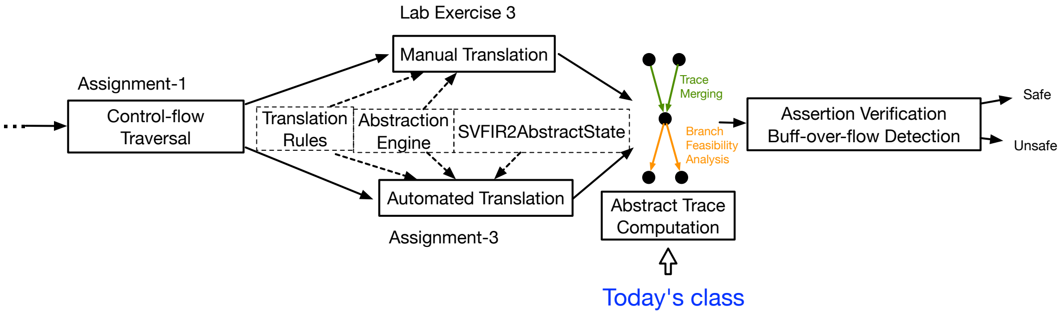
Lab: Abstract Interpretation

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);
}
```

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

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

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

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

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

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"));
...
```

```
----------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);
}
```

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));
...
```

Abstract execution

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

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[p].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[p].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);
}
```

```
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[p].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);
```

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

Source code

Abstract execution

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

```
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, 0]
----------------------------------
```

es after Line 15:

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

Widening delay with `i=0`.

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;
}
```

```
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;
}
```

```
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, 2]
----------------------------------
```

es after Line 15:

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

Widening delay with `i=2`.

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, 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

Abstract Execution Pseudo Code

Algorithm 1: Abstract execution guided by WTO (part 1)

Input: $G_{ICFG} : \langle V_c, E_c \rangle$ funcToWTO: WTO of each function, *preAbsTrace*: the abstract states before each control node, *postAbsTrace*: the abstract states after each control node

```
Function handleFunc(func) :
    WTO := funcToWTO[func];
    for co ∈ WTO do
        if co is node then
            handleICFGNode(co)
        else if co is cycle then
            handleCycle(co)

Function handleICFGNode(n):
    if hasInEdgesES (n) then
    else
        return
    getInEdgesES(n);
    for stmt ∈ n.statements do
        handleStatement(stmt)
    if n is call node then
        handleCallSite(n)
    else
        detectBug (n)
```

Algorithm 2: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h])
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h])
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)
```

Abstract Execution Pseudo Code

Algorithm 3: Abstract execution guided by WTO (part 3)

```
Function handleStatement(ℓ):
    tmpAS := preAbsTrace[ℓ]
    if ℓ is ConsStmt or AddrStmt then
        initSVFVar(ℓ.rhs)
        tmpAS[ℓ.lhs] := tmpAS[ℓ.rhs]
    else if ℓ is CopyStmt then
        tmpAS[ℓ.lhs] := tmpAS[ℓ.rhs]
    else if ℓ is BinaryStmt then
        tmpAS[ℓ.res] := tmpAS[ℓ.op1] ⊗̂ tmpAS[ℓ.op2]
    else if ℓ is PhiStmt then
        rhsVal := UnknownAbsVal
        for op ∈ ℓ.ops do
            rhsVal.join_with(tmpAS[op])
        tmpAS[ℓ.res] := rhsVal
    else if ℓ is GepStmt then
        gepAbsVal := UnknownAbsVal
        offsetAbsVal := tmpAS[ℓ.offset]
        for idx ∈ [offsetAbsVal.lb(), offsetAbsVal.ub()] do
            gepAbsVal.join_with(getGepObjAddress(ℓ.base, idx))
        tmpAS[ℓ.res] := gepAbsVal
    ...
```

Algorithm 4: Abstract execution guided by WTO (part 4)

```
Function handleStatement(ℓ):
    ...
    else if ℓ is LoadStmt then
        rhsVal := UnknownAbsVal
        for addr ∈ tmpAS[ℓ.rhs] do
            rhsVal.join_with(tmpAS.load(addr))
        tmpAS[ℓ.lhs] := rhsVal
    else if ℓ is StoreStmt then
        for addr ∈ tmpAS[ℓ.lhs] do
            tmpAS.store(addr, tmpAS[ℓ.rhs])
    postAbsTrace[ℓ] := tmpAS
```

A Running Example: Abstract Execution

```c
extern void assert(int);

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

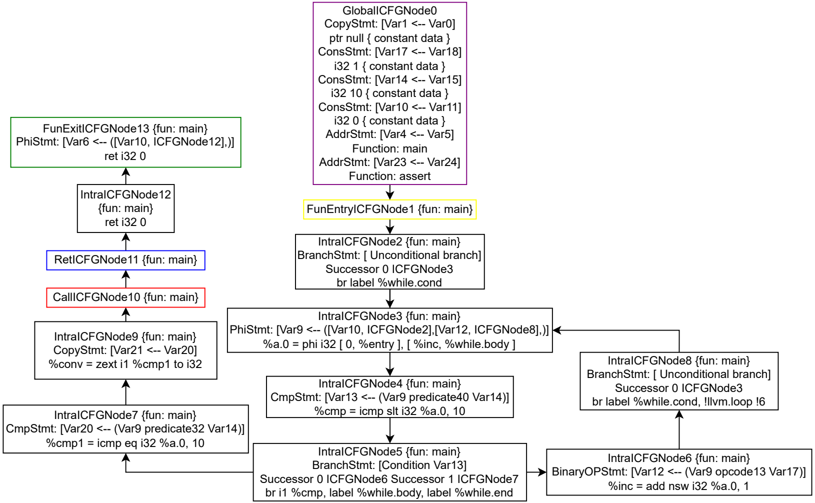
Source Code

Compile to LLVM IR

```llvm
define dso_local i32 @main() {
entry:
  br label %while.cond
while.cond:
  %a.0 = phi i32 [ 0, %entry ], [ %inc, %while.body ]
  %cmp = icmp slt i32 %a.0, 10
  br i1 %cmp, label %while.body, label %while.end
while.body:
  %inc = add nsw i32 %a.0, 1
  br label %while.cond,
while.end:
  %cmp1 = icmp eq i32 %a.0, 10
  %conv = zext i1 %cmp1 to i32
  call void @assert(i32 noundef %conv)
  ret i32 0
}
```

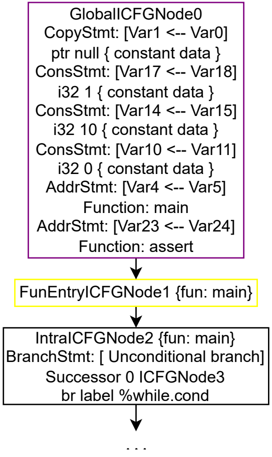
LLVM IR

A Running Example: Abstract Execution



ICFG

A Running Example: Abstract Execution



ICFG

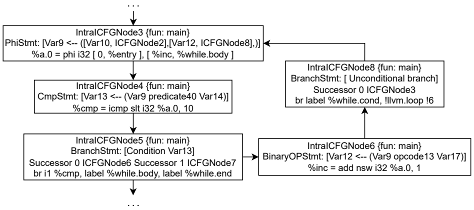
Algorithm 5: Abstract execution guided by WTO

Function `handleStatement(ℓ)`:
  tmpAS := *preAbsTrace*[ℓ]
  if ℓ *is* ConsStmt *or* AddrStmt **then**
    `initSVFVar(ℓ.rhs)`
    tmpAS[ℓ.*lhs*] := *tmpAS*[ℓ.*rhs*]
  else if ℓ *is* CopyStmt **then**
    tmpAS[ℓ.*lhs*] := *tmpAS*[ℓ.*rhs*]
  . . .

$postAbsTrace[\ell_0].varToAbsVal$:

Variable	*Interval*	*MemAddress*
Var0	$\top$	{0x7f00}
Var1	$\top$	{0x7f00}
Var18	[1,1]	$\top$
Var17	[1,1]	$\top$
Var14	[10,10]	$\top$
Var15	[10,10]	$\top$
Var10	[0,0]	$\top$
Var11	[0,0]	$\top$
	. . .	

A Running Example: Abstract Execution



ICFG

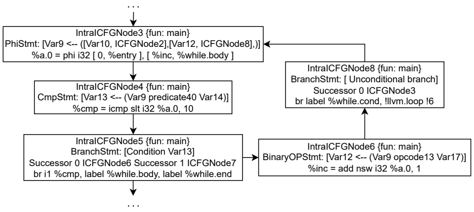
$postAbsTrace[\ell_3].varToAbsVal$:

Variable	*Interval*	*MemAddress*
...		
Var10	[0,0]	$\top$
Var9	[0,0]	$\top$
...		

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]// iter ≡ 1
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h])
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h])
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)
```

A Running Example: Abstract Execution



ICFG

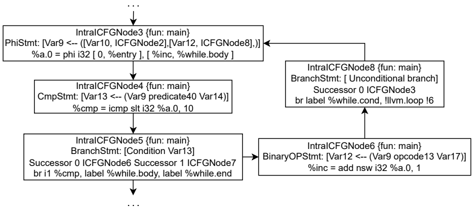
$postAbsTrace[\ell_8].varToAbsVal$:

Variable	*Interval*	*MemAddress*
...		
Var10	[0,0]	$\top$
Var9	[0,0]	$\top$
Var12	[1,1]	$\top$
...		

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h])
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h])
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)// iter ≡ 1
```

A Running Example: Abstract Execution



ICFG

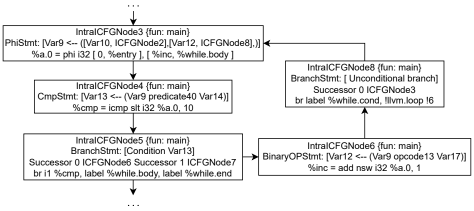
$postAbsTrace[\ell_3].varToAbsVal$:

Variable	Interval	MemAddress
	...	
Var9	[0,1]	$\top$
Var12	[1,1]	$\top$
	...	

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]// iter ≡ 2
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h])
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h])
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)
```

A Running Example: Abstract Execution



ICFG

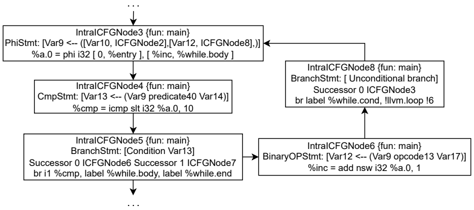
$postAbsTrace[\ell_8].varToAbsVal$:

Variable	*Interval*	*MemAddress*
	...	
Var9	[0,1]	$\top$
Var12	[1,2]	$\top$
	...	

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h])
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h])
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)// iter ≡ 2
```

A Running Example: Abstract Execution



ICFG

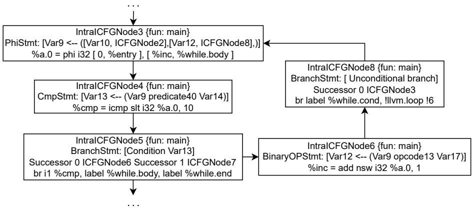
$postAbsTrace[\ell_3].varToAbsVal$:

Variable	*Interval*	*MemAddress*
	...	
Var9	$[0, +\infty]$	$\top$
Var12	[1,2]	$\top$
	...	

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h]) // iter ≡ 3
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h])
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)
```

A Running Example: Abstract Execution



ICFG

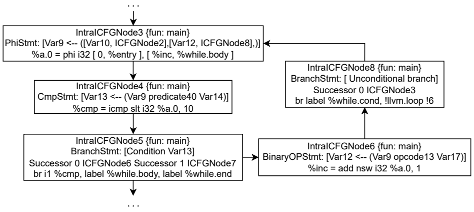
$postAbsTrace[\ell_8].varToAbsVal$:

Variable	*Interval*	*MemAddress*
	...	
Var9	[0,9]	$\top$
Var12	[1,10]	$\top$
	...	

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h])
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h])
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)// iter ≡ 3
```

A Running Example: Abstract Execution



ICFG

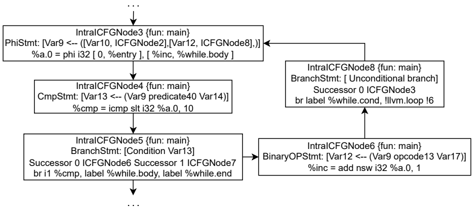
$postAbsTrace[\ell_3].varToAbsVal$:

Variable	*Interval*	*MemAddress*
	...	
Var9	$[0, +\infty]$	$\top$
Var12	[1,10]	$\top$
	...	

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h]) // iter ≡ 4
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h])
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)
```

A Running Example: Abstract Execution



ICFG

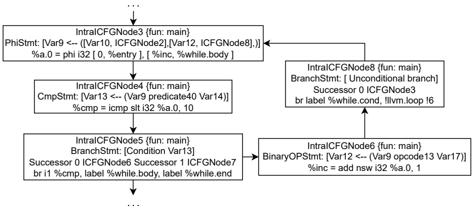
$postAbsTrace[\ell_3].varToAbsVal$:

Variable	*Interval*	*MemAddress*
	...	
Var9	$[0, 10]$	$\top$
Var12	[1,10]	$\top$
	...	

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h])
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h]) // iter ≡ 5
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)
```

A Running Example: Abstract Execution



ICFG

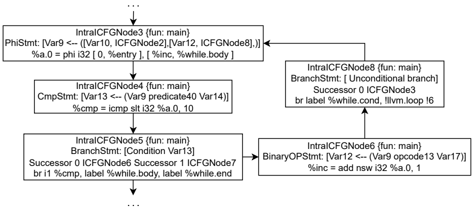
$postAbsTrace[\ell_8].varToAbsVal$:

Variable	*Interval*	*MemAddress*
	...	
Var9	[0,9]	$\top$
Var12	[1,10]	$\top$
	...	

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h])
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h])
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)// iter ≡ 5
```

A Running Example: Abstract Execution



ICFG

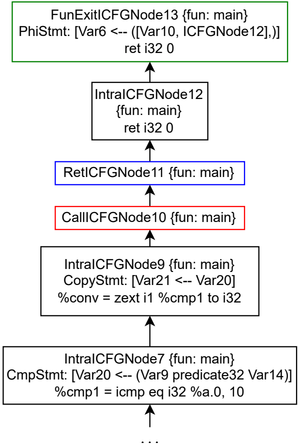
$postAbsTrace[\ell_3].varToAbsVal$:

Variable	*Interval*	*MemAddress*
	...	
Var9	$[0, 10]$	$\top$
Var12	[1,10]	$\top$
	...	

Algorithm 12: Abstract execution guided by WTO (part 2)

```
Function handleCycle(cycle):
    h := head(cycle)
    INC := true
    iter := 0
    while true do
        iter++
        handleICFGNode(h)
        if iter < widen_delay then
            tmpAS := postAbsTrace[h]
        else
            if INC then
                postAbsTrace[h] := tmpAS.widen(postAbsTrace[h])
                if postAbsTrace[h] ≤ tmpAS then
                    ING := false
                    tmpAS := postAbsTrace[h]
                    continue
                tmpAS := postAbsTrace[h]
            else
                postAbsTrace[h] := tmpAS.narrow(postAbsTrace[h]) // iter ≡ 6
                if postAbsTrace[h] ≥ tmpAS then
                    break
                tmpAS := postAbsTrace[h]
        handleCycleBody(cycle)
```

A Running Example: Abstract Execution



ICFG

Algorithm 13: Abstract execution guided by WTO

Function `handleStatement`(ℓ)**:**

  $tmpAS := preAbsTrace[\ell]$

  if ℓ *is* BinaryStmt **then**

    $postAbsTrace[\ell][\ell.res] := preAbsTrace[\ell][\ell.op1] \hat{\otimes} preAbsTrace[\ell][\ell.op2]$

  ...

$postAbsTrace[\ell_7].varToAbsVal$:

Variable	*Interval*	*MemAddress*
...		
Var9	[10,10]	$\top$
Var20	[1,1]	$\top$
...		