An application of KLEE to aerospace industrial software

Juan Francisco García, Daniel Jurjo, Fernando Macías, José F. Morales, Alessandra Gorla

IMDEA Software Institute

June 10, 2021

Motivation

- Testing is a critical process in aerospace software.
- Unit test generation is currently done manually, in a trial and error process.

Aerospace Software Nature

- Loops are bounded.

```
1       for (i = 0; i <= 2; i++) {
2         ...
3       }
4
```

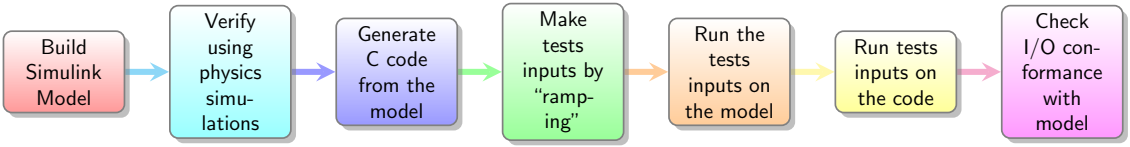
Aerospace Software Nature

The decision variables are mostly:

- Boolean.
- Floating point, usually normalized in a range.

```
1       for (i = 0; i <= 2; i++) {
2         if (validity[i]){
3           if (sensor[i]>threshold){
4             status[i]=1
5           }
6         }
7       }
8
```

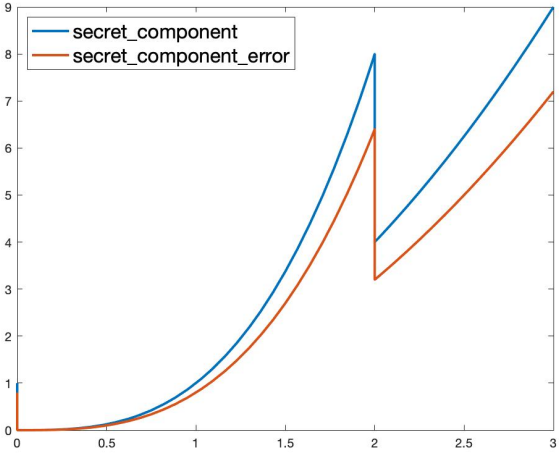
The company approach



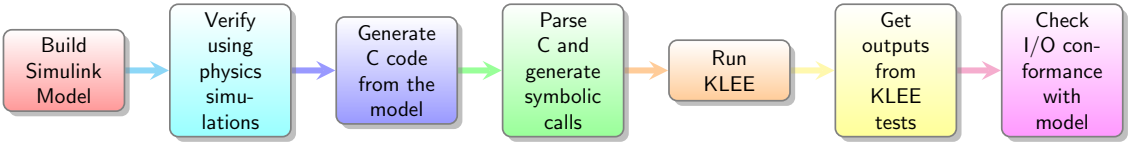
The company approach

- Model is assumed to be correct if it passes series of tests designed from the physical/engineering perspective.
- There is a lack of specification.

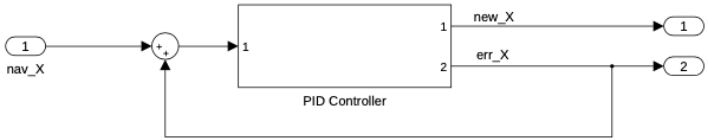
The company approach



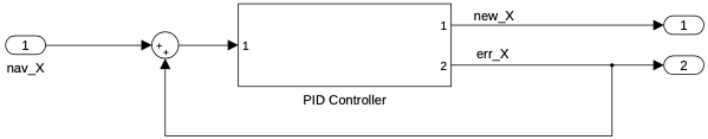
Our approach



Simulink and C code conformance



Simulink and C code conformance



- Make state symbolic, get pairs of values for state and input (S, I).
- It is needed to reach state S in the model to test the input I.

Results

Experiment	Company Approach		Our approach	
	Tests	Decision Coverage[1]	Tests	Decision Coverage[1]
Exp. 1	6000	78%	54	100%
Exp. 2	34	100%	6	100%
Exp. 3	10	100%	2	100%

[1] Measured with Simulink Coverage Tool

Benefits of our approach

- An equivalent process that is automatic.

Benefits of our approach

- An equivalent process that is automatic.
- Reduce human bias.

Benefits of our approach

- An equivalent process that is automatic.
- Reduce human bias.
- Reduce number of tests.

Benefits of our approach

- An equivalent process that is automatic.
- Reduce human bias.
- Reduce number of tests.
- Assure path coverage instead of decision coverage.

Challenge: Floating Point

KLEE does not support floating point. The alternatives we have considered are:

- KLEE-Float.
- Fuzzing.
- Combining KLEE with Fuzzers.

Work in progress: libFuzzer

Why libFuzzer?

- Function oriented.

Work in progress: libFuzzer

Why libFuzzer?

- Function oriented.
- Structure-aware fuzzing.

Work in progress: libFuzzer

Why libFuzzer?

- Function oriented.
- Structure-aware fuzzing.
- Coverage-guided.

Work in progress: libFuzzer

Why libFuzzer?

- Function oriented.
- Structure-aware fuzzing.
- Coverage-guided.
- Custom mutation and recombination function.

Work in progress: libFuzzer

- It does not generate a test for each path.

```
1  for (i = 0; i <= 2; i++) {
2    if (boolean[i]){
3      ...
4    }
5  }
6
```

Work in progress: libFuzzer

- It does not generate a test for each path.
- It provides meaningful values for floating point.

```
1  for (i = 0; i <= 2; i++) {
2    if (fp_value[i]>threshold){
3      ...
4    }
5  }
6
```

Work in progress: KLEE-Float

- It works well on the simpler models.
- It suffers of path explosion on more complex ones.

Future lines

- Evaluate KLEE/KLEE-Float+fuzz with different configurations.
- Memory-based equivalence relation for better oracles.
- Perform symbolic execution of the hardware constants.

Questions

Experiment	Company Approach		Our approach	
	Tests	Decision Coverage[1]	Tests	Decision Coverage[1]
Exp. 1	6000	78%	54	100%
Exp. 2	34	100%	6	100%
Exp. 3	10	100%	2	100%