Contents

- This slide deck summarizes the evaluation of the VSTTE 2010 verification competition
- The comments refer to the versions submitted in time. Versions that arrived late are mentioned on the slides and included in the zip file, but not commented on
- Further discussions are encouraged and should be posted on http://verifythis.cost-ic0701.org/

Overview: Submissions Received

	Problem 1	Problem 2	Problem 3	Problem 4	Problem5
Alexandra.Tsyban	X		X		
anonymousHolHacker					X
holfoot	X				
KeY	X	X			
Leino	X	X	X	X	X (12 mins late)
SPARKuLike	X				
monapoli	X		X		
Resolve					X (before contest)
RobArthan	X		X		
VC Crushers	X		X		
VeriFast	X		X		(Lists only)

http://www.macs.hw.ac.uk/vstte10/Solutions.zip

Problem 1: Sum and Max

- Alexandra Tsyban
  - Isabelle/HOL/Vcg
  - Complete: functional, partial correctness
  - Automation: Axiom, two lemmas, and main proof, all interactive
- KeY
  - Java/JML verifier
  - Completeness: Total correctness for specs including sum/max properties
  - Automation: Proved in 6 seconds
- Monapoli
  - Boogie
  - Completeness: Contracts, but script not shown
  - Automation: Proved in 2 seconds

Problem 1: Sum and Max

- Leino
  - Dafny/Boogie/Z3
  - Completeness: Total correctness
  - Automation: Verified 4 proofs in 2 seconds
- Rob Arthan
  - Proofpower/Z
  - Completeness: Total correctness with functional program, literate presentation
  - Automation: Interactive proof
- SPARKuLike
  - SPARK/Ada
  - Completeness: Total correctness with numeric bounds
  - Automation: 18 VCs; 4 interactive (comprehensive documentation)

Problem 1: Sum and Max

- VC Crushers
  - C verification from contracts
  - Completeness: Partial correctness; Uses unproved lemma
  - Automation: fully automatic in 2.3 seconds
- VeriFast
  - C Verification with separation logic
  - Completeness: Partial correctness; Uses unproved lemmas
  - Automation: Several definitions/lemmas
- HOLFoot
  - HOL/Separation Logic
  - Completeness: Total correctness
  - Automation: Simple interactive proof

Problem 1: KeY Solution

```
class MaxSum {

  int sum;
  int max;

  /*@ public normal_behaviour
    @   requires (\forall int i; 0<=i && i < a.length; a[i] >=
        0);
    @   ensures (\forall int i; 0<=i && i < a.length; max >=
        a[i]);
    @   ensures (a.length > 0 ==>
    @        (\exists int i; 0<=i && i < a.length; max == a[i]));
    @   ensures sum == (\sum int i; 0<= i && i < a.length;
        a[i]);
    @   ensures sum <= a.length * max;
    @*/
  void sumAndMax(int[] a) {
    sum = 0;
    max = 0;
    int k = 0;

    /*@ loop_invariant
      @     (\forall int i; 0<=i && i < k; max >= a[i])
      @  && (k > 0 ==> (\exists int i; 0<=i && i < k; max ==
       a[i]))
      @  && sum == (\bsum int i; 0; k; a[i])
      @  && sum <= k * max
      @  && 0 <= k && k <= a.length
      @  && (k == 0 ==> max == 0);
      @
      @  decreases a.length - k;
      @  modifies max, sum, k;
      @*/
    while(k < a.length){
      if (max < a[k]){
        max = a[k];
      }
      sum += a[k];
      k++;
    }
  }
}
```

Problem 1: Dafny Solution

```
method M(N: int, a: array<int>) returns (sum: int, max: int)
  requires 0 <= N && a != null && |a| == N && (forall k :: 0 <= k && k < N ==> 0 <= a[k]);
  ensures sum <= N * max;
{
  sum := 0;
  max := 0;
  var i := 0;
  while (i < N)
    invariant i <= N && sum <= i * max;
  {
    if (max < a[i]) {
      max := a[i];
    }
    sum := sum + a[i];
    i := i + 1;
  }
}
```

Problem 2: Invert Array

- KeY
  - JML annotated Java, using the KeY tool
  - Complete: functional correctness, termination, and framing
  - Automation: two manual rule instantiations; the rest is automatic
  - Very clear specifications, but longer than the code
- Leino
  - Dafny
  - Completeness: only preconditions specified (full specification arrived at 5:20am)
  - Automation: fully automatic, 2 second verification
  - Elegant specifications, nice notation

Problem 2: KeY Solution

```
/*@ public normal_behaviour
   requires a != b;
   requires a.length == b.length;
   requires (\forall int x; 0 <= x && x < a.length; 0 <= a[x] && a[x] < a.length);
   requires (\forall int x, y; 0 <= x && x < y && y < a.length; a[x] != a[y]);
   requires (\forall int q; 0 <= q && q < a.length; (\exists int w; 0 <= w && w < a.length; a[w] == q));
   assignable b[*];
   ensures (\forall int x, y; 0 <= x && x < y && y < b.length; b[x] != b[y]);
   ensures (\forall int x; 0 <= x && x < b.length; b[a[x]] == x);
   @*/

  public static void invert(int[] a, int[] b) {
    /*@ loop_invariant 0 <= i && i <= a.length
      @    && (\forall int x; 0 <= x && x < i; b[a[x]] == x);
      @  modifies i, b[*];
      @  decreases a.length - i;
      @*/
    for(int i = 0; i < a.length; i++) {
      b[a[i]] = i;
    }
  }
```

Problem 3: List Traversal

- Alexandra.Tsyban
  - Isabelle, based on VCG theory
  - Completeness: partial correctness
  - Automation: ~50 lines of interactive proof
  - Elegance: concise specification; fields encoded as functions
- Leino
  - Dafny
  - Completeness: total correctness
  - Automation: fully automatic, 2.6secs; includes Cons and client code
  - Elegance: concise specification
- Monapoli
  - Boogie
  - Automation: fully automatic, 2.3 secs; uses two unproven lemmas
  - Completeness: partial correctness
  - Elegance: uses several auxiliary functions; fields encoded as functions

Problem 3 (cont'd)

- RobArthan
  - ProofPower-HOL
  - Completeness: total correctness for ML implementation
  - Automation: ~25 lines of interactive proof
  - Elegance: concise and general specification
- VC Crushers
  - VCC
  - Completeness : partial correctness
  - Automation: fully automatic
  - Elegance: much overhead for data structure and data abstraction (invariants)
- VeriFast
  - Java
  - Completeness : partial correctness
  - Automation: fully automatic , including proven lemmas
  - Elegance: uses several auxiliary functions and predicates

Problem 4: N-Queens

- Leino
  - Dafny
  - Completeness: partial solution (IsConsistent is uninterpreted; uses assumptions)
    - Almost complete solution submitted at 6:50am (one assume statement left)
  - Automation: fully automatic, 2.2secs

Problem 5: AmortizedQueue

- Leino
  - Dafny
  - Original partial (only LinkedList), Late: complete
  - Automation: fully automatic 9 sec (12 for whole)
  - Elegant solution, late part only took 12 more min.
- Anonymous HOL Hacker
  - HOL functional code + proof script
  - Completeness: complete, but uses HOL lists
  - Automation: most of proof could have been
  - Specifications just stated as theorems

Problem 5: Solutions out of Competition

- Resolve
  - Done before contest (on web)
  - Complete solution with proofs (but no running code)
  - Automation: proofs are automated with SplitDecision (proof time ~100ms)
  - Two-state loop invariants interesting
  - Fully worked out and proved, some unusual aspects
- VeriFast
  - Java with VeriFast annotations
  - Completeness: Only the List nodes
  - Automation: some close annotations needed
  - Nice partial solution