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

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

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                        // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                       // Store rule
        if q Copy o ∉ E then
          E \leftarrow E \cup \{ q \xrightarrow{Copy} o \}
                                    // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                        // Load rule
11
        if o Copy r ∉ E then
          E <- E U { o Copy r }
                                   // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                        // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
           pushIntoWorklist(x)
18
```

Load

```
Load
                                                         Address
                                                           Сору
                                                                          Store
                                                                     {O2}
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                     // O1
                                                                     {04}
                                                         {O3}
                                     // 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
                                                tail
                                                                                head
  store ptr %0, ptr %q, align 8
  ret void
                                                              Worklist
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                        // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                       // Store rule
        if q Copy o ∉ E then
          E \leftarrow E \cup \{ q \xrightarrow{Copy} o \}
                                    // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                        // Load rule
11
        if o Copy r ∉ E then
          E <- E U { o Copy r }
                                   // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                        // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
           pushIntoWorklist(x)
18
```

```
Load
                                                          Address
                                                           Сору
                                                                           Store
                                                                      {O2},
                                                       {}_{a}\{01\}
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                      // O1
                                                                      {04}
                                                          {O3}
                                      // O2
  %a = alloca ptr. align 8
                                      // 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)
  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
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
  ret void
                                                              WorkList
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                        // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                       // Store rule
        if q Copy o ∉ E then
          E \leftarrow E \cup \{ q \xrightarrow{Copy} o \}
                                    // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                        // Load rule
11
        if o Copy r ∉ E then
          E <- E U { o Copy r }
                                   // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                        // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
           pushIntoWorklist(x)
18
```

```
Load
                                                          Address
                                                                          Store
                                                           Copy
                                                                     {O2}
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                      // O1
                                                                      {04}
                                                          {O3}
                                     // 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
                                                tail
                                                                                head
  store ptr %0, ptr %q, align 8
  ret void
                                                              Worklist
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                     // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
          pushIntoWorklist(x)
18
```

```
Load
                                                          Address
                                                                          Store
                                                           Copy
                                                                     {O2}
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                      // O1
                                                                      {04}
                                                          {O3}
                                     // 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 %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                head
  store ptr %0, ptr %q, align 8
  ret void
                                                              Worklist
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                      // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
          pushIntoWorklist(x)
18
```

```
Load
                                                         Address
                                                           Сору
                                                                          Store
                                                                     {O2}
                                                      %a}{O1}
                                             (01
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                     // O1
                                                                     {04}
                                                         {O3}
                                     // 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 %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                head
  store ptr %0, ptr %q, align 8
  ret void
                                                              WorkWist
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                        // Address rule
          pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                        // Store rule
        if a Copy o ∉ E then
          E \leftarrow E \cup \{ q \xrightarrow{Copy} o \}
                                    // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                        // Load rule
        if o <sup>Copy</sup>→r ∉ E then
          E <- E U { o Copy r }
                                    // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                        // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
           pushIntoWorklist(x)
18
```

```
Load
                                                         Address
                                                           Сору
                                                                          Store
                                                                     {O2}
                                                      %a}{O1}
                                             (01
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                     // O1
                                                                     {04}
                                                         {O3}
                                     // 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 %g, align 8
  store ptr %1, ptr %p, align 8
                                               tail
                                                                                head
  store ptr %0, ptr %q, align 8
                                                             %g %b1 %p %a1
  ret void
                                                             Worklist
```

```
G = < V F > // Constraint Graph
   V: a set of nodes in graph
   E: a set of edges in graph
   Worklist: a vector of nodes
  foreach o Address p do
                                         // Address rule
          pts(p) = \{o\}
         pushIntoWorklist(p)
  while WorkList ≠ Ø do
     p <- popFromWorklist()
     foreach o ∈ pts(p) do
       foreach g Store p do
                                        // Store rule
         if q Copy o ∉ E then
           E \leftarrow E \cup \{ q \xrightarrow{Copy} o \}
                                    // Add copy edge
           pushIntoWorklist(a)
       foreach p Load r do
                                         // Load rule
         if o <sup>Copy</sup>→r ∉ E then
           E <- E U { o Copy r }
                                    // Add copy edge
           pushIntoWorklist(o)
     foreach p Copy X EEdo
                                         // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
17
       if pts(x) changed then
           pushIntoWorklist(x)
18
```

```
Load
                                                         Address
                                                           Сору
                                                                          Store
                                                                     {O2}
                                                      %a}{O1}
                                             (01
define i32 @main() #0 {
entry:
                                     // 01
  %a1 = alloca i8, align 1
                                                                     {04}
                                                         {O3}
                                     // O{O1}
  %a = alloca ptr. align 8
  %b1 = alloca i8, align 1
                                     // O3
                                     // 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
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
                                               tail
                                                                                head
  store ptr %0, ptr %q, align 8
                                                              O3 %g %b1 %p
  ret void
                                                             Worklist
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                        // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                       // Store rule
        if a Copy o ∉ E then
          E \leftarrow E \cup \{ q \xrightarrow{Copy} o \}
                                    // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                        // Load rule
11
        if o Copy r ∉ E then
          E <- E U { o Copy r }
                                   // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                        // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
17
       if pts(x) changed then
           pushIntoWorklist(x)
18
```

```
Address
                                                                             Load
                                                            Copy
                                                                           Store
define i32 @main() #0 {
                                                        6a (O1)
                                              (01
                                                                                    (02)
entry:
  %a1 = alloca i8, align 1
                                     // O2
                                              {01}
  %a = alloca ptr. align 8
                                                          {O3}
                                                                      (04)
                                     // 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
  %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 %q, align 8
                                                              %1 O3 %g %b1
  ret void
                                                                   W
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                        // Address rule
          pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                        // Store rule
        if q Copy o ∉ E then
          E \leftarrow E \cup \{ a \xrightarrow{Copy} o \}
                                    // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                        // Load rule
        if o <sup>Copy</sup>→r ∉ E then
          E <- E U { o Copy r }
                                    // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                        // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
           pushIntoWorklist(x)
18
```

```
Address
                                                                              Load
                                                             Copy
                                                                            Store
define i32 @main() #0 {
                                                        6a (01)
                                              (01)
                                                                                     (02)
entry:
  %a1 = alloca i8, align 1
                                     // O2
// O3{O1}
  %a = alloca ptr. align 8
                                                           {O3}
                                                                       (04)
  %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 %q, align 8
                                                           O3 %1 O3 %a %b1
  ret void
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  WorkList: a vector of nodes
  foreach o Address p do
                                        // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                       // Store rule
        if a Copy_o ∉ E then
          E \leftarrow E \cup \{ a \xrightarrow{Copy} o \}
                                   // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                        // Load rule
        if o Copy r ∉ E then
          E <- E U { o Copy → r }
                                   // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                        // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
           pushIntoWorklist(x)
18
```

```
Address
                                                                             Load
                                                            Copy
                                                                           Store
define i32 @main() #0 {
                                                       6a (01)
                                                                      {O2},
                                             (01)
                                                                                   (02)
entry:
  %a1 = alloca i8, align 1
                                     // 02,
  %a = alloca ptr. align 8
                                                          {O3}
                                                                      (04)
                                                                                     {O2}
                                    // 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 %q, align 8
                                                            04 03 %1 03 %0
  ret void
                                                                   W
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                      // Store rule
        if a Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
          pushIntoWorklist(x)
18
```

```
Address
                                                                              Load
                                                             Copy
                                                                            Store
define i32 @main() #0 {
                                                        6a (01)
                                                                       {O2}<sub>2</sub>
                                              (01)
                                                                                     (02)
entry:
  %a1 = alloca i8, align 1
                                     // O2{O1}
  %a = alloca ptr. align 8
                                                           {O3}
                                                                       (04)
                                                                                       {O2}
  %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 = load ptr. ptr %p, align 8
                                                        %0
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                                                 head
                                                 tail
  store ptr %0, ptr %q, align 8
                                                         O4%0 O4 O3 %1 O3
  ret void
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                     // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
          pushIntoWorklist(x)
18
```

```
Address
                                                                              Load
                                                             Copy
                                                                            Store
define i32 @main() #0 {
                                                        %a}{O1}
                                                                       {O2}<sub>2</sub>
                                              (01)
                                                                                     (02)
entry:
  %a1 = alloca i8, align 1
                                     // O2{O1}
  %a = alloca ptr. align 8
                                                           {O3}
                                                                       (04)
                                                                                       {O2}
  %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) {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 %q, align 8
                                                          %0 O4%0 O4 O3 %1
  ret void
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                      // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
16
       pts(x) \leftarrow pts(x) \cup pts(p)
17
       if pts(x) changed then
          pushIntoWorklist(x)
```

```
Address
                                                                               Load
                                                             Copy
                                                                             Store
define i32 @main() #0 {
                                                        %a}{O1}
                                                                       {O2}<sub>2</sub>
                                              (01)
                                                                                     (02)
entry:
  %a1 = alloca i8, align 1
                                     // O2{O1}
  %a = alloca ptr. align 8
                                                           {O3}
                                                                        (04)
                                                                                       {O2}
  %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) {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 %q, align 8
                                                              %0 O4%0 O4 O3
  ret void
                                                                    W
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                      // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
          pushIntoWorklist(x)
18
```

```
Address
                                                                                Load
                                                              Copy
                                                                              Store
define i32 @main() #0 {
                                                         %a}{O1}
                                                                        {O2}<sub>2</sub>
                                               (01)
                                                                                      (02)
entry:
  %a1 = alloca i8, align 1
                                      // O2
// O3{O1}
  %a = alloca ptr. align 8
                                                            {O3}
                                                                         (04)
                                                                                        {O2}
  %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) {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 %q, align 8
                                                                   %0 Q4%0 Q4
  ret void
                                                                     W
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                      // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
17
          pushIntoWorklist(x)
18
```

```
Address
                                                                               Load
                                                             Copy
                                                                            Store
define i32 @main() #0 {
                                                         6a (01)
                                                                       {O2}<sub>2</sub>
                                              (01)
                                                                                     (02)
entry:
  %a1 = alloca i8, align 1
                                     // O2{O1}
  %a = alloca ptr. align 8
                                                           {O3}
                                                                        (04)
                                                                                       {O2}
  %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) {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 %q, align 8
                                                                 %1 %0 04%0
                                                                                      O4
  ret void
                                                                    W
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                      // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
17
       if pts(x) changed then
          pushIntoWorklist(x)
18
```

```
Address
                                                                                Load
                                                              Copy
                                                                             Store
define i32 @main() #0 {
                                                         6a (01)
                                                                        {O2}<sub>2</sub>
                                               (01)
                                                                                      (02)
entry:
  %a1 = alloca i8, align 1
                                      // O2
// O3{O1}
  %a = alloca ptr. align 8
                                                            {O3}
                                                                         (04)
                                                                                      {02.01}
  %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) {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 %q, align 8
                                                                   O4 %1 %0 O4 --
  ret void
                                                                     W
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                      // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
17
       if pts(x) changed then
          pushIntoWorklist(x)
18
```

```
Address
                                                                               Load
                                                             Copy
                                                                            Store
define i32 @main() #0 {
                                                         %a}{O1}
                                                                       {O2}<sub>2</sub>
                                              (01)
                                                                                     (02)
entry:
  %a1 = alloca i8, align 1
                                     // O2{O1}
  %a = alloca ptr. align 8
                                                           {O3}
                                                                        (04)
                                                                                      {02.01}
  %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) {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 %q, align 8
                                                                 %1 04 %1 %0
  ret void
                                                                    W
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                      // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
       if pts(x) changed then
          pushIntoWorklist(x)
```

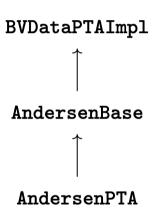
```
Address
                                                                               Load
                                                              Copy
                                                                             Store
define i32 @main() #0 {
                                                         6a (01)
                                                                        {O2}<sub>2</sub>
                                               (01)
                                                                                      (02)
entry:
                                      // 01
  %a1 = alloca i8, align 1
                                      // O2
// O01,02}
  %a = alloca ptr. align 8
                                                           {O3}
                                                                        (04)
                                                                                      {02.01}
  %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) {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 %q, align 8
                                                                   O3 %1 O4
                                                                                       %1
  ret void
                                                                    W
```

```
G = < V F > // Constraint Graph
  V: a set of nodes in graph
  E: a set of edges in graph
  Workl ist: a vector of nodes
  foreach o Address p do
                                      // Address rule
         pts(p) = \{o\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
    p <- popFromWorklist()
    foreach o ∈ pts(p) do
      foreach g Store p do
                                      // Store rule
        if q Copy o ∉ E then
          E <- E U { q Copy → o }
                                  // Add copy edge
          pushIntoWorklist(a)
      foreach p Load r do
                                      // Load rule
11
       if o Copy r ∉ E then
          E <- E U { o Copy r }
                                  // Add copy edge
13
          pushIntoWorklist(o)
    foreach p Copy X EEdo
                                      // Copy rule
       pts(x) \leftarrow pts(x) \cup pts(p)
16
       if pts(x) changed then
          pushIntoWorklist(x)
18
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
                               // 02
%b1 = alloca i8, alian 1
%a = alloca i8*, alian 8
%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 S
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}(0)
            (%a){O1}
    (01)
{01,02}
             {03}
                         {04}
                                     {02,01}
                         {04}
              {03}
            %р
     {01,621(%0)
     tail
                                  head
                  Worklist
```

```
G = < V.E > // Constraint Graph
  V: a set of nodes in graph
   E: a set of edges in graph
  WorkList: a vector of nodes
  foreach o Address P do
                                  // Address rule
        nts(n) = \{0\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
      p ← popFromWorklist()
      foreach o E pts(p) do
         foreach q Store p do
                                   // Store rule
             if q Copy o ∉ E then
                E \leftarrow E \cup \{q^{\frac{\text{Copy}}{\bullet}}o\} / \text{Add copy edge}
                pushIntoWorklist(q)
10
         foreach p Load r do // Load rule
11
             if o Copy r ∉ E then
12
                E \leftarrow E \cup \{o^{Copy} r\} // Add copy edge
13
                pushIntoWorklist(o)
14
      foreach p Copy x ∈ E do // Copy rule
15
          pts(x) \leftarrow pts(x) \cup pts(p)
16
          if pts(x) changed then
17
                 pushIntoWorklist(x)
18
```



 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

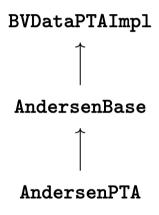


AndersenBase



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