11+ Years of Formal Methods at Galois, Inc.

Lee Pike

Galois, Inc.
leepike@galois.com

A survey of work by many contributors and many stolen slides
Galois Background

- Spun out of university research in functional programming and formal methods.
  - Started in 1999.
  - 30+ employees, primarily technical.
  - Based in Portland, Oregon, USA.
- Technology transition company.
  - Outsourced R&D, product incubation, technology licensing.
  - High assurance software engineering.
We’ve used a few formal methods...

- ACL2
- Isabelle/HOL
- Agda
- PVS
- SAL
- NuSMV
- Yices
- Frama-C
- CBMC
- MiniSAT
- ABC

... and tools we’ve built in-house.

This is a story about the right tool for the right job.

We’ll look at these through application domains.
Domain Specific Languages (DSLs) are high level languages for design capture in particular problem areas.

A program in the DSL is an unambiguous specification that:

- guides and documents implementations;
- can be executed to generate test vectors;
- can be compiled directly to an implementation.

**Ideal Situation:** Reason at a high level about a program in the DSL, and the properties also apply to the low-level implementation.
Cryptol: the Language of Cryptography

Cryptol is a declarative language for describing crypto algorithms.
- Primitives for operations on blocks and streams of bits.
- No assumptions are made about the implementation platform.

An expressive type system ensures consistency.

Download the Cryptol interpreter and give it a try:

http://www.cryptol.net
The Cryptol type system captures important details of interfaces.

The type system is Hindley-Milner plus arithmetic constraints.

Numeric literals are one source of constraints:

\[ 13 : \{ a \} (a \geq 4) \Rightarrow [a] \]

“The literal 13 is represented by a bit vector that requires at least 4 bits to represent”
From the Advanced Encryption Standard:

3.1 Inputs and Outputs

The input and output for the AES algorithm each consist of sequences of 128 bits (digits with values of 0 or 1). These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. The Cipher Key for the AES algorithm is a sequence of 128, 192 or 256 bits. Other input, output and Cipher Key lengths are not permitted by this standard.

In Cryptol:

```
blockEncrypt : {k} (k >= 2, 4 >= k) => ([128], [64*k]) -> [128]
```

“For all k between 2 and 4, first input is a sequence of 128 bits, second input is a sequence of 128, 192 or 256 bits, output is a sequence of 128 bits.”
Cryptol to FPGA Verification

Refine spec for a specific target

Create an FPGA implementation from the target specification

Key
- Galois tools
- Xilinx tools
- Cryptol files
- Formal Models
- Data files
- Input to tool
- Input to designer

Lee Pike
11+ Years of Formal Methods at Galois, Inc.
Cryptol Formal Verification

- Fully automated proving.
  - Safety: no division-by-zero, no index-out-of-bounds.
  - Security: no compiler backdoors.
  - But requires complete unwinding.

- Have a translator to Isabelle/HOL for interactive proving.

- Same techniques apply to general-purpose programming languages.
Cryptol: Variations on a Theme (μCryptol)

- A verifying compiler targeting Rockwell Collins’ AAMP7 microprocessor.
  - Certified EAL7.
  - Intrinsic partitioning: “separation kernel in hardware”

- Automated compiler proofs in ACL2, based on RC’s microprocessor model.

- μCryptol’s type system ensured termination.
Cryptol: Variations on a Theme (SBV)

- A Haskell DSL that efficiently translates to SMT (SMT-Lib).
- Open-source (BSD3), developed by Levent Erkök.
- Try it out: http://hackage.haskell.org/package/sbv
- C code-generator.

```haskell
> :m Data.SBV
> prove $ $(x :: SWord8) \rightarrow x \cdot \text{shiftL} \ 2 \ == \ 4 \cdot x
Q.E.D.
> prove $ forAll ["x"] $(x :: SWord8) \rightarrow x \cdot \text{shiftL} \ 2 \ == \ x
Falsifiable. Counter-example:
x = 128 :: SWord8
```
Typical Security Question: What are the possible effects of changes at the attack surface on the critical data?

Answering this requires an understanding of how information flows through the program.
Analysis is done on a simple flow language:

- The following front end processing is performed to translate C code to the flow language:
  - **Preprocessing:** The C preprocessor.
  - **Parsing:** The Haskell Language.C package.
  - **Simplification:** Normalizing expressions (like CIL).
  - **Variable Classification:** Special handling for address-taken locals and dynamically allocated memory.
  - **Pointer Analysis:** Anderson’s algorithm replaces each indirect reference with a set of direct references.

**Key Property:** The front end processing is *conservative*.

- Every information flow in the C code is translated to an information flow in the flow language.
- **Assumption:** the C code is memory safe.
Visualization

- Information flow is half the battle.
- How do you analyze the flows?
TSE: A Cross Domain Filestore with Read-Down

High Users

High Network

High users/applications see integrated web/filestore with low and high content together

Trusted Service Engine (TSE)
Cross-domain file store

WebDAV, HTTP

Secure read-down

Low Users

Low Network

Low users/applications see simple web/filestore with low content only

Authorization / Authentication Service

2-4 networks
TSE Architectural Principles

1. Factor the security architecture.
2. Minimize the number of components requiring high assurance.
3. Keep each as simple as possible.
4. Use formal methods in critical places.
The BAC directs accesses across disk drives at multiple levels.

- High assurance component.
- Must eliminate data channels between levels.
- Must control timing channels between levels.
The overall security goal is **non-interference**: a partition’s state is not dependent on the actions of any higher level partition.

Formalize von Oheimb’s theory of non-interference in the Isabelle theorem prover.

Implement the BAC as an Isabelle function.

**Non-Interference Verification**: Complete an Isabelle proof that the BAC function satisfies non-interference.

**Model-to-Code Correspondence**: Pretty-print the 800 line C implementation of the BAC from the Isabelle implementation.
Non-Interference Verification

- Memory-safety proof—prove the absence of:
  - Out-of-bounds array access
  - Out-of bounds disk block ID
  - Access to memory undergoing DMA transfer
  - Multiple simultaneous DMA transfers to same memory region
  - Too many simultaneous DMA transfers to a single disk

- Non-Interference proof:
  - Assumes safety.
  - Prove that higher-level actions have no effect on lower levels.
Proof Technique

1. Guess invariants.
2. Try to prove the resulting verification conditions.
3. On all times except the last, fail and go back to step ??.
Machine Learning

- How do you know when you’re done training?
- How do you know that given some training, some input won’t violate your classification constraints?

A verified learner can be used in a critical system to improve optimization.
For decision tree learners, we’ve used SMT/NuSMV to check the output of the learners.

Applied to the temperature-control system for a simulated environment. Can we prove the temperature remains in a safe region?

Next-steps: use non-linear solvers to verify learning for more complex systems.
If the majority of the three engine temperature probes has exceeded 250 degrees, then the cooler is engaged and remains engaged until the temperature of the majority of the probes drop to 250 degrees or less. Otherwise, trigger an immediate shutdown of the engine.

\[
\begin{align*}
\text{engineMonitor} & = \text{do} \\
& \quad \text{trigger} \ "\text{shutoff}" \ (\text{not ok}) \ [\text{arg maj}] \\
& \quad \text{where} \\
& \quad \text{vals} \quad = \text{map} \ \text{externW8} \ [\"\text{tmp\_probe\_0}\", \"\text{tmp\_probe\_1}\", \"\text{tmp\_probe\_2}\"] \\
& \quad \text{exceed} \quad = \text{map} \ (< 250) \ \text{vals} \\
& \quad \text{maj} \quad = \text{majority exceed} \\
& \quad \text{checkMaj} \ = \text{aMajority exceed} \ \text{maj} \\
& \quad \text{ok} \quad = \text{alwaysBeen} \ ((\text{maj} \ &\ & \text{checkMaj}) \implies \text{extern} \ "\text{cooler}"")
\end{align*}
\]
Copilot: a Runtime Monitoring

Libraries

Copilot specification language

Core language

QuickCheck testing

Interpreter

LTL
ptLTL
Regular expressions
clocks
fault-tolerance etc.

Hard real-time back-end + scheduler (C)

Hard real-time back-end (C)

... Kind, other code generators

Type-checking, causality analysis, etc.

CBMC: (C bounded model-checker)
Monitoring an Airspeed System

- Northwest Orient Airlines Flight 6231 (1974)—3 killed
  Increased climb/speed until uncontrollable stall.
- Birgenair Flight 301, Boeing 757 (1996)—189 killed
  One of three pitot tubes blocked; faulty air speed indicator.
- Aeroperú Flight 603, Boeing 757 (1996)—70 killed
  Tape left on the static port(!) gave erratic data.
- Líneas Aéreas Flight 2553, Douglas DC-9 (1997)—74 killed
  Freezing caused spurious low reading, compounded with a failed alarm system.
  Speed increased beyond the plane’s capabilities.
- Air France Flight 447, Airbus A330 (2009)—228 killed
  Airspeed unclear to pilots. Still under investigation.
Formal methods is being actively used at Galois as part of our mission to “create trustworthiness in critical systems”.

We’re often a consumer of FM tools, building on top of them.

The team:

- **Present**: Joe Hurd (lead), Lee Pike, Iavor Diatchki, Aaron Tomb, Joe Hendrix, John Launchbury, Eric Mertens, Rogan Creswick, Joel Stanley.
- **Past**: John Matthews, Paul Graunke, Levent Erkök.

leepike@galois.com
http://www.galois.com

Shameless plug: FMCAD’11 invited panel on FM & control systems (Nov. 2).