Certified Web Services in Ynot:
Programming a web application in a proof assistant

Ryan Wisnesky, Gregory Malecha, Greg Morrisett
Harvard University

Special thanks to Adam Chlipala

July 17, 2009
Goal

- Build a *course gradebook* with strong behavioral guarantees
  - Verify application logic, privacy, I/O behavior
Gradebook Web Application

- Role-based access control:

<table>
<thead>
<tr>
<th>Role</th>
<th>Read</th>
<th>Write</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students</td>
<td>Self</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>TAs</td>
<td>Section</td>
<td>Section</td>
<td>All</td>
</tr>
<tr>
<td>Professors</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>

```
/home>coqc gbsrv.coq
/home>gbsrv grades.dat
Loading grades...
Listening on port 80...
Parsing request...
Checking privacy...
Computing class average...
```

![Diagram of HTTP Request and Response]
Outline

- The Coq Language
  - Imperative programming in Coq
  - Our I/O and networking extensions

- The Gradebook server
  - Specification and implementation
  - Verification overhead
Core ML
+
Specifications as Types
=
Software that is correct by construction
Append in Coq

```coq
data List A = Nil
  | Cons A (List A)

++: \forall A, List A \to List A \to List A
Nil ++ L = L
(Cons a b) ++ L = Cons a (b ++ L)
```
Theorem Proving in Coq

**Theorem** append_associate:

\[ \forall L_1 \ L_2 \ L_3, \]

\[ (L_1 \ ++ \ L_2) \ ++ \ L_3 = L_1 \ ++ \ (L_2 \ ++ \ L_3). \]

**Proof.**

**Induction** \( L_1 \).

... 

Qed.
Benefits of Coq

- Very small proof checker (100s of lines)
- Lightweight, pay as you go verification
- Specification, implementation, and proof of correctness written in the same language
- Extracts to ML
Limitations of Coq

- Coq code must be purely functional, terminating.
- But we need imperative features like general recursion and I/O.
  - Create a type of imperative commands
  - Hoare logic for reasoning about mutation
  - Separation logic for reasoning about memory
Definition swap (p₁ p₂ : ptr) (n₁ n₂ : Nat) :
  Cmd (p₁ → n₁ * p₂ → n₂)
    (fun r : Unit => p₁ → n₂ * p₂ → n₁) :=
  v₁ ← read p₁ ;
  v₂ ← read p₂ ;
  write p₁ ::= v₂ ;
  write p₂ ::= v₁

Generate and solve proof obligations
Definition echo (lsock : Socket) (tr : Trace) :
Cmd (traced tr) :
  (fun r : Unit => ∃ msg rsock, traced [Sent lsock rsock msg , Recd lsock rsock msg ++
          tr] ) :=
  (msg, rsock) ← recv sock ; send lsock msg rsock

Record events using traces.
We do not verify...

- Coq itself
- The ML implementations of our axioms
  - read, write, recv, send, etc
- Extraction from Coq to ML
- The ML compiler
Gradebook Web Application

<table>
<thead>
<tr>
<th>Role</th>
<th>Read</th>
<th>Write</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students</td>
<td>Self</td>
<td>None</td>
<td>All</td>
</tr>
<tr>
<td>TAs</td>
<td>Section</td>
<td>Section</td>
<td>All</td>
</tr>
<tr>
<td>Professors</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>

/home>coqc gbsrv.coq
/home>gbsrv grades.dat
Loading grades...
Listening on port 80...
Parsing request...
Checking privacy...
Computing class average...

Course Grades

Query: AVG [carol cpass] hw1
Result: 67

HTTP → Request

HTML → Response

Certified Application
Architecture

Generic

Certified Application

Store

Gradebook

Server

App Specific

Gradebook Spec

HTML

Many implementations possible

Trace-based I/O Spec

Database Spec

HTTP Request

Many implementations possible

Trace-based I/O Spec

Database Spec

HTTP Request
Gradebook Specification

- Parsing
- Wellformed?
  - Yes
    - Authorization
      - Privacy Respecting?
        - Yes
          - Query Type
            - Get
              - Results
            - Set
              - Mutation
          - Update Grade
          - Return Success
        - No
          - Return Grade
          - Return Average
          - Return Success
    - No
      - Invalid Request
      - Access Denied
Application Server Spec

Http Server

Accept \rightarrow Read Request

\text{Success} \rightarrow HTTP Parse

\begin{cases} 
\text{Invoke App} \\
\text{Fail} 
\end{cases}

\rightarrow Write
Store

- Queries: Select, update, etc
- Verify isomorphism between grades and tuples
- Linked-list implementation

**Theorem:** deserialize (serialize x) = x
Verification Overhead

<table>
<thead>
<tr>
<th></th>
<th>App Server</th>
<th>Parsing</th>
<th>App</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification (LOC)</td>
<td>414</td>
<td>184</td>
<td>231</td>
<td>154</td>
</tr>
<tr>
<td>Implementation</td>
<td>223</td>
<td>269</td>
<td>119</td>
<td>113</td>
</tr>
<tr>
<td>Proofs</td>
<td>231</td>
<td>82</td>
<td>564</td>
<td>99</td>
</tr>
<tr>
<td>Overhead</td>
<td>1.04</td>
<td>.3</td>
<td>4.74</td>
<td>.88</td>
</tr>
<tr>
<td>Compile-time (m:ss)</td>
<td>1:21</td>
<td>0:55</td>
<td>0:32</td>
<td>0:23</td>
</tr>
</tbody>
</table>
Conclusion

- You can program verified imperative software in Coq.
- Language level, “correctness by construction” techniques are scaling up.
- Future directions:
  - Concurrency
  - Other effects
  - Failure modes

ynot.cs.harvard.edu
Compilation


"Extraction"

Executable (native code)