The 2nd Verified Software Competition
Experience Report

Jean-Christophe Filliâtre  Andrei Paskevich  Aaron Stump

VSTTE
Philadelphia, January 28, 2012
• on-site competitions
  ▶ VSTTE 2010 / 2 hours / 5 problems
    (Peter Müller, Natarajan Shankar)
  ▶ FoVeOOS 2011 / 2.5 hours / 3 problems
    (Marieke Huisman, Vladimir Klebanov, Rosemary Monahan)

• long-term challenges
  ▶ VACID-0 / 5 problems
    (Rustan Leino, Michał Moskal)
inspired by the ICFP programming contest
  • more challenging problems
  • over a short period (2/3 days)

but
  • algorithm is given
  • solution = specification + mechanized proof

a completely different evaluation process
  • adequacy of a specification cannot be judged mechanically
• first announcement on Sep 30
  ▶ second call on Oct 7
  ▶ last call on Nov 1 (“one week to go”)

• competition from Nov 8 15:00 UTC to Nov 10 15:00 UTC
  ▶ problems put on the web
  ▶ solutions sent by email

• winner(s) private notification on Dec 12
• **team** work is allowed
  (only teams up to 4 members are eligible for the first prize)
• any software used in the solutions should be **freely** available
  for noncommercial use to the public
• software must be usable on **x86 Linux or Windows**
• participants can **modify** their tools during the competition
find a balance between

- purely applicative vs imperative style
- data structures vs algorithms
- easy vs difficult

5 independent problems
1. Two-Way Sort (50 points)
   sort an array of Boolean values

2. Combinators (100 points)
   call-by-value reduction of SK-terms

3. Ring Buffer (150 points)
   queue data structure in a circular array

4. Tree Reconstruction (150 points)
   build a binary tree from a list of leaf depths

5. Breadth-First Search (150 points)
   search for a shortest path in a directed graph
Participants

- 29 submissions
- 79 participants
  - 8 teams of size 1
  - 6 teams of size 2
  - 4 teams of size 3
  - 10 teams of size 4
  - 1 team of size 9

- ACL2 (1)
- Agda (3)
- ATS (1)
- B (2)
- BLAST (1)
- CBMC (1)
- Coq (7)
- Dafny (6)
- Escher (1)
- Guru (1)
- HIP (1)

- Holfoot (1)
- Isabelle (2)
- KeY (1)
- KIV (1)
- PAT (1)
- PML (1)
- PVS (3)
- Socos (1)
- VCC (2)
- VeriFast (1)
- Ynot (1)
a group of excellent submissions with tied scores
⇒ we opted for 6 medalists: 2 bronze, 2 silver, 2 gold

and they are...
Bronze Medalists (590 points)

- **eam (VCC)**
  - Ernie Cohen
  - Michał Moskal

- **JasonAndNadia (Dafny)**
  - Jason Koenig
  - Nadia Polikarpova
Silver Medalists (595 points)

- SRI (PVS)
  - Sam Owre
  - N. Shankar

- LeinoMuller (Dafny)
  - Rustan Leino
  - Peter Müller
Gold Medalists (600 points)

- acl2-dkms
  - Jared Davis
  - Matt Kaufmann
  - J Strother Moore
  - Sol Swords

- KIV
  - Gidon Ernst
  - Gerhard Schellhorn
  - Kurt Stenzel
  - Bogdan Tofan
some feedback from the organizers
• a larger set of problems
  ▶ Booth algorithm
  ▶ in-place inversion of a permutation
  ▶ stable counting sort
• solutions in Why3
• beta-testing
  ▶ are the problems too easy / too difficult?
  ▶ make a selection
Organization

- **announces** on various mailing lists
- **web page**
  - hosted on the VSTTE web site (Google sites)
    - https://sites.google.com/site/vstte2012/competition
- **mailing list** for the competition
  - Google group vstte-2012-verification-competition
- **mailbox** for submissions
  - vstte-2012-competition@lri.fr
Sequence of Events

- **before the competition**
  - a few discussions on the mailing list or in private
- **during the competition**
  - “night watch” (2 in Europe, 1 in USA)
  - a few questions on the mailing list
- **after the competition**
  - we sent acknowledgment emails (was useful)
  - we invited participants to share their solutions
- **evaluation process**
Evaluation Process

1. proofreading code and specification

2. installing and running tools, inserting errors
1. proofreading code and specification
   ▶ what makes it easy
     • Principle of Least Astonishment
   ▶ what makes it hard
     • $ar[i \rightarrow n(i)]$ as a notation for array access
     • non human-readable format
     • code, spec, and proof tangled

2. installing and running tools, inserting errors
Evaluation Process

1. proofreading code and specification

2. installing and running tools, inserting errors
   ▶ what makes it easy
     • packages
     • tool and prover(s) come together
   ▶ what makes it hard
     • installation issues
● we hope to take part in next competitions
● a submission server would be a good idea
● always hire several organizers, on both sides of the Atlantic
• beta-testing
  Claude Marché, Duckki Oe

• VSTTE 2012 chairs
  Ernie Cohen, Rajeev Joshi, Peter Müller, Andreas Podelski

• publicity
  Gudmund Grov

• technical support
  LRI’s staff
two-way-sort (a: array of boolean) :=
    i <- 0;
    j <- length(a) - 1;
    while i <= j do
        if not a[i] then
            i <- i+1
        elseif a[j] then
            j <- j - 1
        else
            swap(a, i, j);
            i <- i + 1;
            j <- j - 1
        endif
    endwhile
Problem 1: Two-Way Sort

1. Safety. Verify that every array access is made within bounds.

2. Termination. Prove that function `two_way_sort` always terminates.

3. Behavior. Verify that after execution of function `two_way_sort`, the following properties hold.
   3.1 Array `a` is sorted in increasing order.
   3.2 Array `a` is a permutation of its initial contents.
Problem 2: Combinators

**terms**  
\[ t ::= S \mid K \mid (t \, t) \]

**CBV contexts**  
\[ C ::= □ \mid (C \, t) \mid (v \, C) \]

**values**  
\[ v ::= K \mid S \mid (K \, v) \mid (S \, v) \mid ((S \, v) \, v) \]

\[ □[t] = t \]
\[ (C \, t_1)[t] = (C[t] \, t_1) \]
\[ (v \, C)[t] = (v \, C[t]) \]

\[ C[((K \, v_1) \, v_2)] \rightarrow C[v_1] \]
\[ C[((S \, v_2) \, v_3)] \rightarrow C[((v_1 \, v_3) \, (v_2 \, v_3))] \]
Problem 2: Combinators

Implementation Task

1. Implement a function `reduction` which, when given a combinator term \( t \) as input, returns a term \( t' \) such that \( t \rightarrow^* t' \) and \( t' \not\rightarrow \), or loops if there is no such term.

Verification Tasks

1. Prove that if `reduction(t)` returns \( t' \), then \( t \rightarrow^* t' \) and \( t' \not\rightarrow \).
2. Prove that function `reduction` terminates on any term which does not contain \( S \).
3. Consider the meta-language function \( ks \) defined by

   \[
   ks \, 0 = K \\
   ks \, (n + 1) = ((ks \, n) \, K)
   \]

   Prove that `reduction` applied to the term \( (ks \, n) \) returns \( K \) when \( n \) is even, and \( (K \, K) \) when \( n \) is odd.
Problem 3: Ring Buffer

type ring_buffer = record
  data : array of int;  // buffer contents
  size : int;          // buffer capacity
  first: int;          // queue head, if any
  len : int;           // queue length
end
Problem 3: Ring Buffer

create(n: int): ring_buffer :=
  return new ring_buffer(
    data = new array[n] of int;
    size = n; first = 0; len = 0)

clear(b: ring_buffer) :=
  b.len <- 0

head(b: ring_buffer): int :=
  return b.data[b.first]

push(b: ring_buffer, x: int) :=
  b.data[(b.first + b.len) mod b.size] <- x;
  b.len <- b.len + 1

pop(b: ring_buffer): int :=
  r <- b.data[b.first];
  b.first <- (b.first + 1) mod b.size;
  b.len <- b.len - 1;
  return r
Problem 3: Ring Buffer

1. Safety. Verify that every array access is made within bounds.

2. Behavior. Verify the correctness of your implementation w.r.t. the first-in first-out semantics of a queue.

3. Harness. The following test harness should be verified.

```plaintext
test (x: int, y: int, z: int) :=
  b <- create(2);
  push(b, x);
  push(b, y);
  h <- pop(b); assert h = x;
  push(b, z);
  h <- pop(b); assert h = y;
  h <- pop(b); assert h = z;
```
Problem 4: Tree Reconstruction

1, 3, 3, 2
Problem 4: Tree Reconstruction

```
type tree
Leaf(): tree
Node(l:tree, r:tree): tree

type list
is_empty(s: list): boolean
head(s: list): int
pop(s: list)
```

1, 3, 3, 2
Problem 4: Tree Reconstruction

\[
\text{build\_rec}(d: \textbf{int}, s: \textbf{list}): \textbf{tree} := \\
\quad \text{if is\_empty}(s) \text{ then fail; endif} \\
\quad h <- \text{head}(s); \\
\quad \text{if } h < d \text{ then fail; endif} \\
\quad \text{if } h = d \text{ then pop}(s); \text{ return Leaf(); endif} \\
\quad l <- \text{build\_rec}(d+1, s); \\
\quad r <- \text{build\_rec}(d+1, s); \\
\quad \text{return Node}(l, r) \\
\]

\[
\text{build}(s: \textbf{list}): \textbf{tree} := \\
\quad t <- \text{build\_rec}(0, s); \\
\quad \text{if not is\_empty}(s) \text{ then fail; endif} \\
\quad \text{return t} \\
\]
1. **Soundness.** Verify that whenever function `build` successfully returns a tree the depths of its leaves are exactly those passed in the argument list.

2. **Completeness.** Verify that whenever function `build` reports failure there is no tree that corresponds to the argument list.

3. **Termination.** Prove that function `build` always terminates.

4. **Harness.** The following test harness should be verified:
   - Verify that `build` applied to the list 1, 3, 3, 2 returns the tree `Node(Leaf, Node(Node(Leaf, Leaf), Leaf))`.
   - Verify that `build` applied to the list 1, 3, 2, 2 reports failure.
Problem 5: Breadth-First Search

```plaintext
bfs(source: vertex, dest: vertex): int :=
V <- {source}; C <- {source}; N <- {};
d <- 0;
while C is not empty do
    remove one vertex v from C;
    if v = dest then return d; endif
    for each w in succ(v) do
        if w is not in V then
            add w to V;
            add w to N;
        endif
    endfor
    if C is empty then
        C <- N;
        N <- {};
        d <- d + 1;
    endif
endwhile
fail "no path"
```
Problem 5: Breadth-First Search

1. **Soundness.** Verify that whenever function `bfs` returns an integer $n$ this is indeed the length of the shortest path from source to dest.

   A partial score is attributed if it is only proved that there exists a path of length $n$ from source to dest.

2. **Completeness.** Verify that whenever function `bfs` reports failure there is no path from source to dest.