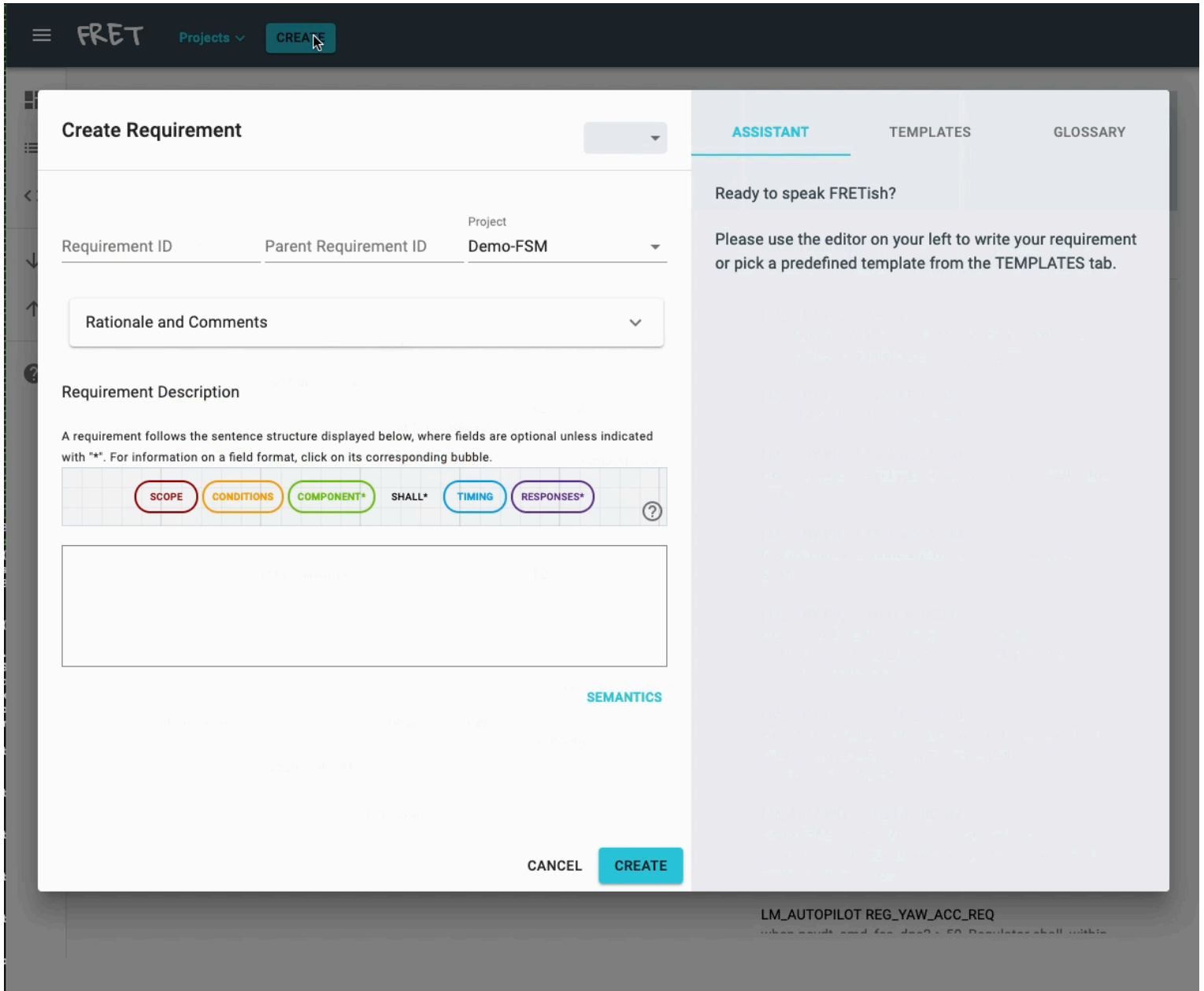
Timings

- 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



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*

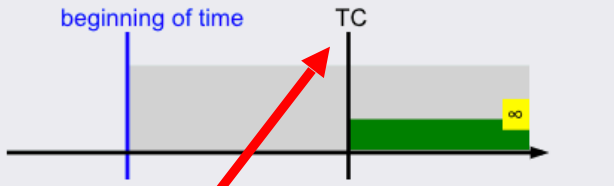
if altitude_hold_selected the altitude_hold_autopilot shall always satisfy maintain_altitude

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

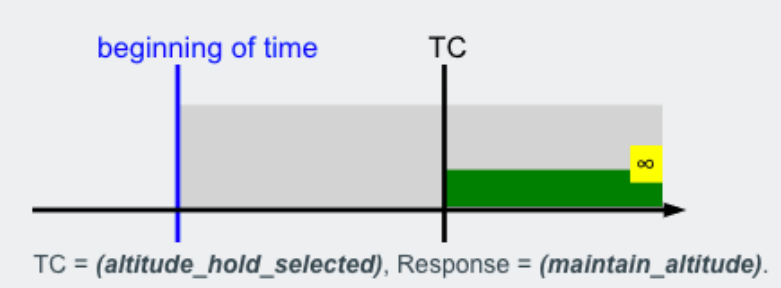
```
(H ((H (! (altitude_hold_selected))) |
(maintain_altitude)))
```

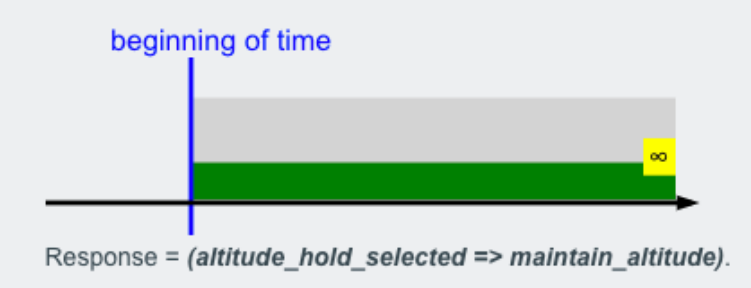
Target: ***altitude_hold_autopilot*** component.

getting to the right requirement

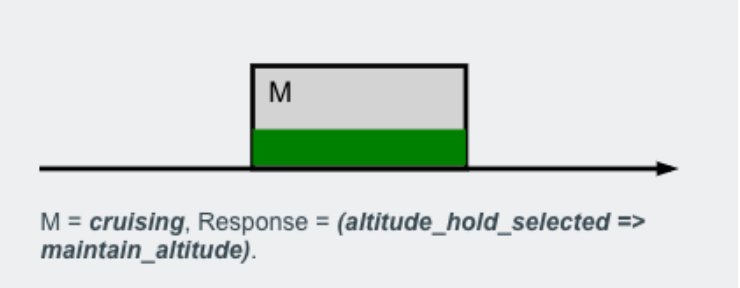
TAKE1: if altitude_hold_selected the altitude_hold_autopilot shall always satisfy maintain_altitude

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
- 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:

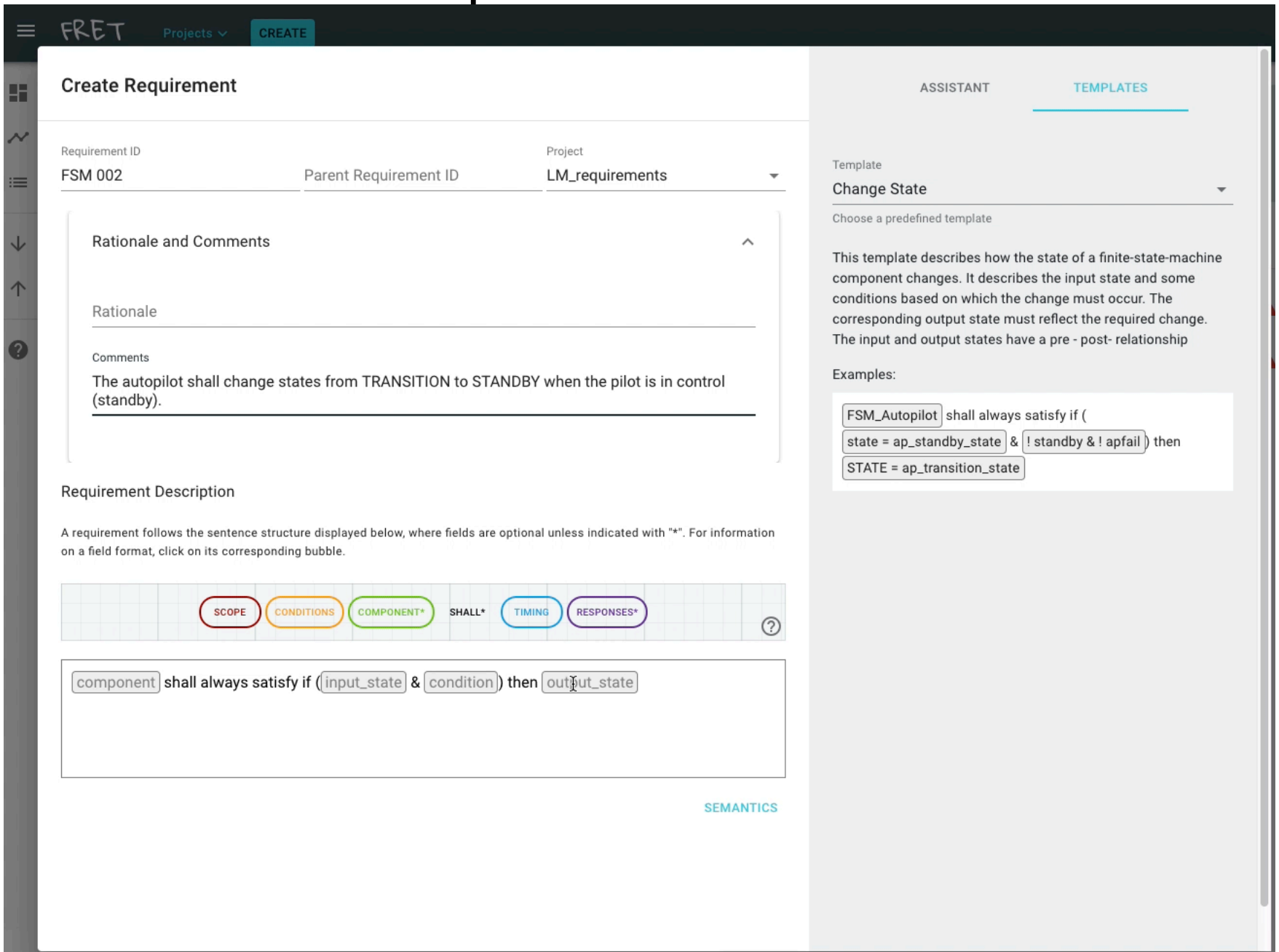
- The autopilot shall change states from TRANSITION to STANDBY when the pilot is in control (standby).
- The autopilot shall change states from TRANSITION to NOMINAL when the system is supported and sensor data is good.
- The autopilot shall change states from NOMINAL to MANEUVER when the sensor data 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



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

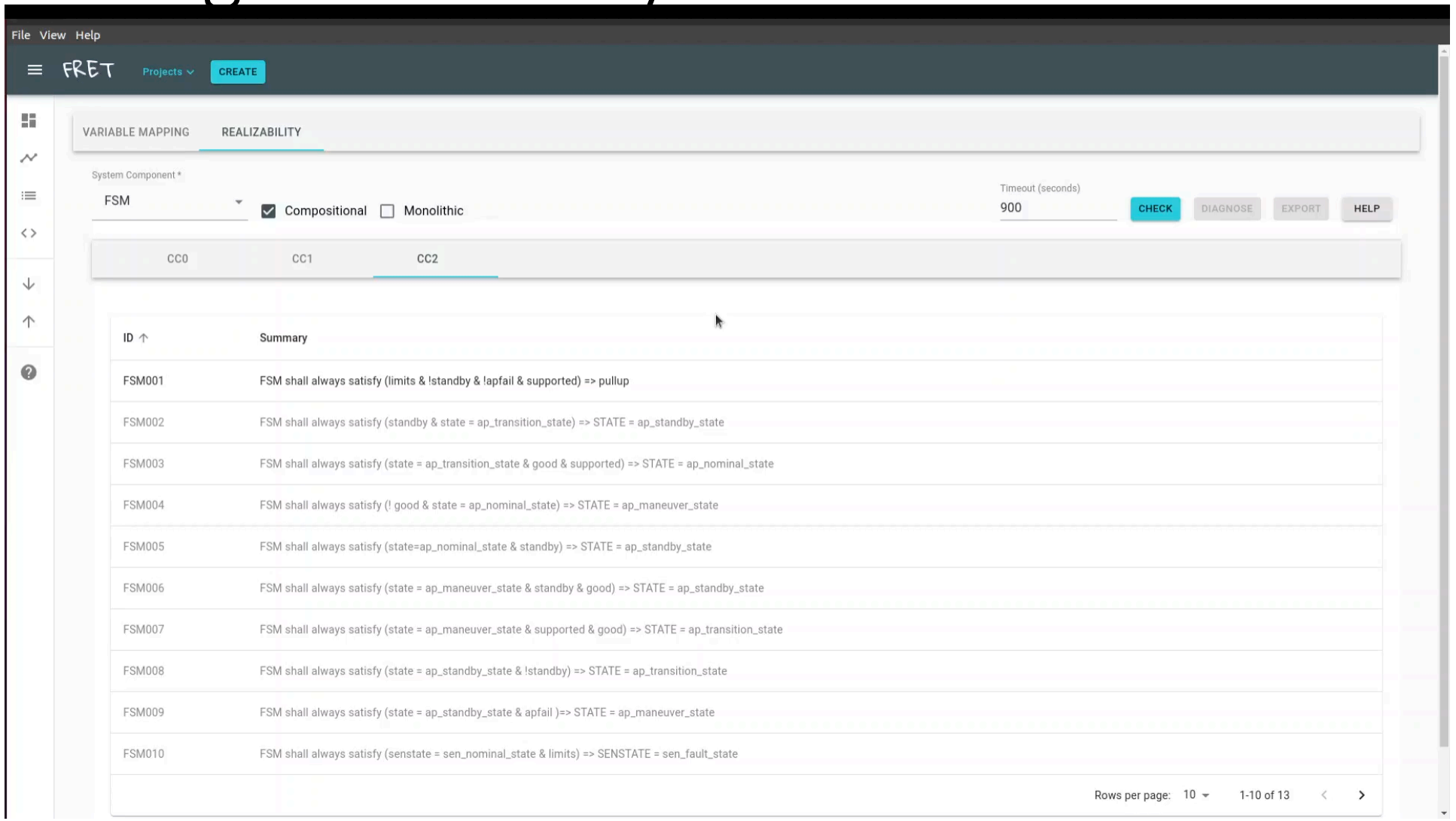
- Unrealizable requirements: Diagnostic analysis
  - Identify minimal sets of unrealizable requirements in specification
  - Counterexamples
  - Simulation of conflicting requirements

- 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

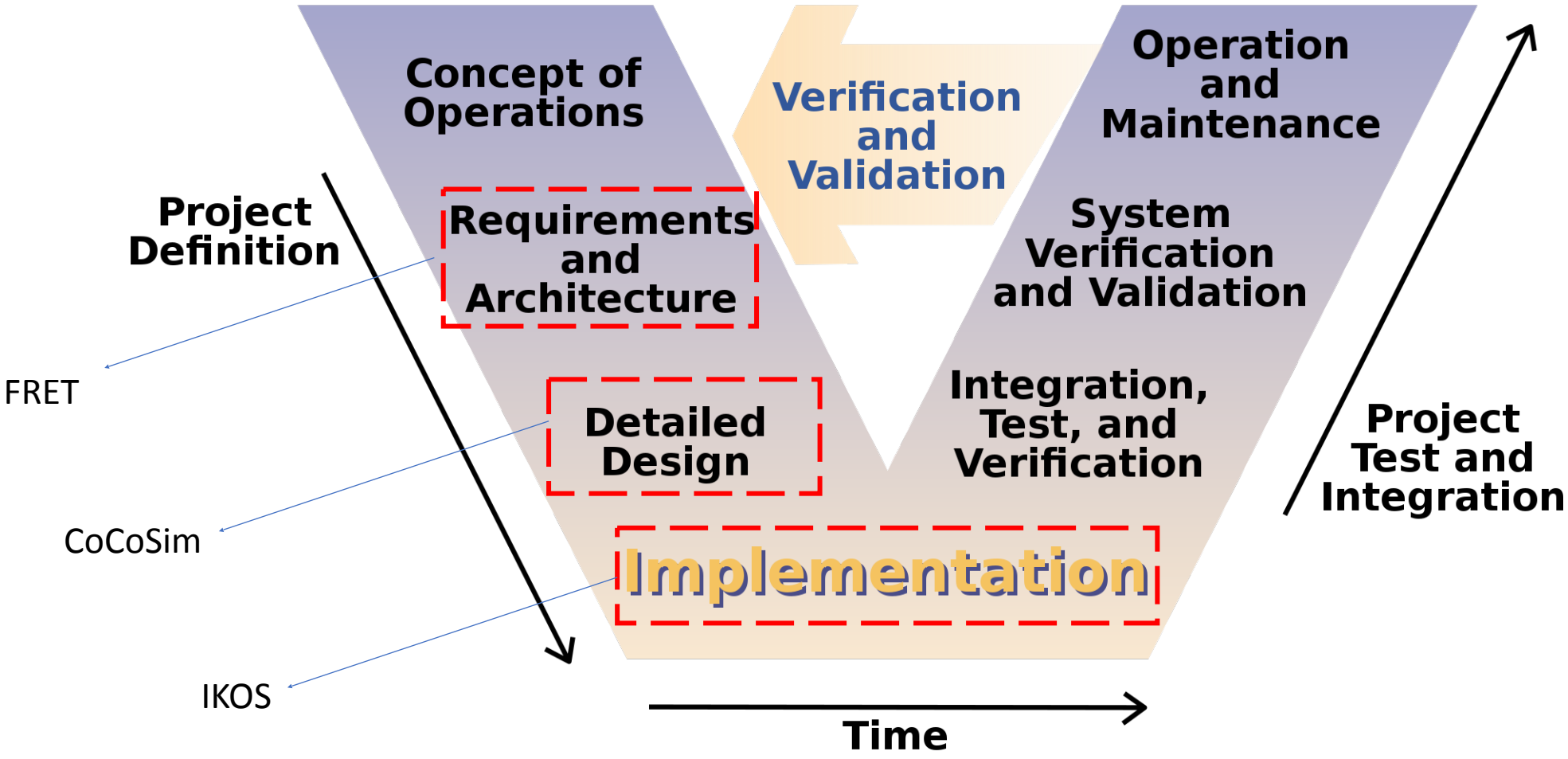
Checking consistency



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



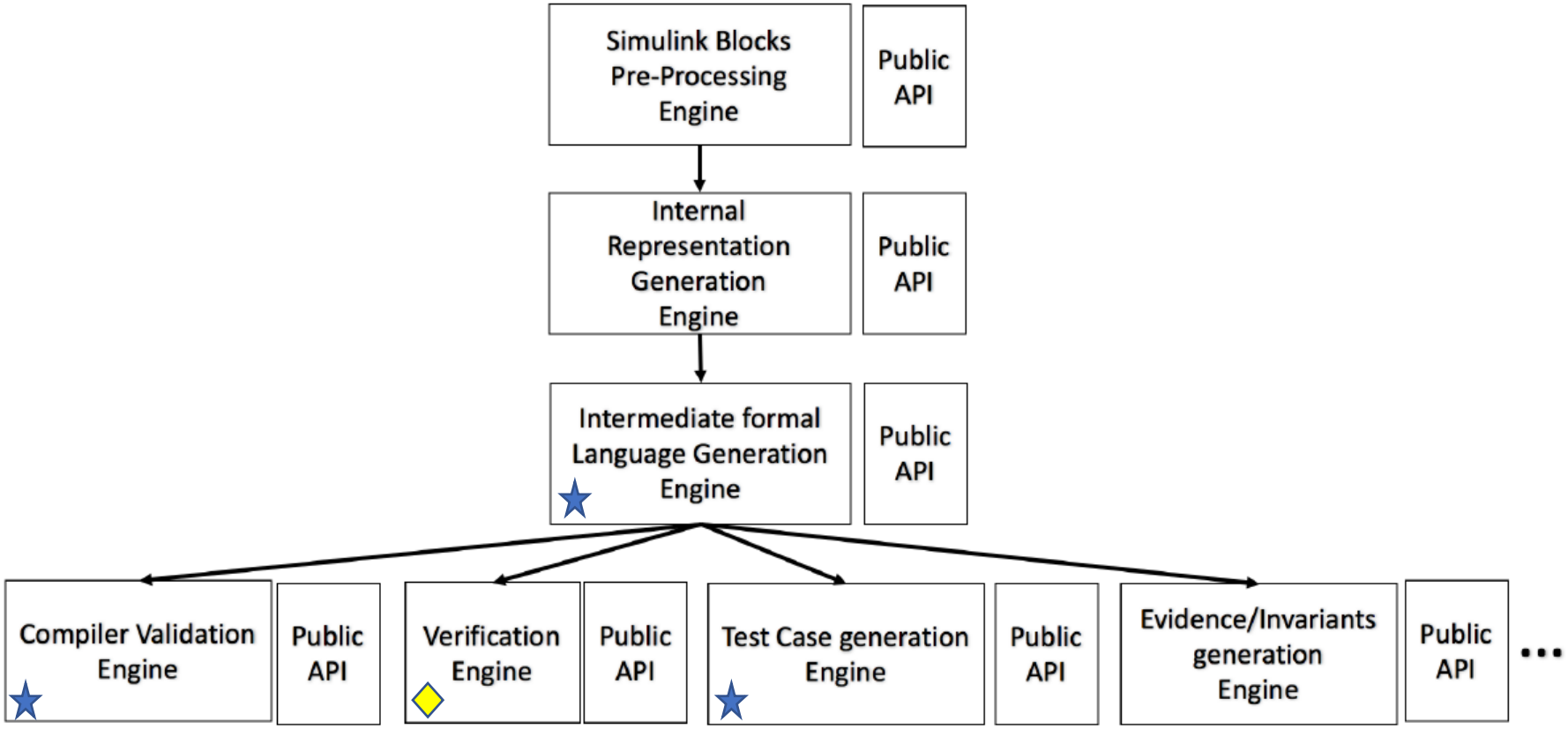
Osborne, Leon F., Jeff Brummond, Robert Hart, Mohsen Zarean, and Steven M. Conger. Clarus: Concept of operations. No. FHWA-JPO-05-072. United States. Federal Highway Administration, 2005.

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

- Bourbouh, Hamza, Marie Farrell, Anastasia Mavridou, Irfan Sljivo, 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 Bourbouh, 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 Bourbouh, 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.
- Bourbouh, Hamza, Pierre-Loïc 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.
- Bourbouh, Hamza, Guillaume Brat, and Pierre-Loïc Garoche. "CoCoSim: an automated analysis framework for Simulink/Stateflow." In *Model Based Space Systems and Software Engineering-European Space Agency Workshop (MBSE 2020)*. 2020.

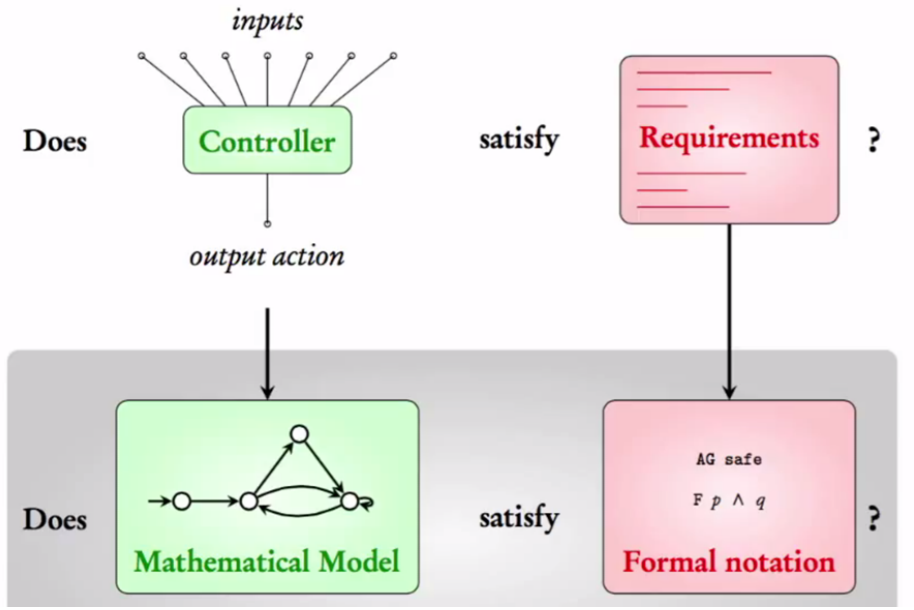
CoCoSim Architecture



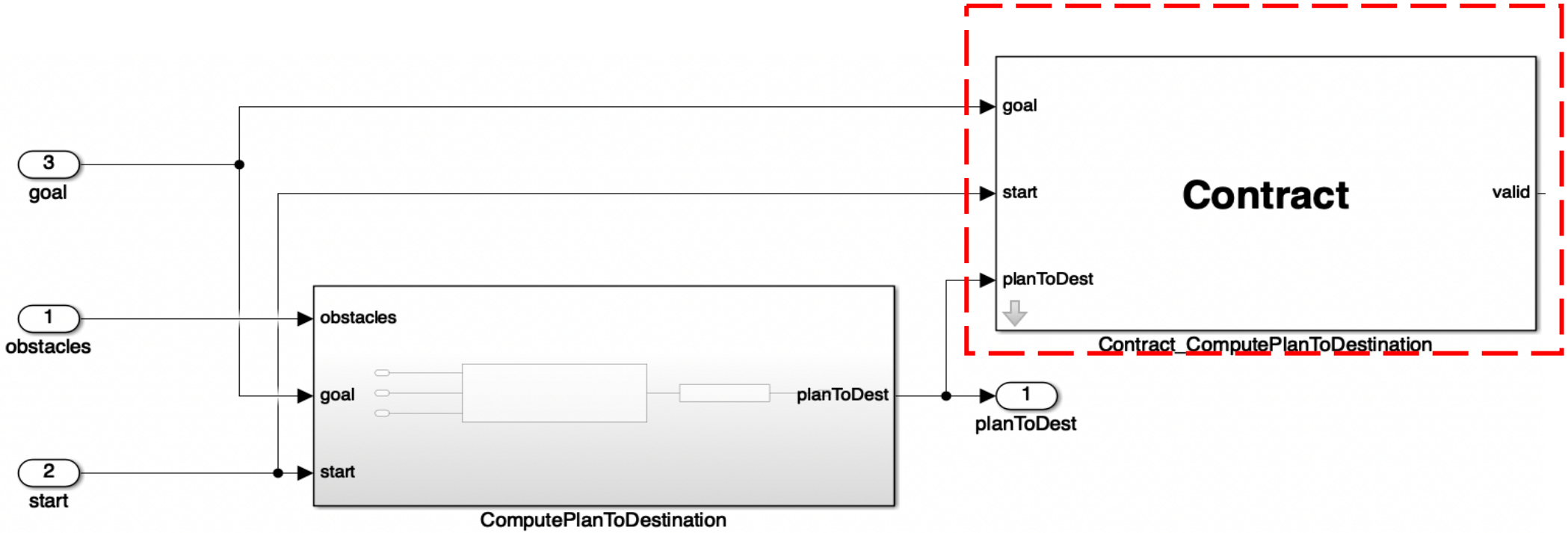
★ LustreC ◇ Kind 2

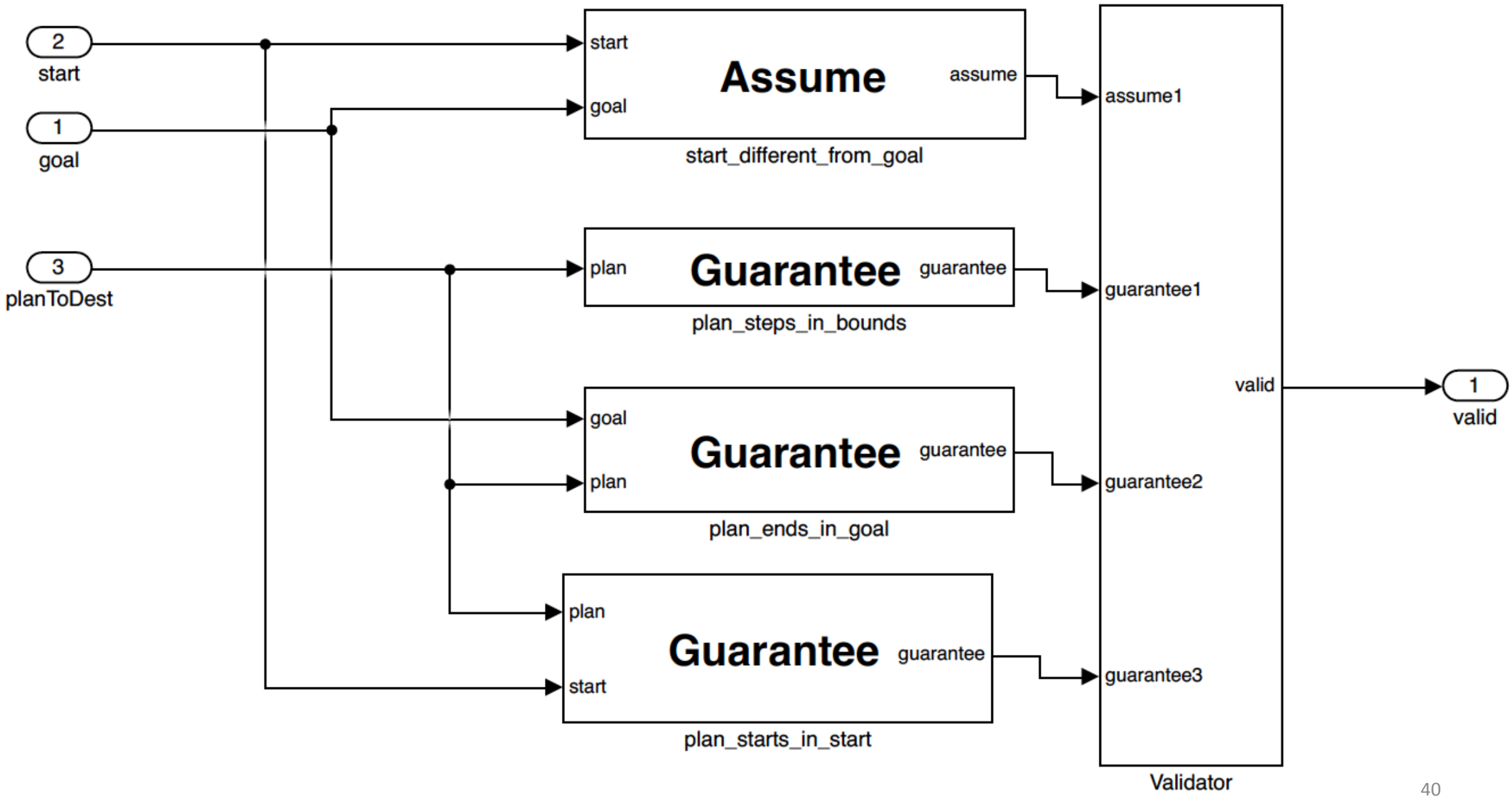
Verification

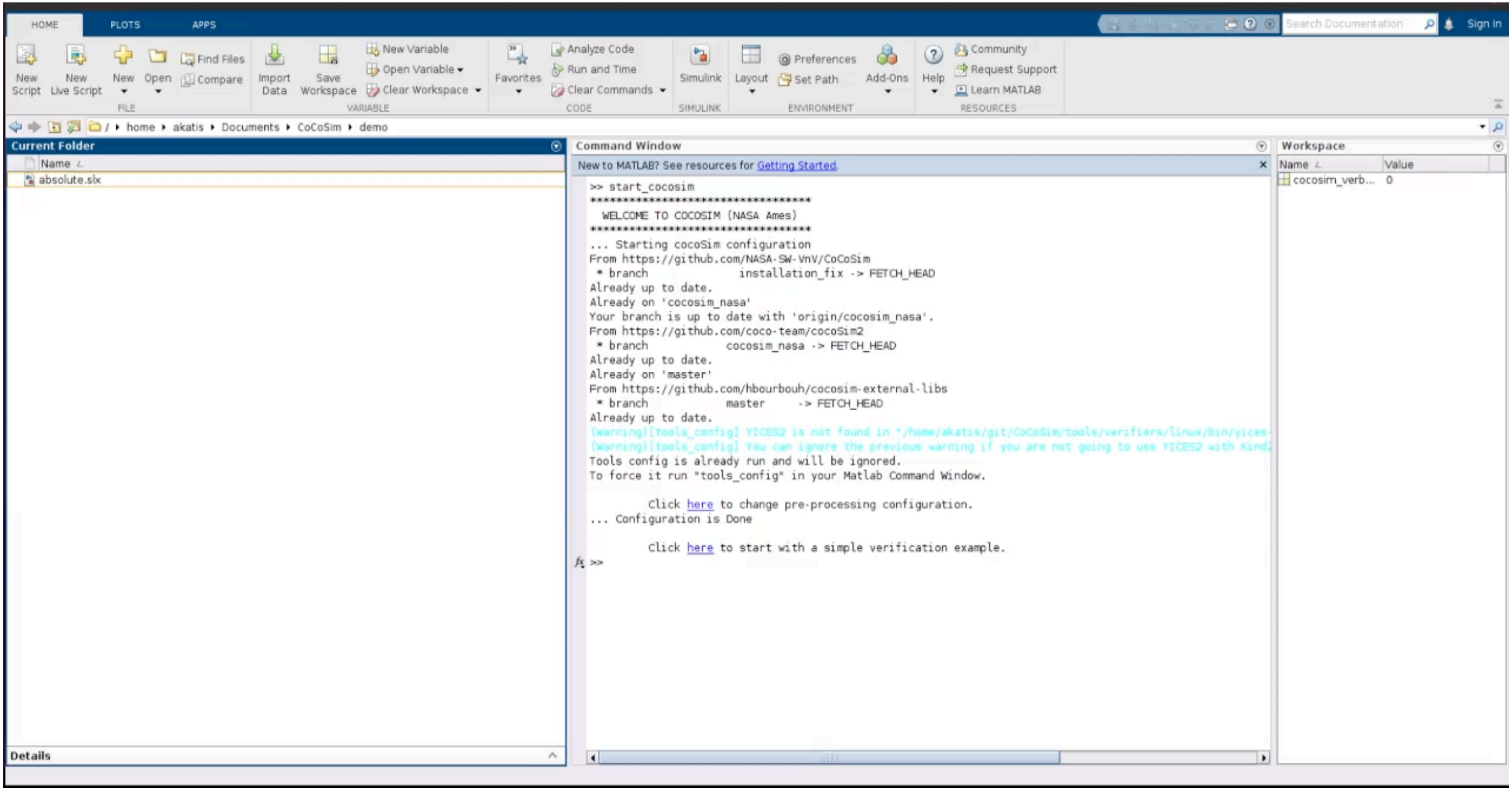
- Formal verification: Prove that a system **always** conforms to our requirements
- By constructing a mathematical proof!
- Model checking



Requirements are expressed using the CoCoSim library.




2
start
1
goal
3
planToDest
start
goal
Assume
assume
start_different_from_goal
plan
Guarantee
guarantee
plan_steps_in_bounds
goal
plan
Guarantee
guarantee
plan_ends_in_goal
plan
start
Guarantee
guarantee
plan_starts_in_start
assume1
guarantee1
guarantee2
guarantee3
valid
Validator
1
valid


HOME
PLOTS
APPS
Search Documentation
Sign In
New Script
New Live Script
New
Open
Find Files
Compare
FILE
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
Favorites
Analyze Code
Run and Time
Clear Commands
CODE
Simulink
SIMULINK
Layout
Preferences
Set Path
Add-Ons
ENVIRONMENT
Help
Community
Request Support
Learn MATLAB
RESOURCES
/ ▸ home ▸ akatis ▸ Documents ▸ CoCoSim ▸ demo
Current Folder
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 Kind2
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.
>>
Workspace
Name
Value
cocosim_verb...
0

Compositional Verification

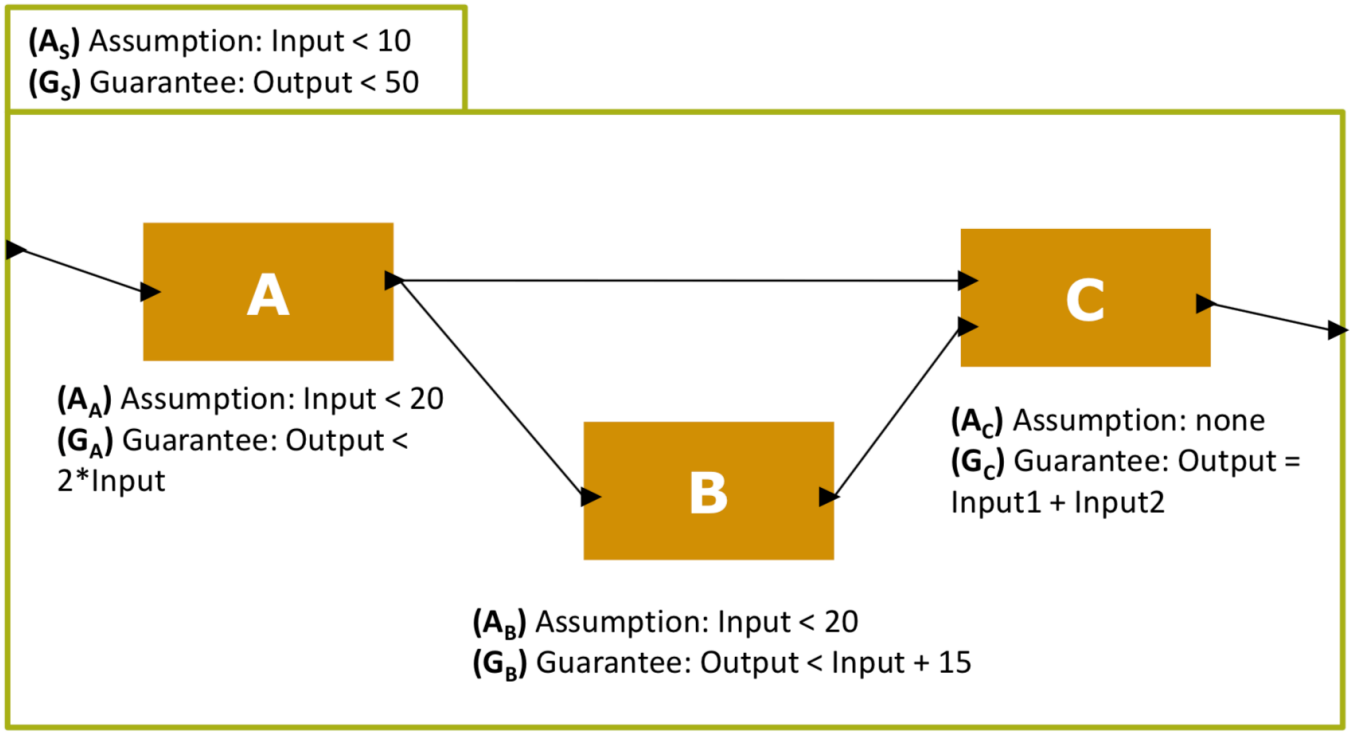
Example (to prove)

$A_S \rightarrow A_A$

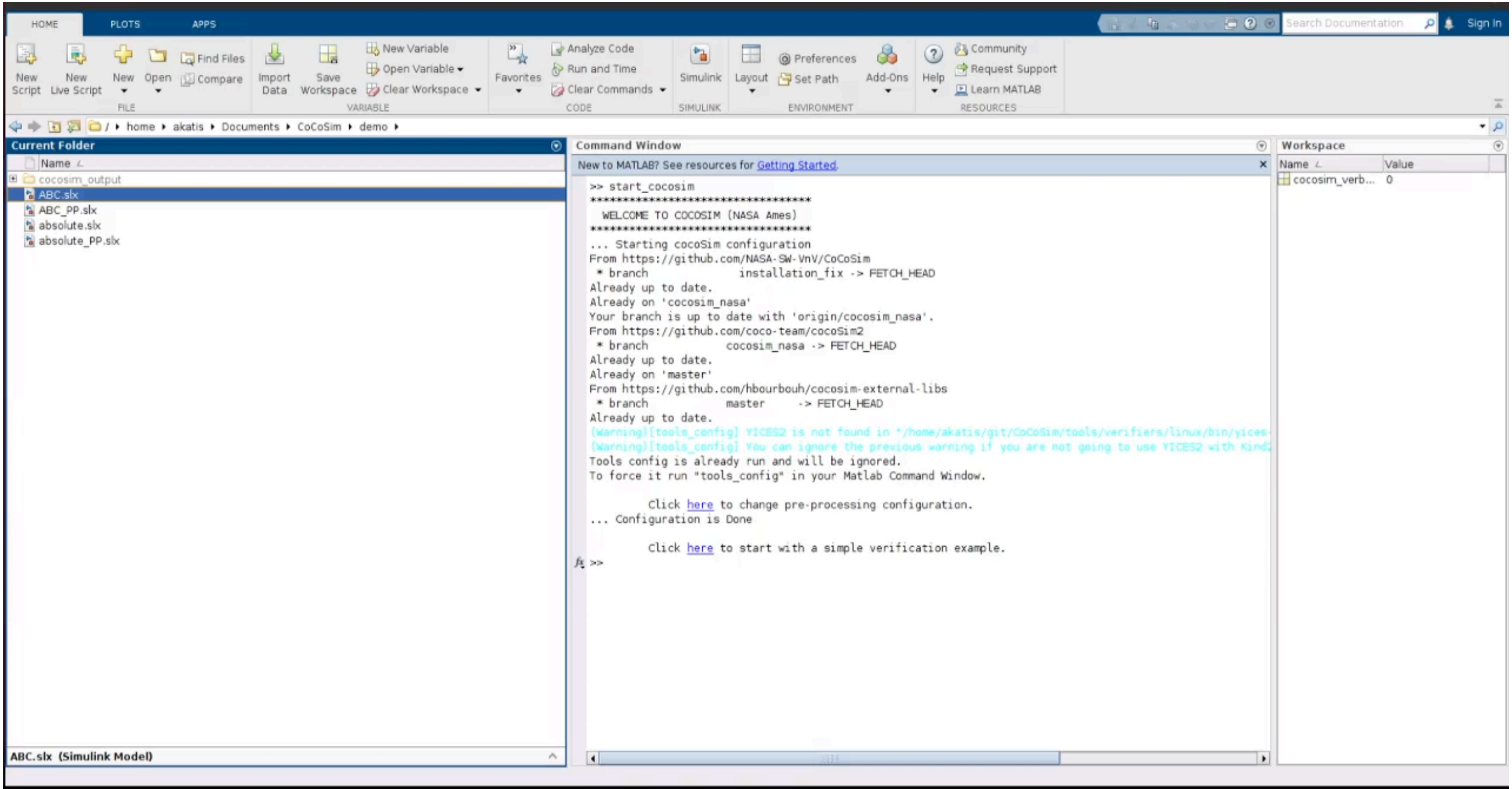
$A_S \wedge G_A \rightarrow A_B$

$A_S \wedge G_A \wedge G_B \rightarrow A_C$

$A_S \wedge G_A \wedge G_B \wedge G_C \rightarrow G_S$



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


HOME
PLOTS
APPS
Search Documentation
Sign In
New Script
New Live Script
New
Open
Find Files
Compare
FILE
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
Favorites
Analyze Code
Run and Time
Clear Commands
CODE
Simulink
SIMULINK
Layout
Preferences
Set Path
ENVIRONMENT
Add-Ons
Help
Community
Request Support
Learn MATLAB
RESOURCES
/ ▸ 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 Kind2
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.
>>
Workspace
Name
Value
cocosim_verb...
0

Compositional Verification

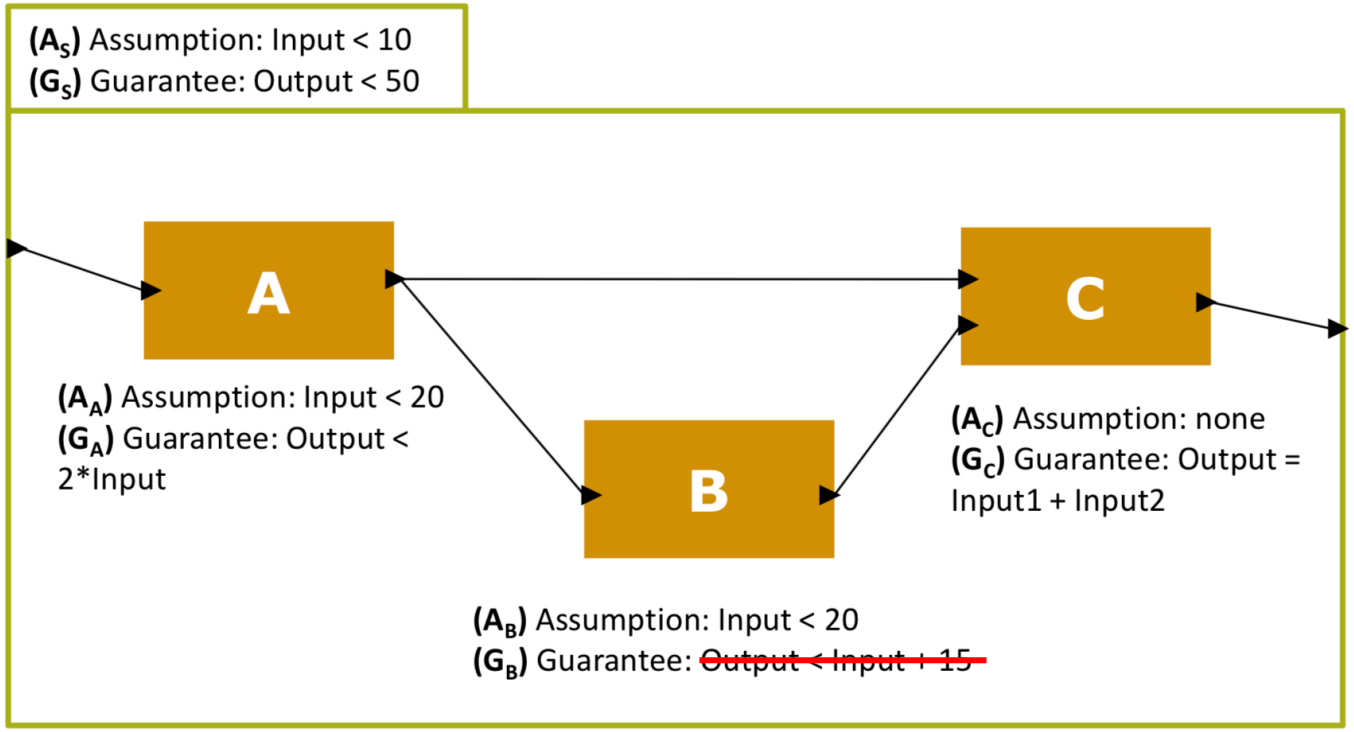
Example (to prove)

$A_S \rightarrow A_A$

$A_S \wedge G_A \rightarrow A_B$

$A_S \wedge G_A \wedge G_B \rightarrow A_C$

$A_S \wedge G_A \wedge G_B \wedge G_C \rightarrow G_S$



$$G_B = Output < Input + 50$$

Verification Summary

ABC_PP/S/C__abstracted/C

Result
✓

ABC_PP/S/B__abstracted/B

Result
✓

ABC_PP/S/A__abstracted/A

Result
✗

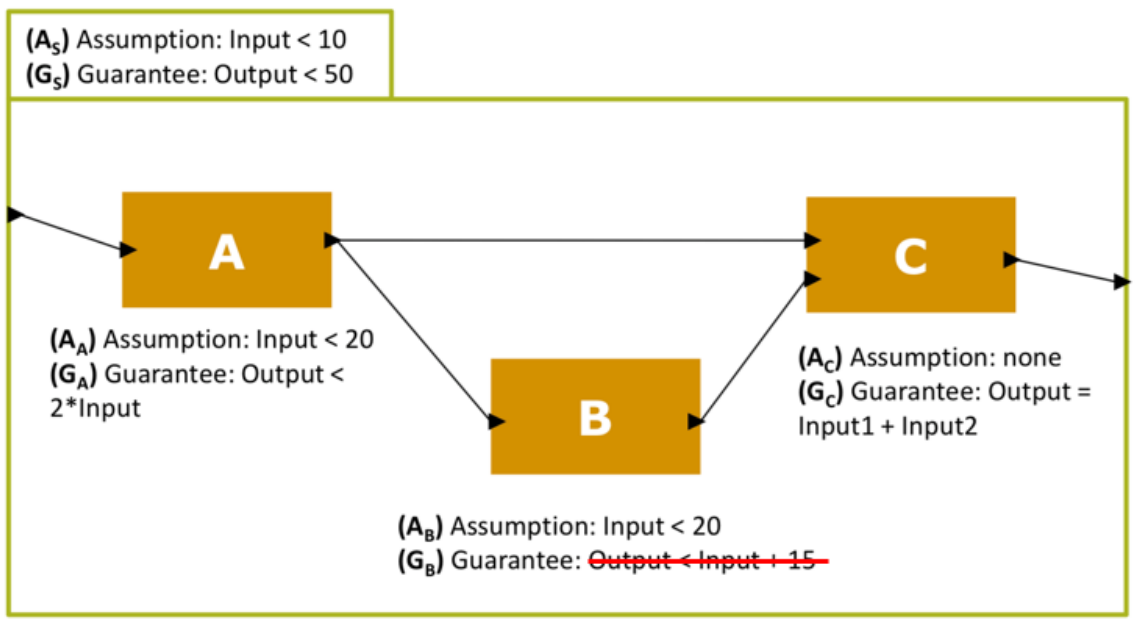
ABC_PP/S

Abstract C	Abstract B	Abstract A	Abstract C	Result
✓	✓	✓	✓	✗
✓	×	✓	✓	✓

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

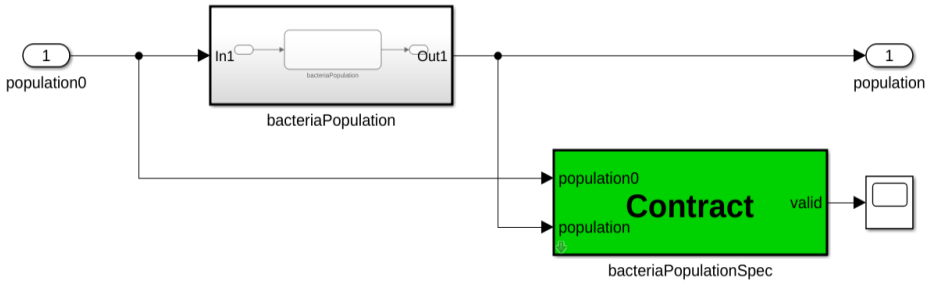
ABC_PP/S

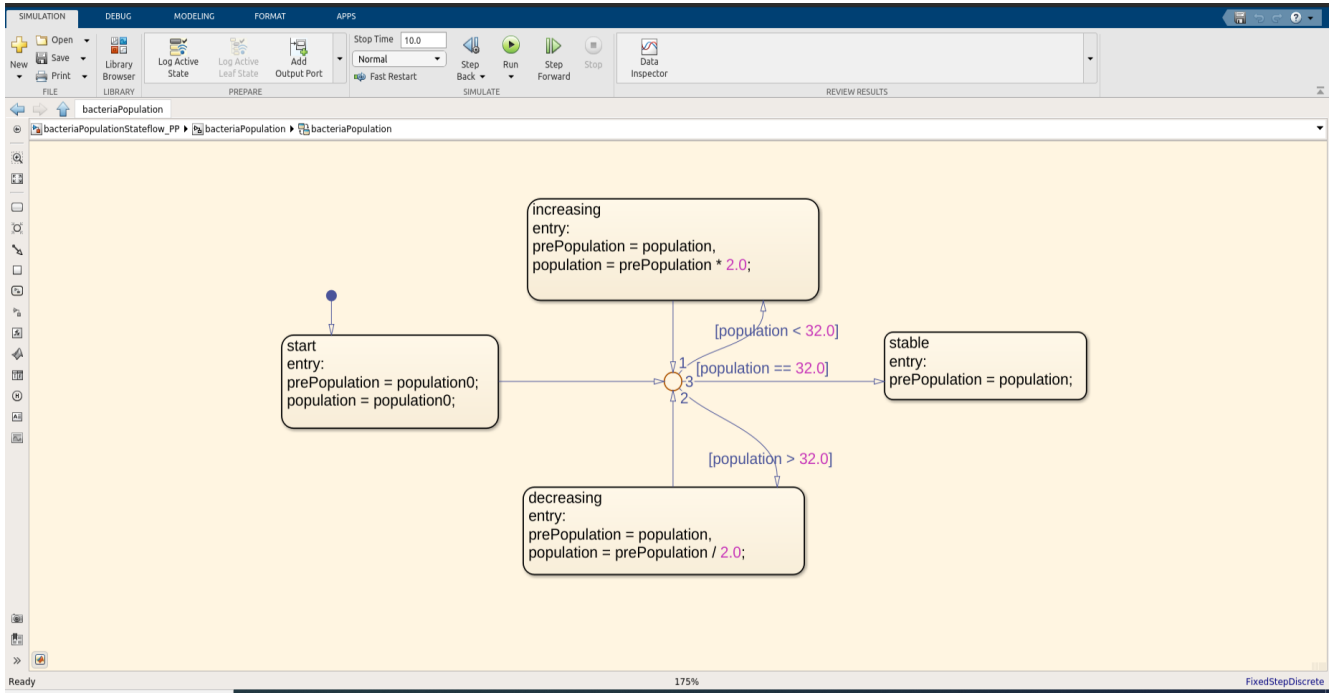
Time	0
Input (input)	9
Output (output)	50
C__abstracted (local)	50
A__abstracted (local)	17
B__abstracted (local)	33



$$G_B = Output < Input + 50$$

Stateflow verification

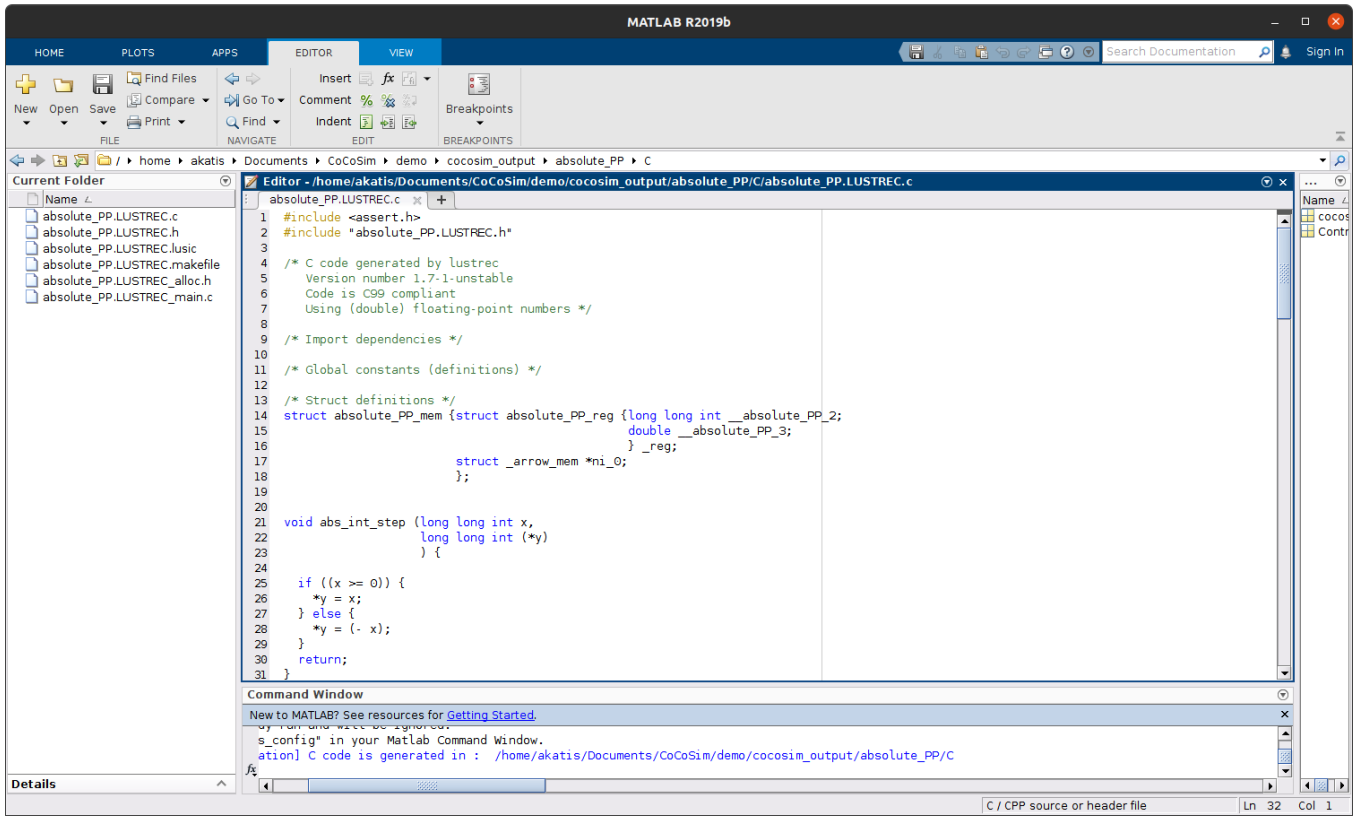




Bourbouh, Hamza, Pierre-Loic Garoche, Christophe Garion, Arie Gurfinkel, Temesghen Kahsai, and Xavier Thirioux. "Automated analysis of Stateflow models." (2017): 144-161.

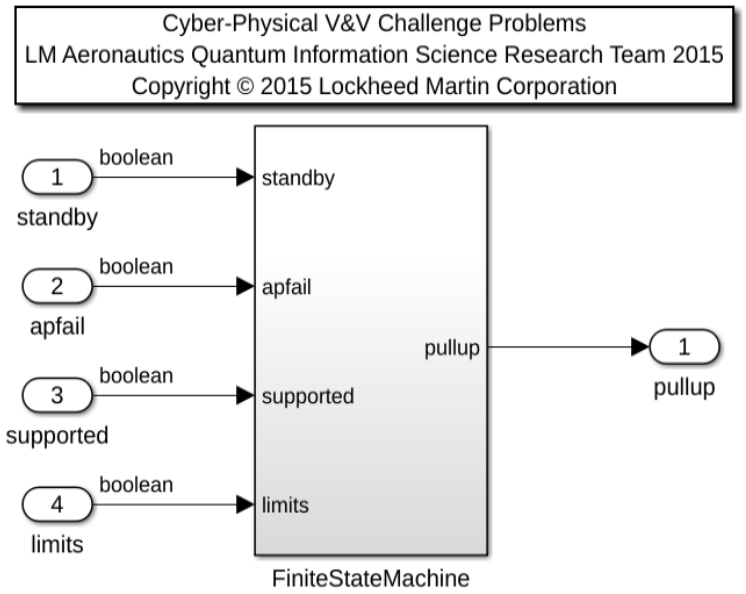
Code Generation

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



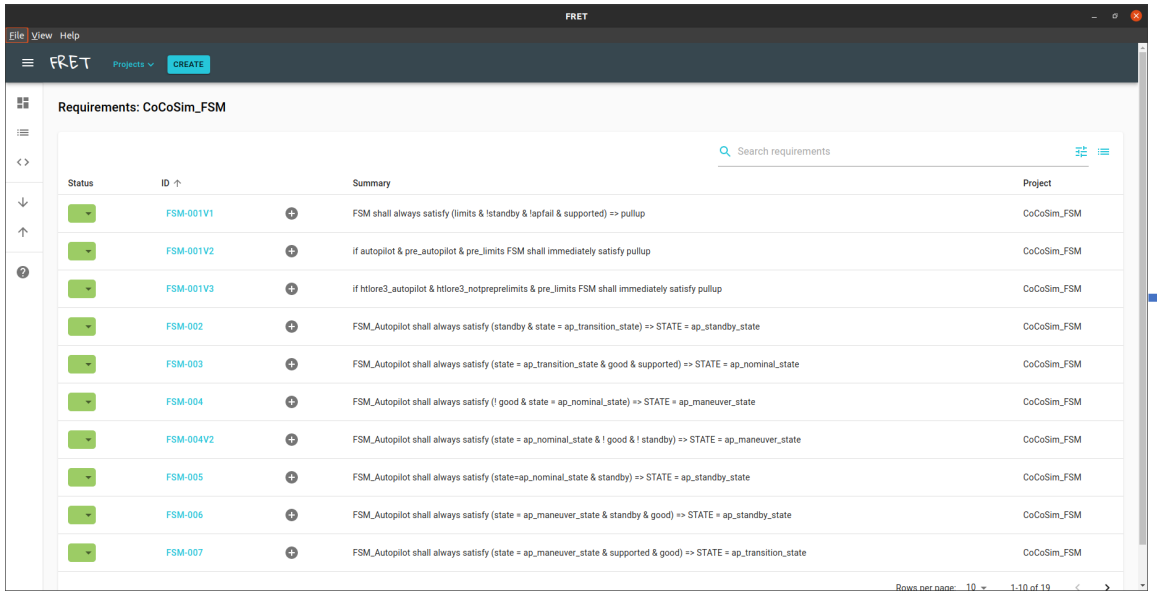
Connection with FRET

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

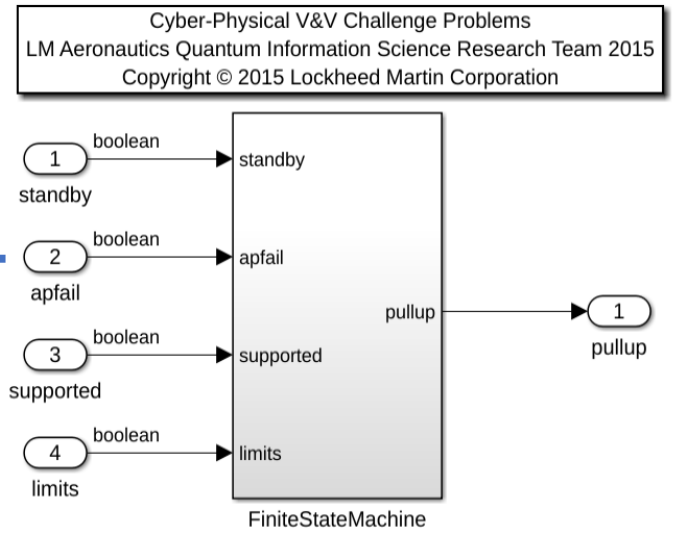


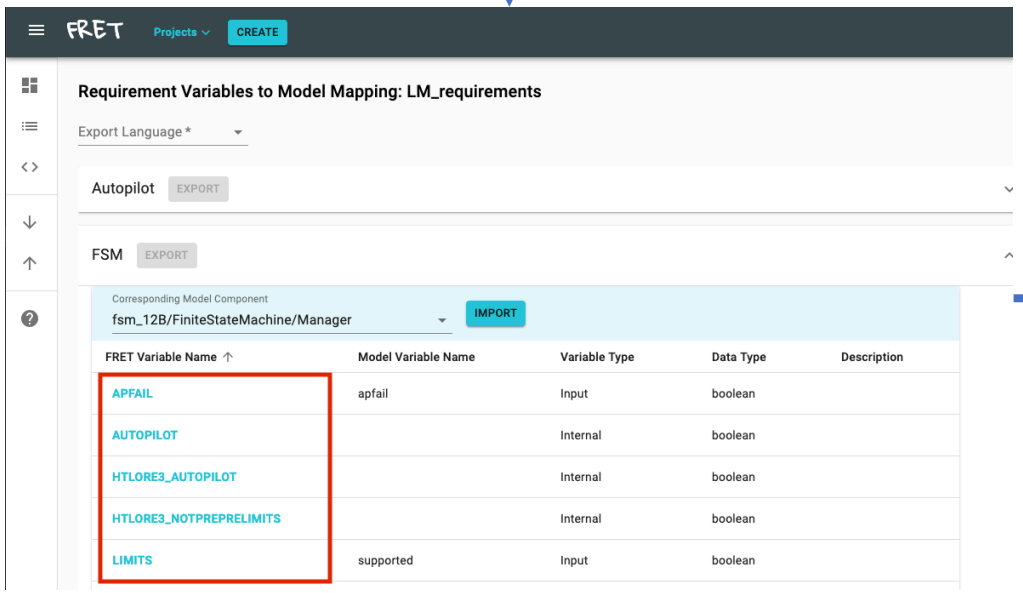
Mavridou, A., Bourbouh, H., Garoche, P.L., Giannakopoulou, D., Pressburger, T. and Schumann, J., 2020, March. Bridging the gap between requirements and Simulink model analysis. In Joint 26th International Conference on Requirements Engineering: Foundation for Software Quality Workshops, Doctoral Symposium, Live Studies Track, and Poster Track.

FRET Requirements

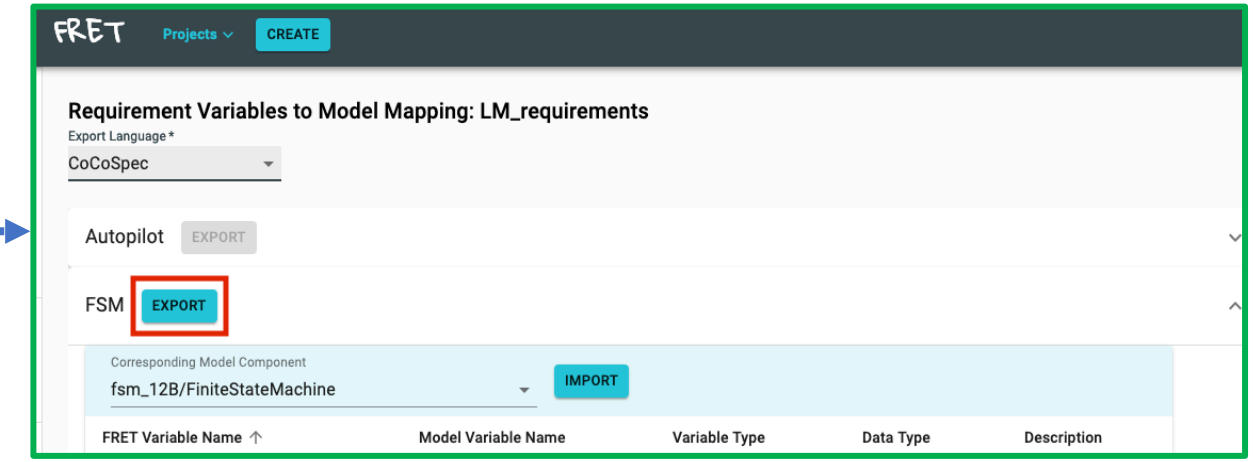


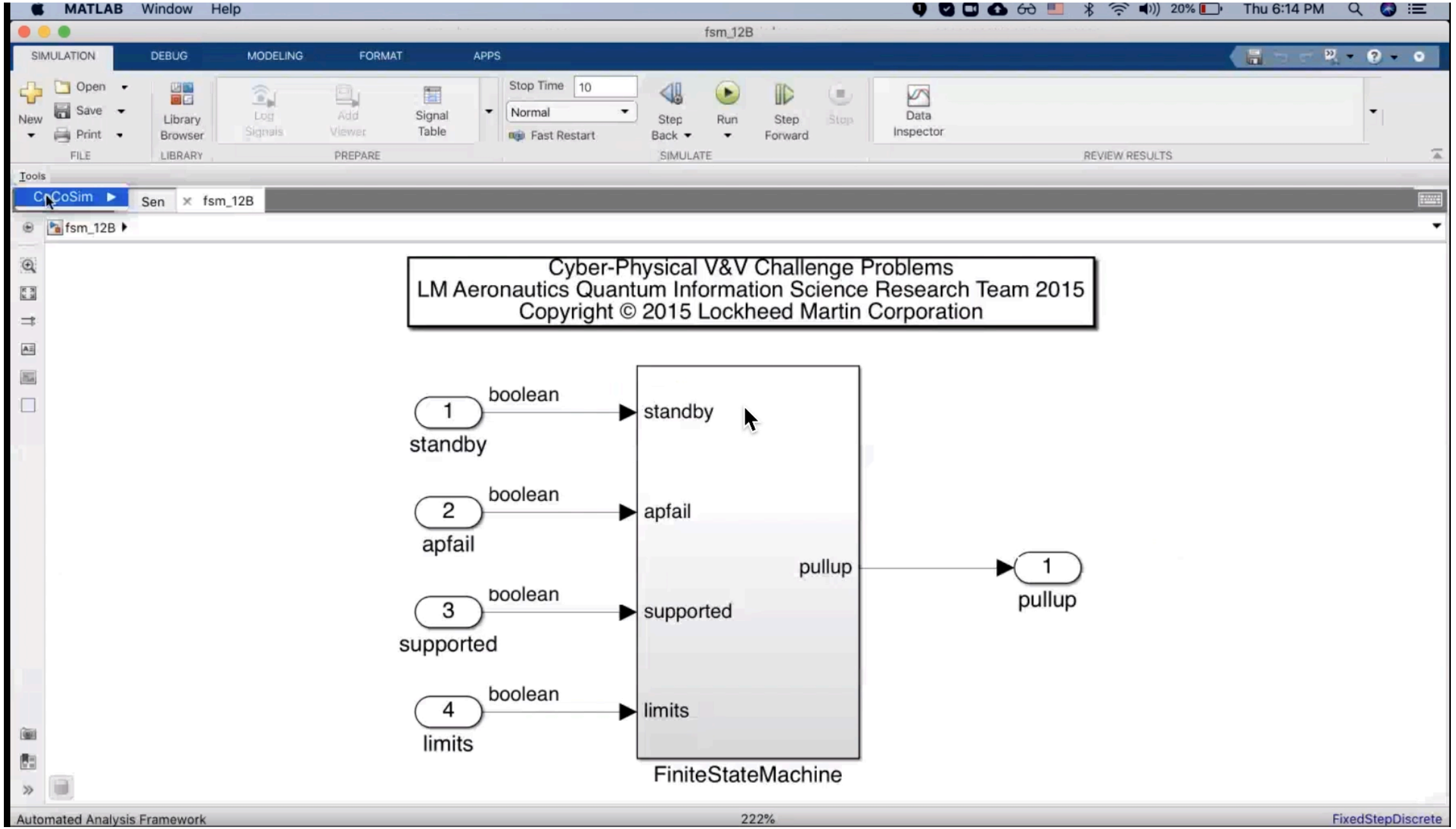
Model Information (JSON)





Export CoCoSpec (Lustre) + Traceability data

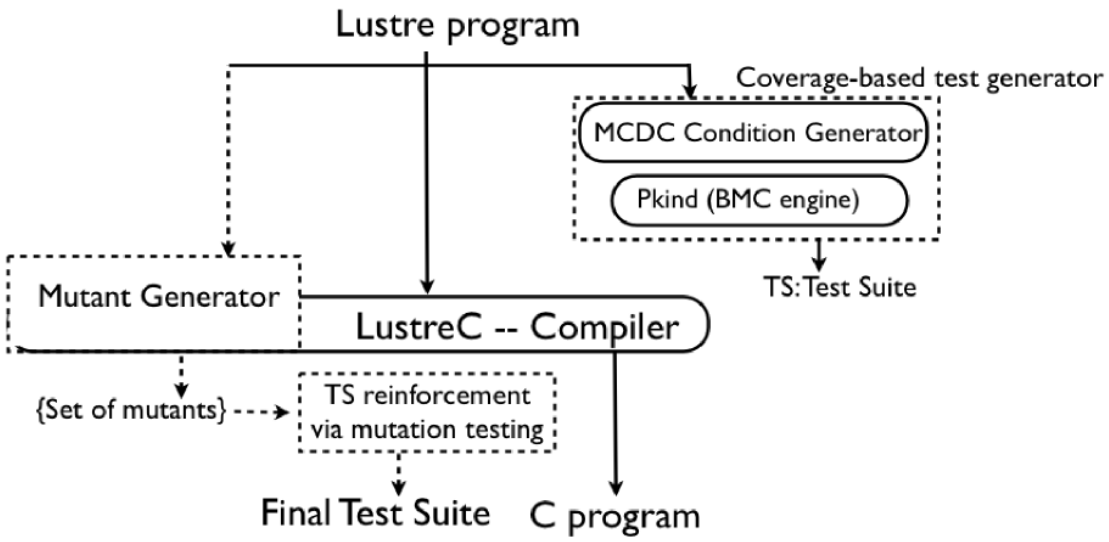


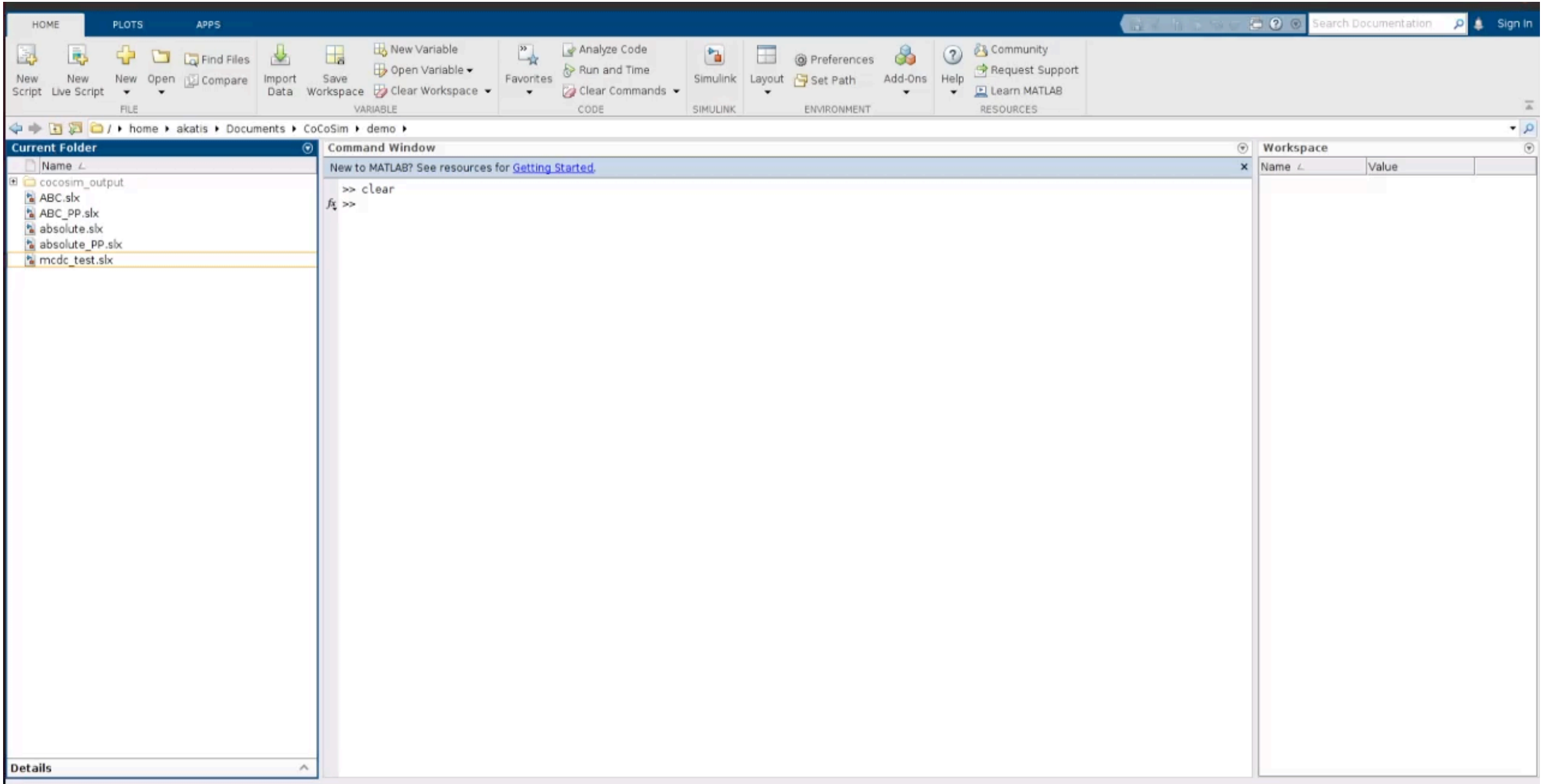

MATLAB Window Help
Thu 6:14 PM
fsm_12B
SIMULATION
DEBUG
MODELING
FORMAT
APPS
New
Open
Save
Print
Library Browser
Log Signals
Add Viewer
Signal Table
Stop Time
10
Normal
Fast Restart
Step Back
Run
Step Forward
Stop
Data Inspector
FILE
LIBRARY
PREPARE
SIMULATE
REVIEW RESULTS
Tools
CoSim
Sen
fsm_12B
Cyber-Physical V&V Challenge Problems
LM Aeronautics Quantum Information Science Research Team 2015
Copyright © 2015 Lockheed Martin Corporation
1
standby
boolean
2
apfail
boolean
3
supported
boolean
4
limits
boolean
standby
apfail
supported
limits
pullup
1
pullup
FiniteStateMachine
Automated Analysis Framework
222%
FixedStepDiscrete

Test-case generation

- Random test generation (Simulink)
- MC/DC
- Mutation-based

} via LustreC (Work in Progress)




HOME
PLOTS
APPS
Search Documentation
Sign In
New Script
New Live Script
New
Open
Find Files
Compare
FILE
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
VARIABLE
Favorites
Analyze Code
Run and Time
Clear Commands
CODE
Simulink
SIMULINK
Layout
Preferences
Set Path
Add-Ons
ENVIRONMENT
Help
Community
Request Support
Learn MATLAB
RESOURCES
/ ▸ home ▸ akatis ▸ Documents ▸ CoCoSim ▸ demo ▸
Current Folder
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
>>
Workspace
Name
Value

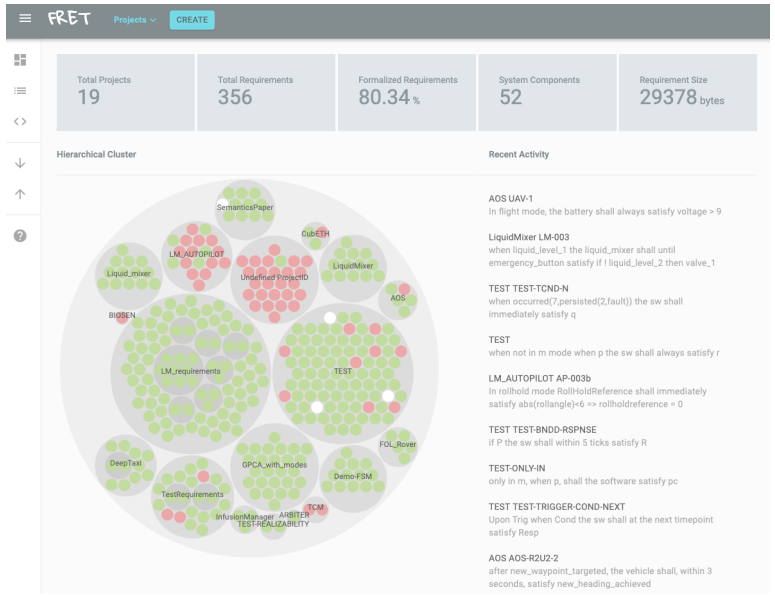
Library	# supp. Blocks	% supp. Blocks	Unsupported blocks
Discontinuities	11/12	91%	Backlash
Discrete	19/21	90%	Discrete PID Controller, Discrete PID Controller (2DOF)
Logic & Bit Operations.	18/19	95%	Extract Bits
Lookup Tables.	9/9	100%	
Math Operations.	31/37	83%	Algebraic Constraint, Complex to Magnitude-Angle, 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 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)