Certified Systems Development in Ynot

Ryan Wisnesky, Gregory Malecha, Greg Morrisett
{ryan,gmalecha,greg}@cs.harvard.edu

Special thanks to Adam Chlipala
Outline

- A 3-tier web application in Ynot
  - Provably correct
  - Runs
  - Looks like ML

Beware this code, I have not run it, only proven it correct...
Our Web App: Gradebook

- Role-based access control

<table>
<thead>
<tr>
<th></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>

- Correctness depends on
  - Enforcing permissions
  - Responding correctly

- Code, Spec, Proof all written in Coq
Specification

- Parsing
- Authorization
- Valid?
  - Yes
    - Privacy Respecting?
      - Yes
        - Query Type
          - Get
            - Results
              - Return Grade
          - Set
            - Update Grade
              - Mutation
                - Return Success
            - Average
              - Return Average
    - No
      - Invalid Request
      - Access Denied
Architecture

HTTP Request

Certified Application

Store

Gradebook

Server

HTML

Gradebook Spec

Database Spec

Trace-based I/O Spec

Many implementations possible

Generic

App Specific

Generic App Specific

Many implementations possible
let swap (p1 p2 : int ref) : unit =
  let v1 = ! p1 in
  let v2 = ! p2 in
  p1 ::= v2 ;
  p2 ::= v1
```haskell
swap :: IORef Int → IORef Int → IO ()
swap p1 p2 = do v1 ← ! p1
              v2 ← ! p2
              p1 ::= v2
              p2 ::= v1
```

Monad to encapsulate effects

Explicit Sequencing
Definition swap (p1 p2: ptr) (v1 v2: nat) :

STsep (p1 → v1 * p2 → v2)
(fun r:unit => p1 → v2 * p2 → v1) :=

v1 ← ! p1 ;

v2 ← ! p2 ;
p1 ::= v2 ;
p2 ::= v1.

Can be made computationally irrelevant

Dependent type

Generate proof obligations

p1 → v1 * p2 → v2

==> p2 → v2 * p1 → v1
Architecture

- HTTP Request
- Server
- HTML

Can store grades in heap

Store

Certification

Gradebook

Gradebook Specification

Can store grades in heap
Architecture: Our Contribution

Certified Application

Store

Gradebook

Server

Requiring I/O

HTML

Store grades on disk

Trace-based I/O Spec

HTTP Request

Database Spec
Extending Ynot

Using Haskell-style types for send and recv

Definition echo (lsock: SockAddr) :
  STsep (
    empty
    (fun r:unit => empty) :=
    (msg,rsock) ← recv sock ;
    send lsock msg rsock.

Receive and Send don't use the heap

Doesn't capture the external effects
Extending Ynot

Definition echo (lsock: SockAddr)
  (tr : Trace) :
  STsep (traced tr)
    (fun r:unit => Exists msg rsock,
       traced (Sent lsock rsock msg ::
              Recv lsock rsock msg ::
              tr)) :=
    (msg,rsock) ← recv sock ;
    send lsock msg rsock.

Record events using traces.
From echo to HTTP Applications

Certified Application

Store

Gradebook

Server

HTTP Request

Trace-based I/O Spec

HTML
Application Server Spec

Echo Server
- Read
- Write

Http Server
- Accept
  - Read Request
    - Success
      - HTTP Parse
        - Invoke App
          - Fail
            - Write
Highlights: Store

- Spec: “database-style” queries
  - Select, update, delete, insert, aggregate
- Certified mapping between grades and tuples
- Imperative implementation uses a heap-based linked-list
- Theorem: deserialize (serialize x) = x
Line count

<table>
<thead>
<tr>
<th></th>
<th>App Server</th>
<th>Parsing</th>
<th>App</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model (LOC)</td>
<td>414</td>
<td>184</td>
<td>231</td>
<td>154</td>
</tr>
<tr>
<td>Implemenation</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>
Compilation

Coq Code (*.v)

Coq Object Files (*.vo)

OCaml Code (*.ml, *.mli)

OCaml Library Code (*.ml, *.mli)

Ocaml Modules (*.cmo, *.cmi)

Executable (native code)

coqc

coqc

“Extraction”

ocamlc

ocamlc
Conclusions

● We can build certified systems in a way like writing Haskell or ML.

● What we don't verify
  ● Non-termination
  ● OCaml “foreign functions”

● Demo (afterward)
  ● See: http://ynot.cs.harvard.edu/
Future Directions

- Add concurrency
- More realistic database
- Resource constraints & failure modes
Persistenence

- Persistance by string serialization

Parameter serial: Table n -> string
Parameter deserial: string -> option (Table n)
Parameter serial_deserial: forall (tbl: Table n),
  deserial (serial tbl) = Some tbl.

Parameter serialize : forall (r: db) (m: Table n),
  STsep (rep r m)
  (fun res:string =>
    rep r m * [str = serial m]).