

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

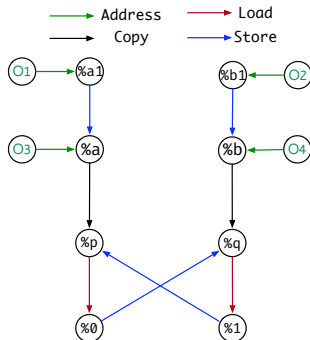
Assignment 3: Quiz + A Coding Task

- 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>

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

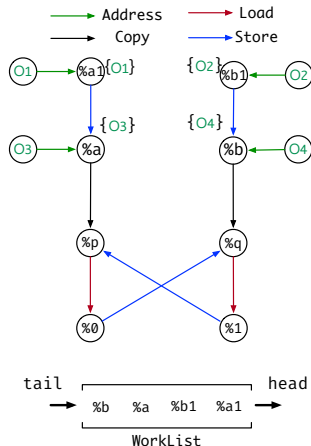


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

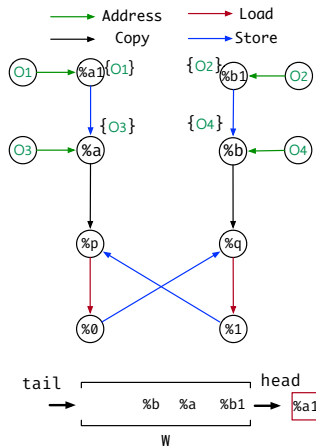


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach r  $\xrightarrow{\text{Load}}$  p do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

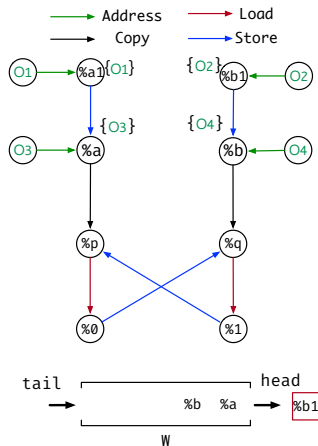


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

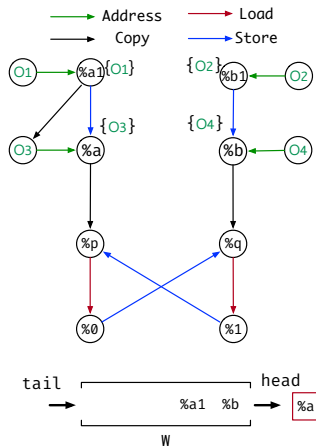


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

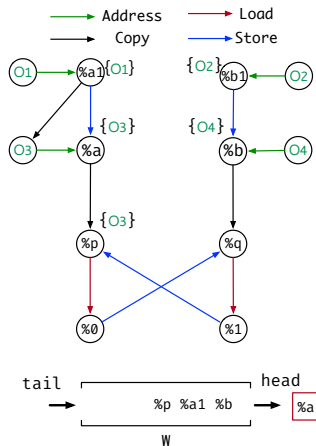


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```


Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

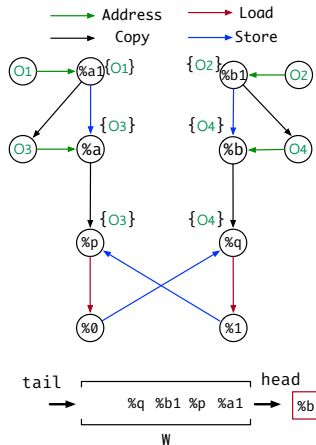


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

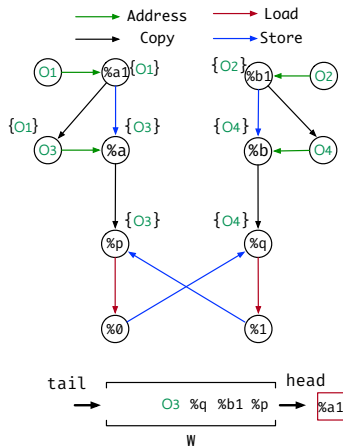


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, align 1      // O1
%b1 = alloca i8, align 1      // O2
%a = alloca i8*, align 8      // O3
%b = alloca i8*, align 8      // O4
store i8* %a1, i8** %a, align 8
store i8* %b1, i8** %b, align 8
call void @swap(i8** %a, i8** %b)
ret i32 0
}
define void @swap(i8** %p, i8** %q)
#0 {
entry:
%0 = load i8** %p, align 8
%1 = load i8** %q, align 8
store i8* %1, i8** %p, align 8
store i8* %0, i8** %q, align 8
ret void
}
```



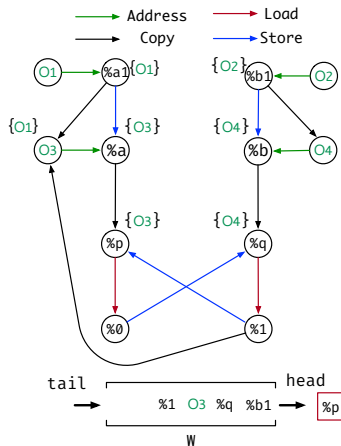
```
G = < V, E > // Constraint Graph
V: a set of nodes in graph
E: a set of edges in graph
WorkList: a vector of nodes

1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule
2   pts(p) = {o}
3   pushIntoWorklist(p)
4 while WorkList  $\neq \emptyset$  do
5   p  $\leftarrow$  popFromWorklist()
6   foreach o  $\in$  pts(p) do
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge
10        pushIntoWorklist(q)
11      foreach r  $\xrightarrow{\text{Load}}$  p do // Load rule
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge
14          pushIntoWorklist(o)
15      foreach x  $\in$  E do // Copy rule
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)
17        if pts(x) changed then
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```



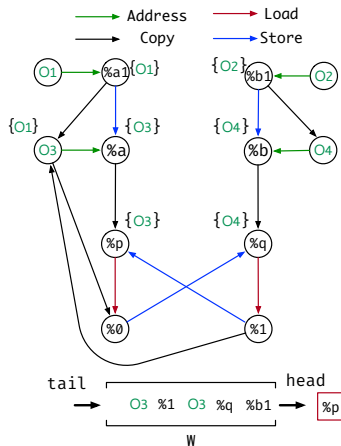
```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, align 1      // O1
%b1 = alloca i8, align 1      // O2
%a = alloca i8*, align 8      // O3
%b = alloca i8*, align 8      // O4
store i8* %a1, i8** %a, align 8
store i8* %b1, i8** %b, align 8
call void @swap(i8** %a, i8** %b)
ret i32 0
}

define void @swap(i8** %p, i8** %q)
#0 {
entry:
%0 = load i8** %p, align 8
%1 = load i8** %q, align 8
store i8* %1, i8** %p, align 8
store i8* %0, i8** %q, align 8
ret void
}
```



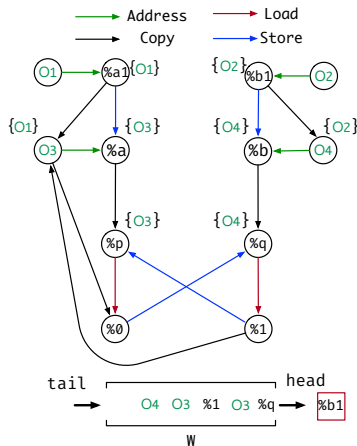
```
G = < V, E > // Constraint Graph
V: a set of nodes in graph
E: a set of edges in graph
WorkList: a vector of nodes

1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule
2   pts(p) = {o}
3   pushIntoWorklist(p)
4 while WorkList  $\neq \emptyset$  do
5   p  $\leftarrow$  popFromWorklist()
6   foreach o  $\in$  pts(p) do
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge
10        pushIntoWorklist(q)
11      foreach r  $\xrightarrow{\text{Load}}$  p do // Load rule
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge
14          pushIntoWorklist(o)
15      foreach x  $\in$  E do // Copy rule
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)
17        if pts(x) changed then
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

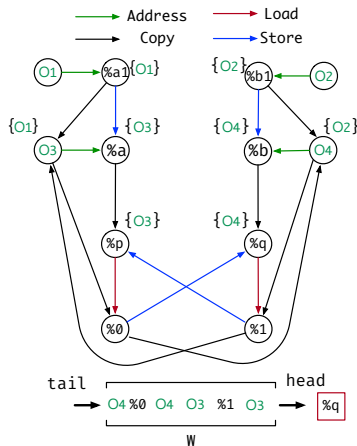


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```



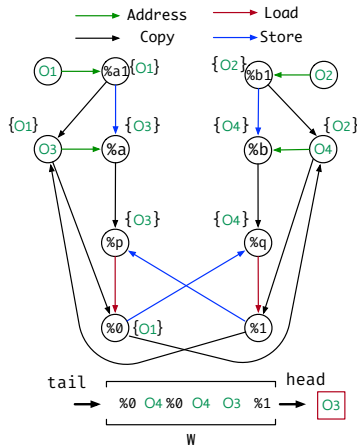
```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14        pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, align 1      // O1
%b1 = alloca i8, align 1      // O2
%a = alloca i8*, align 8      // O3
%b = alloca i8*, align 8      // O4
store i8* %a1, i8** %a, align 8
store i8* %b1, i8** %b, align 8
call void @swap(i8** %a, i8** %b)
ret i32 0
}

define void @swap(i8** %p, i8** %q)
#0 {
entry:
%0 = load i8** %p, align 8
%1 = load i8** %q, align 8
store i8* %1, i8** %p, align 8
store i8* %0, i8** %q, align 8
ret void
}
```



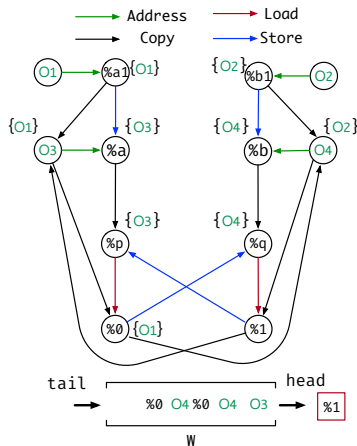
```
G = < V, E > // Constraint Graph
V: a set of nodes in graph
E: a set of edges in graph
WorkList: a vector of nodes

1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule
2   pts(p) = {o}
3   pushIntoWorklist(p)
4 while WorkList  $\neq \emptyset$  do
5   p  $\leftarrow$  popFromWorklist()
6   foreach o  $\in$  pts(p) do
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge
10        pushIntoWorklist(q)
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge
14          pushIntoWorklist(o)
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)
17        if pts(x) changed then
18          pushIntoWorklist(x)
```


Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, align 1      // O1
%b1 = alloca i8, align 1      // O2
%a = alloca i8*, align 8      // O3
%b = alloca i8*, align 8      // O4
store i8* %a1, i8** %a, align 8
store i8* %b1, i8** %b, align 8
call void @swap(i8** %a, i8** %b)
ret i32 0
}
define void @swap(i8** %p, i8** %q)
#0 {
entry:
%0 = load i8** %p, align 8
%1 = load i8** %q, align 8
store i8* %1, i8** %p, align 8
store i8* %0, i8** %q, align 8
ret void
}
```



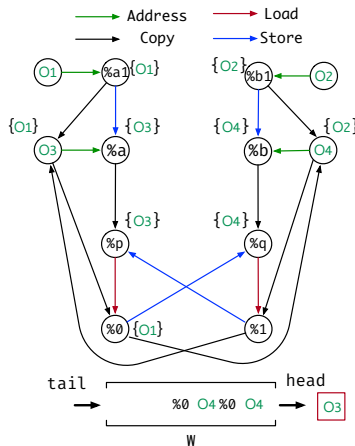
```
G = < V, E > // Constraint Graph
V: a set of nodes in graph
E: a set of edges in graph
WorkList: a vector of nodes

1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule
2   pts(p) = {o}
3   pushIntoWorklist(p)
4 while WorkList  $\neq \emptyset$  do
5   p  $\leftarrow$  popFromWorklist()
6   foreach o  $\in$  pts(p) do
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge
10        pushIntoWorklist(q)
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge
14          pushIntoWorklist(o)
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)
17        if pts(x) changed then
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

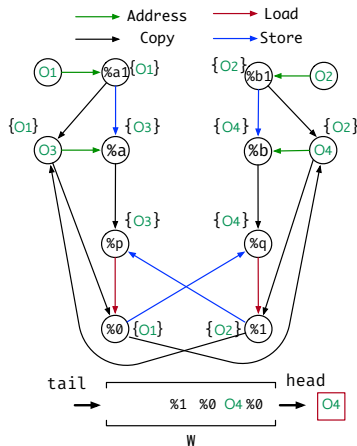


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, align 1      // O1
%b1 = alloca i8, align 1      // O2
%a = alloca i8*, align 8      // O3
%b = alloca i8*, align 8      // O4
store i8* %a1, i8** %a, align 8
store i8* %b1, i8** %b, align 8
call void @swap(i8** %a, i8** %b)
ret i32 0
}
define void @swap(i8** %p, i8** %q)
#0 {
entry:
%0 = load i8** %p, align 8
%1 = load i8** %q, align 8
store i8* %1, i8** %p, align 8
store i8* %0, i8** %q, align 8
ret void
}
```



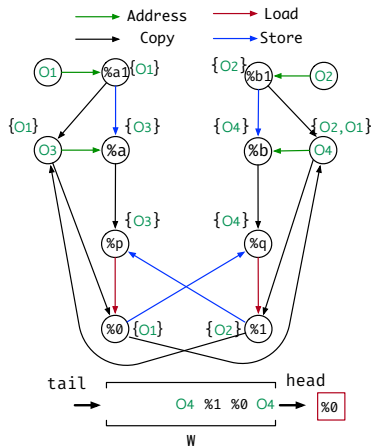
```
G = < V, E > // Constraint Graph
V: a set of nodes in graph
E: a set of edges in graph
WorkList: a vector of nodes

1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule
2   pts(p) = {o}
3   pushIntoWorklist(p)
4 while WorkList  $\neq \emptyset$  do
5   p  $\leftarrow$  popFromWorklist()
6   foreach o  $\in$  pts(p) do
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge
10        pushIntoWorklist(q)
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge
14          pushIntoWorklist(o)
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)
17        if pts(x) changed then
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

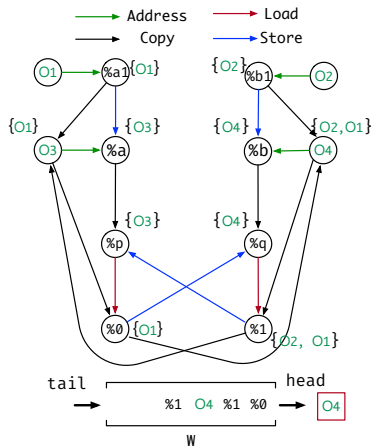


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```

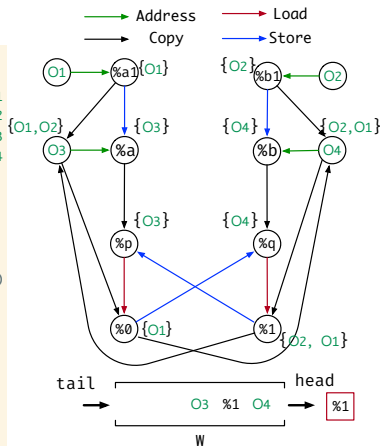


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 8  
  ret void  
}
```



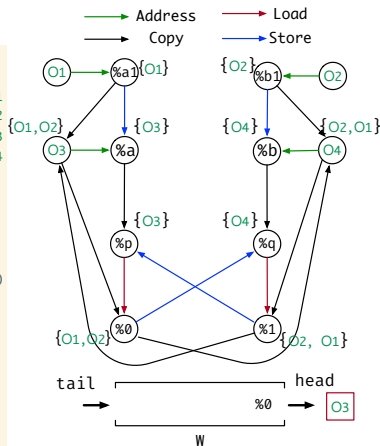
```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {
entry:
%a1 = alloca i8, align 1           // O1
%b1 = alloca i8, align 1           // O2
%a = alloca i8*, align 8           // O3
%b = alloca i8*, align 8           // O4
store i8* %a1, i8** %a, align 8
store i8* %b1, i8** %b, align 8
call void @swap(i8** %a, i8** %b)
ret i32 0
}

define void @swap(i8** %p, i8** %q)
#0 {
entry:
%0 = load i8** %p, align 8
%1 = load i8** %q, align 8
store i8* %1, i8** %p, align 8
store i8* %0, i8** %q, align 8
ret void
}
```



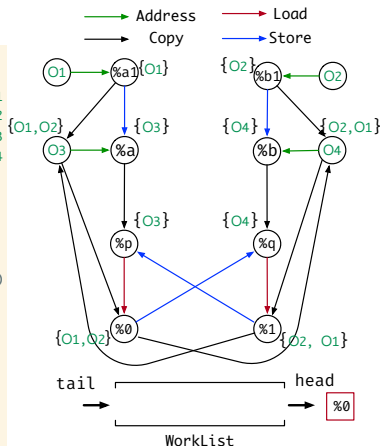
```
G = < V, E > // Constraint Graph
V: a set of nodes in graph
E: a set of edges in graph
WorkList: a vector of nodes

1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule
2   pts(p) = {o}
3   pushIntoWorklist(p)
4 while WorkList  $\neq \emptyset$  do
5   p  $\leftarrow$  popFromWorklist()
6   foreach o  $\in$  pts(p) do
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge
10        pushIntoWorklist(q)
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge
14          pushIntoWorklist(o)
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)
17        if pts(x) changed then
18          pushIntoWorklist(x)
```

Andersen's Pointer Analysis

Algorithm

```
define i32 @main() #0 {  
  entry:  
  %a1 = alloca i8, align 1      // O1  
  %b1 = alloca i8, align 1      // O2  
  %a = alloca i8*, align 8      // O3  
  %b = alloca i8*, align 8      // O4  
  store i8* %a1, i8** %a, align 8  
  store i8* %b1, i8** %b, align 8  
  call void @swap(i8** %a, i8** %b)  
  ret i32 0  
}  
  
define void @swap(i8** %p, i8** %q)  
#0 {  
  entry:  
  %0 = load i8** %p, align 8  
  %1 = load i8** %q, align 8  
  store i8* %1, i8** %p, align 8  
  store i8* %0, i8** %q, align 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  
1 foreach o  $\xrightarrow{\text{Address}}$  p do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList  $\neq \emptyset$  do  
5   p  $\leftarrow$  popFromWorklist()  
6   foreach o  $\in$  pts(p) do  
7     foreach q  $\xrightarrow{\text{Store}}$  p do // Store rule  
8       if q  $\xrightarrow{\text{Copy}}$  o  $\notin$  E then  
9         E  $\leftarrow$  E  $\cup$  {q  $\xrightarrow{\text{Copy}}$  o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach p  $\xrightarrow{\text{Load}}$  r do // Load rule  
12        if o  $\xrightarrow{\text{Copy}}$  r  $\notin$  E then  
13          E  $\leftarrow$  E  $\cup$  {o  $\xrightarrow{\text{Copy}}$  r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p  $\xrightarrow{\text{Copy}}$  x  $\in$  E do // Copy rule  
16        pts(x)  $\leftarrow$  pts(x)  $\cup$  pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```


Assignment Structure

BVDataPTAImpl



AndersenBase



AndersenPTA

- You will be working on AndersenPTA's `solveWorklist` method.

Assignment Structure

BVDataPTAImpl



AndersenBase



AndersenPTA

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

Assignment Structure

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.

Assignment Structure

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. :)