It's QEDs All the Way Down

David S. Hardin   Konrad L. Slind

Trusted Systems Group
Advanced Technology Center
Rockwell Collins

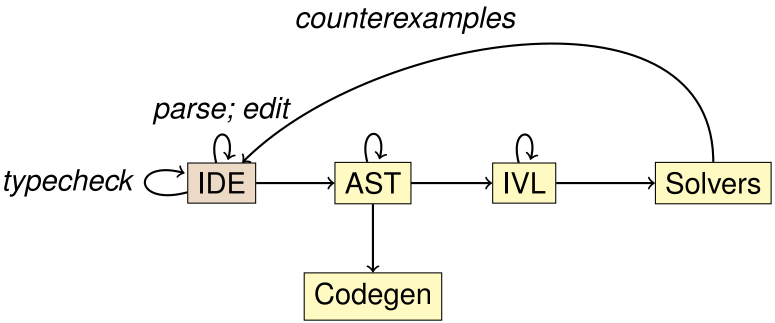
© 2016 Rockwell Collins. All rights reserved.

Introduction

- Traditional Formal Verification efforts have suffered from a fundamental issue: verification artifacts are only models of the system that is to be verified
  - Many translators from engineering artifacts have been built; few are themselves verified
- Further, there has been no verified path from an executable specification expressed in a theorem proving environment to machine code
  - Many systems offer code generation, but not verified code generation
- But what if such a verified path existed, from engineering artifact to formal reasoning to code generation, allowing “QEDs all the way down”?
- This talk reports on our experiences with such an environment

© 2016 Rockwell Collins. All rights reserved.

Verification Tool Design Pattern



- IDE = Interactive Development Environment
- AST = Abstract Syntax Tree
- IVL = Intermediate Verification Language

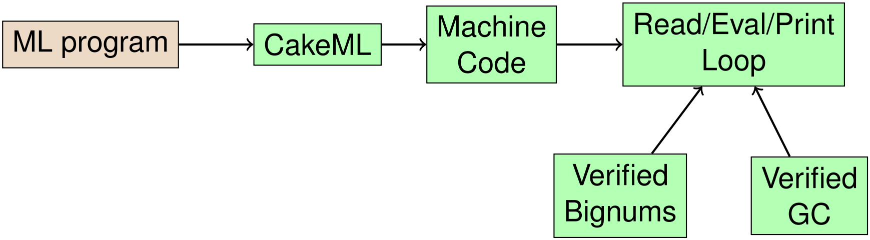
© 2016 Rockwell Collins. All rights reserved.

CakeML — `https://cakeml.org`

- CakeML is a functional programming language with a proven-correct compiler and runtime system
- CakeML is based on a substantial subset of Standard ML
- Its semantics is specified in higher-order logic
- The compiler algorithm, also specified in higher-order logic, has been proven to transform CakeML programs to semantically equivalent machine code
- The compiler implementation is a CakeML program that has been verified by a bootstrap method that executes the compiler specification on the compiler implementation to create a proven-correct compiler binary
- The correctness proofs use validated instruction set models

© 2016 Rockwell Collins. All rights reserved.

CakeML Use Case 1: Verified ML Compile, REPL



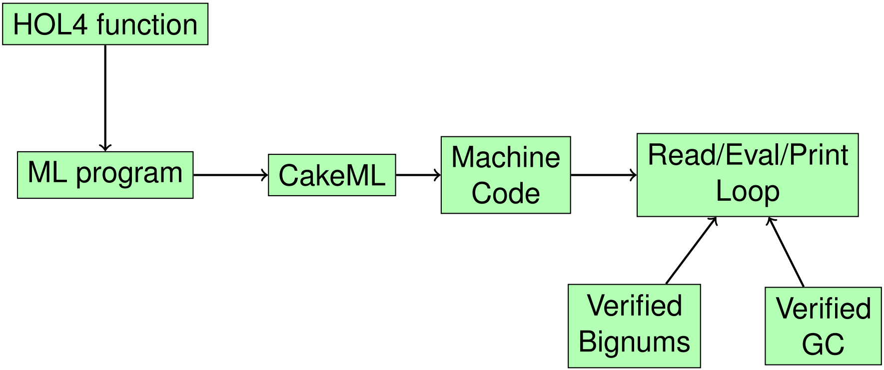
© 2016 Rockwell Collins. All rights reserved.

CakeML Use Case 1

- CakeML is utilized much as any other language compiler
- Code generation is not yet on par with production compilers, but improving
- Read/Eval/Print Loop utilized for execution and validation testing
- Verified runtime support, specifically for bignums and Garbage Collection
- x86, ARM, PowerPC instruction sets supported

© 2016 Rockwell Collins. All rights reserved.

CakeML Use Case 2: Verified HOL Compile, REPL



© 2016 Rockwell Collins. All rights reserved.

From Logic to ML to Machine Code by Proof

Given : ML bigstep evaluation relation **evaluate**
Define: "Evaluating *exp* in environment *env* yields a value *v* with property *P*" :

$$\textbf{Eval}\ \mathit{env}\ \mathit{exp}\ P = \exists v.\ \textbf{evaluate} \ldots \mathit{env} \ldots \mathit{exp}\ (\ldots v) \wedge P(v)$$

P is used to relate *values* resulting from ML computation with the corresponding logical objects (which stem from the original logic definitions)

© 2016 Rockwell Collins. All rights reserved.

Example Translation Theorem

Theorem about function application

$$
\begin{array}{l}
\vdash \textbf{Eval}\ \mathit{env}\ e_1\ ((a \longrightarrow b)\ f) \land \textbf{Eval}\ \mathit{env}\ e_2\ (a\ x) \\
\quad \Rightarrow \\
\textbf{Eval}\ \mathit{env}\ (e_1\ e_2)\ (b\ (f\ x))
\end{array}
$$

"If e_1 evaluates to a value denoted by function f, having logical type $a \rightarrow b$; and e_2 evaluates to an element x having logical type a, then $(e_1\ e_2)$ evaluates to a value corresponding to $f\ x$, having logical type b."

Such theorems are proved for all the AST constructors, and used in a bottom-up, syntax-directed manner for new function definitions

© 2016 Rockwell Collins. All rights reserved.

Examples of Function $\Rightarrow$ ML Translation

In the CakeML distribution:

- (Okasaki) Queues (Bankers, Batched, Hood-Melville, Implicit, Physicists, Real-time)
- (Okasaki) Heaps (Binomial, Leftist, Pairing, Splay)
- Ordered Sets (Red-Black, Unbalanced)
- Crypto algorithms (AES, RC6, TEA)
- Others (primality tester, copying garbage collector, parser generator)
- CakeML compiler itself

Our work: regex compiler (90 definitions, 650 lines of ML text)

© 2016 Rockwell Collins. All rights reserved.

Example: Splay Heads

$$\textbf{datatype } \alpha \textbf{ heap} = \textbf{Empty} \mid \textbf{Tree } (\alpha \textit{heap})\ \alpha\ (\alpha \textit{heap})$$

- Splay trees are a close relative of balanced binary search trees, but they maintain no explicit balance information.
- Instead, every operation blindly restructures the tree using some simple transformations that tend to increase balance.
- Every operation runs in O(log n) amortized time.
- Well suited for implementing heaps.
- Operations: **insert**, **merge**, **findMin**, **delMin**

© 2016 Rockwell Collins. All rights reserved.

Some Splay Heap Theorems (Proved in HOL4)

$$
\begin{array}{l}
\vdash [\![\textbf{insert}\ x\ H]\!] = \{x\} \oplus [\![H]\!] \\
\vdash [\![\textbf{merge}\ H_1\ H_2]\!] = [\![H_1]\!] \oplus [\![H_2]\!] \\
\vdash [\![\textbf{delMin}\ H]\!] = [\![H]\!] \setminus \{\textbf{findMin}\ H\} \\
\vdash \textbf{isHeap}\ H_1 \wedge \textbf{isHeap}\ H_2 \Rightarrow \textbf{isHeap}\ (\textbf{merge}\ H_1\ H_2) \\
\vdash H \neq \textbf{Empty} \wedge \textbf{isHeap}\ H \Rightarrow \textbf{isHeap}\ (\textbf{delMin}\ H) \\
\vdash H \neq \textbf{Empty} \wedge \textbf{isHeap}\ H \\
\quad \Rightarrow \textbf{findMin}\ H \in [\![H]\!]\ \wedge \forall y. y \in [\![H]\!] \Rightarrow \textbf{findMin}\ H \leq y
\end{array}
$$

- $[\![-]\!]$: map to multiset;
- $\oplus$: multiset union;
- $\setminus$: multiset difference

Example Translation : delMin

Removes smallest element from the heap, possibly doing some rebalancing. The smallest element is leftmost in the tree. Expressed in stylized HOL as follows:

$$
\begin{array}{l}
\textbf{delMin}\ (\textbf{Tree Empty}\ x\ b) = b \\
\textbf{delMin}\ (\textbf{Tree}\ (\textbf{Tree Empty}\ x\ b)\ y\ c) = \textbf{Tree}\ b\ y\ c \\
\textbf{delMin}\ (\textbf{Tree}\ (\textbf{Tree}\ a\ x\ b)\ y\ c) = \textbf{Tree}\ (\textbf{delMin}\ a)\ x\ (\textbf{Tree}\ b\ y\ c)
\end{array}
$$

Translation theorem:

$$
\begin{array}{l}
\vdash \textbf{Eval}\ \mathit{env}\ (\textbf{Var}\ \mathit{delMin}) \\
\qquad (((\textbf{SPLAYHEAP}\ a)\ x \longrightarrow \textbf{SPLAYHEAP}\ a)\ \textbf{delMin})
\end{array}
$$

Constraint: x is a splayheap on which **delMin** is defined (patterns are not complete)

© 2016 Rockwell Collins. All rights reserved.

Generated ML for delMin

```
fun delMin x =
 case x
  of Empty => raise Bind
   | Tree(v9,v8,v7) =>
      case v9
       of Empty => v7
        | Tree(v6,v5,v4) =>
             case v6
               of Empty => Tree(v4,v8,v7)
                | Tree(v3,v2,v1) =>
                    Tree(delMin (Tree(v3,v2,v1)),
                         v5,
                         Tree(v4,v8,v7));
```

© 2016 Rockwell Collins. All rights reserved.

Application to Imperative Languages: Guardol

Guardol is a Domain-Specific Language for cross-domain guards:

- Provides a single language to program many different guards
- Integrates highly automated formal verification with development
- Supports high-assurance code generation
- Guardol is a traditional imperative language, but with ML-style pattern matching
- Regular Expression matching supported via the regex_match primitive; verified compilation of regex_match to DFAs via Brzozowski's derivative method accomplished in HOL4

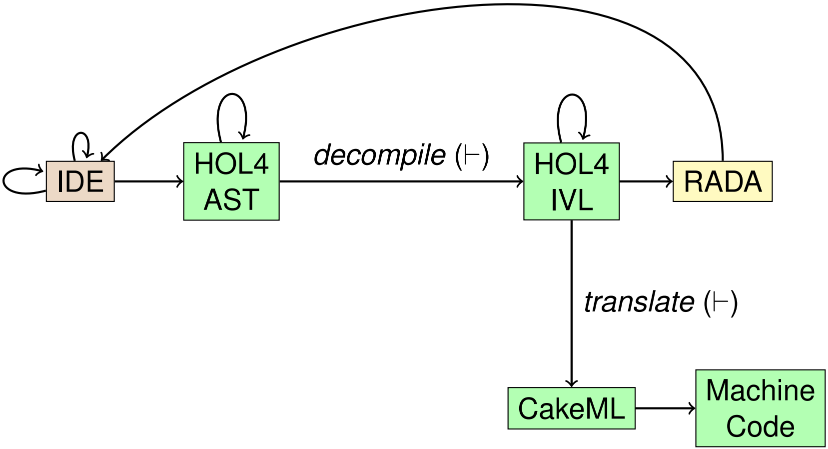
© 2016 Rockwell Collins. All rights reserved.

Sound Code Generation for Guardol

- Decompilation maps from Guardol operational semantics to HOL datatypes and functions (By formal proof)
- Translation maps from HOL datatypes and functions to ML datatypes and functions (By formal proof)
- By transitivity we obtain a verified map from Guardol to CakeML, and so to binary (By formal correctness proof of CakeML compiler (POPL 2014))
- By the proofs, we have that the behavior of the binary in the CakeML REPL has the properties proved about the Guardol source program

© 2016 Rockwell Collins. All rights reserved.

Guardol Sound Code Generation Toolchain



© 2016 Rockwell Collins. All rights reserved.

Results

- We have utilized verified regex DFA compilation to produce high-assurance, high-performance hardware-based regular expression guards
  - Able to guard UDP packets at Gigabit Ethernet line speed rate
- We generated verified x86 binaries for regular expression guards using CakeML, and validated the x86 code using test cases executed in the CakeML REPL
- We have instantiated the Verification Tool Design Pattern for other imperative languages, namely sizable subsets of the Swift and Rust languages

© 2016 Rockwell Collins. All rights reserved.

Current Work: Regex Extensions I

- To support numeric intervals we added the following “interval” form to the regex parser:

  ```
  \i{lo,hi}
  ```

  allowing much more precise time specs, *e.g.*:

  ```
  \i{1,31}\i{1,12}\i{1970,2025}\i{0,23}\i{0,59}\i{0,59}
  ```

- Intervals can utilize a variety of number representations: *e.g.*, `255` can take either 3 bytes (ASCII) or 2 bytes (twos-complement integer) or 1 (unsigned)
- Capturing intervals with regexs is by no means original (J.R. Büchi wouldn’t have been surprised by this in 1960)
- Doesn’t seem commonly supported in regex packages, which require monstrous regexs to match even simple intervals

© 2016 Rockwell Collins. All rights reserved.

Current Work: Regex Extensions II

- We want to check the generated monstrous regexs via proof
- The above regex (call it r) generates the proof obligation

$$
\begin{array}{l}
\forall s.\ \textbf{regex\_match}(r, s) \iff \\
\quad \exists w_1 w_2 w_3 w_4 w_5 w_6. \\
\qquad s = w_1 w_2 w_3 w_4 w_5 w_6 \ \wedge \\
\qquad 1 \leq \mathbb{N}(w_1) \leq 31 \ \wedge \\
\qquad 1 \leq \mathbb{N}(w_2) \leq 12 \ \wedge \\
\qquad 1970 \leq \mathbb{N}(w_3) \leq 2025 \ \wedge \\
\qquad 0 \leq \mathbb{N}(w_4) \leq 23 \ \wedge \\
\qquad 0 \leq \mathbb{N}(w_5) \leq 59 \ \wedge \\
\qquad 0 \leq \mathbb{N}(w_6) \leq 59
\end{array}
$$

- Which should be automatically proved (work in progress)

© 2016 Rockwell Collins. All rights reserved.

Current Work: Regex Extensions III

- This *translation-validation* style approach extends the assurance story of the regex compiler to the extended language including intervals
- Currently applying to generate a software guard for GPS messages over CANBUS
- Other applications: filtering UTF-8 encoded strings, scenarios requiring “full packet inspection” of messages involving numbers

© 2016 Rockwell Collins. All rights reserved.

Future Work

- Explore use of refinement in the theorem prover, similar to Eric Smith's work, to produce higher-performance verified binaries from specifications in logic
- Explore use of idioms such as ACL2's single-threaded object (stobj) syntactic restrictions to allow in-place updates
- Continue to work with CakeML team to improve the compiler, and support other mainstream languages

© 2016 Rockwell Collins. All rights reserved.

THE END

© 2016 Rockwell Collins. All rights reserved.