Fast Synthesis of Fast Collections

Calvin Loncaric
Emina Torlak
Michael D. Ernst

University of Washington
Data structures are everywhere

Lists, maps, and sets solve many problems

8.3. collections — Container datatypes

Source code: Lib/collections/__init__.py

This module implements specialized container datatypes provo to Python's general purpose built-in containers, dict, list, se
Cozy synthesizes collections

- Correct by construction
- Specifications orders-of-magnitude shorter than implementations, synthesized in < 90 seconds
- Equivalent performance to human-written code
Myria Analytics Storage

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)
}

Insert an entry into the data structure

Retrieve entries

Myria Analytics Storage
Myria Analytics Storage

Specification:

Entry has:

- queryId : Int,
- subqueryId : Int,
- fragmentId : Int,
- start, end : Long,
...

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)

}
```
Cozy synthesizes collections

**Specification:**

Entry has:
- `field1 : Type1`,
- `field2 : Type2`,
- ...

Cozy

```java
class Structure {
    void add(Entry e)
    void remove(Entry e)
    void update(Entry e, ...)
    Iterator<Entry> retrieveA(...)  // retrieveA: all e where condition
    Iterator<Entry> retrieveB(...)  // retrieveB: all e where condition
}
```
Trivial Solution

retrieve: all e where P(e, input)

List<Entry> data;

Iterator<Entry> retrieve(input) {
    for e in data:
        if P(e, input):
            yield e
}

There has to be a better way!
In the quest for a good solution, the search space of "all possible programs" is simply too large.

**Specification** → **Outline** → **Implementation**

**Intractable**

- void add(Entry e)
- void remove(Entry e)
- void update(Entry e, ...)
- Iterator retrieveA(...)
- Iterator retrieveB(...)

**Tractable**

- void add(Entry e)
- void remove(Entry e)
- void update(Entry e, ...)
- Iterator retrieveA(...)
- Iterator retrieveB(...)

Entry has:
- field1, field2, ...
- retrieveA: all e where condition
- retrieveB: all e where condition

**specific** enough to describe asymptotic performance

**general** enough to encode a data structure succinctly
Outlines

Plans for retrieving entries

- **All** ( )
- **HashLookup** ( outline, field = var )
- **BinarySearch** ( outline, field > var )
- **Concat** ( outline₁, outline₂ )
- **Filter** ( outline, predicate )
Outlines $\rightarrow$ Implementations

HashLookup\( (\text{All()}, \quad e.\text{queryId} = q ) \)

class Structure {
  T data;
  Iterator<Entry> retrieve(q) {
    ... 
  }
}

Outlines $\rightarrow$ Implementations
class Structure {
    T data;
    Iterator<Entry> retrieve(q) {
        ...
    }
}
class Structure {
    HMap<K, V> data;
    Iterator<Entry> retrieve(q) {
        ... 
    }
}
Outlines → Implementations

```java
class Structure {
    HMap<int, V> data;
    Iterator<Entry> retrieve(q) {
        ...
    }
}

V = ArrayList<Entry>
V = LinkedList<Entry>
```

HashLookup (data,
    e.queryId = q)

V = ArrayList<Entry>  V = LinkedList<Entry>
class Structure {
    HMap<int, V> data;
    Iterator<Entry> retrieve(q) {
        ...
    }
}
HashLookup (data, e.queryId = q )

```java
class Structure {
    HMap<int, V> data;

    Iterator<Entry> retrieve(q) {
        v = data.get(q);
        return v.iterator();
    }
}
```
Specification $\rightarrow$ Outline
Specification → Outline

Remembers all examples; only reasons about examples collected thus far.

CEGIS

Inductive Synthesizer

Verifier

Candidate

counterexample

- or -
certification of correctness

Must ensure the outline is correct for all possible inputs and all possible data structure states.

∀ I ∀ S, out = \{ e | e ∈ S \land P(I, e) \}
Cost Model

Filter (\(\text{All}()\), e.queryId = q )

\(O(n)\) \(\geq\) \(O(1)\)

HashLookup (\(\text{All}()\), e.queryId = q )

\(O(1)\)

\(O(1)\)

Cozy prefers outlines with lower cost
Inductive Synthesis

Enumerative Search

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

Concat(HashLookup(…),…)
vs
Concat(Filter(…),…)

correct on all current examples

Concat(HashLookup(…),…)
vs
Concat(Filter(…),…)

size 1

size 2

All

HashLookup(All, x=y)
Filter(All, x=y)
BinarySearch(All, x>y)
Outline Verification

Specification:

\[ \{ e \mid e \in S \land P(I, e) \} \]

retrieve: all \( e \) where 
\[ e \text{.queryId} = q \text{ and } \ldots \]

Hash Lookup:

\[ \{ e \mid e \in S \land Q(I, e) \} \]

representative predicate \( Q \)

\[ e \text{.queryId} = q \]
Outline Verification

\{ e \mid e \in S \land P(I, e) \} \equiv \{ e \mid e \in S \land Q(I, e) \}

Yes if and only if for all \( I, e \):
\[ P(I, e) = Q(I, e) \]

Equivalence can be checked with an SMT solver.
Evaluation

• Improve correctness ✓

• Save programmer effort ✓

• Match performance ✓
Case studies

- **Myria:** analytics
  - Analytics data indexed by timespan and by request ID

- **Bullet:** volume tree
  - Stores axis-aligned bounding boxes for fast collision detection

- **ZTopo:** tile cache
  - Tracks map tiles in a least-recently-used cache

- **Sat4j:** variable metadata
  - Tracks information about each variable in the formula

11 bugs

15 bugs

7 bugs
Specifications vs. Implementations

Myria
ZTopo
Sat4j
Bullet

Lines of code

Myria: Original 22, Spec 269
ZTopo: Original 25, Spec 1383
Sat4j: Original 11, Spec 292
Bullet: Original 23, Spec 2582
Synthesis Time

Outline Synthesis  Auto-Tuning

Time (s)

Myria  ZTopo  Sat4j  Bullet
Performance

- Original
- Synthesized

Data structures are nearly identical

Binary search tree vs. space partitioning tree

Small overhead; performance dominated by other factors

Original implementation has worst-case linear time

ZTopo

Bullet

Sat4j

Myria
Related Work

• J. Earley: “High level iterators and a method for automatically designing data structure representation” (1974)
  • Hard-coded rewrite rules

  • Enumerate possible views & indexes based on query syntax and use the planner to decide which ones to keep

• P. Hawkins et al: “Data representation synthesis” (2011)
  • Enumerate representations and use a planner to implement retrieval operations; conjunctions of equalities only
Implementation outlines make the problem tractable

Synthesis completes < 90 seconds

Cozy generates correct code, and matches handwritten implementation performance

Special thanks to:

Michael Ernst
Emina Torlak
also Haoming Liu & Daniel Perelman