Lab: Abstract Interpretation

(Week 8)

Yulei Sui

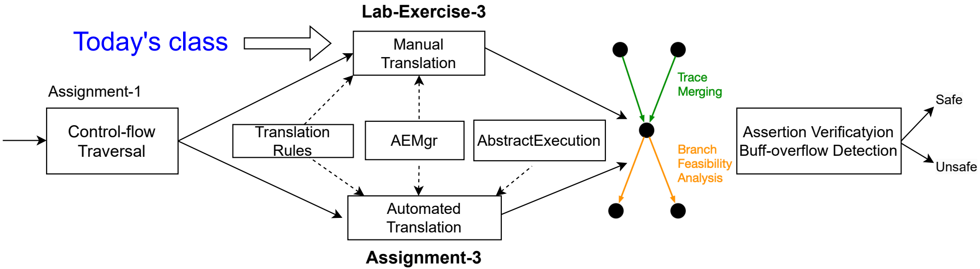
School of Computer Science and Engineering

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");
as[p] = AddressValue(getMemObjAddress("malloc"));
...
```

```
----------Var and Value-----------
Var4 (malloc)      Value: 0x7f000004
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");
as[p] = AddressValue(getMemObjAddress("malloc"));
as[q] = AddressValue(getGepObjAddress("p", 0));
...
```

```
----------Var and Value-----------
Var4 (malloc)       Value: 0x7f000004
Var1 (p)            Value: 0x7f000004
Var2 (q)            Value: 0x7f000005
----------------------------------
```

`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");
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-----------
Var4 (malloc)          Value: 0x7f000004
Var1 (p)               Value: 0x7f000004
Var2 (q)               Value: 0x7f000005
Var3 (x)               Value: [10, 10]
Var5 (0x7f000005)      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");
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-----------
Var4 (malloc)           Value: 0x7f000004
Var1 (p)                Value: 0x7f000004
Var2 (q)                Value: 0x7f000005
Var3 (x)                Value: [10, 10]
Var5 (0x7f000005)       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:

```
----------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");

AbstractState as_after_if;
AbstractValue cmp_true = as[argv] > IntervalValue(10, 10);
// feasibility checking
cmp_true.meet_with(IntervalValue(1, 1));
if (!cmp_true.getInterval().isBottom()) {
    AbstractState as_true = as;
    as_true[x] = as_true[x] + 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);

}
```

Source code

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

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

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

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

as_false:

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

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");

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

AbstractValue cmp_false = as[argv] > IntervalValue(10, 10);
// feasibility checking
cmp_false.meet_with(IntervalValue(0, 0));
if (!cmp_false.getInterval().isBottom()){
    AbstractState as_false = as;
    as_false[x] = as_false[x] + IntervalValue(2, 2);
    //Join the states at the control-flow joint point
    as_after_if.joinWith(as_false);
}
as = as_after_if;
```

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

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

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;
}
```

```
NodeID a = getNodeID("a");
as[a] = IntervalValue(0, 0);
bool increasing = true;
AbstractState entry_as = as;
AbstractState pre_as = as;
AbstractState post_as = as;
for (int i = 0; ; ++i) {
    ...
}
...
```

as, entry_as, pre_as and post_as:

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

The initialization of a.

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 i = 0; ; ++i) {
 3     AbstractState tmp_as;
 4     tmp_as.joinWith(post_as);
 5     tmp_as.joinWith(entry_as);
 6     as = tmp_as;
 7     if (i < 3) {
 8         pre_as = as;
 9     } else {
10         // widen and widen fixpoint
11         ...
12     }
13     as[a].meet_with(IntervalValue(
14         IntervalValue::minus_infinity(), 9));
15     as[a] = as[a] + IntervalValue(1, 1);
16     post_as = as;
17 }
18 ...
```

pre_as after Line 8:

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

as after Line 15:

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

Widening delay with `i=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 i = 0; ; ++i) {
 3     AbstractState tmp_as;
 4     tmp_as.joinWith(post_as);
 5     tmp_as.joinWith(entry_as);
 6     as = tmp_as;
 7     if (i < 3) {
 8         pre_as = as;
 9     } else {
10         // widen and widen fixpoint
11         ...
12     }
13     as[a].meet_with(IntervalValue(
14         IntervalValue::minus_infinity(), 9));
15     as[a] = as[a] + IntervalValue(1, 1);
16     post_as = as;
17 }
18 ...
```

pre_as after Line 8:

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

as after Line 15:

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

Widening delay with `i=1`.

Source code   Translation for Abstract execution   Abstract trace

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 i = 0; ; ++i) {
 3     AbstractState tmp_as;
 4     tmp_as.joinWith(post_as);
 5     tmp_as.joinWith(entry_as);
 6     as = tmp_as;
 7     if (i < 3) {
 8         pre_as = as;
 9     } else {
10         // widen and widen fixpoint
11         ...
12     }
13     as[a].meet_with(IntervalValue(
14         IntervalValue::minus_infinity(), 9));
15     as[a] = as[a] + IntervalValue(1, 1);
16     post_as = as;
17 }
18 ...
```

pre_as after Line 8:

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

as after Line 15:

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

Widening delay with `i`=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;
}
```

```
1  ...
2  for (int i = 0; ; ++i) {
3      ...
4      if (i < 3) {
5          pre_as = as;
6      } else {
7          // widen and widen fixpoint
8          if (increasing) {
9              as = pre_as.widening(as);
10             if (pre_as >= as) {
11                 pre_as = as;
12                 increasing = false;
13                 continue;
14             }
15             pre_as = as;
16         } else {
17             // narrow
18         }
19     }
20     as[a].meet_with(IntervalValue(
21         IntervalValue::minus_infinity(), 9));
22     as[a] = as[a] + IntervalValue(1, 1);
23     post_as = as;
24 }
25 ...
```

pre_as before Line 9:

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

as before Line 9:

```
----------Var and Value-----------
Var1 (a)              Value: [0, 3]
----------------------------------
```

as after Line 9:

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

Widening stage where i=3.

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;
}
```

```
 1 ...
 2 for (int i = 0; ; ++i) {
 3     ...
 4     if (i < 3) {
 5         pre_as = as;
 6     } else {
 7         // widen and widen fixpoint
 8         if (increasing) {
 9             as = pre_as.widening(as);
10             if (pre_as >= as) {
11                 pre_as = as;
12                 increasing = false;
13                 continue;
14             }
15             pre_as = as;
16         } else {
17             // narrow
18         }
19     }
20     as[a].meet_with(IntervalValue(
21         IntervalValue::minus_infinity(), 9));
22     as[a] = as[a] + IntervalValue(1, 1);
23     post_as = as;
24 }
25 ...
```

pre_as before Line 9:

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

as before Line 9:

```
----------Var and Value-----------
Var1 (a)                Value: [0, 9]
----------------------------------
```

as after Line 9:

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

Widening stage where i=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 i = 0; ; ++i) {
 3      ...
 4      if (i < 3) {
 5          pre_as = as;
 6      } else {
 7          // widen and widen fixpoint
 8          if (increasing) {
 9              ...
10          } else {
11              as = pre_as.narrowing(as);
12              if (as >= pre_as) {
13                  break;
14              }
15              pre_as = as;
16          }
17      }
18      as[a].meet_with(IntervalValue(
19          IntervalValue::minus_infinity(), 9));
20      as[a] = as[a] + IntervalValue(1, 1);
21      post_as = as;
22  }
23  ...
```

pre_as before Line 11:

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

as before Line 11:

```
----------Var and Value-----------
Var1 (a)              Value: [0, 9]
----------------------------------
```

as after Line 11:

```
----------Var and Value-----------
Var1 (a)              Value: [0, 9]
----------------------------------
```

Narrowing stage where i=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 i = 0; ; ++i) {
3      ...
4      if (i < 3) {
5          pre_as = as;
6      } else {
7          // widen and widen fixpoint
8          if (increasing) {
9              ...
10         } else {
11             as = pre_as.narrowing(as);
12             if (as >= pre_as) {
13                 break;
14             }
15             pre_as = as;
16         }
17     }
18     as[a].meet_with(IntervalValue(
19         IntervalValue::minus_infinity(), 9));
20     as[a] = as[a] + IntervalValue(1, 1);
21     post_as = as;
22 }
23 ...
```

pre_as before Line 11:

```
----------Var and Value-----------
Var1 (a)              Value: [0, 9]
---------------------------------
```

as before Line 11:

```
----------Var and Value-----------
Var1 (a)              Value: [0, 9]
---------------------------------
```

as after Line 11:

```
----------Var and Value-----------
Var1 (a)              Value: [0, 9]
---------------------------------
```

Narrowing stage where i=6.

Source code   Translation for Abstract execution   Abstract trace

A Loop Example

After exiting loop

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

 svf_assert(a == 10);

  return 0;
}
```

```
...
for (int i = 0; ; ++i) {
    ...
}
getExitState(as, a);
```

as:

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

After analyzing loop.

Source code

Translation for Abstract execution

Abstract trace