# Lab: Abstract Interpretation

## (Week 8)

Yulei Sui

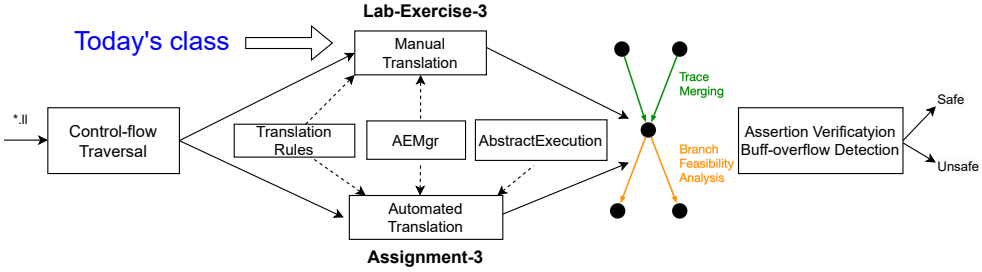
School of Computer Science and Engineering

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**

# Today's class



# Quiz-3 + Lab-Exercise-3

- Quiz-3 (5 points) (due date: **23:59, Tuesday, Week 10**)
  - Abstract domain and soundness
  - Handling loops with widening and narrowing
- Lab-Exercise-3 (5 points) (due date: **23:59, Tuesday, 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`

# Lab-3 Exercise: Manual Translation to Compute Abstract States

- Let us look at how to write abstract execution code to analyze examples of a loop-free and a loop C-like code by manually collecting abstract states at each program statement to form the abstract trace
- You will need to finish all the coding tests in **AEMgr.cpp** under **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", 1);
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", 1);
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", 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 ...
```

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

`getGepObjAddress` returns the field address of the aggregate object *p*

The virual address also in the form of `0x7f.. + 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 }
```

Source code

```
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);
9 ...
```

Translation for Abstract execution

```
----------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 ox7f000005

load the value from ox7f000005 to x

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 }
```

Source code

```
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`.

Translation for Abstract execution

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

assertion checking

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 $5 \leq$ 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 ...
```

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]
-----------------------------------
```

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

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]
---------------------------------
```

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  AEState 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      AEState 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     AEState 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:

```
----------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  AEState 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      AEState 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     AEState 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#4-widening-and-narrowing-implementation-for-the-below-loop-example-in-lecture-slides

# 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  }
```

```
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 ...
```

cur_head_as after Line 11:

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

body_as after Line 22:

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

cur_iter = 0.

Source code

Translation for Abstract execution

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  }
```

```
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 ...
```

cur_head_as after Line 11:

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

body_as after Line 22:

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

cur_iter = 1..

Source code

Translation for Abstract execution

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  }
```

```
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 ...
```

cur_head_as after Line 11:

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

body_as after Line 22:

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

cur_iter = 2..

Source code

Translation for Abstract execution

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

# 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 11:

```
----------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 11:

```
----------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 11:

```
----------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 i = 0; ; ++i) {
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 7:

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

exit_as after Line 13:

```
----------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    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;
11 }
```

`svf_assert` checking is done in `test.cpp`.

Translation for Abstract execution

exit_as at Line 15:

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

After analyzing loop.

Abstract trace