A Case Study in Automated Verification

Jason Kirschenbaum,
Heather Harton and Murali Sitaraman

Introduction

- Goal is to investigate automatic verification of extensions to software components
  - Including the development process
- Selection Sort Example
  - Simple
  - Theory Development
  - Specifications

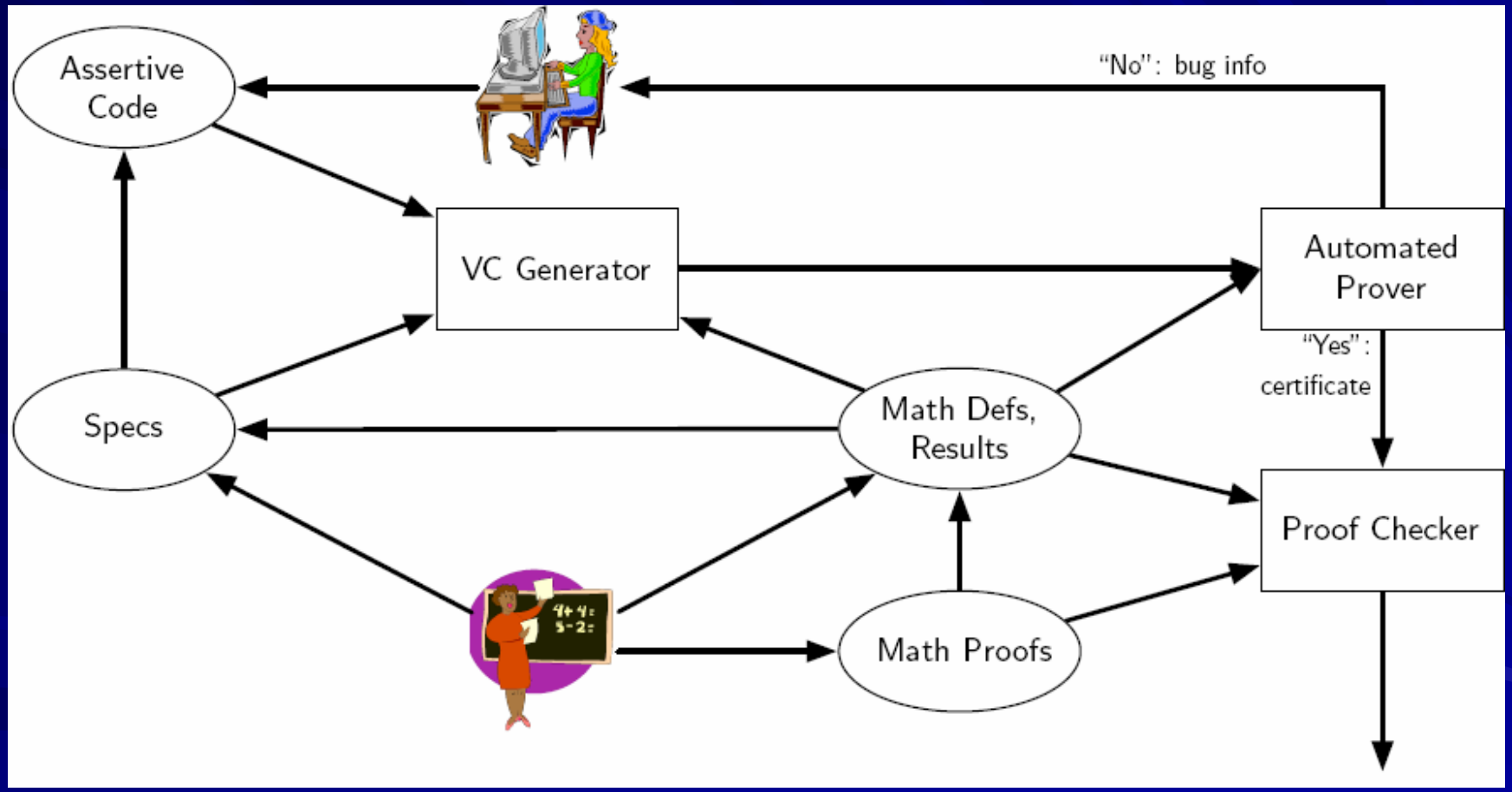
Focus of this Work

- Full Behavioral Verification
- Language with Clean Semantics
- Modular Verification
- Generic Specification and Implementation

Resolve Language

- Design by Contract
- Model Based Specifications
- Modular Reasoning
- Layered Implementations
- Parameterized Code
- Value Semantics
- Alias Avoidance
- Total Correctness
- Code Annotation Constructs

Vision of Development Process



Isabelle

- Higher Order
- Large Library of Verified Theories
- Both Automated and User-guided proof methods

Selection Sort Specification

```
Enhancement Sort_Capability(def
                              LEQV(x,y: Entry): B)
            for Queue_Template;
      requires Is_Total_Preordering(LEQV);

    Definition IsNondecreasing(a: Str(Entry)): B=
      (for all b,c: Str(Entry),
          for all x,y: Entry,
            if a = b o <x> o <y> o c
              then LEQV(x,y));

    Operation Sort_Q(updates Q: Queue);
        ensures IsNondecreasing (Q)
                and IsPermutation(#Q,Q);
end Sort_Capability;
```

```
Begin String_Theory

Definition String( Str(G : Set) : Set,
                   empty_string : Str(G),
                   ext( a : Str(G), x : G): B =

...

Inductive Definition on b : Str(G : Set)
                of (a : Str(G)) o (b : Str(G)) is
...

Lemma ConcatenationAssociative
 a o (b o c) = (a o b) o c

Definition IsPermutation where
 IsPermutation a b = ...

Lemma PermutationCommutative
 IsPermutation (x o y) (y o x)

...
End String_Theory
```

Figure 1: Example of String Theory

Challenges

- Human Errors
  - Specification bugs
- Tool Weaknesses
  - Lemmas involving permutation
  - Proof is too long for Isabelle to find

Challenge 1: Human Errors

```
Definition IsASmallest(a: Str(Entry);
                       x: Entry): B=
    (for all b,c: Str(Entry),
      for all y: Entry,
        if a = b o <y> o c then LEQV(x,y));

Operation Remove_Min(updates Q: Queue;
                     replaces min: Entry);
   requires |Q| /= 0;
   ensures IsPermutation(Q o <min>, #Q)
           and IsASmallest(Q, min);
-- code omitted

 Procedure Sort_Q(updates Q: Queue);
  Var sorted:Queue;
  Var min:Entry;
  While (Length(Q) /= 0)
   changing Q, sorted, min;
   maintaining IsPermutation(Q o sorted, #Q)
               and IsNondecreasing(sorted);
   decreasing |Q|;
  do
     Remove_Min(Q, min);
     Enqueue(min, sorted);
  end;
  Q :=: sorted;

 end Sort_Q;
```

Figure 2: Original Sort_Q Implementation

Corrected Implementation

```
Definition IsASmallest(a: Str(Entry);
                       x: Entry): B=
      (for all b,c: Str(Entry),
        for all y: Entry,
          if a = b o <y> o c then LEQV(x,y));

Operation Remove_Min(updates Q: Queue;
                     replaces min: Entry);
    requires |Q| /= 0;
    ensures IsPermutation(Q o <min>, #Q)
            and IsASmallest(Q, min);
-- code omitted

 Procedure Sort_Q(updates Q: Queue);
  Var sorted:Queue;
  Var min:Entry;
  While (Length(Q) /= 0)
   changing Q, sorted, min;
   maintaining IsPermutation(Q o sorted, #Q)
               and IsNondecreasing(sorted);
   decreasing |Q|;
  do
     Remove_Min(Q, min);
     Enqueue(min, sorted);
  end;
  Q :=: sorted;

 end Sort_Q;
```

Figure 2: Original Sort_Q Implementation

```
Operation Remove_Min(updates Q: Queue;
                     replaces min: Entry);
    requires |Q| /= 0;
    ensures IsPermutation(Q o <min>, #Q)
            and IsASmallest(#Q, min);
-- code omitted

 Procedure Sort_Q(updates Q: Queue);
  Var sorted:Queue;
  Var min:Entry;
  While (Length(Q) /= 0)
   changing Q, sorted, min;
   maintaining IsPermutation(Q o sorted, #Q)
      and IsNondecreasing(sorted)
      and
       (for all y: Entry,
          if IsSubstring (<y>, Q) then
              IsNondecreasing(sorted o <y>));
   decreasing |Q|;
   do
     Remove_Min(Q, min);
     Enqueue(min, sorted);
   end;
   Q :=: sorted;

 end Sort_Q;
```

Figure 3: Corrected Sort_Q Implementation

Challenge 2: Automation

- Verification Condition Proofs Issues
  - Permutation Development
  - Length of Proof

First Type of Problem VC

```
lemma 4:
"[|
 (min_int <= 0);
 (0 < max_int) ;
 (Max_Length > 0);
 is_initial min;
 (length Q <= Max_Length);
 length Q ~= 0;
 Q = (<min2> o Q3);
 IsPermutation ((temp1 o Q2) o <min1>) Q;
 IsASmallest (temp1 o <min1>) min1;
 (length Q2 ~= 0);
 Q2 = (<x1> o Q1);
 x1 <= min1
|]
==>
 IsPermutation (((temp1 o <min1>)
                 o Q1) o <x1>) Q "
apply auto
apply (unfold IsPermutation_def)
apply (auto)
apply (drule_tac x=x in spec)
apply auto
done
```

Figure 4: First Case of Problem VCs

Second Type of Problem VC

```
lemma 10:
"[|
 (min_int <= 0);
 (0 < max_int);
 (Max_Length > 0);
 (length Q <= Max_Length);
 (IsPermutation (Q2 o sorted1) Q);
 (IsNondecreasing sorted1);
 ALL y. Is_SubString <y> Q2
        --> (IsNondecreasing (sorted1 o <y>));
 (length Q2 ~= 0);
 (IsPermutation (Q1 o <min1>) Q2);
 (IsASmallest Q2 min1)
|]
==>
 ALL y. Is_SubString <y> Q1 -->
    (IsNondecreasing ((sorted1 o <min1>) o <y>))"
apply auto
apply (rule ND4)
apply auto
done
```

Figure 5: Second Case of Problem VCs

```
lemma ND4:
"
 [|
  IsNondecreasing (a o <y>);
  y <= b
 |]
==>
   IsNondecreasing ((a o <y>) o <b>)
"
```

Figure 6: Lemma ND4

Proposed Theory Development

```
Begin String_Theory

...

Lemma PermutationCommutative
 IsPermutation (x o y) (y o x)
     ...

Definition Is_Conformal_w (R: (x: G, y: G) : B,
                           a : Str(G)) : B
 = for all x, y.
        If Is_Substring(<x> o <y>, a)
         Then R(x,y)

...

Definition Universally_Relates_to(
                           R: (x: G, y: G) :B,
                           a : Str(G),
                           b : Str(G))i : B
 = for all x, y. If Is_Substring(<x>, a)
                    and Is_Substring(<y>, b)
                 Then R(x,y)

...

...
End String_Theory
```

Figure 9: Definitions of Is_Conformal_w and Universally_Relates_to

Updated Implementation

```
    Definition IsPreceding ...

    Operation Remove_Min(updates Q: Queue;
                         replaces min: Entry);
       requires |Q| /= 0;
       ensures IsPermutation(Q o <min>, #Q)
               and IsPreceding(<min>, #Q);
-- code omitted

     Procedure Sort_Q(updates Q: Queue);
      Var sorted:Queue;
      Var min:Entry;
      While (Length(Q) /= 0)
       changing Q, sorted, min;
       maintaining IsPermutation(Q o sorted, #Q)
            and IsNondecreasing(sorted)
            and IsPreceding(<min>, Q);
       decreasing |Q|;
       do
        Remove_Min(Q, min);
        Enqueue(min, sorted);
       end;
       Q :=: sorted;

     end Sort_Q;
```

Figure 7: Modified Theory Development for Specification

Questions?