Generalized Data Structure Synthesis

Calvin Loncaric
Michael D. Ernst
Emina Torlak
Stateful modules are much more complicated than their specifications

<table>
<thead>
<tr>
<th></th>
<th>Implementation</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZTopo</td>
<td>1383</td>
<td>41</td>
</tr>
<tr>
<td>Sat4j</td>
<td>292</td>
<td>42</td>
</tr>
<tr>
<td>Openfire</td>
<td>1992</td>
<td>157</td>
</tr>
<tr>
<td>Lucene</td>
<td>68</td>
<td>36</td>
</tr>
</tbody>
</table>

Lines of code
User Visibility in Openfire

Can Alice see whether Bob is online?

Yes, if:

- Alice “subscribed” to Bob AND Bob approved it
- Bob is a member of a group $g$ that is visible to everyone
- Alice and Bob are in a group together
- Alice is in group $g_1$, Bob is in group $g_2$, and $g_2$ is a “child” of $g_1$
User visibility in code

boolean isRosterItem(JID user) {
  return rosterItems.get(user.toBareJID()) != null || implicitFrom.get(user.toBareJID()) != null;
}

void groupUserAdded(Group group, ..., JID addedUser) {
  ... // Get the roster of the added user.
  if (server.isLocal(addedUser)) { ... }
  ... // Get the RosterItem for the "local" user to add.
  RosterItem item = getRosterItem(addedUser);
  ... // Check if the item already includes the shared group.
  if (item.getSharedGroups().contains(group)) {
    ... // Set subscription type to BOTH if the roster user belongs to a shared group
    item.setSubStatus(RosterItem.SUB_BOTH);
    ... // Add the new group to the list of groups to check
    sharedGroups.addAll(item.getSharedGroups());
  }
}

Only correct when these maps are in the correct state!
N/A
Cozy

Spec
- short
- self-documenting
- inefficient

.java
- verbose
- invariants
- efficient
Cozy

state \texttt{ints} : Bag\textless Int\textgreater

\textbf{op add}(i : Int)
\texttt{ints}.add(i)

query \textbf{findmin}()
return \texttt{min ints}

state \texttt{m} : Int

\textbf{op add}(i : Int)
\texttt{m} = \texttt{min(m, i)}

query \textbf{findmin}()
return \texttt{m}

Abstraction relation:
\( m = \text{min}(\text{ints}) \)
Cozy

state `ints` : Bag<Int>

op add(i : Int)
   `ints`.add(i)

query findmin()
   return (min `ints`)
state **ints** : Bag<Int>

**op add(i : Int)**

**ints.add(i)**

**query findmin()**

**return (min ints)**
state \texttt{ints} : \texttt{Bag\{Int\}}

\texttt{op add(i : Int)}

\texttt{ints.add(i)}

query \texttt{findmin()}

return \texttt{m}

\textit{Synthesize a better representation}

\textit{Cost model}
state **ints** : Bag<Int>

state **m** : Int

**op add(i : Int)**

```java
ints.add(i)
```

**m = min(m, i)**

**query findmin()**

```java
return m
```

---

**Cozy**

**Spec**

**Synthesize a better representation**

**Cost model**

**Maintain** the new representation
state \( m : \text{Int} \)

op add(i : \text{Int})
\[ m = \min(m, i) \]

query findmin()

return \( m \)
query findmin()
return (min \text{ints})

\text{Abstraction relation}

m = \text{min ints}
return m
for size in [1, 2, ...]:
    for exp in all_expressions(size):
        if correct(exp):
            return exp

“improves cost” is part of correctness

yield and keep searching

skip semantically-identical exps

memoize

bias

bounded verification

Smart Brute
Force Search

“improves cost” is part of correctness
Cozy is biased toward common patterns

for size in [1, 2, ...]:

\[ x + 1 \]

\[
\text{MakeMap}(\text{user_ids}, \lambda u . \text{the } \{x| x \in \text{users}, x.id=u\} )[\text{user_id}]
\]
Iterative Discovery

Spec

Figure out how to *maintain* the new representation

Synthesize a better implementation with a new representation

Cost model

Implementation
Cost Optimization

Expression + Invariants → Cost model → =, <, >, or ?

Challenge: Cardinality Estimation
Iterative Discovery

Spec

Synthesize a better implementation with a new representation

Cost model

Figure out how to maintain the new representation

Implementation
state **ints** : Bag<Int>
state **m** : Int

op add(\texttt{i} : Int)
\texttt{ints}.add(\texttt{i})
\texttt{m} = 

**query** findmin()
**return** \texttt{m}

**Idea:** leverage our incredible query synthesizer
State Maintenance

**State**

```java
state ints : Bag<Int>
state m : Int = min ints

op add(i : Int)
    ints.add(i)
    m = new_min(i)

query findmin()
return m

query new_min(i)
return (min (ints U {i})))
```

**Idea:** leverage our incredible query synthesizer

Leave optimization to later iterations
state \texttt{ints} : \texttt{Bag\langle\texttt{Int}\rangle}
state \texttt{m} : \texttt{Int}

\texttt{op add(i : Int)}
  \texttt{\hspace{1cm} ints.add(i)}
  \texttt{\hspace{1cm} m = new_min(i)}

\texttt{query findmin()}
  \texttt{return m}

\texttt{query new_min(i)}
  \texttt{return (min (ints \cup \{i\}))}

\textbf{Idea:} leverage our incredible query synthesizer
Case Studies

Goals
- Less effort
- Same performance
- No new bugs

ZTopo  Map tile cache
Sat4j  Internal metadata
Openfire  Visibility enforcement
Lucene  Streaming document statistics
Performance

Loss

Win

Speedup

ZTopo
+0%

Sat4j
-15%

Openfire
+6%

Lucene
+0%

1x
## Correctness

<table>
<thead>
<tr>
<th>ZTopo</th>
<th>Handwritten</th>
<th>Cozy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Sat4j</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Openfire</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>Lucene</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Related Work

Iterator Inversion (1975)
  Rewrite rules

SETL (1975—)
  Manual separation of algorithm and data arrangement

Programming by Refinement (1990—)
  Manual iterative transformation

Representation Synthesis (2011)
  Limited specification language, special-purpose techniques

Cozy 1.0 (2016)
  Limited specification language, special-purpose techniques
Data Structure Synthesis

Acknowledgements
Professors Mike and Emina
Students Daniel, David, and Haoming

Lower effort
Equal performance
Fewer bugs

https://cozy.uwplse.org