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

Lab-Exercise-3

Manual Translation

AEMgr

AbstractExecution

Assignment-3

Control-flow Traversal

Translation Rules

Automated Translation

Trace Merging

Branch Feasibility Analysis

Safe

Unsafe

*.ll

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
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.
- **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
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

```c
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);
}
```

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

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

AEState:printAbstractState()
```

Source code  Translation for Abstract execution  Abstract trace
A Loop-Free Example

```c
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

Translation for Abstract execution

Abstract trace

---Var and Value---

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>0x7f000004</td>
</tr>
</tbody>
</table>

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

Each SVF object starts with 0x7f + its ID.
A Loop-Free Example

```c
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 | Translation for Abstract execution | Abstract trace

----------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.. + VarID
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 }
A Loop-Free Example

```c
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 }
```

---

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
A Branch Example

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

--- Var and Value ---

Var1 (argv) Value: [5, 15]

assume 5 ≤ argv ≤ 15

Source code  Translation for Abstract execution  Abstract trace
A Branch Example

```c
int main(int argv) {
    int x = 10;
    if(argv > 10)
        x++;
    else
        x += 2;
    svf_assert(x <= 12);
}
```

Source code  Translation for Abstract execution  Abstract trace

---

### Abstract Trace

**as:**

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv</td>
<td>[5, 15]</td>
</tr>
<tr>
<td>x</td>
<td>[10, 10]</td>
</tr>
</tbody>
</table>

**as_true:**

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv</td>
<td>[5, 15]</td>
</tr>
<tr>
<td>x</td>
<td>[11, 11]</td>
</tr>
</tbody>
</table>

---
A Branch Example

```c
int main(int argv) {
    int x = 10;
    if (argv > 10) {
        x++;
    } else {
        x = 2;
    }
    svf_assert(x <= 12);
}
```

**Source code**

**Translation for Abstract execution**

```
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);
}
```

**Abstract trace**

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv</td>
<td>[5, 15]</td>
</tr>
<tr>
<td>x</td>
<td>[10, 10]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv</td>
<td>[5, 15]</td>
</tr>
<tr>
<td>x</td>
<td>[11, 11]</td>
</tr>
</tbody>
</table>

A Branch Example

```c
int main(int argv) {
    int x = 10;
    if (argv > 10) {
        x++;
    } else {
        x += 2;
    }
    svf_assert(x <= 12);
}
```

Source code Translation for Abstract execution Abstract trace

```c
..._plural
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);
}
..._plural
```

as:

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

as_true:

--- Var and Value ---

as_false:

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

Software Security Analysis 2024
https://github.com/SVF-tools/Software-Security-Analysis
A Branch Example

```c
int main(int argv) {
    int x = 10;
    if (argv > 10)
        x++;
    else
        x += 2;
    svf_assert(x <= 12);
}
```

svf_assert checking is done in test.cpp.

---

Source code | Translation for Abstract execution | Abstract trace
---|---|---

as_after_if, as:

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv</td>
<td>[5, 15]</td>
</tr>
<tr>
<td>x</td>
<td>[11, 12]</td>
</tr>
</tbody>
</table>

as_true:

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv</td>
<td>[5, 15]</td>
</tr>
<tr>
<td>x</td>
<td>[11, 11]</td>
</tr>
</tbody>
</table>

as_false:

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>argv</td>
<td>[5, 15]</td>
</tr>
<tr>
<td>x</td>
<td>[12, 12]</td>
</tr>
</tbody>
</table>
A Loop Example

Before entering loop

```c
int main() {
    int a = 0;
    while(a < 10) {
        a += 1;
    }
    svf_assert(a == 10);
    return 0;
}
```

---

Source code | Translation for Abstract execution | Abstract trace
---|---|---

Implementation available here:

A Loop Example

Widening delay stage

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

cur_head_as after Line 11:

<table>
<thead>
<tr>
<th>Var and Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var1 (a) Value: [0, 0]</td>
</tr>
</tbody>
</table>

body_as after Line 22:

<table>
<thead>
<tr>
<th>Var and Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Var1 (a) Value: [1, 1]</td>
</tr>
</tbody>
</table>
A Loop Example

Widening delay stage

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

Source code  Translation for Abstract execution  Abstract trace

```c
for (int cur_iter = 0;; ++cur_iter) {
    if (cur_iter >= widen_delay) {
        // Handle widening and narrowing after widen_delay
        ...
    } else {
        // Handle widen_delay, update cycle head's state
        cur_head_as = entry_as;
        cur_head_as.joinWith(body_as);
    }
    // Handle loop body by propagating head's state
    // meet with loop condition and enter loop body;
    body_as = cur_head_as;
    body_as[a].meet_with(Interval(minus_infinity(), 9));
    body_as[a] = body_as[a].getInterval() + Interval(1, 1);
}
```

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..
A Loop Example

Widening delay stage

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

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..
A Loop Example

Widening Stage

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

...for (int cur_iter = 0;; ++cur_iter) {
  if (cur_iter >= widen_delay) {
    // Handle widening and narrowing after widen_delay
    AEState prev_head_as = cur_head_as;
    // Update head's state by joining with 'entry_as' and 'body_as'
    cur_head_as = entry_as;
    cur_head_as.joinWith(body_as);
    if (increasing) { // Widening phase
      AEState after_widen = prev_head_as.widening(cur_head_as);
      cur_head_as = after_widen;
      if (cur_head_as == prev_head_as) {
        increasing = false;
        continue;
      }
    } else { // Narrow phase after widening
      AEState after_narrow = prev_head_as.narrowing(cur_head_as);
      cur_head_as = after_narrow;
      if (cur_head_as == prev_head_as) // fix-point reached
        break;
    }
  } else { // Handle widen delay
    ...
  }
}
// Handle loop body ...
// Handle loop exit ...

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 }

For (int cur_iter = 0;; ++cur_iter) {
    if (cur_iter >= widen_delay) {
        // Handle widening and narrowing after widen_delay
        AEState prev_head_as = cur_head_as;
        // Update head's state by joining with 'entry_as' and 'body_as'
        cur_head_as = entry_as;
        cur_head_as.joinWith(body_as);
        if (increasing) { // Widening phase
            AEState after_widen = prev_head_as.widening(cur_head_as);
            cur_head_as = after_widen;
            if (cur_head_as == prev_head_as) {
                increasing = false;
                continue;
            }
        } else { // Narrow phase after widening
            AEState after_narrow = prev_head_as.narrowing(cur_head_as);
            cur_head_as = after_narrow;
            if (cur_head_as == prev_head_as) // fix-point reached
                break;
        }
    } else { // Handle widen delay
        ... 
    }
}
// Handle loop body
...
// Handle loop exit
...

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

```c
int main()
{
int a = 0;
while (a < 10) {
    a += 1;
}
svf_assert(a == 10);
return 0;
}
```

prev_head_as after Line 5:

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>[0, +∞]</td>
</tr>
</tbody>
</table>

cur_head_as after Line 11:

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>[0, 10]</td>
</tr>
</tbody>
</table>

body_as after Line 26 (handle loop body):

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>[1, 10]</td>
</tr>
</tbody>
</table>

Narrowing stage where cur_iter=5.
A Loop Example

Narrowing Stage

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

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.
A Loop Example

Handle Loop Exit

---

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

---

exit\_as after Line 7:

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>[0, 10]</td>
</tr>
</tbody>
</table>

---

exit\_as after Line 13:

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>[10, 10]</td>
</tr>
</tbody>
</table>

Exiting loop.

---

Source code | Translation for Abstract execution | Abstract trace
A Loop Example

Handle Loop Exit

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

```c
... for (int cur_iter = 0;; ++cur_iter) {
    ...
} // 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     return exit_as;
10 }
```

svf_assert checking is done in test.cpp.

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