Experience Report: a Do-it-Yourself High-Assurance Compiler

Lee Pike  Galois, Inc. <leepike@galois.com>
Nis Wegmann  University of Copenhagen
Sebastian Niller  Unaffiliated
Alwyn Goodloe  NASA Langley Research Center
Do-It-Yourself
Do-It-Yourself

© 2012 Galois, Inc.
Do-It-Yourself

Etsy

Night Bracelet

Pitting Sterling Silver and Aqua Crystal Swarovski Crystals.

$94.00 USD

Delicate Ball Chain, Neon Pink, and Silver Ball Chain Bracelet.

$15.95 USD

Textured Ribbon Pendant

Buntheman

$18.00 USD

Necklace "tree of words"

SGSTAGTH

$20.00 USD

Leaf Branch and Metal Bird Charm Pendant.

$16.50 USD

Sterling Silver Chain & Filigree Charm.

Sensa You

$22.00 USD

© 2012 Galois, Inc.
High-Assurance

Compilers
Principles, Techniques, and Tools
Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman

© 2012 Galois, Inc.
3 Not-So-Secret Weapons

1. Embedded domain-specific languages (EDSLs)
2. A verifying (not verified) compiler approach
3. Open source testing/verification libraries & tools
National and Space Administration
National Aeronautics and Space Administration
Copilot: a Run-Time Monitoring DSL

- Embedded DSL in Haskell
- Synthesize monitors for real-time embedded systems
- Stream language
- Generates Misra-like C
- Constant time, constant memory
  - Synthesized scheduler
  - No RTOS needed
Sample Copilot specification

### Haskell

```
fib :: [Word32]
fib = [0, 1] ++ zipWith (+) fib (drop 1 fib)
```

### Copilot

```
fib :: Stream Word32
fib = [0, 1] ++ (fib + drop 1 fib)
```

Special constructs for input (*sampling*) and output (*triggers*)
Copilot Architecture

Libraries

- bounded LTL
- ptLTL
- regular expressions
- clocks
- fault-tolerance
- etc.

Copilot specification language

Domain-specific type-checking

Interpreter

Core language

Atom back-end
(C)

SBV back-end
(C)
Lessons in DIY Assurance

- Who monitors the monitor?

- Challenges:
  - EDSLs encourage rapid language design changes
  - Industrial work often doesn't “pay” for assurance (but wants it)
Lessons in DIY Assurance

Solution: DIY assurance

- Turing incomplete DSLs, Turing complete macros
- Multi-level type-checking
- Cheap testing & proofs
- Unified host language
Lesson #1: Turing-Incompleteness

Turing in completeness means:

- Compiler writing is simplified
- Compiler reasoning is better (e.g., termination analysis)
- Security is improved
- Automated verification has a chance of working!
Lesson #1: Turing-Incompleteness

Turing incompleteness means:

- Compiler writing is simplified
- Compiler reasoning is better (e.g., termination analysis)
- Security is improved
- Automated verification has a chance of working!

Have your cake and eat it, too:
In an embedded DSL, the host language is Turing-complete!

Programs specialized at compile time.

© 2012 Galois, Inc.
Lesson #2: Multi-Level Type-Checking

- Lean on Haskell's type system in the (DSL's) compiler's internal representations: e.g., GADTs
  - Leave the type system twice:
    - Pretty-print C
    - Translating between EDSLs (type-safe dynamic typing*).
  - And ensure you aren't abusing it: **Safe Haskell**

Lesson #2: Multi-Level Type-Checking

- Lean on Haskell's type system in the (DSL's) compiler's internal representations: e.g., GADTs
  - Leave the type system twice:
    - Pretty-print C
    - Translate between EDSLs (type-safe dynamic typing*).
  - And ensure you aren't abusing it: **Safe Haskell**

- Then a little domain-specific type-checking:
  - Productiveness:
    - Rejected: \[ x :: \text{Stream Word64} \]
    \[ x = [0] ++ \text{drop 1} \ x \]
  - Inputs are consistently typed (e.g., external functional calls)

Lesson #3: Cheap Testing & Proofs

QuickCheck:
- Small DSLs make program generation easy with good coverage
- Test ~1.5M programs/day
Lesson #3: Cheap Testing & Proofs

QuickCheck:
- Small DSLs make program generation easy with good coverage
- Test ~1.5M programs/day

Then **prove** back-ends agree:
- Model-checking works (better) with Turing incomplete DSLs
- EDSL simplifies driver generation

http://www.cprover.org/cbmc/
Lesson #4: a Unified Host Language

Embedded DSLs are a paradigm shift for safety-critical languages

- Fewer front-end, type-checker bugs
- “Bolting-on” new tools within the type system (no marshalling)
- The macro language is a build system, too!
Conclusions
Conclusions

Verified compiler

- Expensive
- Specialized skills
- Hard to make repairs
- But flawless when it works
Conclusions

Verified compiler

- Expensive
- Specialized skills
- Hard to make repairs
- But flawless when it works

© 2012 Galois, Inc.
Conclusions

Verified compiler

- Expensive
- Specialized skills
- Hard to make repairs
- But flawless when it works

DIY assurance

- Cheap
- Quick to build
- Easy to repair
- An “90% solution”


© 2012 Galois, Inc.