

Lab: Abstract Interpretation

(Week 8)

Yulei Sui, Mohamad Barbar, Hanyuan Li, Angela He

School of Computer Science and Engineering

University of New South Wales, Australia

Lab-2 Marks and Lab-3 Code Template

- Lab-2 marks are out and let us go through Quiz-2 and Exercise-2!
- Remember to `git pull` or `docker pull` to get the code template for **Lab-Exercise-3**

Quiz-3 + Lab-Exercise-3

- Quiz-3 (5 points) (due date: **23:59, Wednesday, Week 10**)
 - Abstract domain and soundness
 - Handling loops with widening and narrowing
- Lab-Exercise-3 (5 points) (due date: **23:59, Wednesday, Week 10**)
 - **Goal:** Coding exercise to manually update abstract trace based on abstract execution rules and verify the assertions embedded in the code.
 - **Specification:** <https://github.com/SVF-tools/Software-Security-Analysis/wiki/Lab-Exercise-3>

A Loop-Free Example

```
1 struct A{int f0;};  
2 void main() {  
3     struct A *p;  
4     int *q;  
5     int x;  
6     p = malloc;  
7     q = &(p→f0);  
8     *q = 10;  
9     x = *q;  
10  
    svf_assert(x == 10);  
11 }
```

```
1 NodeID p = getNodeID("p");  
2 NodeID q = getNodeID("q");  
3 NodeID x = getNodeID("x");  
4 // ...
```

-----Var and Value-----

AEState:printAbstractState()

Source code

Translation for Abstract execution

Abstract trace

A Loop-Free Example

```
1 struct A{int f0;};  
2 void main() {  
3   struct A *p;  
4   int *q;  
5   int x;  
6   p = malloc;  
7   q = &(p→f0);  
8   *q = 10;  
9   x = *q;  
10  
   svf_assert(x == 10);  
11 }
```

```
1 NodeID p = getNodeID("p");  
2 NodeID q = getNodeID("q");  
3 NodeID x = getNodeID("x");  
4 NodeID malloc = getNodeID("malloc");  
5 as[p] = AddressValue(getMemObjAddress("malloc"));  
6 // ...
```

```
-----Var and Value-----  
Var1 (p)           Value: 0x7f000004  
-----
```

0x7f000004 (or 2130706436 in decimal)

represents the virtual memory
address of this object

Each SVF object starts with 0x7f + its ID.

Source code

Translation for Abstract execution

Abstract trace

A Loop-Free Example

```
1 struct A{int f0;};  
2 void main() {  
3   struct A *p;  
4   int *q;  
5   int x;  
6   p = malloc;  
7   q = &(p->f0);  
8   *q = 10;  
9   x = *q;  
10  
   svf_assert(x == 10);  
11 }
```

```
1 NodeID p = getNodeID("p");  
2 NodeID q = getNodeID("q");  
3 NodeID x = getNodeID("x");  
4 NodeID malloc = getNodeID("malloc");  
5 as[p] = AddressValue(getMemObjAddress("malloc"));  
6 as[q] = AddressValue(getGepObjAddress("p", 0));  
7 // ...
```

-----Var and Value-----	
Var2 (q)	Value: 0x7f000001
Var1 (p)	Value: 0x7f000004

getGepObjAddress returns the field address of the aggregate object p
The virtual address also in the form of $0x7f.. + \text{VarID}$

Source code

Translation for Abstract execution

Abstract trace

A Loop-Free Example

```
1 struct A{int f0;};  
2 void main() {  
3   struct A *p;  
4   int *q;  
5   int x;  
6   p = malloc;  
7   q = &(p→f0);  
8   *q = 10;  
9   x = *q;  
10  
   svf_assert(x == 10);  
11 }
```

```
1 // ...  
2 NodeID malloc = getNodeID("malloc");  
3 as[p] = AddressValue(getMemObjAddress("malloc"));  
4 as[q] = AddressValue(getGepObjAddress("p", 0));  
5 as.storeValue(q, IntervalValue(10, 10));  
6 as[x] = as.loadValue(q);
```

-----Var and Value-----	
Var3 (x)	Value: [10, 10]
Var2 (q)	Value: 0x7f000001
Var1 (p)	Value: 0x7f000004
Var5 (0x7f000001)	Value: [10, 10]

store value of 5 to address 0x7f000005

load the value from 0x7f000005 to x

Source code

Translation for Abstract execution

Abstract trace

A Loop-Free Example

```
1 struct A{int f0;};  
2 void main() {  
3   struct A *p;  
4   int *q;  
5   int x;  
6   p = malloc;  
7   q = &(p→f0);  
8   *q = 10;  
9   x = *q;  
10  
   svf_assert(x == 10);  
11 }
```

```
1 NodeID p = getNodeID("p", 1);  
2 NodeID q = getNodeID("q");  
3 NodeID x = getNodeID("x");  
4 NodeID malloc = getNodeID("malloc");  
5 as[p] = AddressValue(getMemObjAddress("malloc"));  
6 as[q] = AddressValue(getGepObjAddress("p", 0));  
7 as.storeValue(q, IntervalValue(10, 10));  
8 as[x] = as.loadValue(q);
```

svf_assert checking is done in test.cpp.

```
-----Var and Value-----  
Var3 (x)           Value: [10, 10]  
Var2 (q)           Value: 0x7f000001  
Var1 (p)           Value: 0x7f000004  
Var5 (0x7f000001)  Value: [10, 10]  
-----
```

assertion checking

Source code

Translation for Abstract execution

Abstract trace

A Branch Example

```
1 int main(int argv) {  
  // 5 ≤ argv ≤ 15  
  int x = 10;  
2  if(argv > 10)  
3    x ++;  
4  else  
5    x += 2;  
6  
  svf_assert(x <= 12);  
7 }
```

Source code

```
1 NodeID argv = getNodeID("argv");  
2 as[argv] = IntervalValue(5, 15);  
3 ...
```

Translation for Abstract execution

```
-----Var and Value-----  
Var1 (argv)      Value: [5, 15]  
-----
```

assume $5 \leq \text{argv} \leq 15$

Abstract trace

A Branch Example

```
1 int main(int argv) {  
2   int x = 10;  
3   if(argv > 10)  
4     x++;  
5   else  
6     x += 2;  
7  
   svf_assert(x <= 12);  
8 }
```

```
1 NodeID argv = getNodeID("argv");  
2 as[argv] = IntervalValue(5, 15);  
3 NodeID x = getNodeID("x");  
4 as[x] = IntervalValue(10, 10);  
5 ...
```

```
-----Var and Value-----  
Var1 (argv)      Value: [5, 15]  
Var2 (x)         Value: [10, 10]  
-----
```

Source code

Translation for Abstract execution

Abstract trace

A Branch Example

```
1 int main(int argv) {  
2   int x = 10 ;  
3   if(argv > 10)  
4     x ++;  
5   else  
6     x += 2;  
7  
   svf_assert(x <= 12);  
8 }
```

Source code

```
1 // ...  
2 AESTate as_after_if;  
3 AbstractValue cmp_true = as[argv].getInterval() >  
4   IntervalValue(10, 10);  
5  
6 // feasibility checking  
7 cmp_true.meet_with(IntervalValue(1, 1));  
8 if (!cmp_true.getInterval().isBottom()) {  
9   AESTate as_true = as;  
10  as_true[x] = as_true[x].getInterval() +  
11    IntervalValue(1, 1);  
12  //Join the states at the control-flow joint point  
13  as_after_if.joinWith(as_true);  
14 }  
15 // ...
```

Translation for Abstract execution

as:

-----Var and Value-----	
Var1 (argv)	Value: [5, 15]
Var2 (x)	Value: [10, 10]

as_true:

-----Var and Value-----	
Var1 (argv)	Value: [5, 15]
Var2 (x)	Value: [11, 11]

Abstract trace

A Branch Example

```
1 int main(int argv) {  
2   int x = 10;  
3   if(argv > 10)  
4     x ++;  
5   else  
6     x += 2;  
7  
   svf_assert(x <= 12);  
8 }
```

```
1 // ...  
2 AESTate as_after_if;  
3 // ...  
4 AbstractValue cmp_false = as[argv].getInterval() >  
5   IntervalValue(10, 10);  
6  
7 cmp_false.meet_with(IntervalValue(0, 0));  
8 if (!cmp_false.getInterval().isBottom()){  
9   AESTate as_false = as;  
10  as_false[x] = as_false[x].getInterval() +  
11    IntervalValue(2, 2);  
12  as_after_if.joinWith(as_false);  
13 }  
14 ...
```

as:

```
-----Var and Value-----  
Var1 (argv)      Value: [5, 15]  
Var2 (x)         Value: [10, 10]  
-----
```

as_true:

```
-----Var and Value-----  
Var1 (argv)      Value: [5, 15]  
Var2 (x)         Value: [11, 11]  
-----
```

as_false:

```
-----Var and Value-----  
Var1 (argv)      Value: [5, 15]  
Var2 (x)         Value: [12, 12]  
-----
```

Source code

Translation for Abstract execution

Abstract trace

A Branch Example

```
1 int main(int argv) {  
2   int x = 10;  
3   if(argv > 10)  
4     x ++;  
5   else  
6     x += 2;  
7  
   svf_assert(x <= 12);  
8 }
```

```
1 ...  
2 AState as_after_if;  
3 AbstractValue cmp_true = as[argv].getInterval() >  
4   IntervalValue(10, 10);  
5 // feasibility checking  
6 cmp_true.meet_with(IntervalValue(1, 1));  
7 if (!cmp_true.getInterval().isBottom()) {  
8   AState as_true = as;  
9   as_true[x] = as_true[x].getInterval() +  
10     IntervalValue(1, 1);  
11   //Join the states at the control-flow joint point  
12   as_after_if.joinWith(as_true);  
13 }  
14  
15 AbstractValue cmp_false = as[argv].getInterval() >  
16   IntervalValue(10, 10);  
17 cmp_false.meet_with(IntervalValue(0, 0));  
18 if (!cmp_false.getInterval().isBottom()){  
19   AState as_false = as;  
20   as_false[x] = as_false[x].getInterval() +  
21     IntervalValue(2, 2);  
22   as_after_if.joinWith(as_false);  
23 }  
24 as = as_after_if;
```

svf_assert checking is done in test.cpp.

as_after_if, as:

-----Var and Value-----	
Var1 (argv)	Value: [5, 15]
Var2 (x)	Value: [11, 12]

as_true:

-----Var and Value-----	
Var1 (argv)	Value: [5, 15]
Var2 (x)	Value: [11, 11]

as_false:

-----Var and Value-----	
Var1 (argv)	Value: [5, 15]
Var2 (x)	Value: [12, 12]

Source code

Translation for Abstract execution

Abstract trace

A Loop Example

Before entering loop

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6   svf_assert(a == 10);  
7   return 0;  
8 }
```

```
1 AESTate entry_as;  
2 AESTate cur_head_as;  
3 AESTate body_as;  
4 AESTate exit_as;  
5 u32_t widen_delay = 3;  
6  
7 // Compose 'entry_as' (a = 0)  
8 NodeID a = getNodeID("a");  
9 entry_as[a] = IntervalValue(0, 0);  
10 bool increasing = true;  
11 for (int cur_iter = 0;; ++cur_iter) {  
12   ...  
13 }  
14 ...
```

entry_as:

-----Var and Value-----	
Var1 (a)	Value: [0, 0]

The initialization of a.

Source code

Translation for Abstract execution

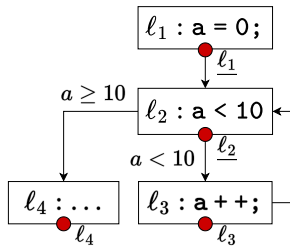
Abstract trace

Implementation available here:

[https://github.com/SVF-tools/Software-Security-Analysis/wiki/Lab-Exercise-3#](https://github.com/SVF-tools/Software-Security-Analysis/wiki/Lab-Exercise-3#4-widening-and-narrowing-implementation-for-the-below-loop-example-in-lecture-slides)

[4-widening-and-narrowing-implementation-for-the-below-loop-example-in-lecture-slides](https://github.com/SVF-tools/Software-Security-Analysis/wiki/Lab-Exercise-3#4-widening-and-narrowing-implementation-for-the-below-loop-example-in-lecture-slides)

Control flow graph



Control Flow Graph

```
1 AESTate entry_as;      // l_1
2 AESTate cur_head_as;   // l_2
3 AESTate body_as;       // l_3
4 AESTate exit_as;       // l_4
5
6 u32_t widen_delay = 3;
7
8 // Compose 'entry_as' (a = 0)
9 NodeID a = getNodeID("a");
10 entry_as[a] = IntervalValue(0, 0);
11
12 bool increasing = true;
13
14 for (int cur_iter = 0;; ++cur_iter) {
15     ...
16 }
17 ...
```

A Loop Example

Widening delay stage

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6  
7   svf_assert(a == 10);  
8 }
```

Source code

```
1 ...  
2 for (int cur_iter = 0;; ++cur_iter) {  
3   if (cur_iter >= widen_delay) {  
4     // Handle widening and narrowing after widen_delay  
5     ...  
6   }  
7   else {  
8     // Handle widen_delay, update cycle head's state  
9     // via joining entry_as and body_as  
10    cur_head_as = entry_as;  
11    cur_head_as.joinWith(body_as);  
12  }  
13  // Handle loop body by propagating head's state  
14  // meet with loop condition and enter loop body;  
15  body_as = cur_head_as;  
16  body_as[a].meet_with(Interval(minus_infinity(), 9));  
17  body_as[a] = body_as[a].getInterval() + Interval(1, 1);  
18 }  
19 // Handle loop exit  
20 ...
```

Translation for Abstract execution

cur_head_as after Line 10:

```
-----Var and Value-----  
Var1 (a)                      Value: [0, 0]  
-----
```

body_as after Line 16:

```
-----Var and Value-----  
Var1 (a)                      Value: [1, 1]  
-----
```

cur_iter = 0

Abstract trace

A Loop Example

Widening delay stage

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6  
7   svf_assert(a == 10);  
8 }
```

Source code

```
1 // ...  
2 for (int cur_iter = 0;; ++cur_iter) {  
3   if (cur_iter >= widen_delay) {  
4     // Handle widening and narrowing after widen_delay  
5     // ...  
6   }  
7   else {  
8     // Handle widen_delay, update cycle head's state  
9     // via joining entry_as and body_as  
10    cur_head_as = entry_as;  
11    cur_head_as.joinWith(body_as);  
12  }  
13  // Handle loop body by propagating head's state  
14  // meet with loop condition and enter loop body;  
15  body_as = cur_head_as;  
16  body_as[a].meet_with(Interval(minus_infinity(), 9));  
17  body_as[a] = body_as[a].getInterval() + Interval(1, 1);  
18 }  
19 // Handle loop exit  
20 // ...
```

Translation for Abstract execution

cur_head_as after Line 10:

```
-----Var and Value-----  
Var1 (a)      Value: [0, 1]  
-----
```

body_as after Line 16:

```
-----Var and Value-----  
Var1 (a)      Value: [1, 2]  
-----
```

cur_iter = 1

Abstract trace

A Loop Example

Widening delay stage

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6  
   svf_assert(a == 10);  
7   return 0;  
8 }
```

Source code

```
1 // ...  
2 for (int cur_iter = 0;; ++cur_iter) {  
3   if (cur_iter >= widen_delay) {  
4     // Handle widening and narrowing after widen_delay  
5     // ...  
6   } else {  
7     // Handle widen_delay, update cycle head's state  
8     // via joining entry_as and body_as  
9     cur_head_as = entry_as;  
10    cur_head_as.joinWith(body_as);  
11  }  
12  // Handle loop body by propagating head's state  
13  // meet with loop condition and enter loop body;  
14  body_as = cur_head_as;  
15  body_as[a].meet_with(Interval(minus_infinity(), 9));  
16  body_as[a] = body_as[a].getInterval() + Interval(1, 1);  
17 }  
18 // Handle loop exit  
19 // ...
```

Translation for Abstract execution

cur_head_as after Line 10:

```
-----Var and Value-----  
Var1 (a)      Value: [0, 2]  
-----
```

body_as after Line 16:

```
-----Var and Value-----  
Var1 (a)      Value: [1, 3]  
-----
```

cur_iter = 2

Abstract trace

A Loop Example

Widening Stage

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6  
   svf_assert(a == 10);  
7   return 0;  
8 }
```

```
1 ...  
2 for (int cur_iter = 0;; ++cur_iter) {  
3   if (cur_iter >= widen_delay) {  
4     // Handle widening and narrowing after widen_delay  
5     AESTate prev_head_as = cur_head_as;  
6     // Update head's state by joining with 'entry_as' and 'body_as'  
7     cur_head_as = entry_as;  
8     cur_head_as.joinWith(body_as);  
9     if (increasing) { // Widening phase  
10      AESTate after_widen = prev_head_as.widening(cur_head_as);  
11      cur_head_as = after_widen;  
12      if (cur_head_as == prev_head_as) {  
13        increasing = false;  
14        continue;  
15      }  
16    } else { // Narrow phase after widening  
17      AESTate after_narrow = prev_head_as.narrowing(cur_head_as);  
18      cur_head_as = after_narrow;  
19      if (cur_head_as == prev_head_as) //fix-point reached  
20        break;  
21    }  
22  } else { // Handle widen delay  
23    ...  
24  }  
25  // Handle loop body  
26  ...  
27 }  
28 // Handle loop exit  
29 ...
```

prev_head_as after Line 5:

-----Var and Value-----	
Var1 (a)	Value: [0, 2]

cur_head_as after Line 11:

-----Var and Value-----	
Var1 (a)	Value: [0, +∞]

body_as after Line 26 (handle loop body):

-----Var and Value-----	
Var1 (a)	Value: [1, 10]

Widening stage where cur_iter=3.

Source code

Translation for Abstract execution

Abstract trace

19

A Loop Example

Widening Stage

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6  
   svf_assert(a == 10);  
7  
   return 0;  
8 }
```

```
1 ...  
2 for (int cur_iter = 0;; ++cur_iter) {  
3   if (cur_iter >= widen_delay) {  
4     // Handle widening and narrowing after widen_delay  
5     AESTate prev_head_as = cur_head_as;  
6     // Update head's state by joining with 'entry_as' and 'body_as'  
7     cur_head_as = entry_as;  
8     cur_head_as.joinWith(body_as);  
9     if (increasing) { // Widening phase  
10      AESTate after_widen = prev_head_as.widening(cur_head_as);  
11      cur_head_as = after_widen;  
12      if (cur_head_as == prev_head_as) {  
13        increasing = false;  
14        continue;  
15      }  
16    } else { // Narrow phase after widening  
17      AESTate after_narrow = prev_head_as.narrowing(cur_head_as);  
18      cur_head_as = after_narrow;  
19      if (cur_head_as == prev_head_as) //fix-point reached  
20        break;  
21    }  
22  } else { // Handle widen delay  
23    ...  
24  }  
25  // Handle loop body  
26  ...  
27 }  
28 // Handle loop exit  
29 ...
```

prev_head_as after Line 5:

-----Var and Value-----	
Var1 (a)	Value: [0, +∞]

cur_head_as after Line 15:

-----Var and Value-----	
Var1 (a)	Value: [0, +∞]

Widening stage where cur_iter=4.

Source code

Translation for Abstract execution

Abstract trace

A Loop Example

Narrowing Stage

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6  
   svf_assert(a == 10);  
7   return 0;  
8 }
```

```
1 ...  
2 for (int cur_iter = 0;; ++cur_iter) {  
3   if (cur_iter >= widen_delay) {  
4     // Handle widening and narrowing after widen_delay  
5     AESTate prev_head_as = cur_head_as;  
6     // Update head's state by joining with 'entry_as' and 'body_as'  
7     cur_head_as = entry_as;  
8     cur_head_as.joinWith(body_as);  
9     if (increasing) { // Widening phase  
10      AESTate after_widen = prev_head_as.widening(cur_head_as);  
11      cur_head_as = after_widen;  
12      if (cur_head_as == prev_head_as) {  
13        increasing = false;  
14        continue;  
15      }  
16    } else { // Narrow phase after widening  
17      AESTate after_narrow = prev_head_as.narrowing(cur_head_as);  
18      cur_head_as = after_narrow;  
19      if (cur_head_as == prev_head_as) //fix-point reached  
20        break;  
21    }  
22  } else { // Handle widen delay  
23    ...  
24  }  
25  // Handle loop body  
26  ...  
27 }  
28 // Handle loop exit  
29 ...
```

prev_head_as after Line 5:

-----Var and Value-----	
Var1 (a)	Value: [0, +∞]

cur_head_as after Line 18:

-----Var and Value-----	
Var1 (a)	Value: [0, 10]

body_as after Line 26 (handle loop body):

-----Var and Value-----	
Var1 (a)	Value: [1, 10]

Narrowing stage where cur_iter=5.

Source code

Translation for Abstract execution

Abstract trace

A Loop Example

Narrowing Stage

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6  
   svf_assert(a == 10);  
7  
   return 0;  
8 }
```

```
1 ...  
2 for (int cur_iter = 0;; ++cur_iter) {  
3   if (cur_iter >= widen_delay) {  
4     // Handle widening and narrowing after widen_delay  
5     AESTate prev_head_as = cur_head_as;  
6     // Update head's state by joining with 'entry_as' and 'body_as'  
7     cur_head_as = entry_as;  
8     cur_head_as.joinWith(body_as);  
9     if (increasing) { // Widening phase  
10      AESTate after_widen = prev_head_as.widening(cur_head_as);  
11      cur_head_as = after_widen;  
12      if (cur_head_as == prev_head_as) {  
13        increasing = false;  
14        continue;  
15      }  
16    } else { // Narrow phase after widening  
17      AESTate after_narrow = prev_head_as.narrowing(cur_head_as);  
18      cur_head_as = after_narrow;  
19      if (cur_head_as == prev_head_as) //fix-point reached  
20        break;  
21    }  
22  } else { // Handle widen delay  
23    ...  
24  }  
25  // Handle loop body  
26  ...  
27 }  
28 // Handle loop exit  
29 ...
```

prev_head_as after Line 5:

-----Var and Value-----	
Var1 (a)	Value: [0, 10]

cur_head_as after Line 20:

-----Var and Value-----	
Var1 (a)	Value: [0, 10]

Narrowing stage where cur_iter=6.

Source code

Translation for Abstract execution

Abstract trace

A Loop Example

Handle Loop Exit

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6   svf_assert(a == 10);  
7   return 0;  
8 }
```

Source code

```
1 ...  
2 for (int cur_iter = 0;; ++cur_iter) {  
3   ...  
4 }  
5 // Propagate head_as to loop exit  
6 exit_as = cur_head_as;  
7 // Process loop exit condition (a>=10)  
8 exit_as[x].meet_with(IntervalValue(10, plus_infinity()));  
9  
10 return exit_as;
```

Translation for Abstract execution

exit_as after Line 6:

-----Var and Value-----	
Var1 (a)	Value: [0, 10]

exit_as after Line 8:

-----Var and Value-----	
Var1 (a)	Value: [10, 10]

Exiting loop.

Abstract trace

A Loop Example

Handle Loop Exit

```
1 int main() {  
2   int a = 0;  
3   while(a < 10) {  
4     a ++;  
5   }  
6  
   svf_assert(a == 10);  
7  
8   return 0;  
}
```

Source code

```
1 ...  
2 for (int cur_iter = 0;; ++cur_iter) {  
3   ...  
4 }  
5 // Propagate head_as to loop exit  
6 exit_as = cur_head_as;  
7 // Process loop exit condition (a>=10)  
8 exit_as[x].meet_with(IntervalValue(10, plus_infinity()));  
9  
10 return exit_as;  
11 }
```

svf_assert checking is done in test.cpp.

Translation for Abstract execution

exit_as at Line 10:

-----Var and Value-----	
Var1 (a)	Value: [10, 10]

After analyzing loop.

Abstract trace

Software Verification Competition

- Next week, we will introduce the **Software Verification Competition**
- Competition website: <https://sv-comp.sosy-lab.org/>