Assignment 3

Yulei Sui

University of Technology Sydney, Australia

Assignment 3: Quiz + A Coding Task

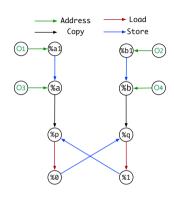
- A quiz (10 points)
 - Data dependence
 - Constraint graph
 - Andersen's points-to analysis

Assignment 3: Quiz + A Coding Task

- A quiz (10 points)
 - Data dependence
 - Constraint graph
 - Andersen's points-to analysis
- One coding task (15 points)
 - Goal: implement Andersen's pointer analysis by solving the constraint graph of a program.
 - Specification and code template: https: //github.com/SVF-tools/Teaching-Software-Analysis/wiki/Assignment-3
 - SVF CPP API https:

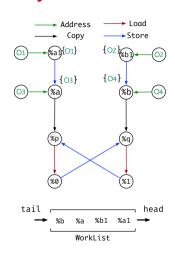
```
//github.com/SVF-tools/Teaching-Software-Analysis/wiki/SVF-CPP-API
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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 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
```



```
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) = \{n\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
      p ← popFromWorklist()
      foreach o E pts(p) do
         foreach g 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
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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 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
```



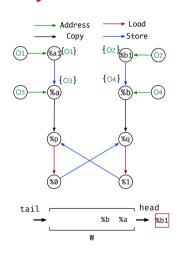
```
G = < V,E > // Constraint Graph
  V: a set of nodes in graph
   E: a set of edges in graph
  WorkList: a vector of nodes
  foreach o Address P do
                                  // Address rule
        nts(n) = \{0\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
      p ← popFromWorklist()
      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
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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 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
```

```
→ Load
           Address
            Copy
                        → Store
        %a1{O1}
                     {04}
          {03}
(03
 tail
                         %b1
               %h
                   %a
                  W
```

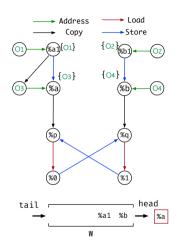
```
G = < V.E > // Constraint Graph
  V: a set of nodes in graph
   E: a set of edges in graph
  WorkList: a vector of nodes
  foreach o Address P do
                                  // Address rule
        nts(n) = \{0\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
      p ← popFromWorklist()
      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
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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 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
```



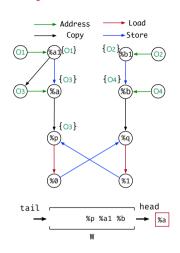
```
G = < V.E > // Constraint Graph
  V: a set of nodes in graph
   E: a set of edges in graph
  WorkList: a vector of nodes
  foreach o Address P do
                                  // Address rule
        nts(n) = \{0\}
        pushIntoWorklist(p)
  while WorkList ≠ Ø do
      p ← popFromWorklist()
      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
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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 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
```



```
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)
         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
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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 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
```



```
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
          pts(x) \leftarrow pts(x) \cup pts(p)
16
          if pts(x) changed then
17
                 pushIntoWorklist(x)
18
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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 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
```

```
→ Load
          Address
            Copy
                        → Store
                    {O2}(02)
        %a1{O1}
(01
         {03}
                     {04}
(03
          {03}
                     {04}
 tail
              %g %b1 %p %a1
                  W
```

```
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)
         foreach p Load r do // Load rule
             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
          pts(x) \leftarrow pts(x) \cup pts(p)
16
          if pts(x) changed then
                 pushIntoWorklist(x)
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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
```

```
→ Load
             Address
               Copy
                          → Store
                      {O2}
          %a1{O1}
   (01)
{01}
            {03}
                       {04}
             {03}
                       {04}
    tail
                 O3 %a %b1 %p
                    W
```

```
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
          pts(x) \leftarrow pts(x) \cup pts(p)
16
          if pts(x) changed then
                 pushIntoWorklist(x)
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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
```

```
→ Load
              Address
               Copy
                            → Store
                        {O2}(02)
           %a1{O1}
   (01)
{01}
            {03}
                        {04}
             {03}
                        {04}
           (%0
    tail
                  %1 O3 %a %b1
                     W
```

```
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
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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
```

```
→ Load
                Address
                 Copy
                               → Store
                          {O2}<sub>(2h1)</sub>-
            %a1{O1}
   (01
{01}
              {03}
                           {04}
               {03}
                           {04}
             %р
             (%0
     tail
                O3 %1 O3 %g %b1
                        W
```

```
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)
11
         foreach p Load r do // Load rule
             if o Copy r ∉ E then
12
                E \leftarrow E \cup \{o^{Copy} r\} // Add copy edge
13
                pushIntoWorklist(o)
14
15
      foreach p Copy x ∈ E do // Copy rule
           pts(x) \leftarrow pts(x) \cup pts(p)
16
           if pts(x) changed then
17
                  pushIntoWorklist(x)
18
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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
```

```
→ Load
              Address
               Copy
                           → Store
                       {O2}(0)
           %a1{O1}
   (01
{O1}
                                    {02}
            {03}
                       {04}
             {03}
                       {04}
           %р
           (%0
    tail
               O4 O3 %1 O3 %g --
```

```
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
          pts(x) \leftarrow pts(x) \cup pts(p)
16
          if pts(x) changed then
                 pushIntoWorklist(x)
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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}<sub>%h1</sub>, →
            (%a){O1}
   (01)
{01}
             {03}
                           {04}
                                         {02}
               {03}
                           {04}
             %р
             %a
     tail
                                    head
             O4 %0 O4 O3 %1 O3
                       W
```

```
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 g 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)
         foreach p Load r do // Load rule
             if o Copy r ∉ E then
12
13
                E \leftarrow E \cup \{o^{Copy} r\} // Add copy edge
                pushIntoWorklist(o)
14
15
      foreach p Copy x ∈ E do // Copy rule
           pts(x) \leftarrow pts(x) \cup pts(p)
16
           if pts(x) changed then
17
                 pushIntoWorklist(x)
18
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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}<sub>%h1</sub>, →
            (%a){O1}
   (01)
{01}
              {03}
                           {04}
                                         {02}
               {03}
                           {04}
             %р
             %0){O1}
     tail
                                    head
              %0 O4 %0 O4 O3 %1 →
                        W
```

```
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
          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
%b1 = alloca i8, alian 1
                               // O2
%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}
            {03}
                        {04}
                                     {02}
             {03}
                        {04}
           %р
           %0){O1}
    tail
                                 head
                %0 O4 %0 O4 O3
                     W
```

```
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
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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}
            {03}
                        {04}
                                      {02}
              {03}
                        {04}
            %р
            %0){O1}
    tail
                                 head
                    %0 O4 %0 O4
                      W
```

```
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 g 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
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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}
            {03}
                        {04}
                                     {02}
              {03}
                        {04}
            %р
            %0\{O1}
                        1021/8
    tail
                                 head
                   %1 %0 O4 %0
                     W
```

```
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
          pts(x) \leftarrow pts(x) \cup pts(p)
16
          if pts(x) changed then
                 pushIntoWorklist(x)
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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}<sub>%h1</sub>, →
             (%a){O1}
   (01)
{01}
              {03}
                           {04}
                                        {02,01}
               {03}
                           {04}
             %р
             %0\{O1}
                           {O2} } %
     tail
                                     head
                      O4 %1 %0 O4 --
                        W
```

```
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
          pts(x) \leftarrow pts(x) \cup pts(p)
16
          if pts(x) changed then
17
                 pushIntoWorklist(x)
18
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
%b1 = alloca i8, alian 1
                               // O2
%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}
            {03}
                        {04}
                                    {02,01}
              {03}
                        {04}
            %р
            %0\{O1}
    tail
                                 head
                  %1 O4 %1 %0
                     W
```

```
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
          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
%b1 = alloca i8, alian 1
                               // O2
%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}(02)
            (%a){O1}
    (01)
{01,02}
             {03}
                         {04}
                                     {02,01}
              {03}
                         {04}
            %р
            %0){O1}
     tail
                                  head
                     O3 %1 O4
                      W
```

```
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
          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
%b1 = alloca i8, alian 1
                               // O2
%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
```

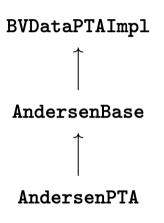
```
Address
                              Load
                 Copy
                             → Store
                         {O2}(02)
            (%a){O1}
    (01)
{01,02}
             {03}
                         {04}
                                     {02,01}
               {03}
                         {04}
             %р
     {01,62}(%0)
     tail
                                  head
                       W
```

```
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)
      foreach p Copy x ∈ E do // Copy rule
          pts(x) \leftarrow pts(x) \cup pts(p)
16
          if pts(x) changed then
                 pushIntoWorklist(x)
```

```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // 01
                               // 02
%b1 = alloca i8, alian 1
%a = alloca i8*, alian 8
                               // 03
%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}(02)
            (%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.

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.
- 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. :)