# CoCoSim: Hamza Bourbouh

## Tentative web input:

CoCoSim is an automated analysis tool for verifying and validating Simulink and Stateflow models. CoCoSim can be used to automatically perform test case generation and to evaluate the validity of user-supplied safety requirements. CoCoSim can also be used to generate C and/or Rust code from Simulink or Stateflow models.

CoCoSim refers to **Co**ntract-based **Co**mpositional verification of **Sim**ulink models. This tool was initially developed through a collaboration between NASA, CMU and IOWA University and is currently supported by NASA Ames Research Center.

The main use of CoCoSim is to provide verification and validation of user-supplied Simulink or Stateflow programs. CoCoSim is structured to operate much like a compiler, sequencing a series of translation steps leading, eventually to either the production of source code, or to the call of a verification tool. CoCoSim is a toolbox that can be called directly from the Matlab Simulink environment using the following commands:

$cd "PATH\\_TO\\_CoCoSim/scripts"

$./install\\_cocosim

This downloads, compiles and installs in a local folder the external tools required by

CoCoSim; and then it downloads from a public repositories the standard library routines needed by CoCoSim. The default libraries are come from two sources: one defined by CMU and one developed within NASA. Both libraries are complimentary and perform different preprocessing. The libraries contain the solvers (model checkers) which need to be downloaded and installed on computer.

CoCoSim allows users to express safety properties as synchronous observers within the Simulink model or StateFlow Diagram. Synchronous observers are code segments that implement assume-guarantee reasoning by taking state variables as input and then producing one or more Boolean outputs. Although this may sound complicated, synchronous observers are simply block diagrams that compare inputs to generate a logic True or False answer. As an example, Fig. 1 shows the addition of two synchronous observers to a Simulink model of an actuator controller. The observers produce Boolean outputs that are either True or False depending on the outcome of the sampled inputs. The logic within the observer blocks represents the safety properties as combinations of state inputs that should always produce a True or False result.

CoCoSim includes a software tool to ease the process of expressing safety properties as synchronous observers. After starting CoCoSim and opening model, the user simply selects the option “Create Property” from the graphical user interface to introduce a synchronous observer specification block within the target subsystem. The user selects

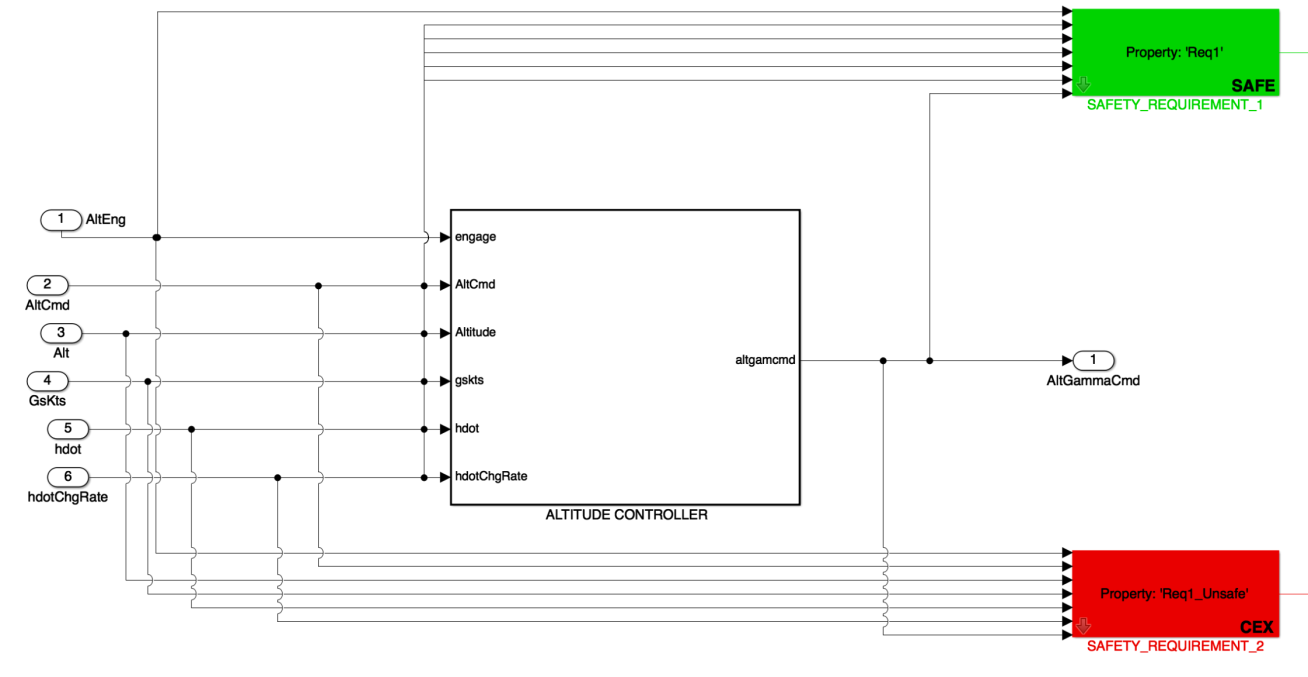


Figure 1: Addition of synchronous observers to actuator controller Simulink Model.

the various inputs to be used, enters the desired logic comparisons, and creates a unique name for the Boolean output.

Formalizing requirements as synchronous observers is a powerful approach to support the verification and validation of reactive systems. The specification of requirements as model elements requires a minimal amount of work but produces numerous benefits. Itsupports existing test activities but also enables the use of formal methods. Each formalized requirement acts as a test oracle. The synchronous observer defines

a predicate. Therefore its Boolean output corresponds to the validity of the expressed

requirement. CoCoSim uses formal model checkers to verify the assertions poised by the synchronous observers.

Figure 2 illustrates the types of CoCoSim analysis possible. The process begins with the user input of a Simulink Model or StateFlow diagram that has been modified to include the synchronous observer blocks describing the safety properties. CoCoSim first performs pre-processing of the Simulink model or Stateflow diagram to create a new version in which complex Simulink blocks have been replaced with a combination of more basic blocks, but in which the Simulink semantics have been preserved. The pre-processed model is the output of the Front End in Fig. 2. This model is then used as input for the Compiler (middle block of Fig. 2) which converts the pre-processed models into either Lustre, C, or Rust code. The compilation from Simulink to these codes represents a hierarchical block-by-block compilation.

The resulting code can be used as input to the third part of CoCoSim which comprises the Solvers that are used to verify the safety properties. The Solvers currently include Zustre, Kind2, Jkind, and LustreC. Zustre and Kind2 are bounded model checkers that

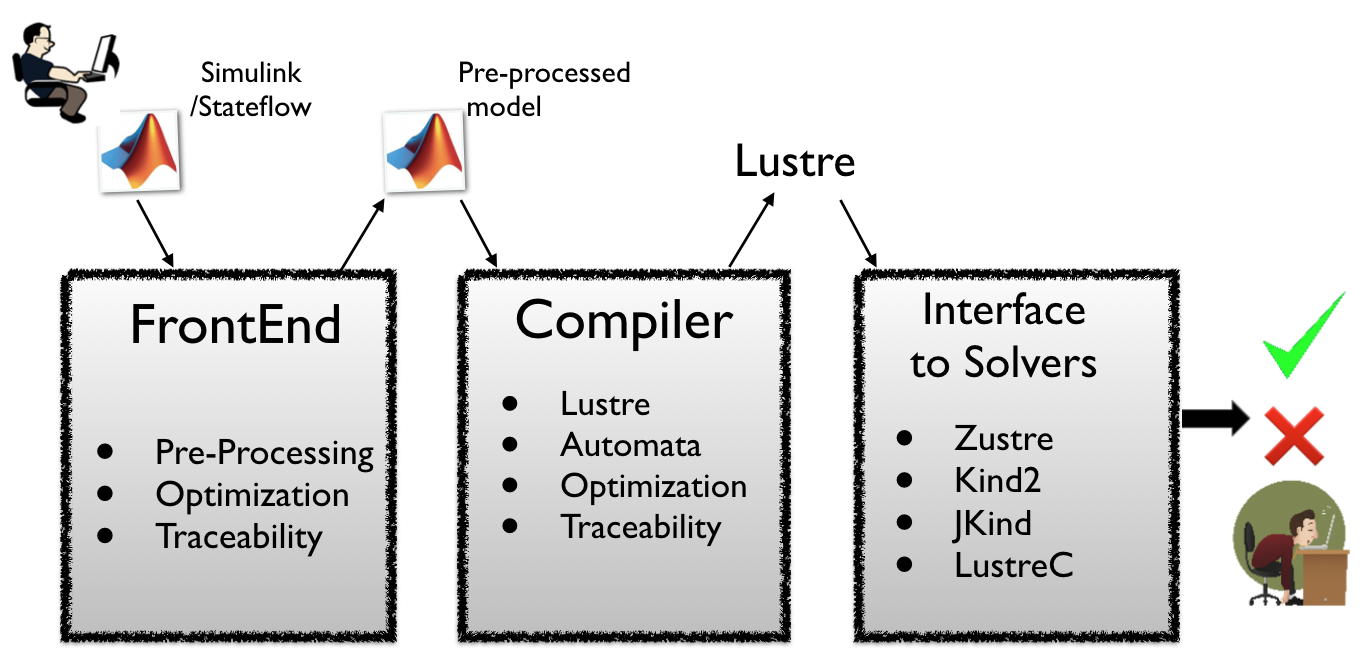


Figure 2: CoCoSim analysis process diagram.

are called by CoCoSim to take the Lustre code as input and then perform verification of the safety properties expressed as the synchronous observers. The result of the safety analysis is reported back to the Simulink environment. In case the property supplied is falsified, CoCoSim provides means to simulate the counterexample trace in the Matlab environment. Zustre and Kind2 run on Unix operating systems. They can be used to provide input to the Evidence/Invariants Generation Engine. Traceability is maintained between Simulink, Lustre, and the test case results so that error sources can be traced. (JKind provides similar analysis capability but only runs on the Java operating system.) Using the LustreC tool, CoCoSim may also be used to validate the compiler outputs or to perform test case generation. Figure 3 illustrates the analysis pathways in CoCoSim. In that diagram, the blocks labeled “Public API” indicate software libraries that are populated with user supplied tools as well as the CoCoSim supplied libraries.

CoCoSim offers many of the same features offered by the Mathworks Simulink Design Verifier (SDV) software, including a graphical user interface, but also has some additional benefits. For one thing, CoCoSim is open-source, free software (that is customizable by the user) and offers almost the same features as SDV. Unlike SDV, CoCoSim supports all types of Simulink blocks including those with nonlinearities. Additionally, whereas SDV is limited to smaller code segments, CoCoSim can handle really big programs. SDV does not disclose what model-checking techniques are used for verification, but CoCoSim offers Zustre, Kind2, and JKind bounded model checkers and the documentation on how to use them. CoCoSim allows users to add other model-checkers through the API. A very important feature provided by CoCoSim, that SDV does not offer, is compositional verification. In one study, compositional verification using Kind2 was able to prove that more than 90% of the code was correct, compared to 0% for SDV. CoCoSim also offers test-case generation to support evidence for MC/DC coverage and proof that requirements are correct.

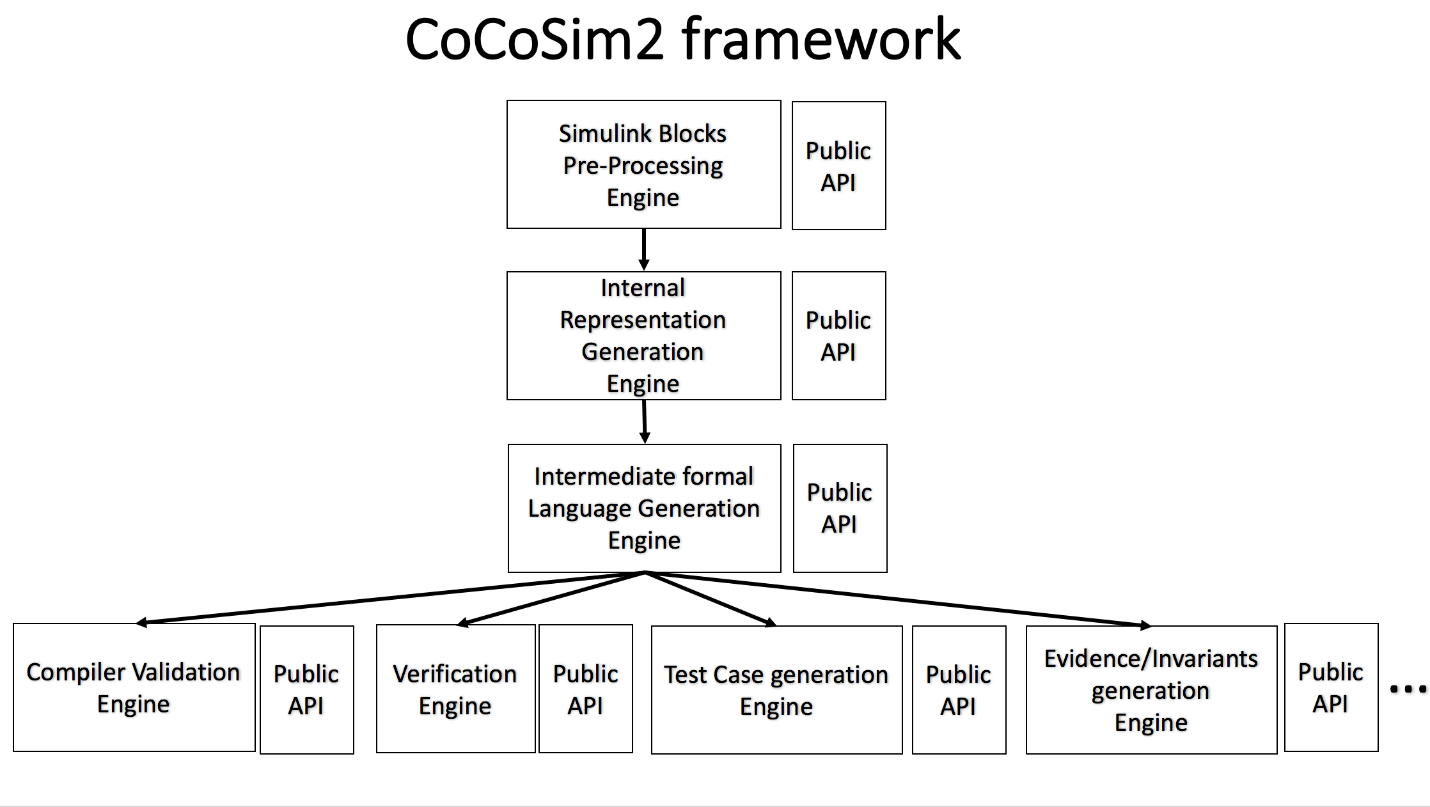


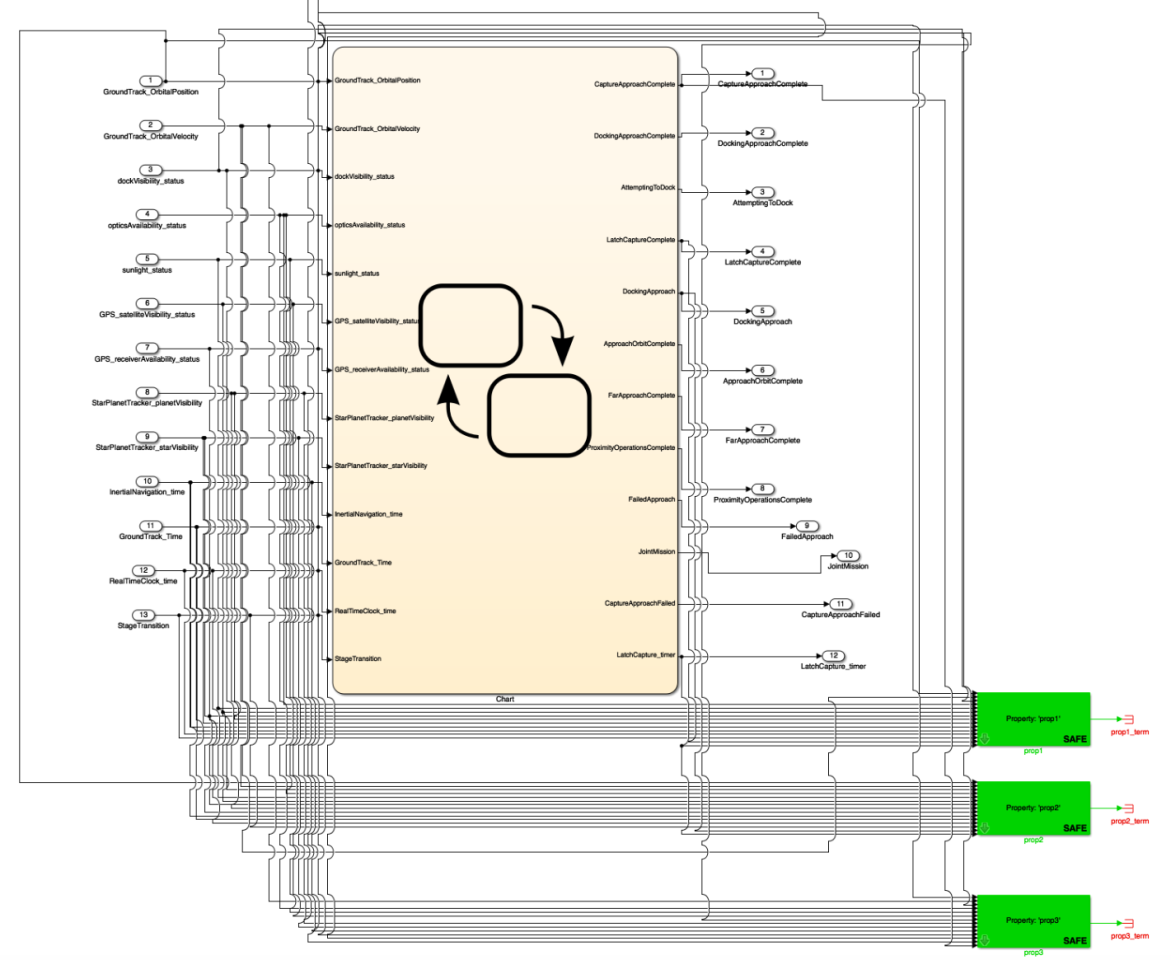
Figure 3: CoCoSim Framework.

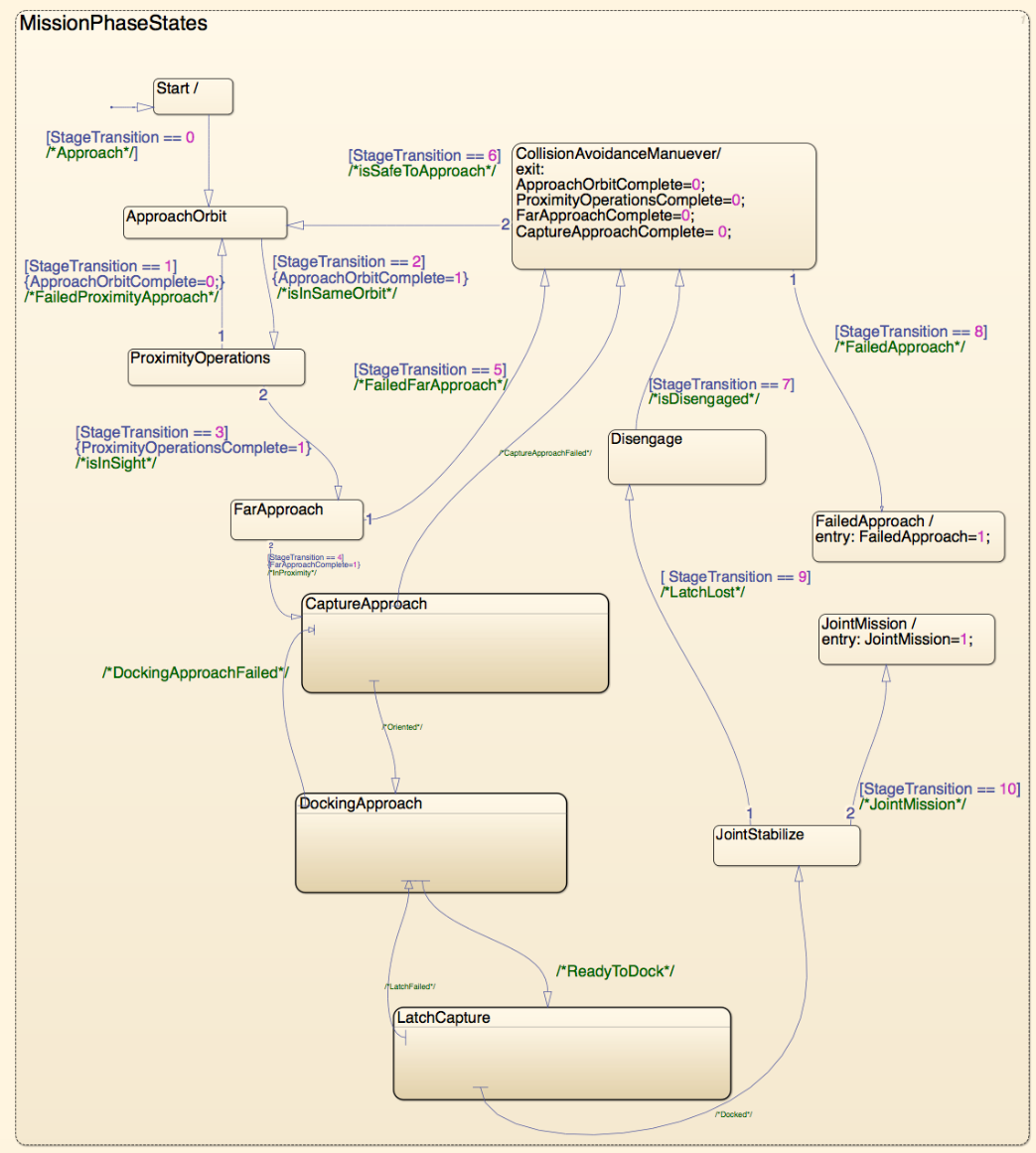
CoCoSim has been used in the analysis of a commercial aircraft control system implemented as a large, high-fidelity, Simulink model that included avionics,

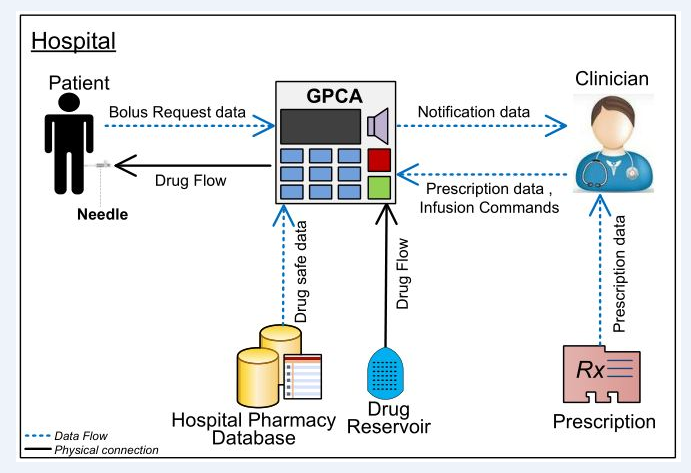
(with transport delay), actuators, engines, landing gear, nonlinear aerodynamics,

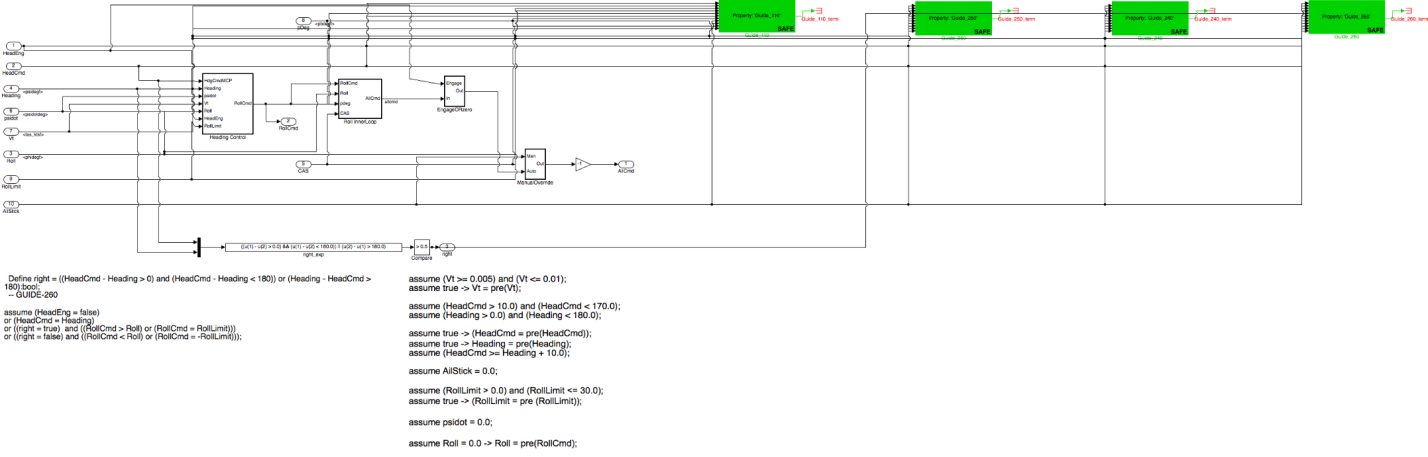
sensors (including noise), aircraft parameters, equations of motion, and gravity. The model consists of approximately 5700 Simulink blocks. CoCoSim was used to study the guidance and controls models and their properties. CoCoSim was used to verify 10 high-level safety requirements of the model. Five requirements were proved to be safe, while the other 5 were proved to be unsafe and counterexamples producing the unsafe states were provided). The Simulink model can be found in <https://github.com/coco-team/benchmarks/tree/master/Simulink/tcm> .

## Photo Options:









## Reference Documents:

<https://easychair.org/publications/paper/fPz>

Picture related to the NASA docking approach, that might be interesting for the RSE website too.