Lab: Data-Dependence and Pointer Aliasing

(Week 3)

Yulei Sui

School of Computer Science and Engineering University of New South Wales, Australia

Quiz-1 + Lab-Exercise-1 + Assignment-1

- A set of quizzes on WebCMS (5 points)
 - LLVM compiler and its intermediate representation
 - Code graphs (including ICFG and PAG)
- Lab-Exercise-1 (5 points)
 - Implement a graph traversal on a general graph
- Assignment-1 (20 points)
 - Control-flow: Implement a context-sensitive graph traversal on a CodeGraph (i.e., ICFG) and print feasible paths from a source node to a sink node on the graph
 - Data-flow: Implement Andersen's inclusion-based constraint solving for points-to analysis
 - Implement a taint checker using control-flow analysis and data-flow analysis.

Quiz-1 + Lab-Exercise-1 + Assignment-1

- A set of guizzes on WebCMS (5 points)
 - LLVM compiler and its intermediate representation
 - Code graphs (including ICFG and PAG)
- Lab-Exercise-1 (5 points)
 - Implement a graph traversal on a general graph
- Assignment-1 (20 points)
 - Control-flow: Implement a context-sensitive graph traversal on a CodeGraph (i.e., ICFG) and print **feasible** paths from a source node to a sink node on the graph
 - Data-flow: Implement Andersen's inclusion-based constraint solving for points-to analysis
 - Implement a taint checker using control-flow analysis and data-flow analysis.
 - Specification and code template: https: //github.com/SVF-tools/Software-Security-Analysis/wiki/Assignment-1
 - SVF APIs for control- and data-flow analysis https: //github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API

Algorithm

```
Сору
                                                                           → Store
define i32 @main() #0 {
entry:
                                     // 01
  %a1 = alloca i8, align 1
                                     // O2
  %a = alloca ptr, align 8
                                                                       {04}
                                                           {O3}
                                     // O3
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr, ptr %q, align 8
  store ptr %1, ptr %p, align 8
  store ptr %0, ptr %a, alian 8
  ret void
```

Address

▶ Load

Algorithm

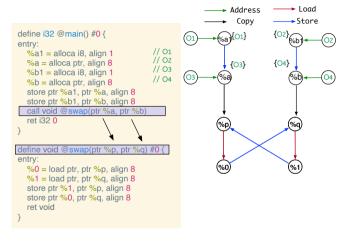
```
Address
                                                           Copy
                                                                           → Store
define i32 @main() #0 {
entry:
                                     // 01
   %a1 = alloca i8. align 1
                                     // O2
  %a = alloca ptr. align 8
                                                                      {04}
                                                          {O3}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
  store ptr %0, ptr %q, align 8
  ret void
```

Load

```
define i32 @main() #0 {
                                              (01
entry:
                                     // 01
  %a1 = alloca i8, align 1
                                     // O2
  %a = alloca ptr. align 8
                                     // O3
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
  store ptr %0, ptr %a, alian 8
  ret void
```

```
Address Load
Copy Store

(O2)
(O3)
(O4)
(O6)
(O6)
(O6)
```



```
Algorithm 1: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
s while Worklist + do
       p := popFromWorklist():
       foreach o \in pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
8
                                                   // Store rule
               if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                               // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                    // Load rule
               if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                               // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
          if pts(x) changed then
18
19
               pushIntoWorklist(x):
       foreach p \xrightarrow{Gep} x \in E do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
               pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
               pushIntoWorklist(x);
```

Algorithm

```
Address
                                                                             Load
                                                                           ► Store
                                                            Copy
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                                                      {04}
                                                          {O3}
                                      // O2
  %a = alloca ptr. align 8
                                      // O3
  %b1 = alloca i8, align 1
                                                                                    (04)
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
  %0 = load ptr. ptr %p, align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
  ret void
                                                              WorkList
```

```
Algorithm 2: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
      pts(p) = o:
      pushIntoWorklist(p):
5 while WorkList \(\perp \) do
      p := popFromWorklist():
      foreach o \in pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                   // Store rule
              if q <sup>Copy</sup> o ∉ E then
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                               // Add copy edge
                   pushIntoWorklist(g):
          foreach p \xrightarrow{Load} r \in E do
                                                    // Load rule
              if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                               // Add copy edge
                  pushIntoWorklist(o);
      foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
          pts(x) := pts(x) \cup pts(p):
          if pts(x) changed then
              pushIntoWorklist(x):
      foreach p \xrightarrow{Gep} x \in E do
                                                     // Gep rule
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
          if pts(x) changed then
              pushIntoWorklist(x);
```

8

9

10

11

12

14

15

17

19

20

21

22

```
Address
                                                                             Load
                                                                           ► Store
                                                            Copy
                                                                      {O2}
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                                                      {04}
                                                          {O3}
                                      // O2
  %a = alloca ptr. align 8
                                      // O3
  %b1 = alloca i8, align 1
                                                                                   (04)
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
                                                       %p)
define void @swap(ptr %p, ptr %q) #0 {
entry:
  %0 = load ptr. ptr %p, align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
  ret void
                                                              WorkList
```

```
Algorithm 3: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                // Address rule
       pts(p) = o:
       pushIntoWorklist(p):
s while WorkList ≠ do
       p := popFromWorklist():
       foreach o \in pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
8
                                                   // Store rule
               if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                               // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                    // Load rule
               if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                               // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
               pushIntoWorklist(x):
       foreach p \xrightarrow{Gep} x \in E do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
21
               pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
               pushIntoWorklist(x);
```

Algorithm

```
Address
                                                                             Load
                                                                           ► Store
                                                            Copy
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                                                      {04}
                                                          {O3}
                                      // O2
  %a = alloca ptr. align 8
                                      // O3
  %b1 = alloca i8, align 1
                                                                                   (04)
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
  %0 = load ptr. ptr %p, align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
                                                                                     %a1
  ret void
                                                              WorkList
```

```
Algorithm 4: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
      pts(p) = o:
      pushIntoWorklist(p):
s while Worklist + do
      p := popFromWorklist():
      foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if q <sup>Copy</sup> o ∉ E then
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                              // Add copy edge
                   pushIntoWorklist(g):
          foreach p \xrightarrow{Load} r \in E do
                                                    // Load rule
              if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                              // Add copy edge
                  pushIntoWorklist(o);
      foreach p \xrightarrow{Copy} x \in E do
                                                   // Copy rule
          pts(x) := pts(x) \cup pts(p):
          if pts(x) changed then
              pushIntoWorklist(x):
      foreach p \xrightarrow{Gep} x \in E do
                                                     // Gep rule
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
          if pts(x) changed then
              pushIntoWorklist(x);
```

8

9

10

11

12

14

15

17

19

20

21

22

```
Address
                                                                             Load
                                                                           ► Store
                                                            Copy
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                                                      {04}
                                                          {O3}
                                      // O2
  %a = alloca ptr. align 8
                                      // O3
  %b1 = alloca i8, align 1
                                                                                   (04)
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
  %0 = load ptr. ptr %p, align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
                                                                      %h
                                                                                    %b1
  ret void
                                                              WorkList
```

```
Algorithm 5: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
s while Worklist + do
       p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                   // Store rule
8
              if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                    // Load rule
              if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                               // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \xrightarrow{Gep} x \in E do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
              pushIntoWorklist(x);
```

```
Address
                                                                             Load
                                                                           → Store
                                                            Copy
define i32 @main() #0 {
                                              (01
entry:
  %a1 = alloca i8, align 1
                                                                      {04}
                                                          {O3}
                                      // O2
  %a = alloca ptr. align 8
                                      // O3
  %b1 = alloca i8, align 1
                                                                                   (04)
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
  %0 = load ptr. ptr %p, align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
                                                                     %a1
                                                                                     %a
  ret void
                                                              World ist
```

```
Algorithm 6: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                 // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
5 while WorkList \(\perp \) do
       p := popFromWorklist():
       foreach o e pts(p) do
           foreach a \xrightarrow{\text{Store}} p \in E do
8
                                                   // Store rule
               if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                               // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                     // Load rule
               if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                               // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
               pushIntoWorklist(x):
       foreach p \xrightarrow{Gep} x \in E do
                                                      // Gep rule
20
          foreach o ∈ pts(p) do
21
               pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
               pushIntoWorklist(x);
```

```
Address
                                                                             I ond
                                                            Copy
                                                                          Store
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                                                      {04}
                                                          {O3}
                                      // O2
  %a = alloca ptr. align 8
                                      // O3
  %b1 = alloca i8, align 1
                                                                                   (04)
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
define void @swap(ptr %p, ptr %g) #0 {
entry:
  %0 = load ptr, ptr %p, align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %g, align 8
  ret void
                                                              WorkWist
```

```
Algorithm 7: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                               // Address rule
      pts(p) = o:
      pushIntoWorklist(p):
  while Worklist + do
      p := popFromWorklist():
      foreach o ∈ pts(p) do
          foreach q \xrightarrow{Store} p \in E do
                                                 // Store rule
              if q <sup>Copy</sup> o ∉ E then
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                             // Add copy edge
                   pushIntoWorklist(g):
          foreach p \xrightarrow{Load} r \in E do
12
                                                  // Load rule
              if o copy r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                             // Add copy edge
14
                  pushIntoWorklist(o);
      foreach p \xrightarrow{Copy} x \in E do
                                                  // Copy rule
17
          pts(x) := pts(x) \cup pts(p):
          if pts(x) changed then
19
              pushIntoWorklist(x):
      foreach p \xrightarrow{Gep} x \in E do
20
                                                    // Gep rule
21
          foreach o ∈ pts(p) do
           | pts(x) := pts(x) ∪ {o.fld};
          if pts(x) changed then
23
              pushIntoWorklist(x);
```

```
Address
                                                                             Load
                                                                           ► Store
                                                            Copy
                                                                      {O2}
define i32 @main() #0 {
                                             (01
entry:
  %a1 = alloca i8, align 1
                                                                      {04}
                                                          {O3}
                                      // O(01)
  %a = alloca ptr. align 8
                                      // O3
  %b1 = alloca i8, align 1
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
                                                          {O3}
                                                                      {04}
  call void @swap(ptr %a, ptr %b)
  ret i32 0
                                                        %p)
define void @swap(ptr %p, ptr %q) #0 {
entry:
  %0 = load ptr. ptr %p, align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
                                                               O3 %a %b1 %p
                                                                                    %a1
  ret void
                                                              Worklist
```

```
Algorithm 8: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
5 while WorkList \(\perp \) do
       p := popFromWorklist():
       foreach o \in pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
8
                                                  // Store rule
              if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                    // Load rule
              if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                              // Add copy edge
14
                  pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p Gep x ∈ E do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
              pushIntoWorklist(x);
```

Algorithm

```
Store
                                                             Copy
define i32 @main() #0 {
                                                                       (02)
                                              (01
                                                                                    (02
entry:
  %a1 = alloca i8, align 1
                                              {01}
  %a = alloca ptr. align 8
                                                                       (Q4)
                                                           {O3}
                                     // O3
  %b1 = alloca i8, align 1
                                              (03
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                           {O3}
                                                                       {04}
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                        (‰
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                                                 head
                                                 tail
  store ptr %0, ptr %a, alian 8
                                                               %1 O3 %a %b
  ret void
                                                                   W
```

Address

Load

```
Algorithm 9: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                 // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
5 while WorkList \(\perp \) do
       p := popFromWorklist():
       foreach o e pts(p) do
           foreach a \xrightarrow{\text{Store}} p \in E do
8
                                                   // Store rule
               if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                               // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                     // Load rule
               if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                               // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
               pushIntoWorklist(x):
       foreach p \xrightarrow{Gep} x \in E do
                                                      // Gep rule
20
          foreach o ∈ pts(p) do
21
               pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
               pushIntoWorklist(x);
```

Algorithm

```
Store
                                                             Copy
define i32 @main() #0 {
                                                                       (02)
                                              (01
                                                                                     (02
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                       (Q4)
                                                           {O3}
                                     // O3(O1)
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                           \{O3\}
                                                                       {04}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                        (%0
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                                                 head
                                                 tail
  store ptr %0, ptr %a, alian 8
                                                           O3 %1 O3 %a %b
  ret void
                                                                   W
```

Address

Load

```
Algorithm 10: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                               // Address rule
      pts(p) = o:
      pushIntoWorklist(p):
5 while WorkList \(\perp \) do
      p := popFromWorklist():
      foreach o \in pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
8
                                                  // Store rule
              if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                   // Load rule
              if o copy r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                              // Add copy edge
14
                  pushIntoWorklist(o);
15
      foreach p Copy x ∈ E do
                                                   // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
              pushIntoWorklist(x):
      foreach p \xrightarrow{Gep} x \in E do
                                                    // Gep rule
20
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
              pushIntoWorklist(x);
```

Algorithm

```
Address
                                                                              Load
                                                                           Store
                                                            Copy
define i32 @main() #0 {
                                                                       (02)
                                              (01
                                                                                    (02
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                       (Q4)
                                                           {O3}
                                     // O3(O1)
                                                                                      {02}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                           \{O3\}
                                                                       {04}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                       (%0
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                                                 head
                                                 tail
  store ptr %0, ptr %a, alian 8
                                                                03 %1 03 %0
  ret void
                                                                   W
```

```
Algorithm 11: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                               // Address rule
      pts(p) = o:
      pushIntoWorklist(p):
s while WorkList + do
      p := popFromWorklist():
      foreach o \in pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                 // Store rule
              if q <sup>Copy</sup> o ∉ E then
                  E := E \cup \{a \xrightarrow{Copy} o\}:
                                             // Add copy edge
                  pushIntoWorklist(g):
          foreach p \xrightarrow{Load} r \in E do
                                                   // Load rule
              if o <sup>Copy</sup> r ∉ E then
                  E := E \cup \{o \xrightarrow{Copy} r\}:
                                             // Add copy edge
                  pushIntoWorklist(o);
      foreach p \xrightarrow{Copy} x \in E do
                                                   // Copy rule
          pts(x) := pts(x) \cup pts(p):
          if pts(x) changed then
              pushIntoWorklist(x):
      foreach p Gep x ∈ E do
                                                    // Gep rule
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
          if pts(x) changed then
              pushIntoWorklist(x);
```

8

9

10

11

12

14

15

17

19

20

22

```
Address
                                                                              Load
                                                                           Store
                                                            Copy
define i32 @main() #0 {
                                                                       (02)
                                              (01
entry:
  %a1 = alloca i8, align 1
                                     1/ 02.
  %a = alloca ptr. align 8
                                                                       (Q4)
                                                          {O3}
                                     // 03(01)
                                                                                      {02}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                                       {04}
                                                           \{O3\}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                                                 head
                                                tail
  store ptr %0, ptr %a, alian 8
                                                        O4%0 O4 O3 %1 O3
  ret void
                                                                   W
```

```
Algorithm 12: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
5 while WorkList \(\perp \) do
       p := popFromWorklist():
       foreach o ∈ pts(p) do
           foreach a \xrightarrow{\text{Store}} p \in E do
                                                   // Store rule
              if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                    // Load rule
              if o copy r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \xrightarrow{Gep} x \in E do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
              pushIntoWorklist(x);
```

Algorithm

```
Store
                                                            Copy
define i32 @main() #0 {
                                                                       (02)
                                              (01
                                                                                    (02
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                       (Q4)
                                                          {O3}
                                     // O3(O1)
                                                                                      {02}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                                       {04}
                                                           \{O3\}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                       (%0){O1}
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                                                 head
                                                tail
  store ptr %0, ptr %a, alian 8
                                                         %0 O4%0 O4 O3 %1
                                                                                    03
  ret void
                                                                   W
```

Address

Load

```
Algorithm 13: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
5 while WorkList \(\perp \) do
       p := popFromWorklist():
       foreach o \in pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                    // Load rule
              if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p Gep x ∈ E do
                                                     // Gep rule
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
              pushIntoWorklist(x);
```

Algorithm

```
Store
                                                             Copy
define i32 @main() #0 {
                                                                       (02)
                                              (01
                                                                                     (02
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                       (Q4)
                                                           {O3}
                                     // 03(01)
                                                                                      {02}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                                       {04}
                                                           \{O3\}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                        (%0){O1}
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                                                 head
                                                 tail
  store ptr %0, ptr %a, alian 8
                                                             %0 O4%0 O4 O3
  ret void
                                                                   W
```

Address

Load

```
Algorithm 14: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
s while WorkList + do
      p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                   // Store rule
              if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                    // Load rule
              if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \xrightarrow{Gep} x \in E do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
              pushIntoWorklist(x);
```

Algorithm

```
Address
                                                                            Store
                                                             Copy
define i32 @main() #0 {
                                                                       (02)
                                              (01
entry:
  %a1 = alloca i8, align 1
                                     1/ 02.
  %a = alloca ptr. align 8
                                                                        (Q4)
                                                           {O3}
                                     // 03(01)
                                                                                       {02}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                                        {04}
                                                           \{O3\}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                        (%0){(01}
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                                                 head
                                                 tail
  store ptr %0, ptr %a, alian 8
                                                                 %0 O4%0 O4
  ret void
                                                                   W
```

Load

```
Algorithm 15: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
s while WorkList + do
      p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                   // Store rule
              if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                    // Load rule
              if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \xrightarrow{Gep} x \in E do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
              pushIntoWorklist(x);
```

Algorithm

```
Address
                                                                            Store
                                                             Copy
define i32 @main() #0 {
                                                                       (02)
                                              (01
                                                                                     (02
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                       (Q4)
                                                           {O3}
                                     // O3(O1)
                                                                                      {02}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                                       {04}
                                                           \{O3\}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                                      {02} (%
                                                        (%0){01}
  \%0 = load ptr. ptr \%p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                                                 head
                                                 tail
  store ptr %0, ptr %a, alian 8
                                                                 %1 %0 04%0
  ret void
                                                                   W
```

Load

20

```
Algorithm 16: 1 Anderson's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes;
2 foreach o Address p do
                                                // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
5 while WorkList \(\perp \) do
       p := popFromWorklist():
       foreach o \in pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if q <sup>Copy</sup> o ∉ E then
9
                   E := E \cup \{a \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                    // Load rule
              if o <sup>Copy</sup> r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                    // Copy rule
17
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p Gep x ∈ E do
                                                     // Gep rule
          foreach o ∈ pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
24
              pushIntoWorklist(x);
```

```
Address
                                                                                Load
                                                                             Store
                                                              Copy
define i32 @main() #0 {
                                                                         (02)
                                               (01
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                         (Q4)
                                                            {O3}
                                      // O3(O1)
  %b1 = alloca i8, align 1
                                                                                       (O4
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
                                                                                               9
  call void @swap(ptr %a, ptr %b)
                                                                         {04}
                                                                                               10
                                                            \{O3\}
                                                                                               11
  ret i32 0
                                                         %p
                                                                                               12
define void @swap(ptr %p, ptr %q) #0 {
                                                                                               14
                                                                                               15
entry:
                                                                        {02} (%
                                                         (%0){01}
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
                                                                                               17
  store ptr %1, ptr %p, align 8
                                                                                   head
                                                  tail
                                                                                               19
  store ptr %0, ptr %a, alian 8
                                                                   O4 %1 %0 O4
  ret void
                                                                                               20
                                                                                               21
                                                                     W
                                                                                               22
                                                                                               23
                                                                                               24
```

```
Algorithm 17: 1 Anderson's Pointer Analysis
              Input: G =< V.E >: Constraint Graph
                      V: a set of nodes in graph
                      E: a set of edges in graph
            1 WorkList := an empty vector of nodes;
            2 foreach o Address p do
                                                           // Address rule
                  pts(p) = o:
                  pushIntoWorklist(p):
\{O2,O1\}_{5} while WorkList \neq do
                  p := popFromWorklist():
                  foreach o \in pts(p) do
                      foreach a \xrightarrow{\text{Store}} p \in E do
                                                              // Store rule
                          if q <sup>Copy</sup> o ∉ E then
                               E := E \cup \{a \xrightarrow{Copy} o\}:
                                                          // Add copy edge
                               pushIntoWorklist(g):
                      foreach p \xrightarrow{Load} r \in E do
                                                               // Load rule
                          if o <sup>Copy</sup> r ∉ E then
                               E := E \cup \{o \xrightarrow{Copy} r\}:
                                                          // Add copy edge
                              pushIntoWorklist(o);
                  foreach p \xrightarrow{Copy} x \in E do
                                                               // Copy rule
                      pts(x) := pts(x) \cup pts(p):
                      if pts(x) changed then
                          pushIntoWorklist(x):
                  foreach p Gep x ∈ E do
                                                                // Gep rule
                      foreach o ∈ pts(p) do
                          pts(x) := pts(x) \cup \{o.fld\};
                      if pts(x) changed then
                          pushIntoWorklist(x);
```

```
Address
                                                                                 Load
                                                                              Store
                                                               Copy
define i32 @main() #0 {
                                                                          (02)
                                                (01
                                                                                        (02
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                          (Q4)
                                                             {O3}
                                      // O3(O1)
  %b1 = alloca i8, align 1
                                                                                        (O4
                                       // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
                                                                                                 9
  call void @swap(ptr %a, ptr %b)
                                                                          {04}
                                                                                                10
                                                             \{O3\}
                                                                                                11
  ret i32 0
                                                          %p
                                                                                                12
define void @swap(ptr %p, ptr %q) #0 {
                                                                                                14
                                                                                                15
entry:
                                                          (%0)<u>{</u>01}
                                                                              (%1
  \%0 = load ptr. ptr \%p. align 8
                                                                                       01}
                                                                                                16
  %1 = load ptr. ptr %g, align 8
                                                                                                17
  store ptr %1, ptr %p, align 8
                                                                                    head
                                                  tail
                                                                                                19
  store ptr %0, ptr %a, alian 8
                                                                  %1 04 %1 %0
  ret void
                                                                                                20
                                                                      W
                                                                                                22
                                                                                                23
                                                                                                24
```

```
Algorithm 18: 1 Anderson's Pointer Analysis
              Input: G =< V.E >: Constraint Graph
                      V: a set of nodes in graph
                      E: a set of edges in graph
            1 WorkList := an empty vector of nodes;
            2 foreach o Address p do
                                                           // Address rule
                  pts(p) = o:
                  pushIntoWorklist(p):
\{O2,O1\}_{5} while WorkList \neq do
                  p := popFromWorklist():
                  foreach o \in pts(p) do
                      foreach a \xrightarrow{\text{Store}} p \in E do
                                                              // Store rule
                          if q <sup>Copy</sup> o ∉ E then
                               E := E \cup \{a \xrightarrow{Copy} o\}:
                                                          // Add copy edge
                               pushIntoWorklist(g):
                      foreach p \xrightarrow{Load} r \in E do
                                                               // Load rule
                          if o <sup>Copy</sup> r ∉ E then
                               E := E \cup \{o \xrightarrow{Copy} r\}:
                                                          // Add copy edge
                              pushIntoWorklist(o);
                  foreach p \xrightarrow{Copy} x \in E do
                                                               // Copy rule
                      pts(x) := pts(x) \cup pts(p):
                      if pts(x) changed then
                          pushIntoWorklist(x):
                  foreach p Gep x ∈ E do
                                                                // Gep rule
                      foreach o ∈ pts(p) do
                          pts(x) := pts(x) \cup \{o.fld\};
                      if pts(x) changed then
                          pushIntoWorklist(x);
```

```
Address
                                                                                Load
                                                                              Store
                                                              Copy
define i32 @main() #0 {
                                                                         (02)
                                               (01)
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                         (Q4)
                                                            {O3}
  %b1 = alloca i8, align 1
                                                                                       (O4
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
                                                                                                9
  call void @swap(ptr %a, ptr %b)
                                                                         {04}
                                                                                               10
                                                             \{O3\}
                                                                                               11
  ret i32 0
                                                         %p
                                                                                               12
define void @swap(ptr %p, ptr %q) #0 {
                                                                                               14
                                                                                               15
entry:
                                                         (%0){O1}
                                                                             (%1
  \%0 = load ptr. ptr \%p. align 8
                                                                                      01}
                                                                                               16
  %1 = load ptr. ptr %g, align 8
                                                                                               17
  store ptr %1, ptr %p, align 8
                                                                                   head
                                                  tail
                                                                                               19
  store ptr %0, ptr %a, alian 8
                                                                    O3 %1 O4
  ret void
                                                                                               20
                                                                     W
                                                                                               22
                                                                                               23
                                                                                               24
```

```
Algorithm 19: 1 Anderson's Pointer Analysis
              Input: G =< V.E >: Constraint Graph
                      V: a set of nodes in graph
                      E: a set of edges in graph
            1 WorkList := an empty vector of nodes;
            2 foreach o Address p do
                                                           // Address rule
                  pts(p) = o:
                  pushIntoWorklist(p):
\{O2,O1\}_{5} while WorkList \neq do
                  p := popFromWorklist():
                  foreach o \in pts(p) do
                      foreach a \xrightarrow{\text{Store}} p \in E do
                                                              // Store rule
                          if q <sup>Copy</sup> o ∉ E then
                               E := E \cup \{a \xrightarrow{Copy} o\}:
                                                          // Add copy edge
                               pushIntoWorklist(g):
                      foreach p \xrightarrow{Load} r \in E do
                                                               // Load rule
                          if o <sup>Copy</sup> r ∉ E then
                               E := E \cup \{o \xrightarrow{Copy} r\}:
                                                          // Add copy edge
                              pushIntoWorklist(o);
                  foreach p \xrightarrow{Copy} x \in E do
                                                               // Copy rule
                      pts(x) := pts(x) \cup pts(p):
                      if pts(x) changed then
                          pushIntoWorklist(x):
                  foreach p Gep x ∈ E do
                                                                // Gep rule
                      foreach o ∈ pts(p) do
                          pts(x) := pts(x) \cup \{o.fld\};
                      if pts(x) changed then
                          pushIntoWorklist(x);
```

Algorithm

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
\%b1 = alloca i8. alian 1
                               1/02
%a = alloca i8*, alian 8
                               // O3
%b = alloca i8*, alian 8
                               // 04
store i8* %a1. i8** %a. alian 8
store i8* %b1. i8** %b. alian 8
call void @swap(i8** %a, i8** %b)
ret i32 0
define void @swap(i8** %p. i8** %a)
#0 {
entry:
%0 = load i8** %p, alian 8
%1 = load i8** %a, alian 8
store i8* %1, i8** %p, alian 8
store i8* %0, i8** %a, alian 8
ret void
```

```
Address
                              ► Load
             Copy
                           --- Store
                       {O2}<sub>%b1</sub>
        %a1\{O1}
         {O3}
                        {04}
                        {04}
           {03}
        %p
                             %a
        <sub>%0</sub>
{01.02}
                                      01}
                                   head
tail
                                        %0
                WorkList
```

```
Algorithm 20: 1 Anderson's Pointer Analysis
              Input: G =< V.E >: Constraint Graph
                       V: a set of nodes in graph
                       E: a set of edges in graph
             1 WorkList := an empty vector of nodes;
            2 foreach o Address p do
                                                            // Address rule
                   pts(p) = o:
                  pushIntoWorklist(p):
\{O2,O1\}_5 while WorkList \neq do
                   p := popFromWorklist():
                   foreach o \in pts(p) do
                       foreach a \xrightarrow{\text{Store}} p \in E do
                                                               // Store rule
                           if q <sup>Copy</sup> o ∉ E then
                               E := E \cup \{a \xrightarrow{Copy} o\}:
                                                           // Add copy edge
                               pushIntoWorklist(g):
                       foreach p \xrightarrow{Load} r \in E do
                                                                // Load rule
                           if o <sup>Copy</sup> r ∉ E then
                               E := E \cup \{o \xrightarrow{Copy} r\}:
                                                           // Add copy edge
                               pushIntoWorklist(o);
                   foreach p \xrightarrow{Copy} x \in E do
                                                                // Copy rule
                       pts(x) := pts(x) \cup pts(p):
                       if pts(x) changed then
                           pushIntoWorklist(x):
                   foreach p \xrightarrow{Gep} x \in F do
                                                                 // Gep rule
                       foreach o ∈ pts(p) do
                          pts(x) := pts(x) \cup \{o.fld\};
                       if pts(x) changed then
                           pushIntoWorklist(x);
```

9

10

11

12

14

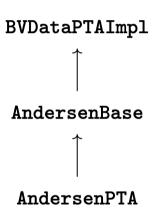
15

16 17

19

20

22



 You will be working on AndersenPTA's solveWorklist method.

BVDataPTAImpl



AndersenBase



AndersenPTA

- You will be working on AndersenPTA's solveWorklist method.
- Constraint graph is the field consCG.

BVDataPTAImpl

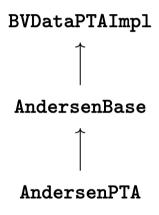


AndersenBase



AndersenPTA

- You will be working on AndersenPTA's solveWorklist method.
- Constraint graph is the field consCG.
- Address edge processing is done for you.



- You will be working on AndersenPTA's solveWorklist method.
- Constraint graph is the field consCG.
- Address edge processing is done for you.
- Note in the API there is a getDirectInEdges/getDirectOutEdges but no getCopyIn/OutEdges. This is intentional, use the Direct variant.
- You will reuse this assignment for assignment 4, make sure it is clean. :)

Lab: Information Flow Tracking

(Week 3)

Yulei Sui

School of Computer Science and Engineering University of New South Wales, Australia

Assignment 1: Taint Tracker

- Implement method readSrcSnkFromFile in Assignment-4.cpp using C++ file reading to configure sources and sinks.
- Implement method printICFGPath to collect the tainted ICFG paths and add each path (a sequence of node IDs) as a string into std::set<std::string> paths similar to Assignment 2
- Implement method aliasCheck to check aliases of the variables at source and sink.

Coding Task

- Code template and specification: https://github.com/SVF-tools/ Teaching-Software-Analysis/wiki/Assignment-4
- Make sure your previous implementations in Assignment-2.cpp and Assignment-3.cpp are in place.
 - Class TaintGraphTraversal in Assignment 4 is a child class of 'ICFGTraversal'. TaintGraphTraversal will use the DFS method implemented in Assignment 2 for control-flow traversal.
 - Andersen's analysis implemented in Assignment 3 will also be used for checking aliases between two pointers.

C++ File Reading

Implement method readSrcSnkFormFile in Assignment-4.cpp to parse the two lines from SrcSnk.txt in the form of

```
source -> { source src set getname update getchar tgetstr }
sink -> { sink mysql_query system require chmod broadcast }
```

Please refer to the following links (among many others) for C++ file reading:

- https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm
- https://www.cplusplus.com/doc/tutorial/files/
- https://linuxhint.com/cplusplus_read_write/
- https://opensource.com/article/21/3/ccc-input-output