Toward a Verified Relational Database Management System

Gregory Malecha, Greg Morrisett,
Avraham Shinnar, **Ryan Wisnesky**

POPL 2010

Overview

- We built a **verified RDBMS**.
  - DB2, Oracle, BerkeleyDB, MySQL, SQLite,…
- Code available at:

  `http://ynot.cs.harvard.edu`

Motivation

- Data management is ubiquitous and difficult; correctness is important.
- RDBMS components have clean specifications with deep underlying theory.
- We can verify compilers and operating systems.

Our RDBMS Functionality

- Data storage
- Data update
- Query optimization
- Single-threaded B+ Tree execution

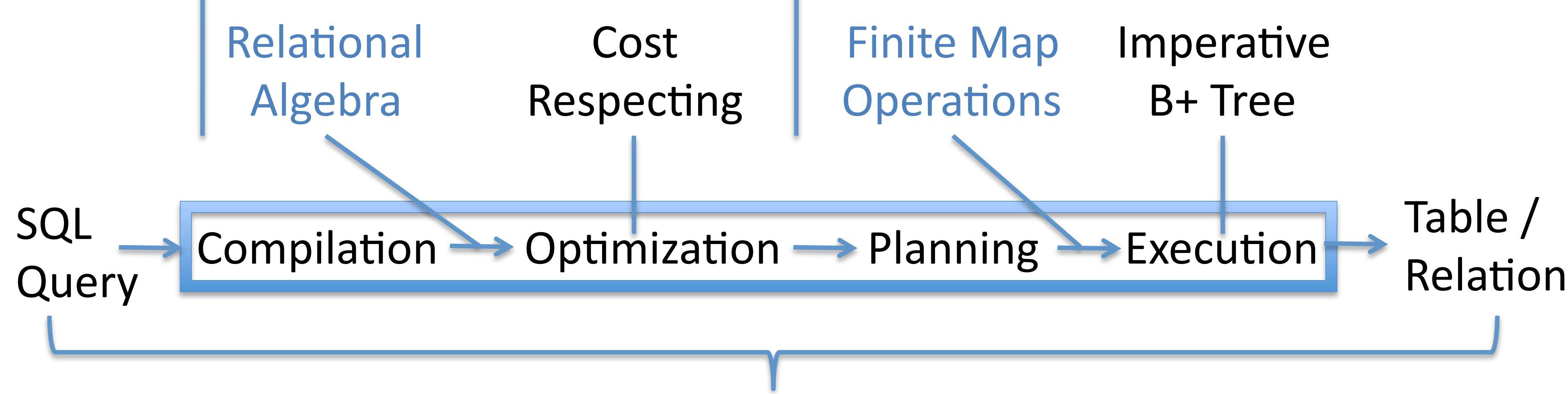
Mechanically verified

RDBMS State of the Art

- Data storage
- Data update
- Query optimization
- Concurrent execution
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability

Our RDBMS Pipeline

(LOC)	Front-End	Back-End
Functional	1200	810
Imperative	0	920
Proof	1140	6880
Proof/Code	**≈ 1**	**≈ 4**



The table the SQL query denotes and
the table the RDBMS returns are equal (partial correctness).

Verification Methodology

- For Purely Functional Code:
  the Coq proof assistant

- For Imperative Code:
  the Ynot extensions to Coq *(Nanevski et al, ICFP '08)*
  - Hoare Type Theory *(Nanevski et al, ICFP '06)*
  - Separation logic *(Reynolds, LICS 02) (O'Hearn et al, POPL '04)*
  - Proof Automation *(Chlipala et al, ICFP '09)*

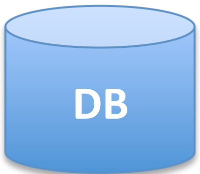
Lessons Learned

Challenge	**Solution**
• Reasoning about queries and relations	• Tuples as heterogeneous lists • Schema-indexed queries
• Modular, first-class imperative finite maps	• Axiomatic interfaces • Hoare Type Theory – Key: higher order iterator
• Reasoning about B+ Trees in separation logic	• Generic, higher order traversal • Multiple invariants

Challenge #1

- Reasoning about queries and relations

DB Model



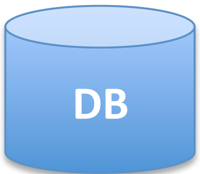
Section Assignments

Student	Year	Section #
John Doe	Freshman	3
Jane Doe	Senior	2

Room Assignments

Course	Room #
CS 51	115
CS 252	319
CS 152	323

DB Model




```
(* nameless, ordered *)
Definition Schema : Type :=
  list Type.
```

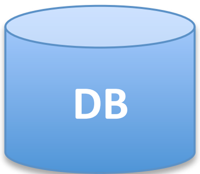
Section Assignments

Student	Year	Section #
John Doe	Freshman	3
Jane Doe	Senior	2

Room Assignments

Course	Room #
CS 51	115
CS 252	319
CS 152	323

DB Model



Section Assignments

Student	Year	Section #
John Doe	Freshman	3
Jane Doe	Senior	2

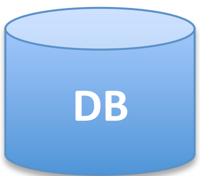
Room Assignments

Course	Room #
CS 51	115
CS 252	319
CS 152	323

```
(* nameless, ordered *)
Definition Schema : Type :=
 list Type.
```

```
Fixpoint Tuple (T: Schema) :
  Type :=
 match T with
  | Nil ⇒ unit
  | Cons a b ⇒ a * Tuple b
 end.
```

DB Model



Section Assignments

Student	Year	Section #
John Doe	Freshman	3
Jane Doe	Senior	2

```
(* nameless, ordered *)
Definition Schema : Type :=
 list Type.
```

```
Fixpoint Tuple (T: Schema) :
  Type :=
 match T with
  | Nil ⇒ unit
  | Cons a b ⇒ a * Tuple b
 end.
```

Room Assignments

Course	Room #
CS 51	115
CS 252	319
CS 152	323

```
(* unordered, no duplicates *)
Definition Table (T: Schema) :
  Type := FSet (Tuple T).
```

Query Abstract Syntax

```
Inductive RAExp              :              Type :=
 | var : ∀(v: name), RAExp
 | union :
     RAExp     → RAExp     → RAExp
 | select : ∀(t: Schema),
     RAExp     → (Tuple t → bool) → RAExp
 | …
 | product :
     RAExp     → RAExp      →
       RAExp             .
```

Query Abstract Syntax

```
Inductive RAExp (G: Context) : Schema → Type :=
  | var : ∀(v: name), RAExp G (G v)
  | union : ∀(t: Schema),
      RAExp G t → RAExp G t → RAExp G t
  | select : ∀(t: Schema),
      RAExp G t → (Tuple t → bool) → RAExp G t
  | …
  | product : ∀(t t’: Schema),
      RAExp G t → RAExp G t’ →
        RAExp G (t ++ t’).
```

Optimization

```
Definition denote T (q: RAExp T)
  : FSet (Tuple T) := …
```

For legibility, ignore context and environment.

Optimization

```
Definition denote T (q: RAExp T)
  : FSet (Tuple T) := …

Definition rewrite T : Type :=
 RAExp T → RAExp T.
```

Optimization

```
Definition denote T (q: RAExp T)
  : FSet (Tuple T) := …

Definition rewrite T : Type :=
 RAExp T → RAExp T.

Definition semantics_preserving T (r: rewrite T) :
  Prop := ∀(q: RAExp T), denote q = denote (r q).
```

Optimization

```
Definition denote T (q: RAExp T)
  : FSet (Tuple T) := …

Definition rewrite T : Type :=
 RAExp T → RAExp T.

Definition semantics_preserving T (r: rewrite T) :
  Prop := ∀(q: RAExp T), denote q = denote (r q).

Definition optimization T : Type :=
  { r   : rewrite T
  ; pf1 : semantics_preserving  r
  ; pf2 : cost_respecting       r }
```

Challenge #2

- First-class, modular imperative finite maps.

Mutable ADTs in Ynot

```
Class myADT : Type := {

  handle : Type;                     Implementation type

  model  : Type := ...;              Model / "Ghost state" type

  rep    : handle → model → heap → Prop;    Representation
                                            invariant

                  "ghost state"
  myOp   : ∀(m : model) (h: handle),
   IO (            rep h m    )      Pre-condition
      (fun r: T ⇒ rep h m_post)      Post-condition

 IO Monad   Return value
```

A Finite Map ADT

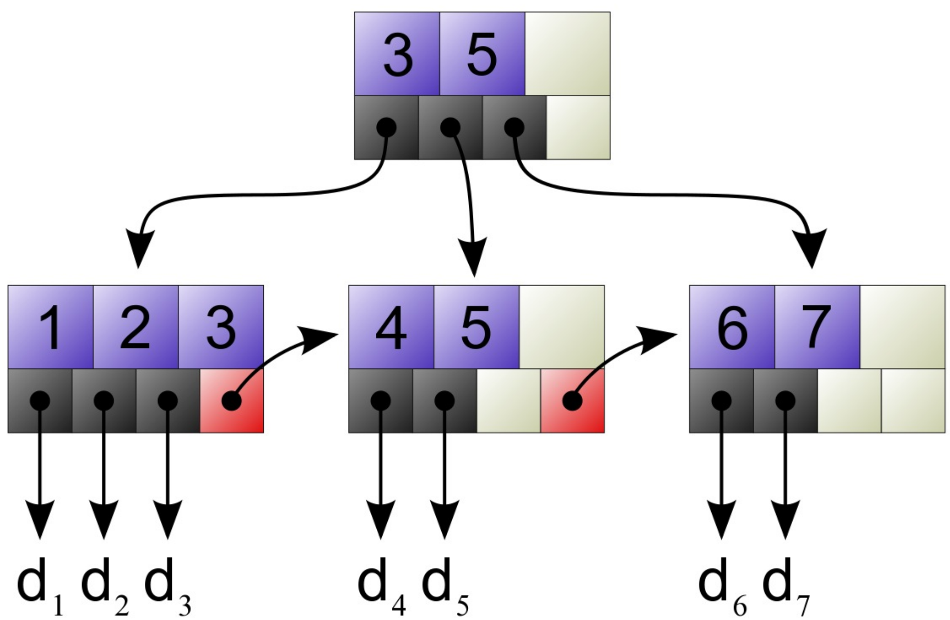
```
Class Fmap (K V: Type) : Type := {

 handle : Type;

 model  : Type := list (K*V);

 rep    : handle → model → heap → Prop;

 add   : ∀(m: model) (k: K) (v: V) (h: handle),
  IO (               rep h m )
     (fun _: unit ⇒ rep h ((k,v)::m));

 lookup  : ... ;
 iterate : ... ;
...
```

Challenge #3

- Reasoning about B+ Trees in separation logic.

B+ Trees

- Generalized Binary Search Trees



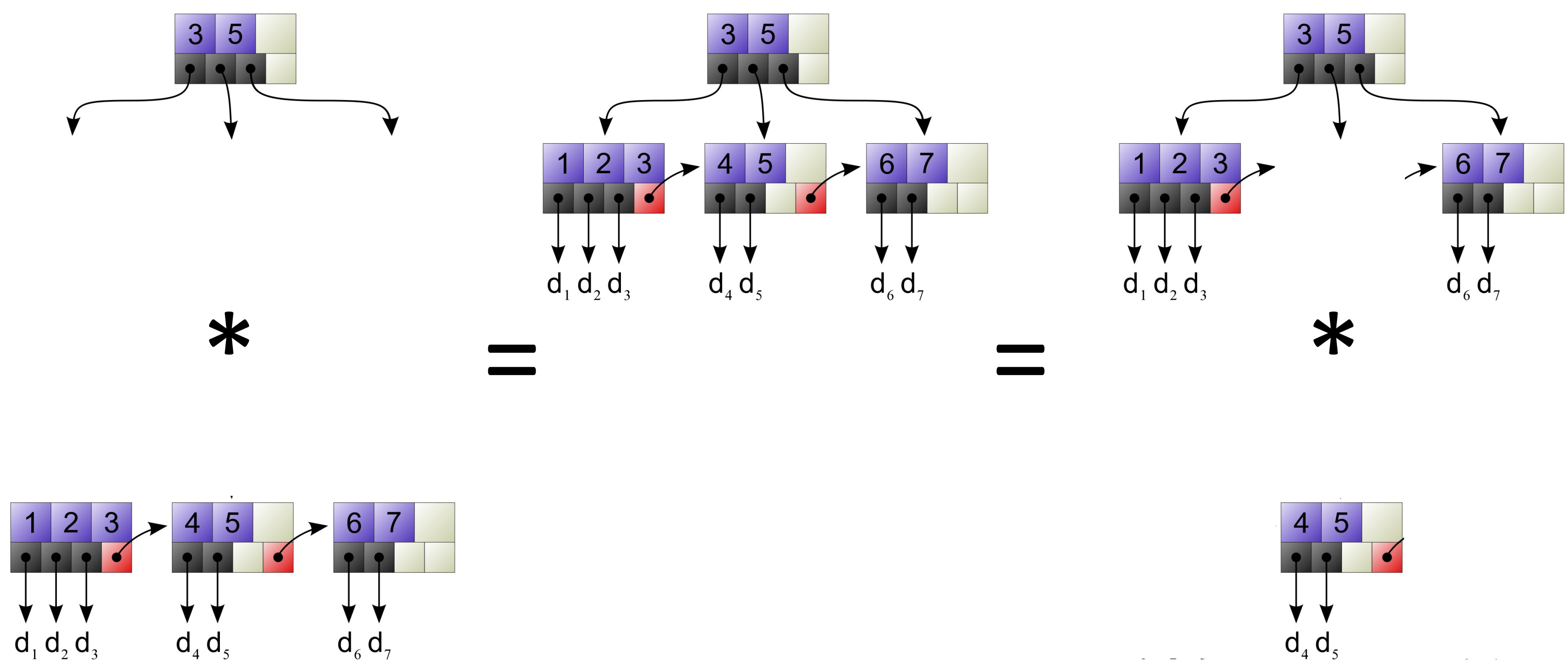
N-way fan-out

Linked fringe for in-order traversal

B+ Tree Invariant

$$
\begin{array}{l}
\texttt{repTree}\; 0\; r\; optr\; (p', ls) \iff \\
\quad [r = p'] * \exists ary.\; r \mapsto \texttt{mkNode}\, 0\; ary\; optr\; * \\
\qquad \texttt{repLeaf}\; ary\; |ls|\; ls \\
\\
\texttt{repTree}\; (h+1)\; r\; optr\; (p', (ls, nxt)) \iff \\
\quad [r = p'] * \exists ary. r \mapsto \texttt{mkNode}\; (h+1)\; ary\; (\texttt{ptrFor}\; nxt) * \\
\qquad \texttt{repBranch}\; ary\; (\texttt{firstPtr}\; nxt)\; |ls|\; ls * \\
\qquad \texttt{repTree}\; h\; (\texttt{ptrFor}\; nxt)\; optr\; nxt \\
\\
\texttt{repLeaf}\; ary\; n\; [v_1, ..., v_n] \iff \\
\quad ary[0] \mapsto \texttt{Some}\; v_1 * ... * ary[n-1] \mapsto \texttt{Some}\; v_n * \\
\quad ary[n] \mapsto \texttt{None} * ... * ary[SIZE-1] \mapsto \texttt{None} \\
\\
\texttt{repBranch}\; ary\; n\; optr\; [(k_1, t_1), ..., (k_n, t_n)] \iff \\
\quad ary[0] \mapsto \texttt{Some}\; (k_1, \texttt{ptrFor}\; t_1) * \\
\qquad \texttt{repTree}\; h\; (\texttt{ptrFor}\; t_1)\; (\texttt{firstPtr}\; t_2)\; t_1\; * ... * \\
\quad ary[n-2] \mapsto \texttt{Some}\; (k_{n-1}, \texttt{ptrFor}\; t_{n-1}) * \\
\qquad \texttt{repTree}\; h\; (\texttt{ptrFor}\; t_{n-1})\; (\texttt{firstPtr}\; t_n)\; t_{n-1} * \\
\quad ary[n-1] \mapsto \texttt{Some}\; (k_n, \texttt{ptrFor}\; t_n) * \\
\qquad \texttt{repTree}\; h\; (\texttt{ptrFor}\; t_n)\; optr\; t_n * \\
\quad ary[n] \mapsto \texttt{None} * ... * ary[SIZE-1] \mapsto \texttt{None}
\end{array}
$$

Multiple B+Tree Views



During Iteration

During Deletion

Related Work

- **B+ Trees in Separation Logic**
  - Local Reasoning, Separation and Aliasing *(Bornat et al, SPACE 04)* [classical conj.]
  - Reasoning about B+ Trees with Operational Semantics *(Sexton et al, Elec. Notes. Theoretical CS 08)*
- **Verified DB Components**
  - Ph.D. Thesis *(Gonzalia, 2006)* [relations in Agda]
  - The Power of Pi *(Oury et al, ICFP '08)* [relational algebra in Agda]
  - A Generic Algebra for Data Collections Based on Constructive Logic *(Rajagopalan et al, LNCS '95)* [generic data in Nuprl]
- **Program Verification**
  - Compcert: Formal certification of a compiler back-end *(Leroy, POPL '06)*
  - A Verified Compiler for an Impure Functional Language *(Chlipala, POPL '10)*
  - seL4: Formal Verification of an OS Kernel *(Klein at al, SOSP '09)* [165k Proof]

Conclusion

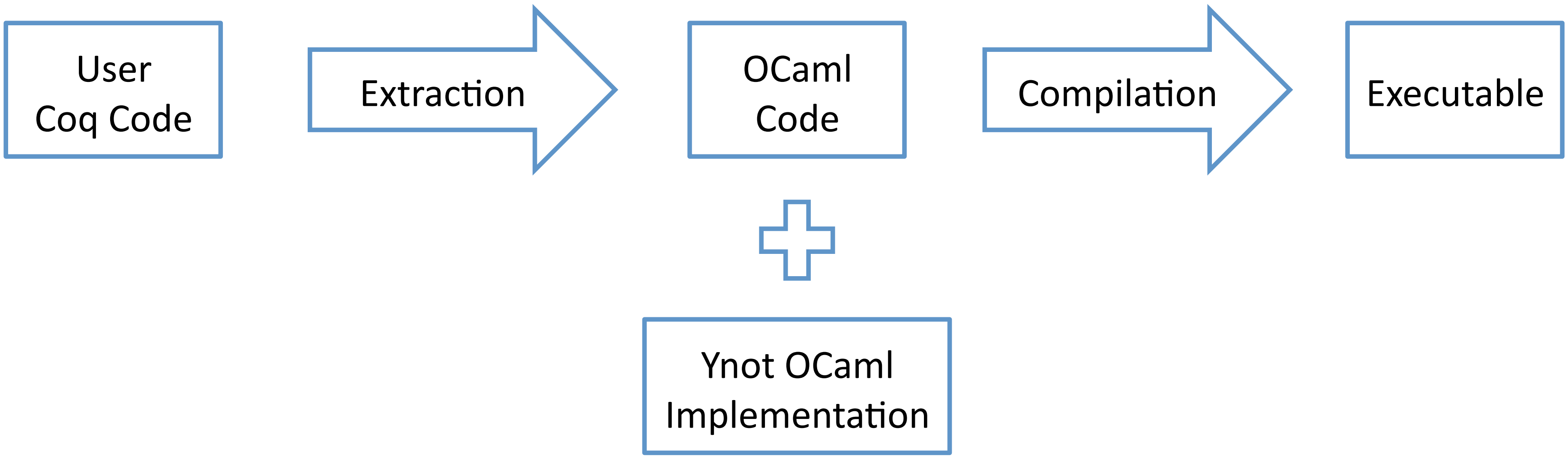
- Verified systems software is now viable.
- Verified RDBMSs are possible.
  – 3 Ph.D. students part-time 3-6 months
  – 30 minutes to verify (3GHz Pentium D, 1GB RAM)

… but still difficult, despite progress.

- Future Work
  – Concurrency and the ACID Properties
  – Dependent types vs traditional types

Questions?

Extraction


User
Coq Code
Extraction
OCaml
Code
Compilation
Executable
Ynot OCaml
Implementation

Cost Model

- Naïve implementation:
  - Set union: O(n*m)
  - Selection: O(n)
- No data statistics
- Conservative approximations
  - Selection preserves cardinality

First-class Sets

```
let user_schema : Schema :=
  load_schema()
in
let user_data : Table user_schema :=
  load_data(user_schema)
in
  process_data(user_schema, user_data)
```

LOC

- SQLite: 65k

- Compcert: 8k Code, 24k Proof, ratio ≈ 4
- seL4: 47.3k Code 165k Proof, ratio ≈ 3.5
- Chlipala: 6.6k Code, 1.8k Proof, ratio ≈ .3