Lab: Abstract Interpretation

(Week 8)

Yulei Sui

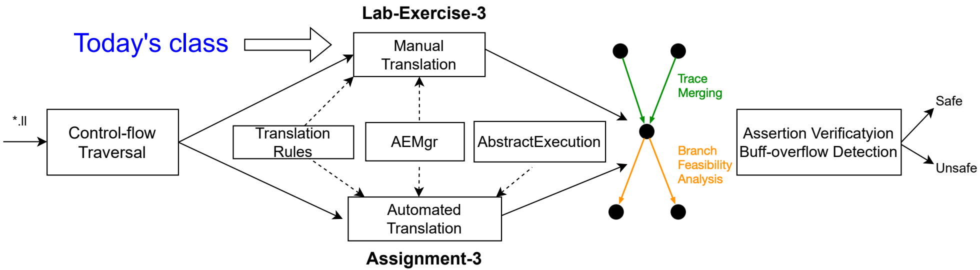
School of Computer Science and Engineering

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 + Assignment-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`

Quiz-3 + Lab-Exercise-3 + Assignment-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`
- Assignment-3 (25 points) (due date: **23:59, Wednesday, Week 11**)
  - **Goal:** Perform automated abstract trace update on ICFG for assertion checking and buffer overflow detection
  - **Specification:** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/Assignment-3`
  - **SVF AE APIs:** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/AE-APIs`

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

```
struct A{int f0;};
void main() {
  struct A * p ;
  int * q ;
  int x ;
  p = malloc;
  q = &(p→f0);
  *q = 10;
  x = *q;

 svf_assert(x == 10);

}
```

```
NodeID p = getNodeID("p", 1);
NodeID q = getNodeID("q");
NodeID x = getNodeID("x");
...
```

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


-----------------------------------
```

AEState:printAbstractState()

Source code

Translation for Abstract execution

Abstract trace

A Loop-Free Example

```
struct A{int f0;};
void main() {
  struct A *p;
  int *q;
  int x;
  p = malloc;
  q = &(p→f0);
  *q = 10;
  x = *q;

 svf_assert(x == 10);

}
```

```
NodeID p = getNodeID("p", 1);
NodeID q = getNodeID("q");
NodeID x = getNodeID("x");
NodeID malloc = getNodeID("malloc");
as[p] = AddressValue(getMemObjAddress("malloc"));
...
```

```
----------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

```
struct A{int f0;};
void main() {
  struct A *p;
  int *q;
  int x;
  p = malloc;
  q = &(p→f0);
  *q = 10;
  x = *q;

 svf_assert(x == 10);

}
```

```
NodeID p = getNodeID("p", 1);
NodeID q = getNodeID("q");
NodeID x = getNodeID("x");
NodeID malloc = getNodeID("malloc");
as[p] = AddressValue(getMemObjAddress("malloc"));
as[q] = AddressValue(getGepObjAddress("p", 0));
...
```

```
----------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

```
struct A{int f0;};
void main() {
  struct A * p ;
  int * q ;
  int x ;
  p = malloc;
  q = &(p→f0);
  *q = 10;
  x = *q;

 svf_assert(x == 10);

}
```

```
NodeID p = getNodeID("p", 1);
NodeID q = getNodeID("q");
NodeID x = getNodeID("x");
NodeID malloc = getNodeID("malloc");
as[p] = AddressValue(getMemObjAddress("malloc"));
as[q] = AddressValue(getGepObjAddress("p", 0));
as.storeValue(q, IntervalValue(10, 10));
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 `ox7f000005`

load the value from `ox7f000005` to `x`

Source code Translation for Abstract execution Abstract trace

A Loop-Free Example

```
struct A{int f0;};
void main() {
  struct A *p;
  int *q;
  int x;
  p = malloc;
  q = &(p→f0);
  *q = 10;
  x = *q;

 svf_assert(x == 10);

}
```

```
NodeID p = getNodeID("p", 1);
NodeID q = getNodeID("q");
NodeID x = getNodeID("x");
NodeID malloc = getNodeID("malloc");
as[p] = AddressValue(getMemObjAddress("malloc"));
as[q] = AddressValue(getGepObjAddress("p", 0));
as.storeValue(q, IntervalValue(10, 10));
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

```
int main(int argv) {
 // 5 ≤ argv ≤ 15
 int x = 10 ;
  if(argv > 10)
    x++;
  else
    x += 2;

 svf_assert(x <= 12);

}
```

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

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

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

Source code

Translation for Abstract execution

Abstract trace

A Branch Example

```
int main(int argv) {
  int x = 10;
  if(argv > 10)
    x++;
  else
    x += 2;

 svf_assert(x <= 12);

}
```

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

as:

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

as_true:

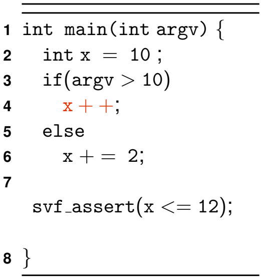
```
----------Var and Value-----------
Var1 (argv)         Value: [5, 15]
Var2 (x)            Value: [11, 11]
----------------------------------
```

Source code

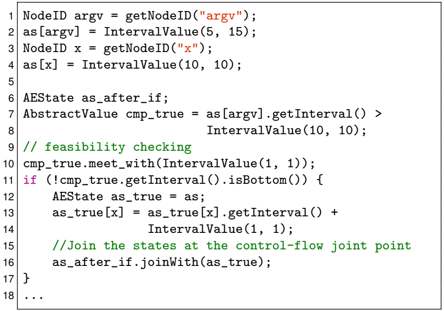
Translation for Abstract execution

Abstract trace

A Branch Example

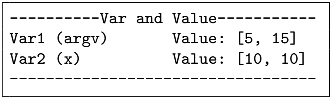


```
int main(int argv) {
  int x = 10;
  if(argv > 10)
    x++;
  else
    x += 2;

 svf_assert(x <= 12);

}
```



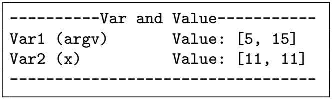
```
NodeID argv = getNodeID("argv");
as[argv] = IntervalValue(5, 15);
NodeID x = getNodeID("x");
as[x] = IntervalValue(10, 10);

AEState as_after_if;
AbstractValue cmp_true = as[argv].getInterval() >
                          IntervalValue(10, 10);
// feasibility checking
cmp_true.meet_with(IntervalValue(1, 1));
if (!cmp_true.getInterval().isBottom()) {
    AEState as_true = as;
    as_true[x] = as_true[x].getInterval() +
                 IntervalValue(1, 1);
    //Join the states at the control-flow joint point
    as_after_if.joinWith(as_true);
}
...
```

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

```
int main(int argv) {
  int x = 10;
  if(argv > 10)
    x++;
  else
    x += 2;

 svf_assert(x <= 12);

}
```

```
...
AEState as_after_if;
AbstractValue cmp_true = as[argv].getInterval() >
                              IntervalValue(10, 10);
// feasibility checking
cmp_true.meet_with(IntervalValue(1, 1));
if (!cmp_true.getInterval().isBottom()) {
    AEState as_true = as;
    as_true[x] = as_true[x].getInterval() +
                     IntervalValue(1, 1);
    //Join the states at the control-flow joint point
    as_after_if.joinWith(as_true);
}

AbstractValue cmp_false = as[argv].getInterval() >
                               IntervalValue(10, 10);
cmp_false.meet_with(IntervalValue(0, 0));
if (!cmp_false.getInterval().isBottom()){
    AEState as_false = as;
    as_false[x] = as_false[x].getInterval() +
                      IntervalValue(2, 2);
    as_after_if.joinWith(as_false);
}
...
```

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

```
int main(int argv) {
  int x = 10;
  if(argv > 10)
    x++;
  else
    x += 2;

svf_assert(x <= 12);

}
```

```
...
AEState as_after_if;
AbstractValue cmp_true = as[argv].getInterval() >
                              IntervalValue(10, 10);
// feasibility checking
cmp_true.meet_with(IntervalValue(1, 1));
if (!cmp_true.getInterval().isBottom()) {
    AEState as_true = as;
    as_true[x] = as_true[x].getInterval() +
                 IntervalValue(1, 1);
    //Join the states at the control-flow joint point
    as_after_if.joinWith(as_true);
}

AbstractValue cmp_false = as[argv].getInterval() >
                               IntervalValue(10, 10);
cmp_false.meet_with(IntervalValue(0, 0));
if (!cmp_false.getInterval().isBottom()){
    AEState as_false = as;
    as_false[x] = as_false[x].getInterval() +
                  IntervalValue(2, 2);
    as_after_if.joinWith(as_false);
}
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

```
int main() {
  int a = 0 ;
  while(a < 10) {
    a++;
  }

 svf_assert(a == 10);

  return 0;
}
```

```
AEState entry_as;
AEState cur_head_as;
AEState body_as;
AEState exit_as;
u32_t widen_delay = 3;

// Compose 'entry_as' (a = 0)
NodeID a = getNodeID("a");
entry_as[a] = IntervalValue(0, 0);
bool increasing = true;
for (int cur_iter = 0;; ++cur_iter) {
    ...
}
...
```

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

```
int main() {
  int a = 0 ;
  while(a < 10) {
    a++;
  }

 svf_assert(a == 10);

  return 0;
}
```

```
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

```
int main() {
  int a = 0 ;
  while(a < 10) {
    a++;
  }

 svf_assert(a == 10);

  return 0;
}
```

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          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 ...
```

Translation for Abstract execution

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.

Abstract trace

A Loop Example

Widening Stage

```
int main() {
  int a = 0 ;
  while(a < 10) {
    a++;
  }

 svf_assert(a == 10);

  return 0;
}
```

```
 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

```
int main() {
  int a = 0;
  while(a < 10) {
    a++;
  }

 svf_assert(a == 10);

  return 0;
}
```

```
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

```
int main() {
  int a = 0 ;
  while(a < 10) {
    a++;
  }

 svf_assert(a == 10);

  return 0;
}
```

```
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 }
```

```
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;
```

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.

Source code

Translation for Abstract execution

Abstract trace

A Loop Example

Handle Loop Exit

```
int main() {
  int a = 0 ;
  while(a < 10) {
    a++;
  }

 svf_assert(a == 10);

  return 0;
}
```

```
...
for (int cur_iter = 0;; ++cur_iter) {
    ...
}
// Propagate head_as to loop exit
exit_as = cur_head_as;
// Process loop exit condition (a>=10)
exit_as[x].meet_with(IntervalValue(10, plus_infinity()));

return exit_as;
}
```

svf_assert checking is done in test.cpp.

exit_as at Line 15:

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

After analyzing loop.

Source code

Translation for Abstract execution

Abstract trace