galois

Copilot: Assured Runtime Verification for Embedded Systems and Hardware

Ryan Scott, Galois Inc.

Focus: safety-critical systems

- Examples: medical devices, aircraft, nuclear power
- In this setting, system failure can result in significant damage, injury, or death, so high levels of assurance are needed
- Copilot achieves this via **runtime verification (RV)**

Copilot: a runtime verification (RV) framework

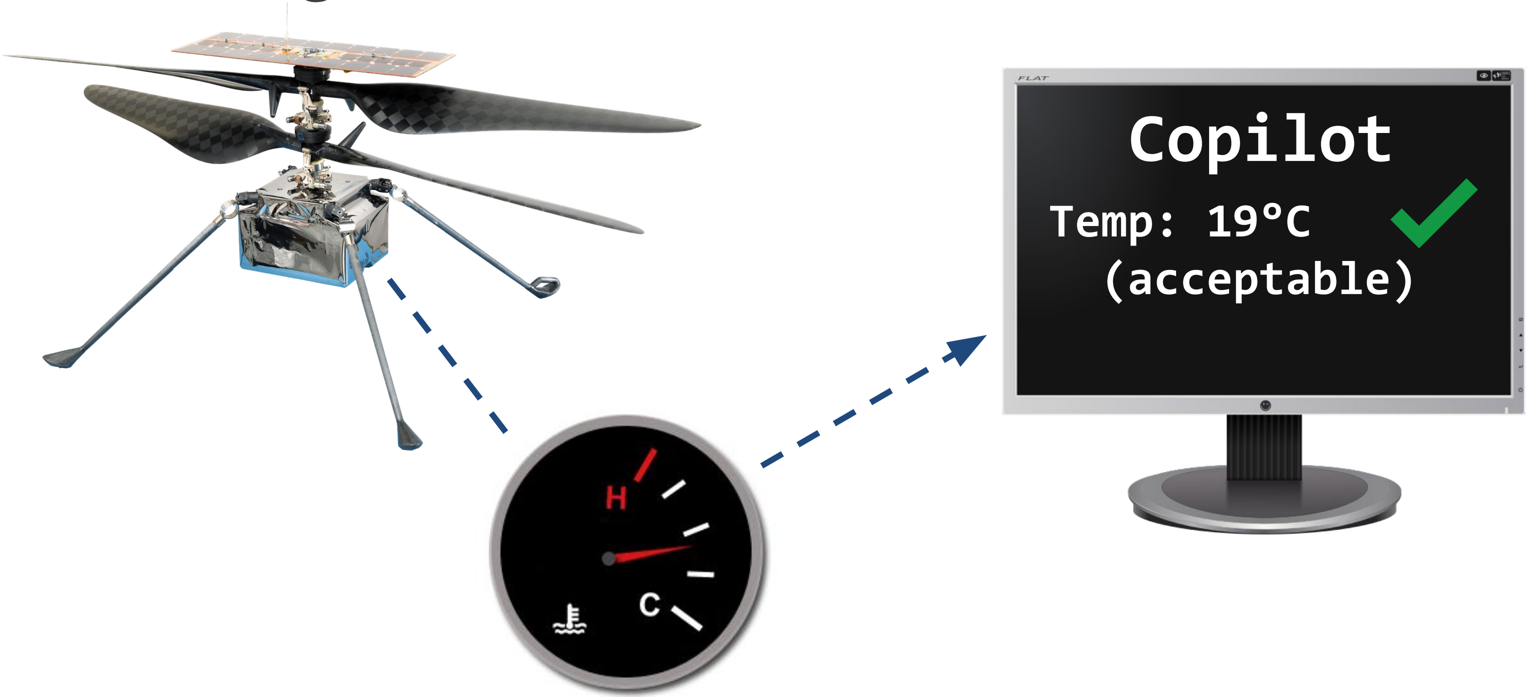
- Copilot is a hard realtime RV framework targeting embedded systems (since 2010) **and hardware (new!)**
- Used at NASA (e.g., to monitor UAV test flights)
- Open source

https://github.com/Copilot-Language/copilot

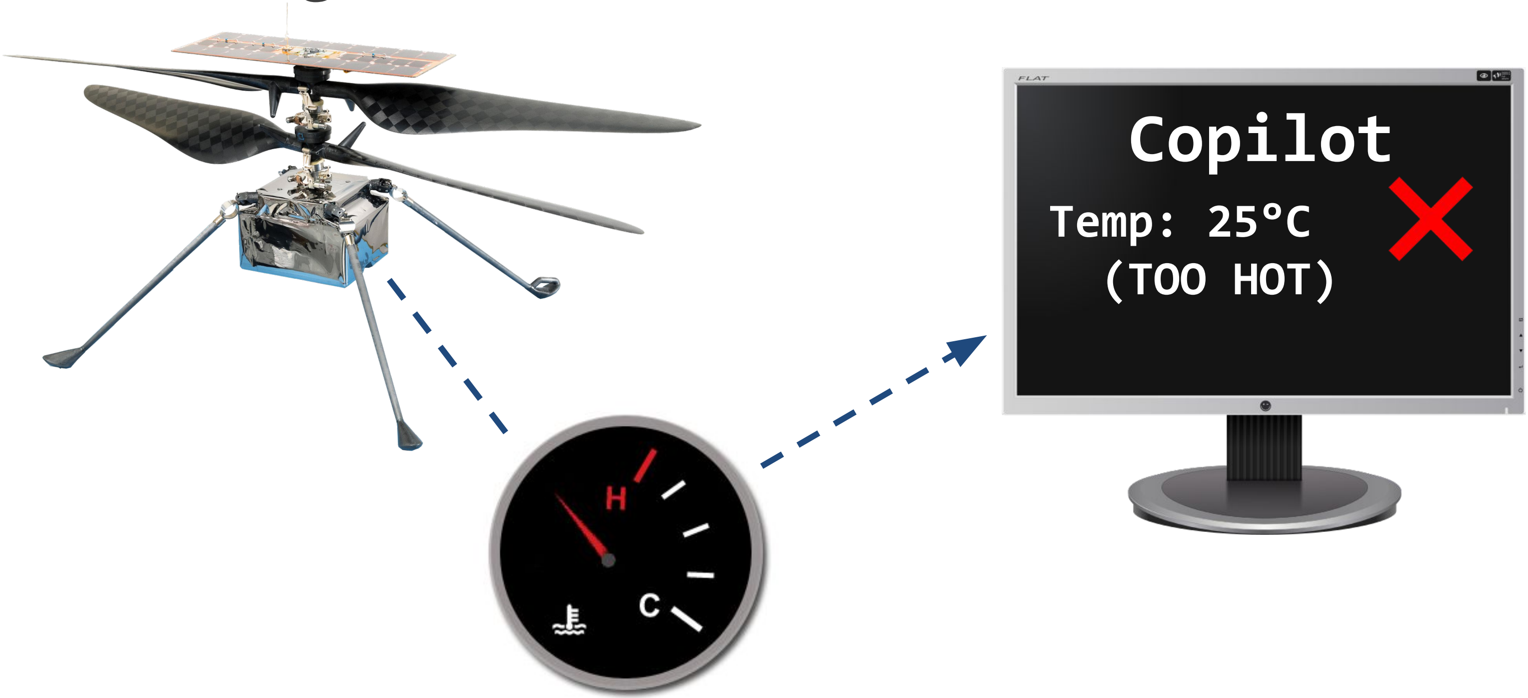
RV at a glance

RV at a glance

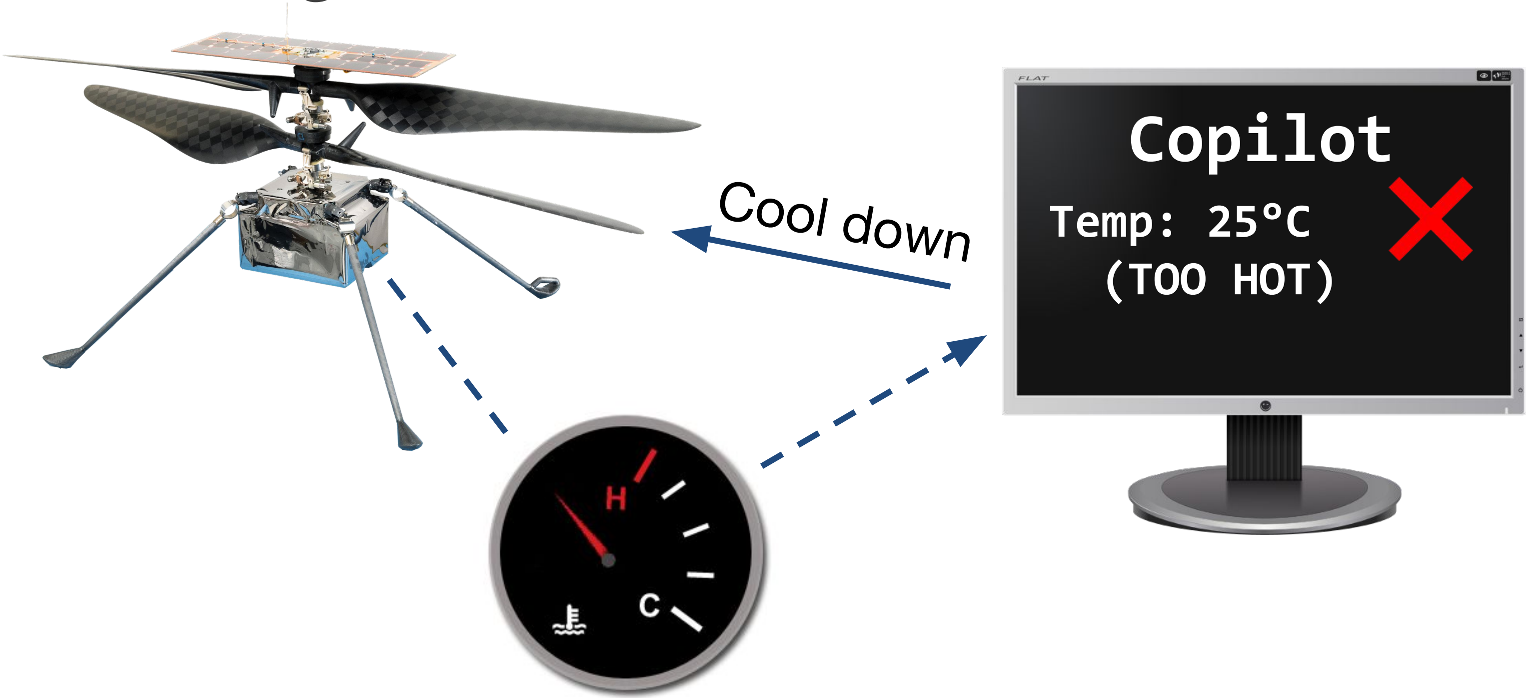
RV at a glance



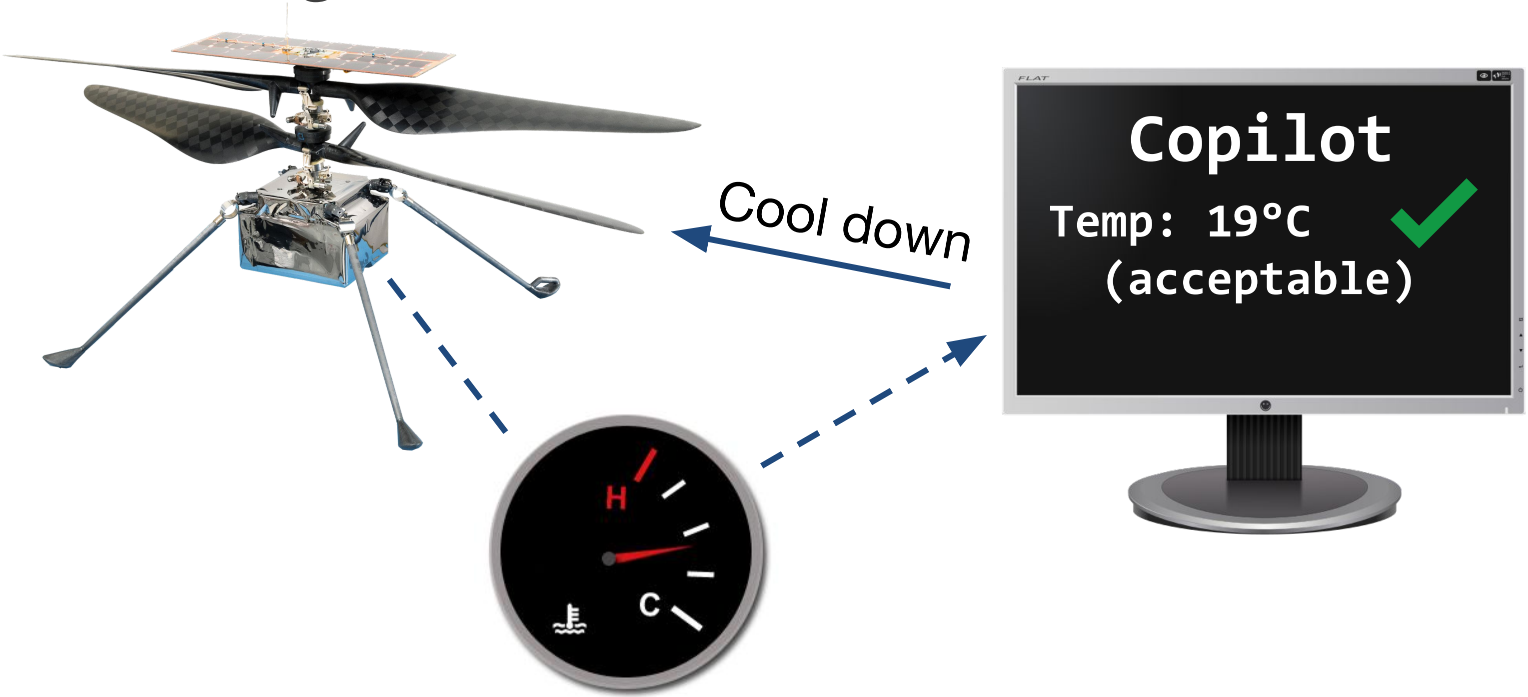
RV at a glance



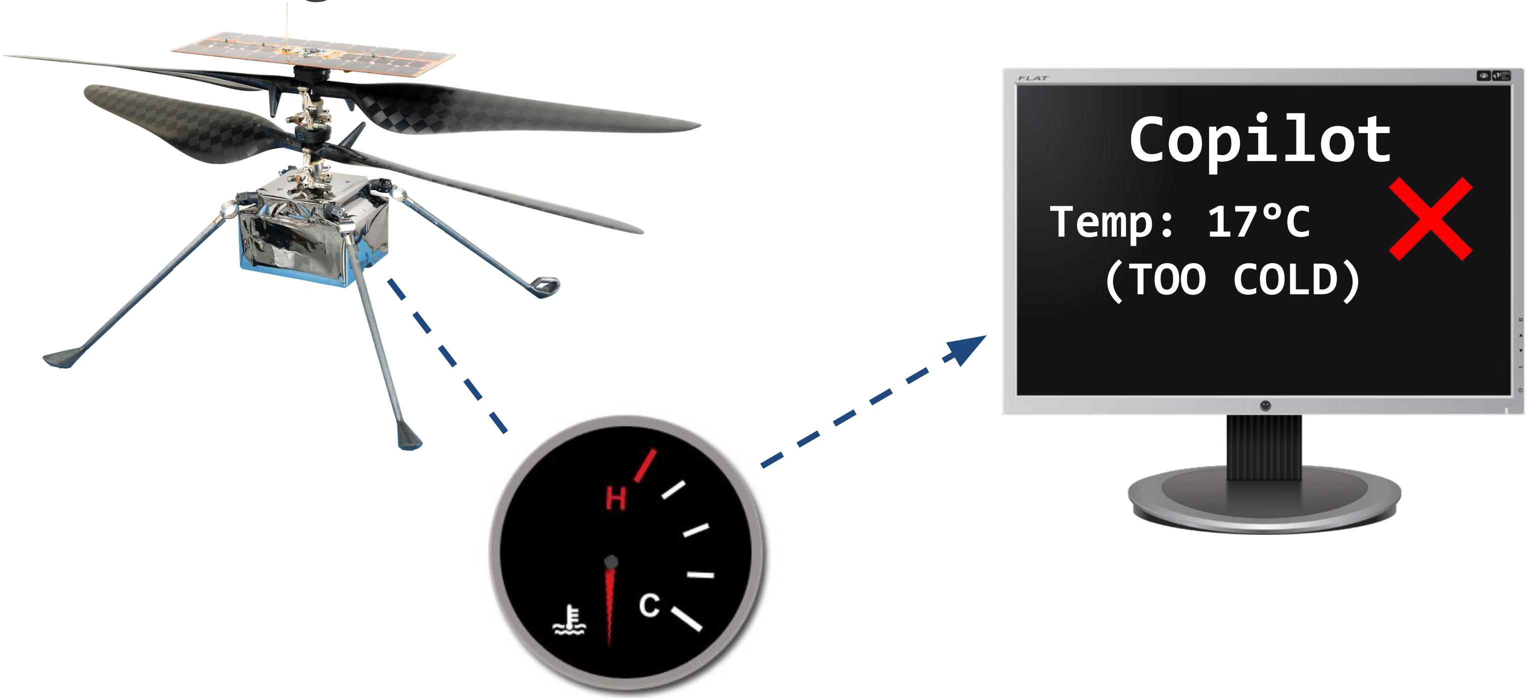
RV at a glance



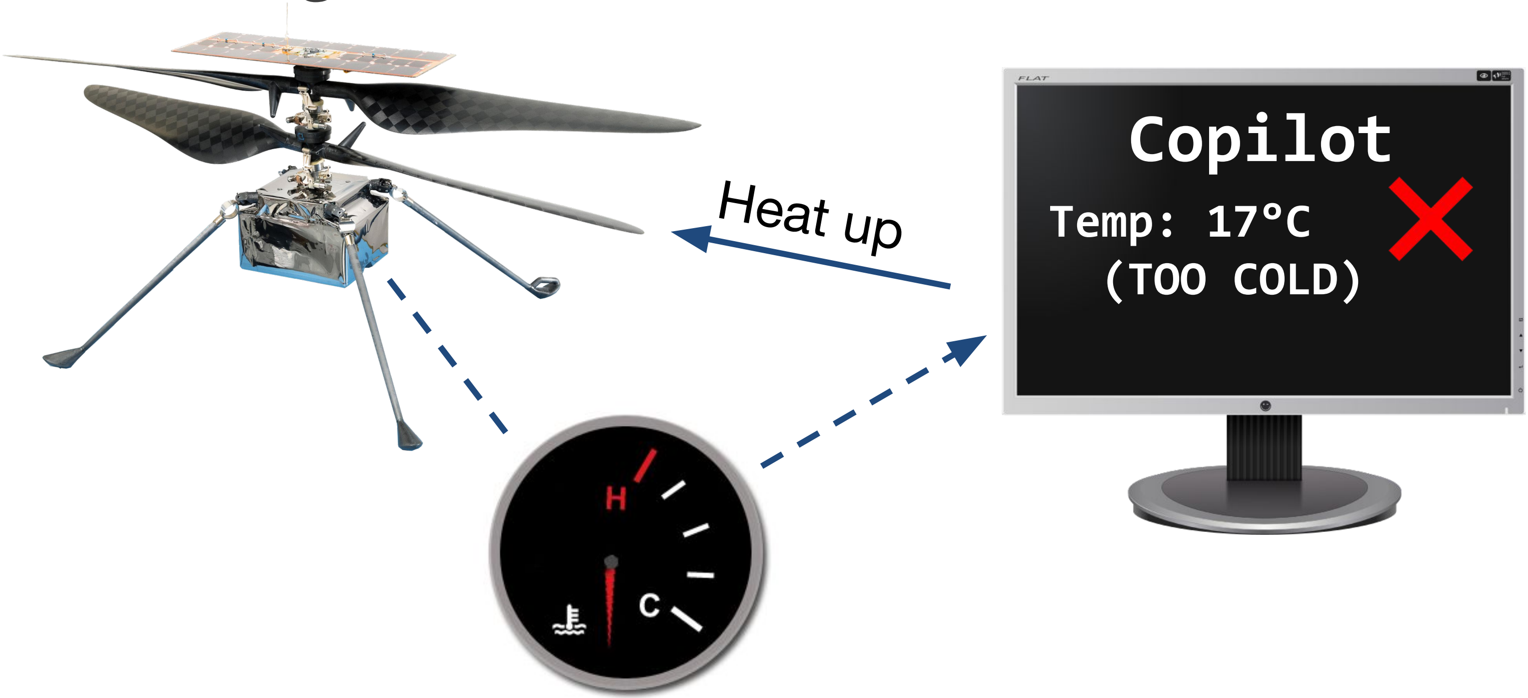
RV at a glance



RV at a glance



RV at a glance



Copilot design constraints

- Monitor code should be constant-space and constant-time
  - No manual memory management, `for` loops, or recursion

- Monitor code should be traceable to high-level requirements
  - Code must be auditable

Copilot: traceability

```
avgTemp :: Stream Float
avgTemp = ...


spec :: Spec
spec = do
  trigger "heaton"
    (avgTemp < 18.0) [arg avgTemp]
  trigger "heatoff"
    (avgTemp > 21.0) [arg avgTemp]
```

Copilot: traceability

```haskell
avgTemp :: Stream Float
avgTemp = ...


spec :: Spec
spec = do
  trigger "heaton"
    (avgTemp < 18.0) [arg avgTemp]
  trigger "heatoff"
    (avgTemp > 21.0) [arg avgTemp]
```

Copilot translator

```c
void heaton(float);
void heatoff(float);

void step(void) {
  ...
  if (heaton_guard()) {
     heaton(heaton_get_arg());
  };
  if (heatoff_guard()) {
     heatoff(heatoff_get_arg());
  };
  ...
}
```

Copilot: traceability

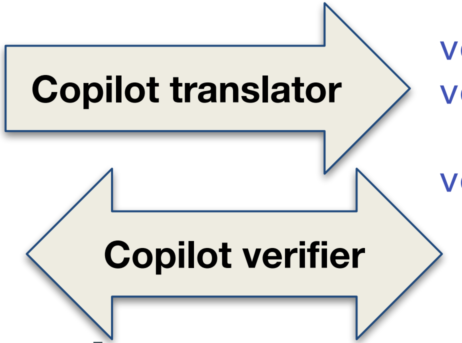
```haskell
avgTemp :: Stream Float
avgTemp = ...


spec :: Spec
spec = do
  trigger "heaton"
    (avgTemp < 18.0) [arg avgTemp]
  trigger "heatoff"
    (avgTemp > 21.0) [arg avgTemp]
```

Copilot translator

```c
void heaton(float);
void heatoff(float);

void step(void) {
  ...
  if (heaton_guard()) {
     heaton(heaton_get_arg());
  };
  if (heatoff_guard()) {
     heatoff(heatoff_get_arg());
  };
  ...
}
```

Copilot: traceability

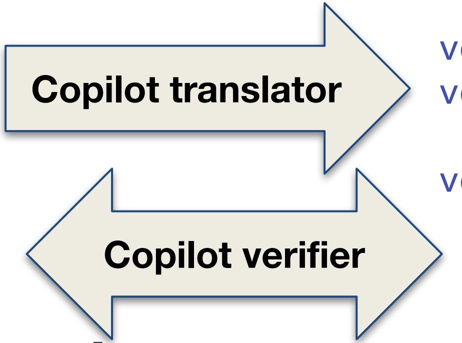
```haskell
avgTemp :: Stream Float
avgTemp = ...


spec :: Spec
spec = do
  trigger "heaton"
    (avgTemp < 18.0) [arg avgTemp]
  trigger "heatoff"
    (avgTemp > 21.0) [arg avgTemp]
```




```c
void heaton(float);
void heatoff(float);

void step(void) {
  ...
  if (heaton_guard()) {
     heaton(heaton_get_arg());
  };
  if (heatoff_guard()) {
     heatoff(heatoff_get_arg());
  };
  ...
}
```

Copilot: traceability

```haskell
avgTemp :: Stream Float
avgTemp = ...


spec :: Spec
spec = do
  trigger "heaton"
    (avgTemp < 18.0) [arg avgTemp]
  trigger "heatoff"
    (avgTemp > 21.0) [arg avgTemp]
```



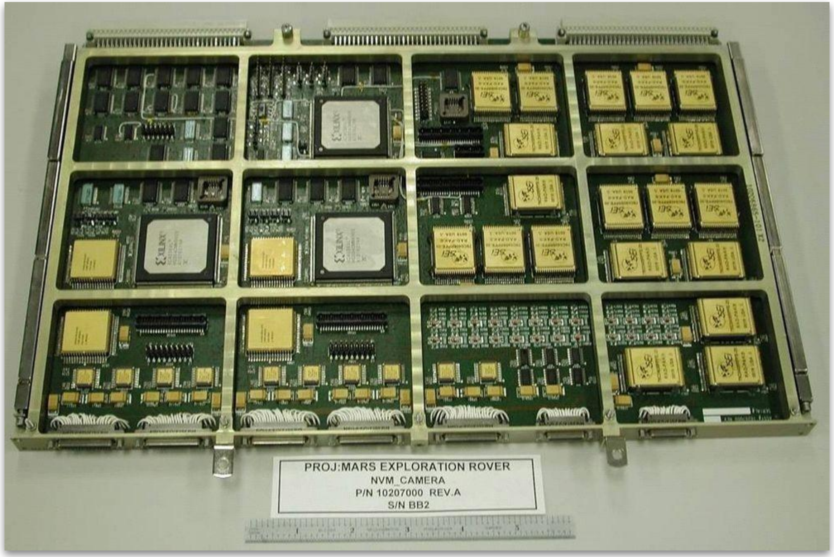

```c
void heaton(float);
void heatoff(float);

void step(void) {
  ...
  if (heaton_guard()) {
     heaton(heaton_get_arg());
  };
  if (heatoff_guard()) {
     heatoff(heatoff_get_arg());
  };
  ...
}
```

https://github.com/Copilot-Language/copilot-verifier

Copilot and hardware

Reasons for hardware RV

- Many critical systems run on FPGAs
  - e.g., Integrated Modular Avionics (IMAs) for space applications
- Need to run monitors in fault-containment regions, separate from the code being monitored
- FPGAs ideal for hard realtime



Which hardware language to use?

VHDL Verilog

SystemVerilog

bluespec

CHISEL

Which hardware language to use?

VHDL Verilog

SystemVerilog

bluespec

CHISEL

Why Bluespec?

- Bluespec's syntaxes are familiar to both Copilot users (Bluespec Haskell) and hardware enthusiasts (Bluespec SystemVerilog)

Why Bluespec?

- Bluespec’s syntaxes are familiar to both Copilot users (Bluespec Haskell) and hardware enthusiasts (Bluespec SystemVerilog)

- Bluespec’s semantics closely correspond with Copilot’s semantics

Why Bluespec?

- Bluespec's syntaxes are familiar to both Copilot users (Bluespec Haskell) and hardware enthusiasts (Bluespec SystemVerilog)
- Bluespec's semantics closely correspond with Copilot's semantics
- Well suited to high levels of assurance
  - Property-based testing (Bluecheck), formal verification (Kami)

Why Bluespec?

- Bluespec's syntaxes are familiar to both Copilot users (Bluespec Haskell) and hardware enthusiasts (Bluespec SystemVerilog)
- Bluespec's semantics closely correspond with Copilot's semantics
- Well suited to high levels of assurance
  - Property-based testing (Bluecheck), formal verification (Kami)
- Can be compiled to Verilog RTL

Copilot versus Bluespec

Copilot versus Bluespec

Values are represented as *streams* that change over time

```
-- 1, 2, 3, 4, 5, ...
countUp :: Stream Word32
countUp = [1] ++ (countUp + 1)
```

Copilot versus Bluespec

Values are represented as *streams* that change over time

```
-- 1, 2, 3, 4, 5, ...
countUp :: Stream Word32
countUp = [1] ++ (countUp + 1)
```

Values are stored in *registers*, whose values can change each clock cycle

```
countUpModule :: Module Empty
countUpModule =
  module
    countUp :: Reg (UInt 32)
      <- mkReg 1;
    ...
    action
      countUp := countUp + 1;
```

Copilot versus Bluespec

Copilot versus Bluespec

External streams represent abstract values that are sampled (e.g., sensor readings)

```
-- Average engine temperature
avgTemp :: Stream Float
avgTemp = extern "avg_temp"
```

Copilot versus Bluespec

External streams represent abstract values that are sampled (e.g., sensor readings)

```
-- Average engine temperature
avgTemp :: Stream Float
avgTemp = extern "avg_temp"
```

Module interfaces can define registers that are defined elsewhere in the hardware

```
interface TempIfc
  avgTemp :: Reg Float

engineModule ::
  Module TempIfc -> Module Empty
```

Copilot | versus | Bluespec

Copilot versus Bluespec

Triggers can fire when a stream satisfies a predicate

```
spec :: Spec
spec = do
  trigger "heaton"
    (avgTemp < 18.0) [arg avgTemp]
  trigger "heatoff"
    (avgTemp > 21.0) [arg avgTemp]
```

Copilot versus Bluespec

Triggers can fire when a stream satisfies a predicate

```
spec :: Spec
spec = do
  trigger "heaton"
    (avgTemp < 18.0) [arg avgTemp]
  trigger "heatoff"
    (avgTemp > 21.0) [arg avgTemp]
```

Rules fire on a particular clock cycle if its condition holds:

```
rules
  "heaton": when (avgTemp < 18.0) ==>
      heaton avgTemp

  "heatoff": when (avgTemp > 21.0) ==>
      heatoff avgTemp
```

Copilot versus Bluespec

Other Copilot language features that Bluespec supports:

- Arrays (via Bluespec's `Vector` package)
- Structs (via Bluespec's `struct` feature)
- Floating-point operations* (via Bluespec's `Float` package)

Copilot to Bluespec: current status

- Developed *Copilot-Bluespec*, which automatically translates Copilot to Bluespec code suitable for FPGA use

- Capable of translating all examples in the Copilot test suite

- Extensive test suite that checks that translated Bluespec matches the behavior of the original Copilot code

https://github.com/Copilot-Language/copilot-bluespec

Future challenges

Challenges

How to correctly translate system requirements to Copilot?

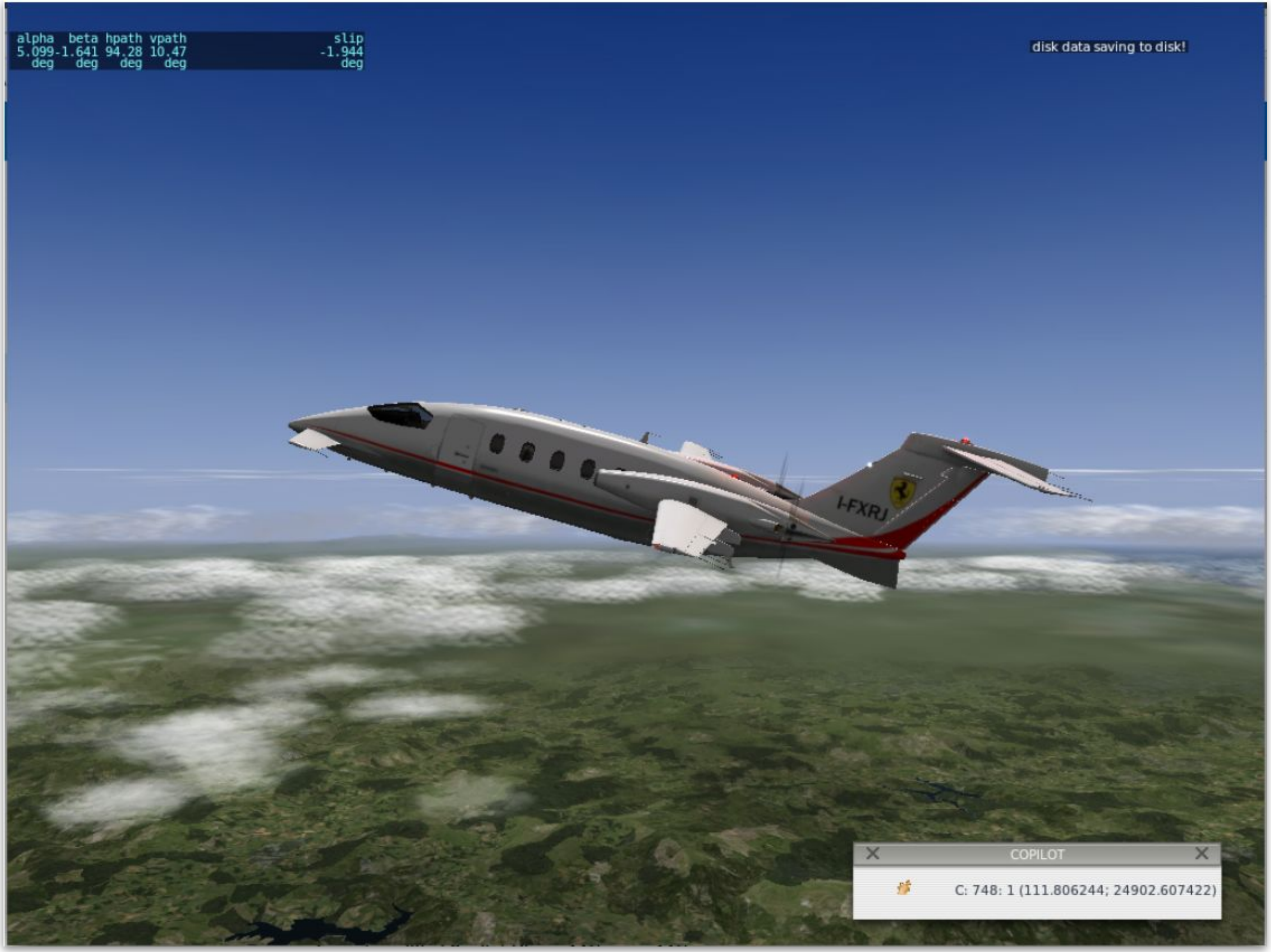
- Many workflows that use monitoring involve English-language requirements, not formally rigorous specifications

- How can we make the process of encoding these requirements into Copilot easier?

Ogma

- Ogma: a tool for converting high-level requirements into runtime monitoring code
- Converts requirements into temporal logic formulae, which can then be translated into trustworthy code written in Copilot, NASA's Core Flight System (cFS), Robot Operating System (ROS), and more

https://github.com/nasa/ogma



alpha beta hpath vpath slip
5.099 -1.641 94.28 10.47 -1.944
deg deg deg deg deg
disk data saving to disk!
I-FXRJ
COPILOT
C: 748: 1 (111.806244; 24902.607422)

Challenges

What is the best way to write a compiler to Bluespec?

- There are no industrial-grade Bluespec pretty-printers, aside from the code used in the official Bluespec compiler
- The Bluespec compiler can't be used as a library
- For now, we've forked the code in the Bluespec compiler to use in our tool
- We should do better: **https://github.com/B-Lang-org/bsc/issues/546**

Challenges

Limited support for floating-point operations

- Basic arithmetic operations, `abs`, and `sqrt` are supported
- `sin`/`cos`, exponentiation, and logarithms not currently supported
- Could consider using floating-point IP or VGM
- Currently porting software implementations of floating-point operations to Bluespec: **https://github.com/B-Lang-org/bsc/discussions/534**

Copilot-Bluespec takeaways

- Copilot and Ogma are robust ecosystems for describing, implementing, and deploying monitors.
- Bluespec is a natural fit for realizing Copilot's style of runtime verification in a hardware setting.
- We are continuing to reduce barriers to entry for integrating runtime verification in high-assurance scenarios.

Copilot: **https://copilot-language.github.io**
Ogma: **https://github.com/nasa/ogma**
Copilot-Bluespec: **https://github.com/Copilot-Language/copilot-bluespec**

Backup slides

Ogma

- Ogma: a tool for translating natural-language requirements into runtime monitoring code
- Translates structured natural language into temporal logic formulae
- The temporal logic can then be translated into trustworthy code written in Copilot, NASA's Core Flight System (cFS), Robot Operating System (ROS), and more

https://github.com/nasa/ogma

Ogma example

Requirements are expressed in structured natural language (FRETish):

```
scope condition component* shall* timing response*
```

NL: "*While flying, if the airspeed is below 100 m/s, the autopilot shall increase the airspeed to at least 100 m/s within 10 seconds.*"

FRETish: `in flight mode if airspeed < 100 the aircraft shall within 10 seconds satisfy (airspeed >= 100)`

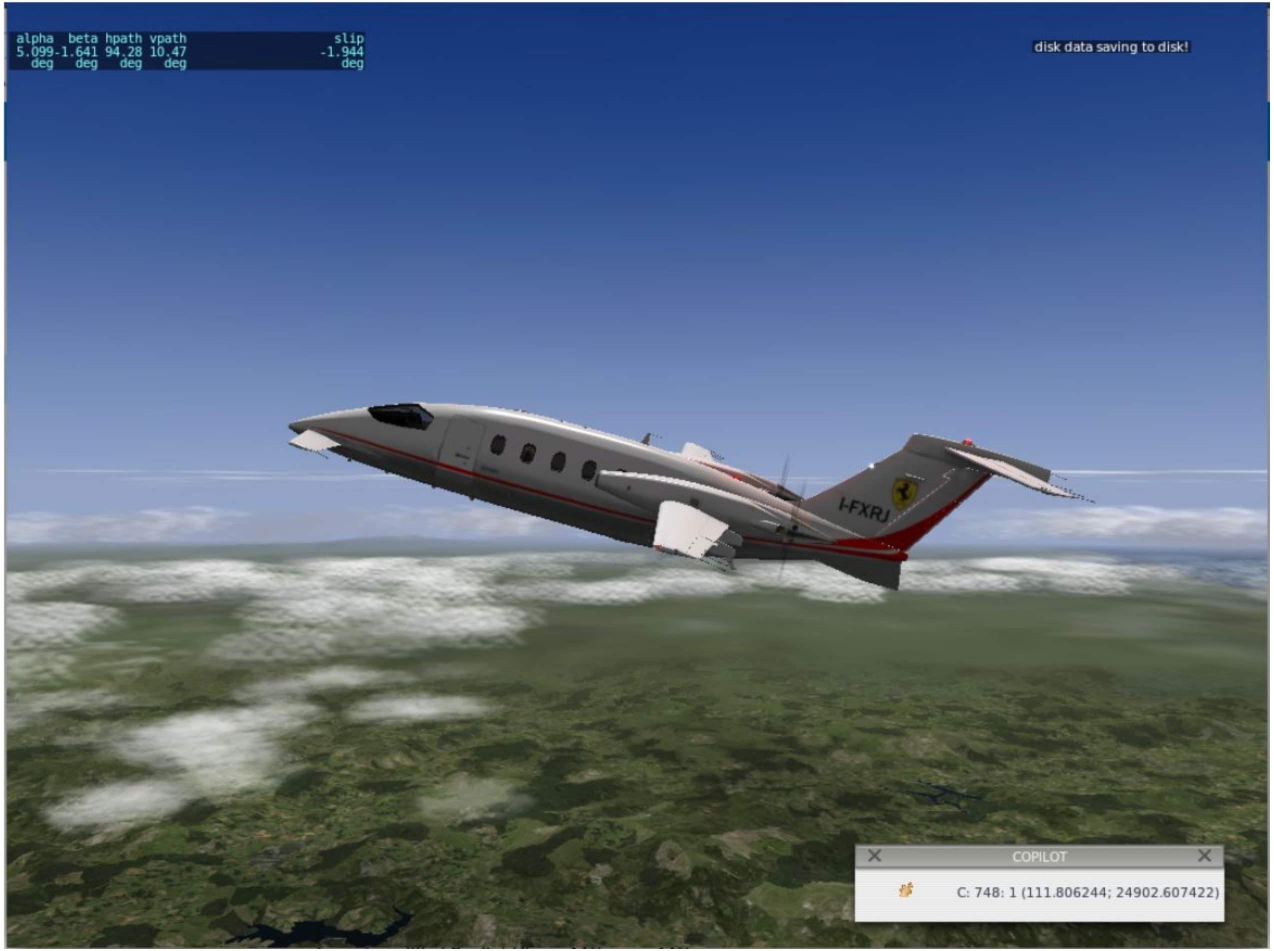
Ogma example: translated to temporal logic

NL: "*While flying, if the airspeed is below 100 m/s, the autopilot shall increase the airspeed to at least 100 m/s within 10 seconds.*"

FRETish: `in flight mode if airspeed < 100 the aircraft shall within 10 seconds satisfy (airspeed >= 100)`

pmLTL: H (Lin_flight → (Y ((($O_{[=10]}$ (((airspeed < 100) & ((Y (!(airspeed < 100))) | Fin_flight)) & (!(airspeed ≥ 100)))) → ($O_{[<10]}$ (Fin_flight | (airspeed ≥ 100)))) S ((($O_{[=10]}$ (((airspeed < 100) & ((Y (!(airspeed < 100))) | Fin_flight)) & (!(airspeed ≥ 100)))) → ($O_{[<10]}$ (Fin_flight | (airspeed ≥ 100)))) & Fin_flight)))) & ((!Lin_flight) S ((!Lin_flight) & Fin_flight)) → ((($O_{[=10]}$ (((airspeed < 100) & ((Y (!(airspeed < 100))) | Fin_flight)) & (!(airspeed ≥ 100)))) → ($O_{[<10]}$ (Fin_flight | (airspeed ≥ 100)))) S ((($O_{[=10]}$ (((airspeed < 100) & ((Y (!(airspeed < 100))) | Fin_flight)) & (!(airspeed ≥ 100)))) → ($O_{[<10]}$ (Fin_flight | (airspeed ≥ 100)))) & Fin_flight)),

where Fin_flight (First timepoint in flight mode) is flight & (FTP | Y !flight), Lin_flight (Last timepoint in flight mode) is !flight & Y flight, FTP (First Time Point) is ! Y true.


alpha beta hpath vpath slip
5.099 -1.641 94.28 10.47 -1.944
deg deg deg deg deg
disk data saving to disk!
I-FXRJ
COPILOT
C: 748: 1 (111.806244; 24902.607422)

Ways to achieve assurance

Prove

Test

Monitor
(*Runtime verification*)

The need for hardware RV

- Many critical systems run on FPGAs or ASICs

- Use case: System Theoretic Process Analyses (STPAs)
  - Methodology for designing safe systems and preventing critical losses
  - Involve interactions between humans, software, and hardware