Verifying Data Structure Correctness in the Hob and Jahob Systems

Patrick Lam, Viktor Kuncak, Karen Zee, Martin Rinard
MIT CSAIL
Massachusetts Institute of Technology
Cambridge, MA 02139
Data Structure Correctness

• Internal correctness (single data structure)
  • Internal representation consistent
  • Procedure specifications sound
    (precondition implies postcondition)
• Client correctness
  (clients use data structures correctly)
• External correctness (multiple data structures)
  • Inclusion constraints
  • Disjointness constraints
  • Equality constraints
Our Goal

• Verify data structure correctness
  • Internal properties
  • Usage properties
  • External properties
• Before program executes
• For all possible program executions
• Two systems
  • Hob (abstract sets of objects)
  • Jahob (abstract sets and relations)
spec module Queue { 
    format Entry;
    sets Content : Entry;
    proc add(e : Entry)
        requires not (e in Content)
        modifies Content
        ensures Content' = Content + e;
    proc removeFirst() returns e : Entry
        requires card(Content)>=1
        modifies Content
        ensures (Content' = Content - e) & (e in Content);
    proc isEmpty() returns b : bool
        ensures b <=> (card(Content)=0);
}
Queue Specification in Hob System

Program is set of modules
Each module has specification

```plaintext
spec module Queue {
  format Entry;
  sets Content : Entry;
  proc add(e : Entry)
    requires not (e in Content)
    modifies Content
    ensures Content' = Content + e;
  proc removeFirst() returns e : Entry
    requires card(Content) >= 1
    modifies Content
    ensures (Content' = Content - e) & (e in Content);
  proc isEmpty() returns b : bool
    ensures b <= (card(Content) = 0);
}
```
spec module Queue {
    format Entry;
    sets Content : Entry;
    proc add(e : Entry)
        requires not (e in Content)
        modifies Content
        ensures Content' = Content + e;
    proc removeFirst() returns e : Entry
        requires card(Content)>=1
        modifies Content
        ensures (Content' = Content - e) & (e in Content);
    proc isEmpty() returns b : bool
        ensures b <=> (card(Content)=0);
}
spec module Queue {
  format Entry;
  sets Content : Entry;
  proc add(e : Entry)
    requires not (e in Content)
    modifies Content
    ensures Content' = Content + e;
  proc removeFirst() returns e : Entry
    requires card(Content)>=1
    modifies Content
    ensures (Content' = Content - e) & (e in Content);
  proc isEmpty() returns b : bool
    ensures b <=> (card(Content)=0);
}

Modules have procedures
Procedures have
preconditions
postconditions
modifies clauses
Queue Specification in Hob System

spec module Queue {
    format Entry;
    sets Content : Entry;
    proc add(e : Entry)
        requires not (e in Content)
        modifies Content
        ensures Content' = Content + e;
    proc removeFirst() returns e : Entry
        requires card(Content) >= 1
        modifies Content
        ensures (Content' = Content - e) & (e in Content);
    proc isEmpty() returns b : bool
        ensures b <=> (card(Content) = 0);
}
spec module Queue {
    format Entry;
    sets Content : Entry;
    proc add(e : Entry)
        requires not (e in Content)
        modifies Content
        ensures Content' = Content + e;
    proc removeFirst() returns e : Entry
        requires card(Content) >= 1
        modifies Content
        ensures (Content' = Content - e) & (e in Content);
    proc isEmpty() returns b : bool
        ensures b <=> (card(Content)=0);
}
What Are We Missing (so far)?

- Implementation
- Verification
  - Connection (implementation and specification)
  - Verification algorithm
impl module Queue {
    format Entry { next : Entry; }
    reference root, tail : Entry;
    proc add(e : Entry) {
        e.next = null;
        if (tail != null) tail.next = e;
        else root = e;
        tail = e;
    }
    proc removeFirst() returns e : Entry {
        Entry res = root;
        if (tail == root) tail = null;
        root = root.next;
        return res;
    }
    proc isEmpty() returns b : bool { return root == null; }
}
Abstraction Modules in Hob

abst module Queue {
    use plugin "PALE";
    Content = {x : Entry | "root<next*>x"};
    invariant "type Entry = {
        data next : Entry;
    }";
    invariant "data root:Entry;"
}
Abstraction Modules in Hob

abst module Queue {
    use plugin "PALE";
    Content = {x : Entry | "root<next*>x"};

    invariant
        "type Entry = {
            data next : Entry;
        }"
        invariant "data root:Entry;"
    }

Identification of verification algorithm (multiple plugins)
Eliminating Errors in Queue Clients

```java
e1 = new Entry();
Queue.add(e1);
Queue.add(e1);
e1 = new Entry();
e2 = Queue.removeFirst();
e3 = Queue.removeFirst();
```
Eliminating Errors in Queue Clients

e1 = new Entry();
Queue.add(e1);
Queue.add(e1);

Error (double add)

e1 = new Entry();
e2 = Queue.removeFirst();
e3 = Queue.removeFirst();
Errors in Queue Clients

```java
e1 = new Entry();
Queue.add(e1);
Queue.add(e1);
```
Error (double add)

```java
e1 = new Entry();
e2 = Queue.removeFirst();
e3 = Queue.removeFirst();
```
Error (empty remove)
Hob Minesweeper Experience

- Abstract Sets of Objects
  - Exposed cells
  - Unexposed cells
  - Mined cells
- State
  - Game Over
Hob Minesweeper Experience

Enforced Constraints
• Individual data structures
  • Correct implementations
  • Used correctly
• Multiple data structures
  • Sets of exposed and unexposed cells are disjoint
  • No mined cells are exposed unless game is over
  • When game is over, all cells are exposed
Hob Minesweeper Experience

Enforced Constraints
- Individual data structures
- Correct implementations
- Used correctly
- Multiple data structures
  - Sets of exposed and unexposed cells are disjoint
  - No mined cells are exposed unless game is over
  - When game is over, all cells are exposed

Rules of Game!
Key Hob Point

Can prove properties that relate directly to concepts that users understand
No Relations in Hob

• Hash table
  Hash.insert(k1, v1);
  Hash.insert(k2, v2);
  v = Hash.lookup(k);

• What Hob can specify
  If k ∈ {k1, k2} then v ∈ {v1, v2}
  Otherwise v = null

• What Hob can’t specify
  If k = k1 then v = v1
  If k = k2 then v = v2
  Otherwise v = null
Jahob

• Annotation and verification system for Java
• Data structure consistency properties
• Includes relations
Jahob Hash Table Implementation

Layered implementation: Hash Table uses Association List
Jahob Hash Table Specification

• Hash table = 
  { <k1, v1>, <k2, v2>, <k3, v3>, <k4, v4>, <k5, v5> }
• insert(k, v) adds <k, v> to Hash table
• lookup(k) returns v such that either
  • <k, v> in Hash table
  • v is null and no <k, x> in Hash table
• Complete functional correctness
  (modulo non-termination) of data structure
Verifying Properties

• Complex properties
  • Full boolean algebra
  • Lots of quantifiers over sets of objects
• Basic idea
  • Derive verification conditions
  • Discharge conditions in Isabelle using
    • Proof assistants
    • Clever decision procedures
    • Manual theorem proving
      (graduate students)
Results

- Hob specification and verification system
- Systems implemented in Hob
  - Data structures (lists, trees, arrays)
  - Minesweeper
  - Hob web server
  - Water (scientific computation)
- Jahob specification and verification system
- Data structures implemented in Jahob
  - Doubly-linked list, array list, association list
  - Hash table, binary search tree
  - Priority queue (balanced tree stored in array)
Future

- Library of verified data structures
- Loop invariant inference
- Program analysis for simplifying data structure verification
Discussion Point

• We can prove/verify very sophisticated properties about lots of data structures
• What properties are useful for people here?