Data Structure Synthesis

Calvin Loncaric
Emina Torlak
Michael D. Ernst
Data structures are everywhere
Data structures are everywhere
Data structures are everywhere
Data structures are everywhere
Data structures are everywhere

Lists, maps, and sets solve many problems
Data structures are everywhere

Lists, maps, and sets solve many problems

What if I need a custom data structure?
Myria Analytics Storage

Request 1

Request 2

time
Myria Analytics Storage

Operations being performed

Request 1

Request 2

time
Myria Analytics Storage

Operations being performed

Request 1

Request 2

time
Myria Analytics Storage

Operations being performed

Request 1

Request 2

time
Myria Analytics Storage

Operations being performed

Request 1

Request 2

**Goal:** efficient retrieval of entries for a particular request ID in a particular timespan
class AnalyticsLog {

    void log(Entry e)

    Iterator<Entry> getEntries(
        int queryId,
        int subqueryId,
        int fragmentId,
        long start,
        long end)

}
Myria Analytics Storage

Insert an entry into the data structure

```java
class AnalyticsLog {
    void log(Entry e)
    Iterator<Entry> getEntries(
        int queryId,
        int subqueryId,
        int fragmentId,
        long start,
        long end)
}
```
Myria Analytics Storage

Insert an entry into the data structure

```java
class AnalyticsLog {
    void log(Entry e)

    Iterator<Entry> getEntries(
        int queryId,
        int subqueryId,
        int fragmentId,
        long start,
        long end)
}
```

Retrieve entries
Myria Analytics Storage

**Specification:**

Entry has:
- queryId,
- subqueryId,
- fragmentId,
- start, end,
...

getEntries: all e where
  e.queryId = queryId and
  e.subqueryId = subqueryId and
  e.fragmentId = fragmentId and
  e.end >= start and
  e.start <= end

```java
class AnalyticsLog {
    void log(Entry e)
    
    Iterator<Entry> getEntries(
        int queryId,
        int subqueryId,
        int fragmentId,
        long start,
        long end)
}
```
Myria Analytics Storage

Specification:

Entry has:
- queryId,
- subqueryId,
- fragmentId,
- start, end,
...

class AnalyticsLog {
    void log(Entry e)

    Iterator<Entry> getEntries(
        int queryId,
        int subqueryId,
        int fragmentId,
        long start, 
        long end
    )
}

getEntries: all e where
    e.queryId = queryId and
    e.subqueryId = subqueryId and
    e.fragmentId = fragmentId and
    e.end >= start and
    e.start <= end
Cozy synthesizes collections
Cozy synthesizes collections

**Specification:**

**Entry** has:
- field1
- field2
- ...

**retrieveA:** all e where
  e.field1 < var1 and ...

**retrieveB:** all e where
  e.field1 > var1 and ...
Cozy synthesizes collections

**Specification:**

- **Entry** has:
  - `field1`
  - `field2`
  - ...

- **retrieveA:** all `e` where `e.field1 < var1` and ...

- **retrieveB:** all `e` where `e.field1 > var1` and ...

**class Structure {**

- `void add(Entry e)`
- `void remove(Entry e)`
- `void update(Entry e, ...)`

- `Iterator<Entry> retrieveA(...)`
- `Iterator<Entry> retrieveB(...)`

**}**
Architecture

Outline


Specification → Inductive Synthesizer → Verifier
Specification:

Entry has:
- field1
- field2
- ...

retrieveA: all e where e.field1 < var1 and ...

retrieveB: all e where e.field1 > var1 and ...
Architecture


Brute-force Search
Architecture

Brute-force Search

More on this shortly!
Architecture


High-level Plan
Architecture

Benchmark to make low-level choices
Outlines

Plans for retrieving entries
Outlines

Plans for retrieving entries

• \textbf{All ( )}
Outlines

Plans for retrieving entries

• **All** ( )

• **HashLookup** ( **outline**, **field** = **var** )
Outlines

Plans for retrieving entries

• **All**( )

• **HashLookup** ( outline, field = var )

• **BinarySearch** ( outline, field > var )
Outlines

Plans for retrieving entries

• **All** ( )

• **HashLookup** ( outline, field = var )

• **BinarySearch** ( outline, field > var )

• **Concat** ( outline, outline )
Outlines

Plans for retrieving entries

• **All** ( )

• **HashLookup** ( outline, field = var )

• **BinarySearch** ( outline, field > var )

• **Concat** ( outline, outline )

• **Filter** ( outline, predicate )
Outlines

Plans for retrieving entries

• **All** ( )

• **HashLookup** ( outline, field = var )

• **BinarySearch** ( outline, field > var )

• **Concat** ( outline, outline )

• **Filter** ( outline, predicate )
Outlines

Plans for retrieving entries

- **All** ( )  
  \[ 1 \]

- **HashLookup** ( outline, field = var )  
  \[ c + 1 \]

- **BinarySearch** ( outline, field > var )

- **Concat** ( outline, outline )

- **Filter** ( outline, predicate )
Outlines

Plans for retrieving entries

• All ( ) 1

• HashLookup ( outline, field = var ) c + 1

• BinarySearch ( outline, field > var ) c + log n

• Concat ( outline, outline )

• Filter ( outline, predicate )
Outlines

Plans for retrieving entries

• **All ( )**  
  
• **HashLookup ( outline, field = var )**  
  
• **BinarySearch ( outline, field > var )**  
  
• **Concat ( outline, outline )**  
  
• **Filter ( outline, predicate )**
Outlines

Plans for retrieving entries

• **All ( )**

• **HashLookup ( outline, field = var )** $c + 1$

• **BinarySearch ( outline, field > var )** $c + \log n$

• **Concat ( outline, outline )** $c_1 + c_2$

• **Filter ( outline, predicate )** $c + n$
Architecture

Specification → Inductive Synthesizer → Verifier → Outline → Brute-force Search

Specification → Outline

Enumerative search
Specification —> Outline

Enumerative search

size 1
Specification —> Outline

Enumerative search

size 1

All

HashLookup(All, x=y)

Filter(All, x=y)

BinarySearch(All, x>y)

size 2

...
Specifications $\rightarrow$ Outline

Enumerative search

size 1

All

size 2

HashLookup(All, x=y)

Filter(All, x=y)

BinarySearch(All, x>y)

...
Specification

Enumerative search

- HashLookup(All, x=y)
- Filter(All, x=y)
- BinarySearch(All, x>y)

...
Specification —> Outline

Enumerative search

size 1
All

size 2
HashLookup(All, x=y)
BinarySearch(All, x>y)

size 3
HashLookup(HashLookup(...), a=b)
Filter(HashLookup(...), p=q)
Filter(BinarySearch(...), x<y)

...
Specification ➔ Outline

Enumerative search

size 1

All

size 2

HashLookup(All, x=y)

BinarySearch(All, x>y)

size 3

HashLookup(HashLookup(...), a=b)

Filter(HashLookup(...), p=q)

Filter(BinarySearch(...), x<y)

correct on all current examples
Outline Verification

**Specification:**

**Entry** has:
- `queryId`
- `subqueryId`
- ...

**retrieve:** all e where
- `e.queryId = q` and ...

**counterexample**
Outline Verification

∀I ∀S,
out = \{ e \mid e \in S \land P(I, e) \}

retrieve: all e where e.queryId = q and ...
Outline Verification

Specification:

Entry has:

- queryId
- subqueryId
...

retrieve: all e where e.queryId = q and ...

∀I ∀S,
out = \{ e \mid e \in S \land P(I, e) \}

HashLookup(
All(),
e.queryId = q)
Outline Verification

**Specification:**
Entry has:
- `queryId`
- `subqueryId`
...

**retrieve:** all `e` where `e.queryId = q` and ...

```
∀ I ∀ S,
out = \{ e \mid e \in S \land P(I, e) \}
```

**HashLookup**

```
HashLookupup(
  All(),
  e.queryId = q)
```

**representative predicate**

```
e.queryId = q
```
Outline Verification

Specification:
Entry has:
queryId
subqueryId
...

retrieve: all e where e.queryId = q and ...

∀I ∀S,
 out = { e | e ∈ S ∧ P(I, e) }

HashLookup
All(),
e.queryId = q

∀I ∀S,
 out = { e | e ∈ S ∧ Q(I, e) }

representative
predicate
e.queryId = q
Outline Verification

**Specification:**
Entry has:
- queryId
- subqueryId
...

retrieve: all e where e.queryId = q and ...

HashLookup (All(),
- e.queryId = q)

∀I ∀S,
out = { e | e ∈ S ∧ P(I, e) }

∀I ∀S,
out = { e | e ∈ S ∧ Q(I, e) }

representative predicate

check equivalence with an SMT solver
Evaluation
Evaluation

- **Myria**: analytics

Bugs at the SQL/Java interface. Unpredictable query planner.
Evaluation

- **Myria**: analytics

  Bugs at the SQL/Java interface. Unpredictable query planner.

- **ZTopo**: tile cache

  Tricky invariant: “state” field on entries reflects its position in the data structure.
Evaluation

- **Myria**: analytics
  
  Bugs at the SQL/Java interface. Unpredictable query planner.

- **ZTopo**: tile cache
  
  Tricky invariant: “state” field on entries reflects its position in the data structure.

- **Bullet**: volume tree
  
  Lots of verbose, handwritten C++ pointer manipulation code. Custom memory allocator.
Evaluation

- **Myria**: analytics
  - Bugs at the SQL/Java interface.
  - Unpredictable query planner.

- **ZTopo**: tile cache
  - Tricky invariant: “state” field on entries reflects its position in the data structure.

- **Bullet**: volume tree
  - Lots of verbose, handwritten C++ pointer manipulation code.
  - Custom memory allocator.

- **Sat4J**: variable metadata
  - Custom map implementation for faster lookups
Performance

- Original
- Synthesized
Performance

Original implementation has worst-case linear time

Myria
Performance

Original implementation has worst-case linear time

Myria
Performance

- Original
- Synthesized

Original implementation has worst-case linear time

Data structures are nearly identical

Myria

ZTopo
Performance

- **Original**
- **Synthesized**

Original implementation has worst-case linear time

Data structures are nearly identical

Binary search tree vs. space partitioning tree

Myria

ZTopo

Bullet
Performance

- **Original**
- **Synthesized**

Original implementation has worst-case linear time.

Data structures are nearly identical.

Binary search tree vs. space partitioning tree.

**Myria**

**ZTopo**

**Bullet**
Performance

- Original
- Synthesized

Original implementation has worst-case linear time

Data structures are nearly identical

Binary search tree vs. space partitioning tree

Small overhead; performance dominated by other factors

Myria
ZTopo
Bullet
Sat4J
Data Structure Synthesis

- Implementation outlines make the problem tractable
- This approach is extensible
- Cozy generates correct code, and can sometimes surpass handwritten implementations

Special thanks to:

Michael Ernst
Emina Torlak
also Haoming Liu & Daniel Perelman