VerifyThis Benchmarks for SV-COMP

Gidon Ernst, (Gregor Alexandru)

LMU Munich, Germany
gidon.ernst@lmu.de

CPA 2019
On-site program verification competition
▶ FoVeOOS 2011, FM 2012, ETAPS 2015–2019
▶ permanent SC; changing organizers each year
▶ ≈20 participants per event, in teams

Goals: describe state-of-the-art; community exchange

1st day: Tutorial, then 3 Challenges, 90 min each
2nd day: Discussions: judging || plenum
On-site program verification competition
  ▶ FoVeOOS 2011, FM 2012, ETAPS 2015–2019
  ▶ permanent SC; changing organizers each year
  ▶ \( \approx 20 \) participants per event, in teams

Goals: describe state-of-the-art; community exchange

1st day: Tutorial, then 3 Challenges, 90 min each

2nd day: Discussions: judging || plenum

Plenty of interesting verification challenges!
But: hard for SV-COMP verifiers
(need quantifiers, \( \backslash \)old, abstractions, inductive lemmas)
On-site program verification competition
- FoVeOOS 2011, FM 2012, ETAPS 2015–2019
- permanent SC; changing organizers each year
- ≈20 participants per event, in teams

Goals: describe state-of-the-art; community exchange

1st day: Tutorial, then 3 Challenges, 90 min each

2nd day: Discussions: judging || plenum

Plenty of interesting verification challenges!
But: hard for SV-COMP verifiers
(need quantifiers, \old, abstractions, inductive lemmas)

You might want to be able to solve them:
6 challenges from 2011/2012 now in sv-benchmarks
Challenge Descriptions

► Given: pseudocode + natural language spec
► Task: write down algorithm, formalize spec, prove correctness
► Characteristics: arrays, heap, concurrency
► Some tools: Dafny, Why3, VeriFast, KeY, KIV, VerCors, Isabelle, Frama-C, Viper, mCRL2, CIVL, ...
► [www.pm.inf.ethz.ch/research/verifythis.html](http://www.pm.inf.ethz.ch/research/verifythis.html)
Example Challenge: Linked Tree

```c
struct node *tree_del(struct node *t, int *min) {
    struct node *r;
    if (!t->left) {
        *min = t->data; r = t->right; free(t); return r;
    } else {
        t->left = tree_del(t->left, min);
        return t;
    }
}

void task(struct node *t) {
    int n = size(t);
    int a = min(t), b; int x[n], y[n-1];
    tree_inorder(t, x);
    assert(a == x[0]);

    struct node *r = tree_del(t, &b);
    assert(a == b);
    tree_inorder(t, y);

    int i;
    assume(0 <= i < n-1);
    assert(x[i+1] == y[i]);
}
```
Example Challenge: Elimination Max

```c
int max(int a[n], int n) {
    assume(0 < n);

    int i = 0, j = n-1;
    while(i < j) {
        if(a[i] < a[j]) i++;
        else j--;
    }

    int k;
    assume(0 <= k < n);
    assert(a[k] <= a[i]);
    return i;
}
```

Invariant (Why3)
\[ \forall k. (0 \leq k < i \lor j < k < n) \implies (a[k] \leq a[i] \land a[k] \leq a[j]) \]

Invariant (KIV)
\[ \exists k. (0 \leq i \leq k \leq j < n) \land a[k] = \text{max}\{a[0], \ldots, a[n-1]\} \]
Example Challenge: Elimination Max

```c
int max(int a[n], int n) {
    assume(0 < n);

    int i = 0, j = n-1;
    while(i < j) {
        if(a[i] < a[j]) i++; else j--;
    }

    int k;
    assume(0 <= k < n);
    assert(a[k] <= a[i]);
    return i;
}
```

▶ Invariant (Why3)
\[
\forall k. (0 \leq k < i \lor j < k < n) \implies (a[k] \leq a[i] \land a[k] \leq a[j])
\]
Example Challenge: Elimination Max

```c
int max(int a[n], int n) {
    assume(0 < n);

    int i = 0, j = n-1;
    while(i < j) {
        if(a[i] < a[j]) i++; else j--;
    }

    int k;
    assume(0 <= k < n);
    assert(a[k] <= a[i]);
    return i;
}
```

► Invariant (Why3)
\[ \forall k. (0 \leq k < i \lor j < k < n) \implies (a[k] \leq a[i] \land a[k] \leq a[j]) \]

► Invariant (KIV)
\[ \exists k. (0 \leq i \leq k \leq j < n) \land a[k] = max\{a[0], .., a[n - 1]\} \]
Example Challenge: Elimination Max

```c
int max(int a[n], int n) {
    assume(0 < n);
    int i = 0, j = n-1;
    while(i < j) {
        if(a[i] < a[j]) i++;
        else j--;
    }
    int k;
    assume(\old(i) <= k < \old(j)+1);
    assert(a[k] <= a[i]);
    return i;
}
```

- **Invariant (Why3)**
  \[ \forall k. \ (0 \leq k < i \lor j < k < n) \implies (a[k] \leq a[i] \land a[k] \leq a[j]) \]

- **Invariant (KIV)**
  \[ \exists k. \ (0 \leq i \leq k \leq j < n) \land a[k] = \max\{a[0], .., a[n - 1]\} \]

- **Sufficient: invariant** \[0 \leq i \leq j < n\] & induction & \old
Demo
Recursive Variant

```c
int max(int a[n], int i, int j, int n, int k) {
    int m;
    if(i >= j) {
        m = i;
    } else {
        if(a[i] < a[j]) i++; else j--;
        m = max(a, i, j, n, k);
    }

    if(i <= k <= j);
    assert(a[k] <= a[m]);
    return m;
}
```

Note trick: fix the $k$ upfront which we want to compare!
Recursive Variant

```c
int max(int a[n], int i, int j, int n, int k) {
    int m;
    if(i >= j) {
        m = i;
    } else {
        if(a[i] < a[j]) i++;
        else j--;
        m = max(a, i, j, n, k);
    }

    if(i <= k <= j);
    assert(a[k] <= a[m]);
    return m;
}
```

Note trick: fix the $k$ upfront which we want to compare!

Should be in reach of SV-COMP verifiers:
no quantifier in procedure summary
Reasoning about Loops

▶ Hoare’s invariant rule

\[
\frac{\{I \land c\} \ p \ \{I\}}{\{I\} \ \text{while}(c)\{p\} \ \{I \land \neg c\}}
\]
Reasoning about Loops

▶ Hoare’s invariant rule

\[
\frac{\{ I \land c \} \quad p \quad \{ I \}}{\{ I \} \quad \text{while}(c)\{ p \} \quad \{ I \land \neg c \}}
\]

▶ Loop specifications (e.g. Morgan, Hehner, Tuerk)

▶ Intuition: treat loop as recursive procedure, use induction
  ▶ base case = loop termination
  ▶ step case = unwind once, assume that rest of program will be safe after remaining iterations
Reasoning about Loops

- Hoare’s invariant rule

\[
\begin{align*}
\{I \land c\} & \quad p \quad \{I\} \\
\{I\} & \quad \text{while}(c)\{p\} \quad \{I \land \neg c\}
\end{align*}
\]

- Loop specifications (e.g. Morgan, Hehner, Tuerk)
- Intuition: treat loop as recursive procedure, use induction
  - base case = loop termination
  - step case = unwind once, assume that rest of program will be safe after remaining iterations

\[
\begin{align*}
\{P_x \land \neg c \land x_0 = x\} & \quad p_2 \quad \{Q^{x_0}_x\} \\
\{P_x \land c \land x_i = x\} & \quad p_1 \quad \{P_x \land Q^{x_i}_{x_n} \Rightarrow Q^x_{x_n}\} \\
\{P_x \land x_0 = x\} & \quad \text{while}(c)\{p_1\}; \quad p_2 \quad \{Q^{x_0}_x\}
\end{align*}
\]
Comparison of Proof Structure

Hoare’s rule

Loop specifications

(Illustration by Gregor Alexandru)
Comparison of Proof Structure

Hoare’s rule

not like k-Induction

(Illustration by Gregor Alexandru)

(not like k-Induction
(which does not use hypothesis about remaining execution)
Proof Sketch

- **Base case:**
  
  ```
  assume(0 < i_0 <= k <= j_0 < n);
  assume(i_0 >= j_0);
  assert(a[k] <= a[i_0]);
  ```
Proof Sketch

Base case:
\[
\begin{align*}
&\text{assume}(0 < i_0 \leq k \leq j_0 < n); \\
&\text{assume}(i_0 \geq j_0); \\
&\text{assert}(a[k] \leq a[i_0]); \\
\end{align*}
\]

Step case:
\[
\begin{align*}
&\text{assume}(0 < i_0 \leq k \leq j_0 < n); \\
&\text{assume}(i_0 < j_0); \\
&\text{if}(a[i_0] < a[j_0]) \ i_1 = i_0 + 1; \ 	ext{else} \ j_1 = j_0 - 1; \\
&\text{assume(IndHyp)}; \\
&\text{assert}(a[k] \leq a[i_n]); \ // \ i_n \text{ denotes final result} \\
\end{align*}
\]

where IndHyp is that for all \( k \) this is safe:
\[
\begin{align*}
&\text{assume}(0 < i_1 \leq k \leq j_1 < n); \ // \ 	ext{different bounds!} \\
&\text{assert}(a[k] \leq a[i_n]); \\
\end{align*}
\]
Onwards to new and difficult tasks!

- Some benchmarks are available, more to come
  - Properties $\rightsquigarrow$ (recursive) procedures
  - Quantifiers $\rightsquigarrow$ nondeterministic choice

Big thanks to: Vladimir Klebanov, Marieke Huisman, Rosemary Monahan, Peter Müller, Mattias Ulbrich, and all VerifyThis organizers.
Onwards to new and difficult tasks!

- Some benchmarks are available, more to come
  - Properties \(\rightsquigarrow\) (recursive) procedures
  - Quantifiers \(\rightsquigarrow\) nondeterministic choice

- Loop specifications:
  - Can be taken from assertions/postconditions, mild generalizations necessary
    (btw: old not required in assertion language)
  - Actual loop invariants typically simple
  - Alternative to existing summarization techniques?

Big thanks to: Vladimir Klebanov, Marieke Huisman, Rosemary Monahan, Peter Müller, Mattias Ulbrich, and all VerifyThis organizers.
Onwards to new and difficult tasks!

- Some benchmarks are available, more to come
  - Properties $\leadsto$ (recursive) procedures
  - Quantifiers $\leadsto$ nondeterministic choice
- Loop specifications:
  - Can be taken from assertions/postconditions, mild generalizations necessary
    (btw: old not required in assertion language)
  - Actual loop invariants typically simple
  - Alternative to existing summarization techniques?
- (watch out for bugs in the slides/tasks)

Big thanks to: Vladimir Klebanov, Marieke Huisman, Rosemary Monahan, Peter Müller, Mattias Ulbrich, and all VerifyThis organizers.