**Combinatorial Optimization of Unit Tests in NASA’s Core Flight System (cFS)**

Dimitris E. Simos, Manuel Leithner, William M. Stanton, Rick Kuhn, Raghu Kacker

**NASA Core Flight System (cFS)**
- Common software for spaceflight missions.
- Focus on mission-specific applications instead of reinventing the wheel.
- Layered architecture allows development on desktop systems and later integration on actual flight hardware.
- Provides unit tests for cFS.
- Mission-specific apps supply their own tests.

**Research Questions**
- How much combinatorial coverage do current tests provide?
- Can we add Covering Arrays to improve it?

**Workflow**
- Extract function signatures and execution trace using `gdb`.
- Create Input Parameter Model from signatures, traces and constants.
- Measure combinatorial coverage using `CAmetrics`.
- Create Covering Array from Input Parameter Model using `CAgen`.

**Additional Variations**
- Covering Arrays that extend existing tests.
- Input Structure Model based on manual partitioning.
- Combined model for `CFE_SB_SubscribeFull()` and `CFE_SB_UnsubscribeFull()`.

**Next Steps**
- Identify additional constraints.
- Construct oracle and test bed.
- Execute tests as part of continuous integration.

**Figures**
- Figure 1: Excerpt of execution trace
- Figure 2: Coverage of (a) existing unit tests, (b) generated MCA(19596; 3, 6, {272, 18, 3, 2, 4, 3}) for `CFE_SB_SubscribeFull()` function
- Figure 3: Per-test and cumulative coverage of (a) existing unit tests, (b) generated MCA(19596; 3, 6, {272, 18, 3, 2, 4, 3}) for `CFE_SB_SubscribeFull()` function

**Conclusion**

**Summary**
- Model extraction of unit tests feasible with dynamic analysis.
- Existing unit tests do not provide much combinatorial coverage.
- Combination of unit and combinatorial testing yields high assurance.

**Challenges**
- Unit tests may not use defined values.
- Identifying constraints requires domain knowledge.
- Testbed and oracle necessary for execution.