Can we make safety-critical embedded systems ultra-reliable?

Modern aircraft and manned spacecraft rely on complex embedded software for guidance, navigation, and control (GN&C). Software failures can lead to the loss of the vehicle and human life.

Characteristics of safety-critical GN&C software:
- Distributed and redundant (for fault-tolerance)
- Hard real-time (i.e., constant time usage)
- Constant memory usage (i.e., no dynamic memory allocation)
- (Usually) C source code

Safety-critical software is rigorously designed using regimented coding standards, testing standards, and formal methods to increase confidence in its correctness. But these do not guarantee correctness at runtime.

Research Challenge

Runtime monitoring for general-purpose software in high-level languages (e.g., Java) is a mature field. We are researching how to adapt runtime monitoring to safety-critical GN&C software with the named characteristics.

A fundamental challenge our research addresses is compositionality: how do monitors and the observed software interact?
- **Functionality**: monitors must not change the functionality of the observed system, unless it violates its specification.
- **Schedulability/Time**: monitors must execute in constant time and not interfere with the timing properties of the observed program.
- **Reliability**: the reliability of the monitor + observed system is greater or equal to the reliability of the observed system alone.
- **Certification**: Monitors should require very few---if any---modifications to the source code of the observed programs.

**SYNCHRONIZATION**

**Tools**

**Code Synthesis**
A correct-by-construction approach generates low-level C code from high-level models. Commercial code-generators like Simulink/Stateflow and SCADE can generate embedded C from high-level models, but they are closed-source so are not extensible.

**Customized Code Generation**

**Atom** is an extensible, open-source, domain-specific library for the Haskell functional programming language. The library rewrites an Atom/Haskell program to embedded C source files. Eaton, Ltd. has used Atom to synthesize control systems for commercial vehicles. Atom-synthesized C is **guaranteed** to have deterministic memory usage and timing. Atom is **not** a new compiler so can reuse all of the language infrastructure provided by Haskell—thus, we get cheap, customized, embedded code generation!

**Copilot**

Copilot synthesizes monitors for safety-critical GN&C software. Copilot is built on top of Atom and is an extensible, open-source set of libraries for specifying and synthesizing monitors, **without** requiring the development of a new languages or compiler infrastructure.

**Prototype**

**Monitors**

**Further Reading**


**Atomic specification**

```plaintext
atom: Atom i
  g0 := do
    -- External reference to value A.
    a := word32 "a"
    b := word32 "b"
    -- External reference to value B.
    A := word32 "A"
    B := word32 "B"
    -- The external running flag.
    running := "running"
    -- A rule to modify A.
    A := value b - value a
    -- A rule to modify B.
    B := value a - value b
    -- A rule to clear the running flag.
    running := "false"
```

**Synthesized embedded C code**

```
... void example(void) {
  if (_clock % 1 == 0) {
    /* example.a_minus_b */
    example_a_minus_b();
    /* example.b_minus_a */
    example_b_minus_a();
    /* example.stop */
    example_stop();
    _clock = _clock + 1;
  }
  ...}
```