Building a High-Assurance Unpiloted Air Vehicle

Lee Pike (speaker), Pat Hickey, James Bielman, Trevor Elliott, John Launchbury, Erlend Hamberg, Thomas DuBuisson

MEMOCODE | Oct 2013
The Problem

Mechanic

Short-range wireless

Long-range wireless

Entertainment

The Challenge

High-Assurance Cyber-Military Systems (HACMS)

PM: Dr. Kathleen Fisher

The “Air Team”

- **Boeing**: military vehicle
- **Galois, Inc.**: autopilot synthesis
- **NICTA**: networking/operating systems
- **Rockwell Collins/Univ. Minn.**: integration and architecture
- **DRAPER/AIS/U. Oxford (Red Team)**: vulnerability analysis
SMACCMPilot

- Secure
- Mathematically
- Assured
- Composition of
- Control
- Models
This Talk

How we have nearly built

- **Ivory**: a memory-safe language/compiler
- **Tower**: an architectural coordination language
- **SMACCMPilot**: a high(er)-assurance autopilot in 2-3 engineer-years (~1 calendar year).
How We Did It

1. Collaborate with a vibrant open-source system/community
2. Build **embedded domain-specific languages (EDSLs)** and type-safe macros
3. Synthesize the architecture
In the Beginning...

There was Arduino
• Simple 8-bit AVR
• For DIY beginners in embedded systems
• ArduPilot Mega Hardware
  AVR Processor: 8 bit, 16MHz, 8k RAM, 256k Flash
ArduPilot

- ArduPilot
- Arduino-based
- Open-source hardware and software
- 25 volunteer developers worldwide
- 1000s of users
- Starting to see commercial use

DIYDrones.com
- 30,000 users, 99% amateurs and hobbyists
- Home of the ArduPilot project
- Emphasis on beginner friendly
ArduPilot Robustness

- Monolithic design
- Platform-specific C/C++
- Hobbyist use-cases
  - No communication security, fault-tolerance
  - But being adopted in security-critical environments
- No regimented testing/verification story
The Hardware Abstraction Layer (HAL)

Gave back to the open-source community.
The foundation for ArduPilot now.
Designing a Language for Safety and Security

Design goal: give the programmer a few centimeters less rope than required to hang herself

- Help ensure
  - Memory safety
  - Timing safety (i.e., easier WCET analysis)
  - Functional correctness

- While being flexible:
  - bit-data manipulation
  - memory-area manipulation
  - “escaping” to/interop with C
  - readable generated code
Just...No.

Stateflow model of Tetris game (included in the Stateflow Demo models from the Mathworks!).

Diagram is essentially a control-flow graph of a program that implements tetris.

*Much* harder to read and modify than an equivalent program.

Model © The Mathworks, 2007
© 2013 Galois, Inc. All rights reserved.
Haskell

- Strong, static, polymorphic type checking and inference
- Pure, higher-order language—no side effects
- Functional programming for modularity: program composition is function composition

What if...

Can we have the high-level abstractions and type-safety of functional programming in embedded systems programming?

Approaches:

• Design a new FP-inspired language/compiler from scratch? **No:**
  • Would take too long
  • No library support

• Take the Haskell compiler and pair it down? **No:**
  • The runtime system is 50KLOCs of C/C--
  • And there's little control over memory usage (it's lazy) and it's a hog--"hello world" takes over 1MB
EDSL

- Building a programming language is hard!
- Get your programming language features for free:
  - Syntax & Parser
  - Type Checker
  - Macro language is type-safe and Turing-complete

“Just” a powerful Haskell library

Haskell

Ivory Language

Ivory Compiler

Ivory language: 2.5KLOCs
Ivory compiler: 1.2KLOCs
Compiling and Running Ivory

User Ivory Program

Ivy Compiler

.hs

Compile

Executable

Code Generator

Generated Code

.C

Make

Build Script

Compile

User Libraries

.RTOS

Run

Firmware

© 2013 Galois, Inc. All rights reserved.
Who's Used EDSLs?

- Eaton: garbage truck controllers
- Boeing: component configuration
- Ericsson: DSP
- Xilinx: FPGA synthesis
- Soostone: high-speed trading
- ...
Ivory Example

Loop over an array adding $x$ to each element:

```ivory
arrayExample :: Def('[ Ref s (Array 4 (Stored Uint8))
                      , Uint8
                      ] :-> ()
arrayExample = proc "arrayExample"
  $ \arr x -> body
  arrayMap
  $ \ix -> do
    v <- deref (arr ! ix)
    store (arr ! ix) (v + x)
```

Type automatically inferred

Map over the elements of the array

Guaranteed dereference $\arr$ at $\ix$

Store $v+x$ at index $\ix$
Haskell as Type-Safe Macro Language

```
arrayExample :: Def('[ Ref s (Array 4 (Stored Uint8)) 
         , Uint8 ] :-> ()
)
arrayExample = proc "arrayExample"
  $ \arr x -> body
  $ arrayMap (arrAdd arr x)
  $ \ix -> do
    v <- deref (arr ! ix)
    store (arr ! ix) (v + x)

arrAdd :: (Num a, SingI len, IvoryStore a)
  => Ref s (Array len (Stored a))
  -> a
  -> Ix len
  -> Ivory eff ()
arrAdd arr x ix = do
  v <- deref (arr ! ix)
  store (arr ! ix) (v + x)
```

Type-safe Haskell function call:  
No overhead in generated code

And arbitrary data-types  
Can be used for arbitrary-length arrays
Macros, Example 2

```
data Cond eff = Cond IBool (Ivory eff ())

(==>) = Cond

cond [] = return ()

cond (Cond b f : cs) = ifte_ b f (cond cs)
```

```
ifte (x >? 100)
  (store result 10)
  (ifte (x >? 50)
    (store result 5)
    (ifte (x >? 0)
      (store result 1)
      (store result 0)))
```

```
cond
[  x >? 100  ==>  store result 10
   ,  x >? 50  ==>  store result 5
   ,  x >? 0   ==>  store result 1
   ,  true     ==>  store result 0
]
```
Ivory Memory-Safety

• No null pointer dereferences
• No out-of-bounds array-indexing
• No unsafe implicit casting
• No unexpected type coercions—even satisfying the C standard!

Distilled ArduPilot bug discovered by Galois:

```c
... uint8_t a = 10;
uint8_t b = 250;
printf("Answer: %i, %i", a-b > 0, (uint8_t)(a-b) > 0);
...
```

Answer: 0, 1
Assuming int > uint8_t
Ivory: What We Removed

• No heap allocation (only stack)
• Unbounded looping combinators
  Except for a single forever combinator
• void type
• Machine-dependent sizes (modulo float, double)
• Side-effecting expressions
• Pointer arithmetic
Ivory: What We Added

• Effect types
  • Allocation effects: “This function can't (stack) allocate memory”
  • Escape effects: “No break is allowed in this loop”
  • Return effects: “This macro cannot contain a return statement”

• References (guaranteed non-null pointers)

• Array map/fold combinators

• Automatic assertions
  • arithmetic underflow/overflow
  • div-by-zero
  • user-specified assertions
Ivory: TBD

• Sum types (unions)
• Fat pointers/strings
• Function pointers
• A better module system
• Interpreters for embedded software
Tower
Tower: a Glue Code Macro Language

- Goal: address the “glue code” problem: task initialization and communication.
  - Specifies how tasks are scheduled and communicate
    - Pub/sub model
  - Provides both time-triggered and event-triggered behaviors
  - Channels (queues) and data-ports (shared data) communication
  - Able to specify both interrupt handlers and user tasks
- Tower is “just” Ivory macros so has all the type-safety guarantees of Ivory—and no new code generator!
Tower example

Signal task \(\rightarrow\) On/off? \(\rightarrow\) LED hardware controller task
Signal Task

```haskell
blink :: SingI n => ChannelSource n (Stored IBool) -> Task ()
blink chan = do
  tx <- withChannelEmitter chan "bTx"
  onPeriod period (body tx period)
  where period = 100 :: Integer

body :: (SingI Nat n, GetAlloc eff ~ Scope cs)
  => ChannelEmitter n (Stored IBool)
  -> Integer
  -> a
  -> Ivory eff ()
body tx period currTime = emitV_ tx (even currTime)
  where
    even = currTime .% (2*p) <? p
    p    = fromIntegral period
```

- **Tower**: Specify the output channel
- **Ivory**: Specify when computation takes place
- **Ivory**: What the task actually does
- **Send 0,1,0,1 ...**
SMACCMMPilot
The Hardware

- ArduPilot Mega Hardware (Legacy)
  AVR Processor: 8 bit, 16MHz, 8k RAM, 256k Flash

- PX4 Hardware (SMACCMPilot)
  ARM Cortex M4 Processor: 32 bit, 168Mhz, 192k RAM, 1024k Flash

SMACCMPilot Architecture

Approx. 5x code generation
SMACCMPilot
An Embedded Systems Software Research Project
We're building open-source autopilot software for small unmanned aerial vehicles (UAVs) using new high-assurance software methods.

The SMACCMPilot autopilot software:
Hardware Guide
Complete instructions for building a SMACCMPilot based quadcopter.

Software Guide
Learn about how the SMACCMPilot software platform works, and how to develop for it.

Open Source
The SMACCMPilot platform is an open-source project, released under a liberal BSD license.

And the technology used to build it:
Ivory Language
SMACCMPilot is the flagship project of a new programming language called Ivory, a domain specific language for safe systems programming.

Ivory Tutorial
Walk through an Ivory program with annotations introducing some of the features of the language.

Tower Framework
Tower is a framework for composing Ivory programs into multithreaded applications.
Lessons Learned/Open Problems

• Memory safety isn't a panacea
  • We still test/debug/verify
  • Traceability from DSLs to object code is necessary
  • But the kinds of bugs is restricted: seg-faults, memory leaks don't happen

• EDSL shortcomings:
  • Reusing a general-purpose type-checker
  • Requires host-language knowledge
  • Abstractions/macros can affect performance
  • Compilation cycle

• Interpreters for embedded systems are hard
• Have not proved architectural properties or verified controllers
Questions