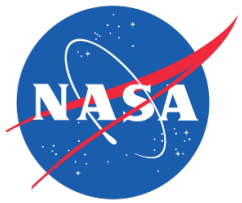
Structuring Copilot Code

Implementing Structs and Other Insights in Copilot

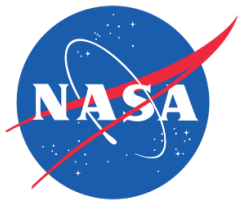
Eli Mendelson

Safety-Critical Avionics: Systems Branch
NASA Langley Research Center

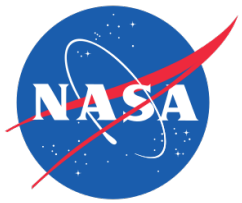
August 6, 2015 / Intern Presentation



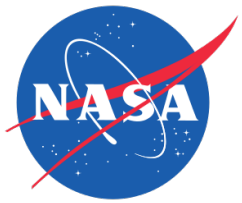
Outline



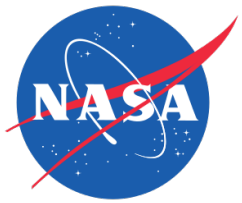
Outline



Outline

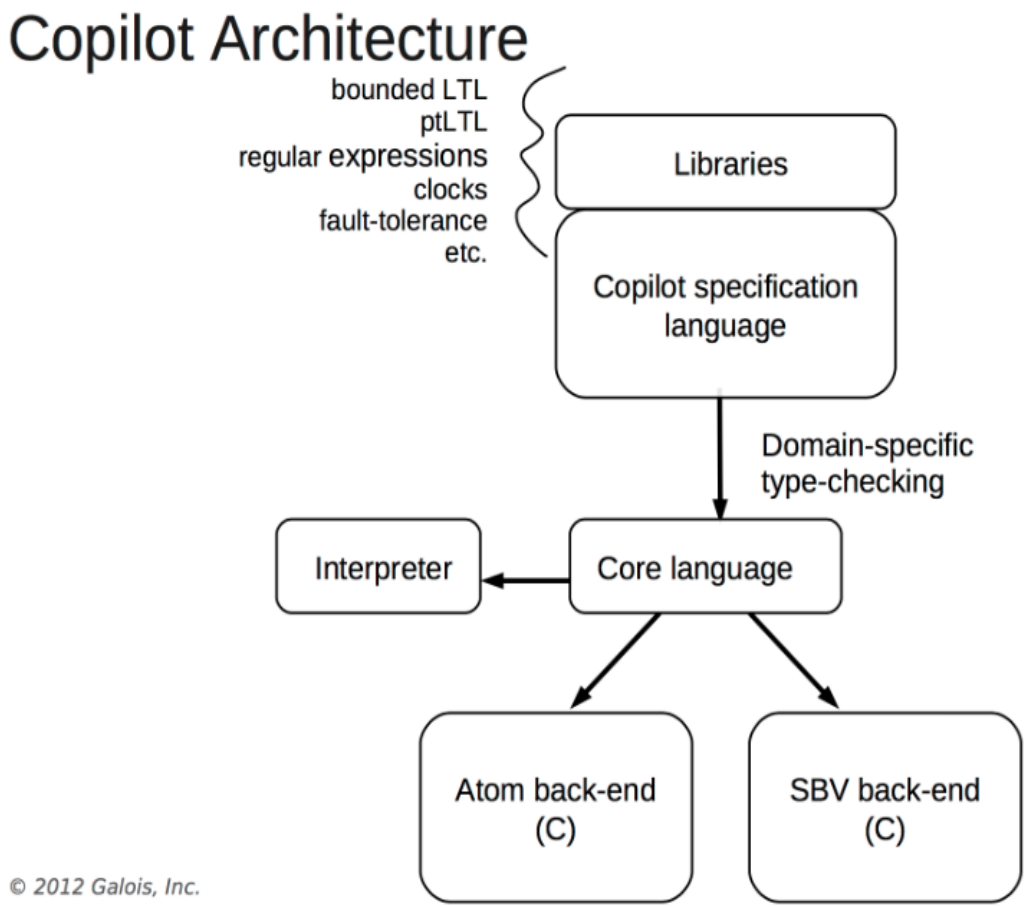


Outline



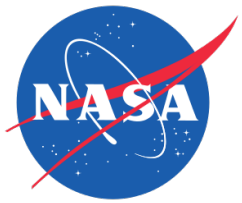
Copilot Pipeline

New Back Ends are Available (CompCert, Frama-C)

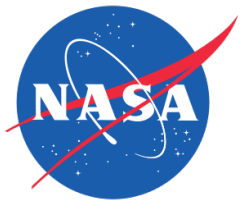


Copilot Back Ends

- All back ends compile to C99 - hard real-time systems
  - Atom
    - Heavily relies on arbitrary changes of state
  - SBV
    - Modular
    - Used to implement structs

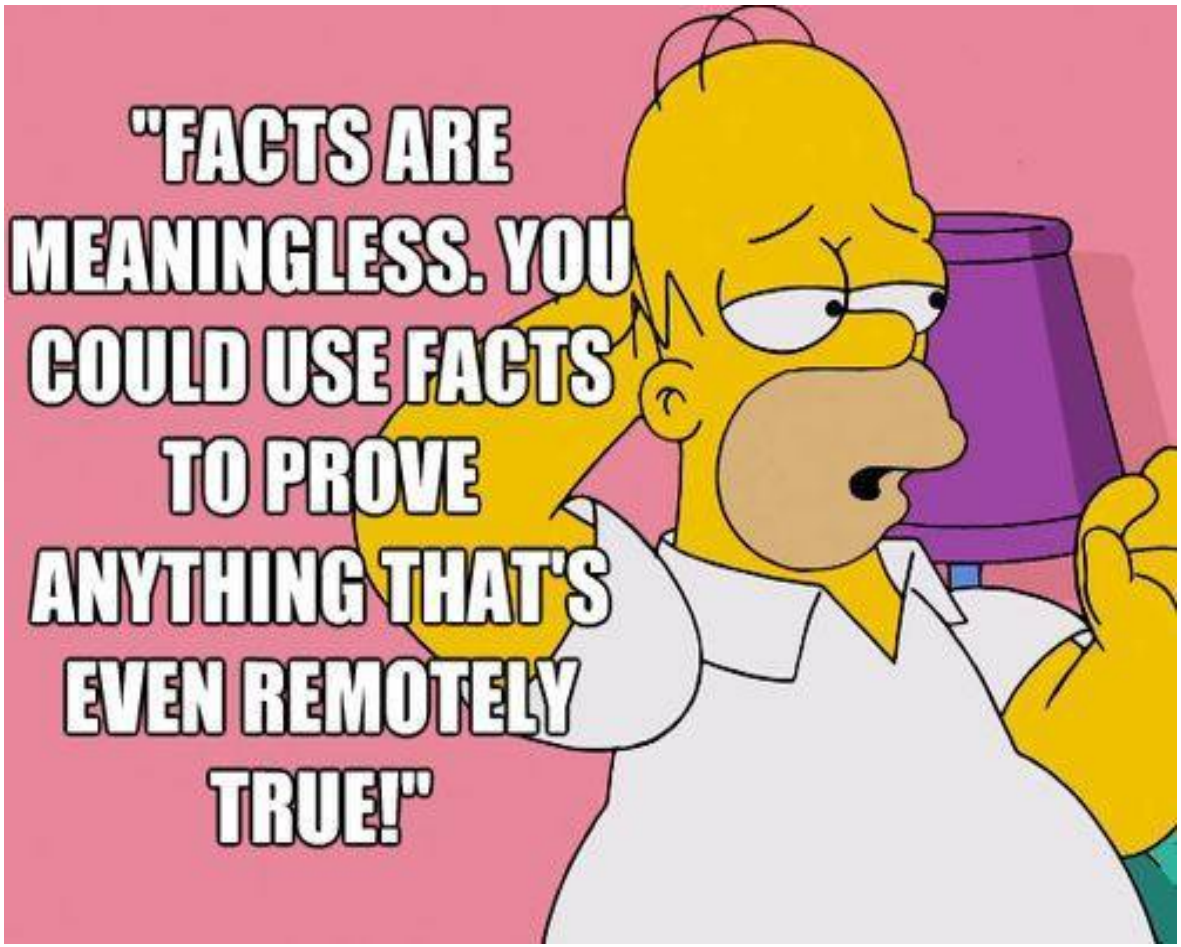


Outline



Homer Got It Right

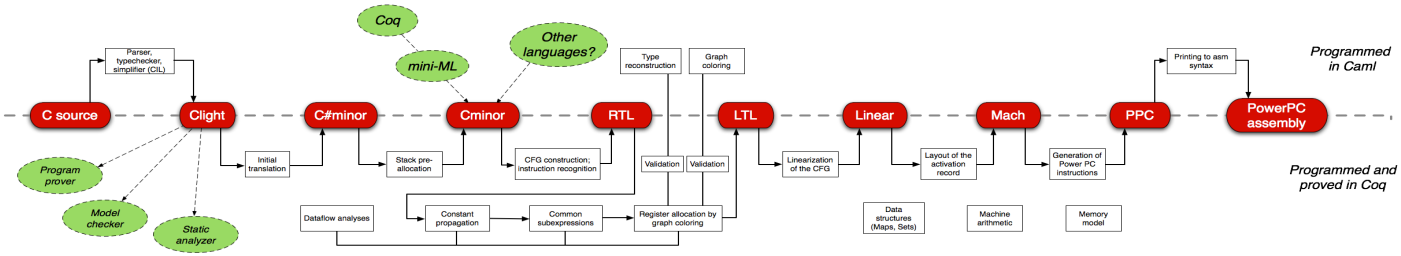
Converting Sensory Input to Safety



New Back Ends
CompCert, Frama-C

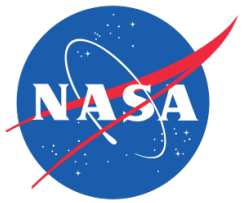
New back ends verify correctness of **monitor code**

- CompCert



- frama C Software Analyzers

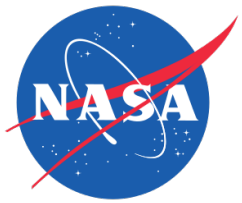
Outline



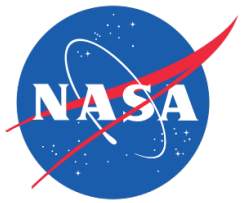
What is a Struct (in C)?

A struct is. . .

- Collection of values (no singular *field type*)
- Contiguous block of memory
- Type-less
- Portable



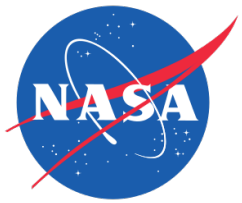
Outline



Monitors
How Can They Be Applied to Structs?

We need monitors to...

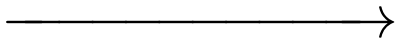
- Keep track of sensory input
- Embedded systems use structs to keep track of sensory values in a specification
- Ensure that a flight system abides by a specification



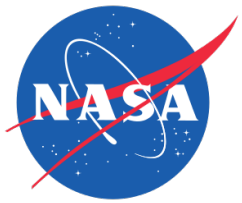
Structs in Monitor Code

Structuring Sensor Values

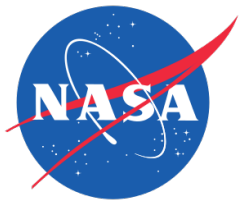
```
uint32_t count;
float sum;
float average;
float
correction;
float
algo_erro_check;
float min;
float max;
bool
start_sampling;
bool
have_correction;
```



```
struct NeutralThrustEstimation {
  uint32_t count;
  float sum;
  float average;
  float correction;
  float algo_erro_check;
  float min;
  float max;
  bool start_sampling;
  bool have_correction;
  };
static struct
NeutralThrustEstimation
neutralThrustEst;
```



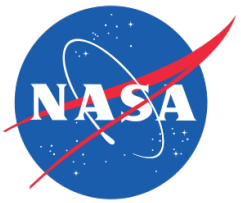
Outline



Haskell Meets Embedded-C

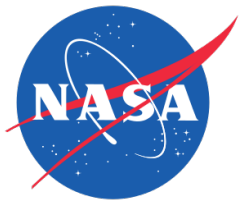
- Haskell
  - Functional language
  - Easy to implement formal methods
    - Verification
    - Model Checking

  vs.
- Embedded C
  - Constant time/memory
  - Output C99 code similar to MISRA

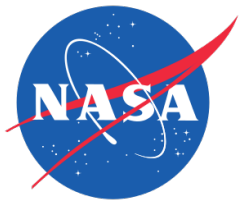


(Dis)Advantages

- Goals/Uses of Haskell and Embedded-C are not aligned
  - Leverage the features of both languages
  - Two sets of limitations
- Compiler produces monitor code
  - Avionics programmer must know Haskell
- New back ends only support subset of C99
  - CompCert does not support long doubles

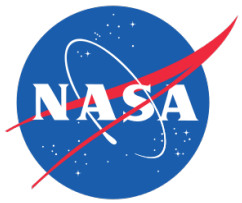


Outline



Examples in Copilot
Haskell Code

```
simple :: Stream Bool
simple = externStruct "simple" [("example", arg example)]
  where
    example :: Stream Word32
    example = 1
...
...
...
run :: Stream Word32
run = simple#"example"
```



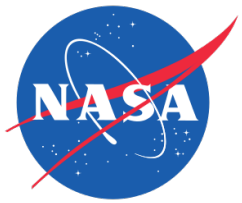
Examples in Copilot
C Code (for monitor)

.h File

```
struct simple {
  uint_32t example;
};
```

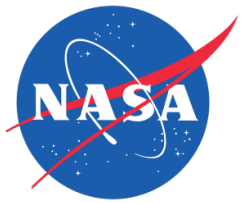
.c File

```
...
...
.. simple.example ..
...
...
```



Semantic Problems

- Structs need to be of type Bool
- Redundant way of implementing structs in Haskell specification
  - All fields must be declared as `Extern` variables, including structs
- Complications surrounding typing structs in Haskell
  - Difficult to nest structs



Summary

- Haskell and C serve vastly different purposes as languages
- Structs are necessary for monitors, but suffer from the disparity of Haskell and C
- Structs are not done being implemented, but significant progress has been achieved

- Outlook
  - **Copilot** will be very powerful as it becomes more robust.
  - Extending Copilot's use to other UAV software