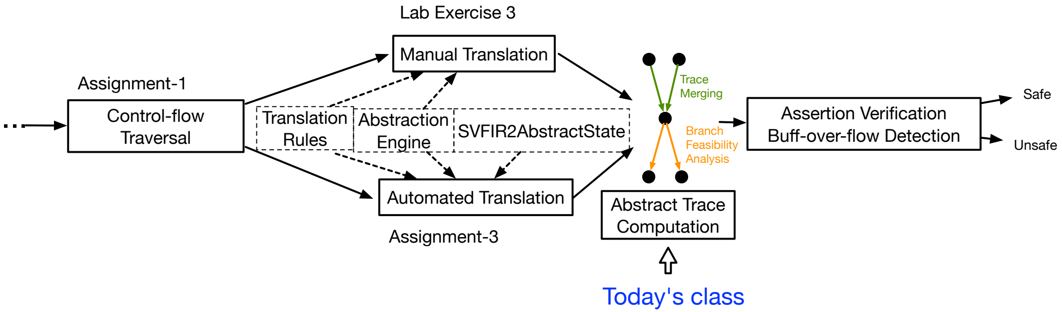
Lab: Abstract Interpretation

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

Today’s class



Quiz-3 + Lab-Exercise-3 + Assignment-3

- Quiz-3 (5 points)
  - Abstract domain and soundness
  - Handling loops with widening and narrowing
- Lab-Exercise-3 (5 points)
  - **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)
  - Abstract domain and soundness
  - Handling loops with widening and narrowing
- Lab-Exercise-3 (5 points)
  - **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)
  - **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 Coding Exercise: Abstract States and Abstract Traces

- 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;
  if(x == 10)
    x ++;
  assert(x == 11);
}
```

Source code

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

Abstract execution

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

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

```
----------Loc and Value-----------

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

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;
  if(x == 10)
    x++;
  assert(x == 11);
}
```

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

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

```
----------Loc and Value-----------

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

`0x7f000004` (or 2130706436 in decimal) represents the virtual memory address of this object

Each SVF object starts with `0x7f` + its ID.

Source code | 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;
  if(x == 10)
    x++;
  assert(x == 11);
}
```

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

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

```
----------Loc and Value-----------

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

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

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

Source code

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;
  if(x == 10)
    x ++;
  assert(x == 11);
}
```

```
NodeID p = getNodeID("p", 1);
NodeID q = getNodeID("q");
NodeID x = getNodeID("x");
es[p] = AddressValue(getMemObjAddress("malloc"));
es[q] = AddressValue(getGepObjAddress("p", 0));
for (auto addr : es[q].getAddrs()) {
    es.store(addr, IntervalValue(10, 10));
}
for (const auto &addr: es[q].getAddrs()) {
    es[x].join_with(es.load(addr));
}
...
```

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

```
----------Loc and Value-----------
0x7f000005         Value: [10, 10]
----------------------------------
```

store value of 5 to address `ox7f000005`

load the value from `ox7f000005` to `x`

Source code

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;
  if(x == 10)
    x ++;
  assert(x == 11);
}
```

Source code

```
NodeID p = getNodeID("p", 1);
NodeID q = getNodeID("q");
NodeID x = getNodeID("x");
es[p] = AddressValue(getMemObjAddress("malloc"));
es[q] = AddressValue(getGepObjAddress("p", 0));
for (auto addr : es[q].getAddrs()) {
    es.store(addr, IntervalValue(10, 10));
}
for (const auto &addr: es[q].getAddrs()) {
    es[x].join_with(es.load(addr));
}
AbstractState es_after_if;
AbstractValue cmp_true = es[x] == IntervalValue(10, 10);
cmp_true.meet_with(IntervalValue(1, 1));
if (!cmp_true.isBottom()) {
    es[x] = es[x] + IntervalValue(1, 1);
}
...
```

Abstract execution

```
----------Var and Value-----------
Var4 (malloc)      Value: 0x7f000004
Var1 (p)           Value: 0x7f000004
Var2 (q)           Value: 0x7f000005
Var3 (x)           Value: [11, 11]
---------------------------------
```

```
----------Loc and Value-----------
0x7f000005         Value: [10, 10]
---------------------------------
```

handle branch

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;
  if(x == 10)
    x ++;
  assert(x == 11);
}
```

Source code

```
NodeID p = getNodeID("p", 1);
NodeID q = getNodeID("q");
NodeID x = getNodeID("x");
es[p] = AddressValue(getMemObjAddress("malloc"));
es[q] = AddressValue(getGepObjAddress("p", 0));
for (auto addr : es[q].getAddrs()) {
    es.store(addr, IntervalValue(10, 10));
}
for (const auto &addr: es[q].getAddrs()) {
    es[x].join_with(es.load(addr));
}
AbstractState es_after_if;
AbstractValue cmp_true = es[x] == IntervalValue(10, 10);
cmp_true.meet_with(IntervalValue(1, 1));
if (!cmp_true.isBottom()) {
    es[x] = es[x] + IntervalValue(1, 1);
}
svf_assert(es[x] == IntervalValue(11, 11);
```

Abstract execution

```
----------Var and Value-----------
Var4 (malloc)      Value: 0x7f000004
Var1 (p)           Value: 0x7f000004
Var2 (q)           Value: 0x7f000005
Var3 (x)           Value: [11, 11]
----------------------------------
```

```
----------Loc and Value-----------
0x7f000005         Value: [10, 10]
----------------------------------
```

assertion checking

Abstract trace

A Loop Example

Before entering loop

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

```
NodeID a = getNodeID("a");
es[a] = IntervalValue(1, 1);
bool increasing = true;
AbstractState entry_es = es;
AbstractState pre_es = es;
AbstractState post_es = es;
for (int i = 0; ; ++i) {
    ...
}
...
```

es, entry_es, pre_es and post_es:

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

The initialization of a.

Source code | 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   assert(a == 10);
7   return 0;
8 }
```

Source code

```
1  ...
2  for (int i = 0; ; ++i) {
3      AbstractState tmp_es;
4      tmp_es.joinWith(post_es);
5      tmp_es.joinWith(entry_es);
6      es = tmp_es;
7      if (i < 3) {
8          pre_es = AbstractState(es);
9      } else {
10         // widen and widen fixpoint
11         ...
12     }
13     es[a].meet_with(IntervalValue(
14         IntervalValue::minus_infinity(), 9));
15     es[a] = es[a] + IntervalValue(1, 1);
16     post_es = es;
17 }
18 ...
```

Abstract execution

pre_es after Line 8:

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

es after Line 15:

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

Widening delay with `i=0`.

Abstract trace

A Loop Example

Widening delay stage

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

```
1  ...
2  for (int i = 0; ; ++i) {
3      AbstractState tmp_es;
4      tmp_es.joinWith(post_es);
5      tmp_es.joinWith(entry_es);
6      es = tmp_es;
7      if (i < 3) {
8          pre_es = AbstractState(es);
9      } else {
10         // widen and widen fixpoint
11         ...
12     }
13     es[a].meet_with(IntervalValue(
14         IntervalValue::minus_infinity(), 9));
15     es[a] = es[a] + IntervalValue(1, 1);
16     post_es = es;
17 }
18 ...
```

pre_es after Line 8:

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

es after Line 15:

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

Widening delay with `i=1`.

Source code | Abstract execution | Abstract trace

A Loop Example

Widening delay stage

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

Source code

```
1  ...
2  for (int i = 0; ; ++i) {
3      AbstractState tmp_es;
4      tmp_es.joinWith(post_es);
5      tmp_es.joinWith(entry_es);
6      es = tmp_es;
7      if (i < 3) {
8          pre_es = AbstractState(es);
9      } else {
10         // widen and widen fixpoint
11         ...
12     }
13     es[a].meet_with(IntervalValue(
14         IntervalValue::minus_infinity(), 9));
15     es[a] = es[a] + IntervalValue(1, 1);
16     post_es = es;
17 }
18 ...
```

Abstract execution

pre_es after Line 8:

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

es after Line 15:

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

Widening delay with `i`=2.

Abstract trace

A Loop Example

Widening stage

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

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

pre_es before Line 9:

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

es before Line 9:

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

es after Line 9:

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

Widening stage where `i`=3.

Source code Abstract execution Abstract trace

A Loop Example

Widening stage

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

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

pre_es before Line 9:

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

es before Line 9:

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

es after Line 9:

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

Widening stage where i=4.

Source code   Abstract execution   Abstract trace

A Loop Example

Narrowing stage

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

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

pre_es before Line 11:

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

es before Line 11:

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

es after Line 11:

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

Narrowing stage where i=5.

Source code

Abstract execution

Abstract trace

A Loop Example

Narrowing stage

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

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

pre_es before Line 11:

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

es before Line 11:

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

es after Line 11:

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

Narrowing stage where `i`=6.

Source code | Abstract execution | Abstract trace

A Loop Example

After exiting loop

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

```
...
for (int i = 0; ; ++i) {
    ...
}
getExitState(es, a);
svf_assert(es[a] == IntervalValue(10, 10));
```

es:

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

After analyzing loop.

Source code   Abstract execution   Abstract trace