

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

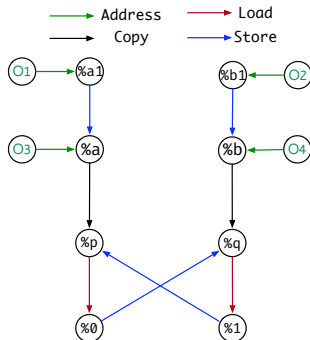
Assignment 3: Quiz + A Coding Task

- 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/SVF-Teaching/wiki/Assignment-3>
 - **SVF CPP API**
<https://github.com/SVF-tools/SVF-Teaching/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  
}
```

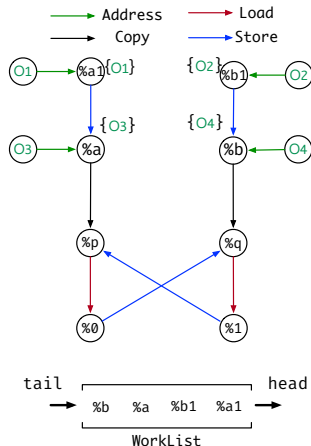


```
G = < V, E >  // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15    foreach p → x ∈ E do // Copy rule  
16      pts(x) ← pts(x) ∪ 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  
}
```

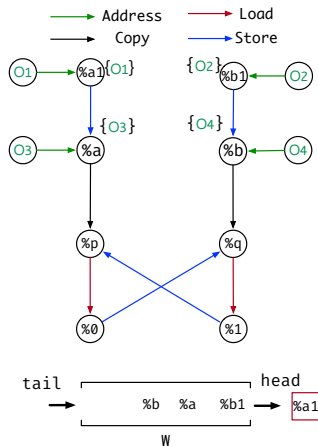


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p → x ∈ E do // Copy rule  
16        pts(x) ← pts(x) ∪ 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  
}
```

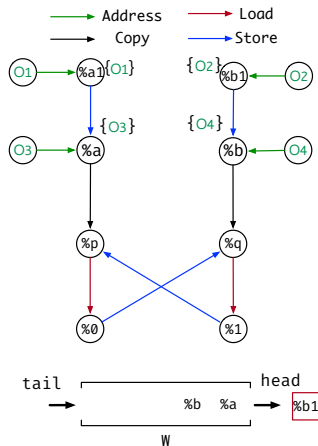


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList != ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p → x ∈ E do // Copy rule  
16        pts(x) ← pts(x) ∪ 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  
}
```

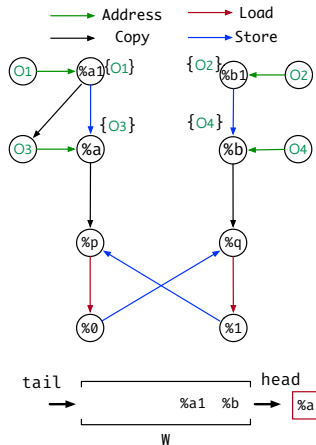


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p → x ∈ E do // Copy rule  
16        pts(x) ← pts(x) ∪ 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  
}
```

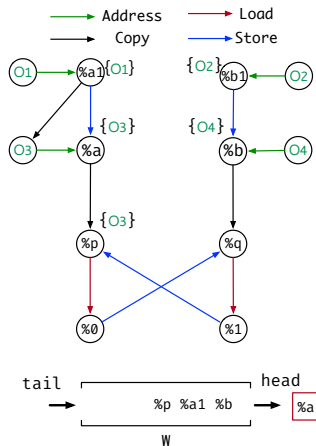


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p → x ∈ E do // Copy rule  
16        pts(x) ← pts(x) ∪ 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  
}
```

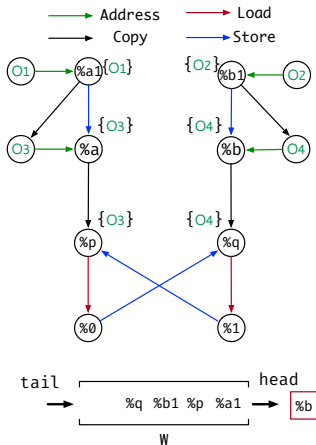


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15  foreach p → x ∈ E do // Copy rule  
16    pts(x) ← pts(x) ∪ pts(p)  
17    if pts(x) changed then  
18      pushIntoWorklist(x)
```

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

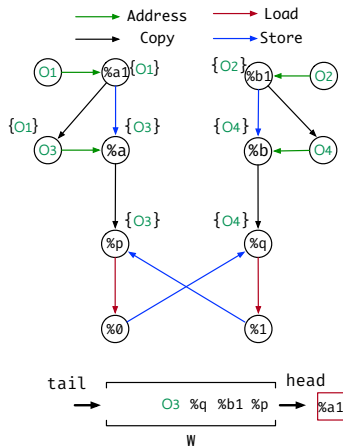
1 foreach address p = &o do // Address rule
2     pts(p) = {o}
3     pushIntoWorklist(p)
4 while Worklist ≠ ∅ do
5     p ← popFromWorklist()
6     foreach o ∈ pts(p) do
7         foreach store *p = q do // Store rule
8             if q → o ∉ E then
9                 E ← E ∪ {q → o} // Add copy edge
10                pushIntoWorklist(q)
11            foreach load r = *p do // Load rule
12                if o → r ∉ E then
13                    E ← E ∪ {o → r} // Add copy edge
14                pushIntoWorklist(o)
15        foreach p → x ∈ E do // Copy rule
16            pts(x) ← pts(x) ∪ 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  
}
```

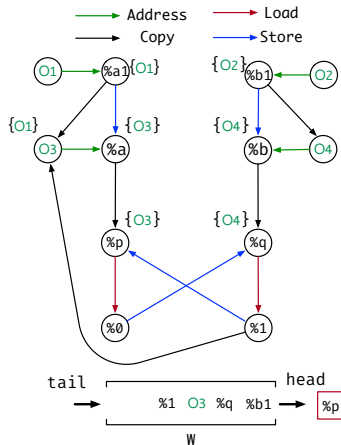


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15  foreach p → x ∈ E do // Copy rule  
16    pts(x) ← pts(x) ∪ 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  
}
```

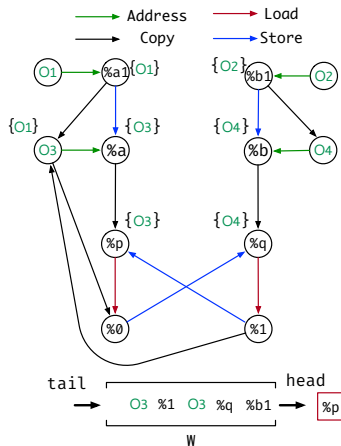


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p → x ∈ E do // Copy rule  
16        pts(x) ← pts(x) ∪ 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  
}
```

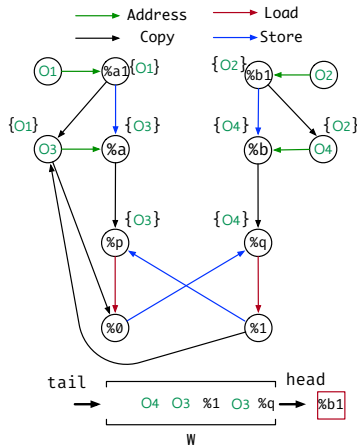


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11     foreach load r = *p do // Load rule  
12       if o → r ∉ E then  
13         E ← E ∪ {o → r} // Add copy edge  
14        pushIntoWorklist(o)  
15   foreach p → x ∈ E do // Copy rule  
16     pts(x) ← pts(x) ∪ 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  
}
```

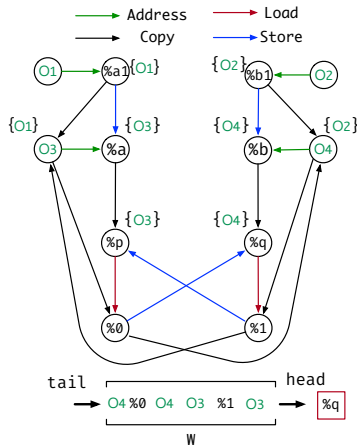


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15  foreach p → x ∈ E do // Copy rule  
16    pts(x) ← pts(x) ∪ 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  
}
```

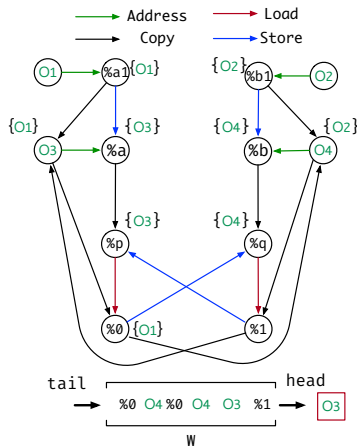


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p → x ∈ E do // Copy rule  
16        pts(x) ← pts(x) ∪ 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
}
```



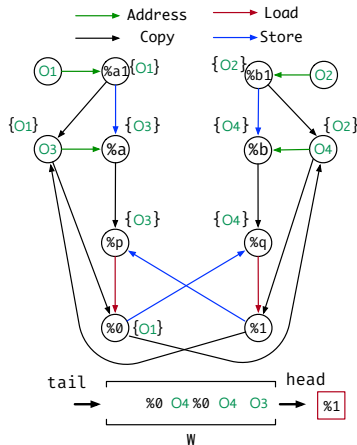
```
G = < V, E > // Constraint Graph
V: a set of nodes in graph
E: a set of edges in graph
WorkList: a vector of nodes

1 foreach address p = &o do // Address rule
2   pts(p) = {o}
3   pushIntoWorklist(p)
4 while WorkList ≠ ∅ do
5   p ← popFromWorklist()
6   foreach o ∈ pts(p) do
7     foreach store *p = q do // Store rule
8       if q → o ∉ E then
9         E ← E ∪ {q → o} // Add copy edge
10        pushIntoWorklist(q)
11     foreach load r = *p do // Load rule
12       if o → r ∉ E then
13         E ← E ∪ {o → r} // Add copy edge
14        pushIntoWorklist(o)
15  foreach p → x ∈ E do // Copy rule
16    pts(x) ← pts(x) ∪ 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  
}
```

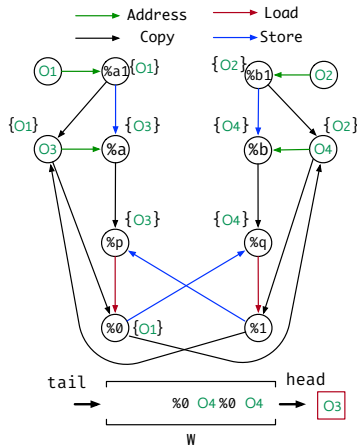


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p → x ∈ E do // Copy rule  
16        pts(x) ← pts(x) ∪ 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  
}
```

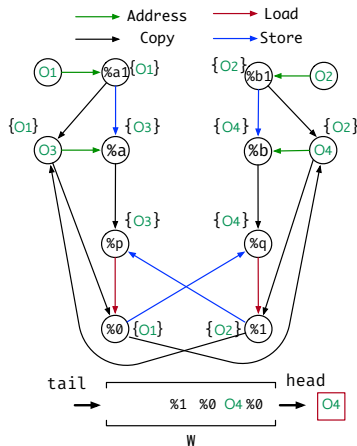


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p → x ∈ E do // Copy rule  
16        pts(x) ← pts(x) ∪ 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  
}
```

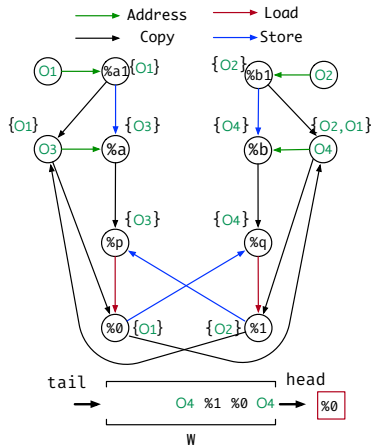


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15  foreach p → x ∈ E do // Copy rule  
16    pts(x) ← pts(x) ∪ 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  
}
```

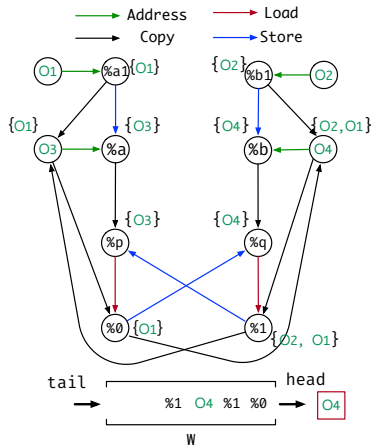


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15  foreach p → x ∈ E do // Copy rule  
16    pts(x) ← pts(x) ∪ 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
}
```



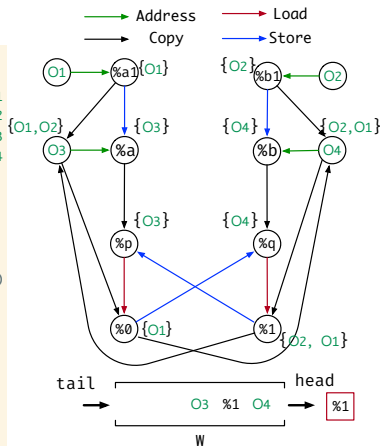
```
G = < V, E > // Constraint Graph
V: a set of nodes in graph
E: a set of edges in graph
WorkList: a vector of nodes

1 foreach address p = &o do // Address rule
2   pts(p) = {o}
3   pushIntoWorklist(p)
4 while WorkList ≠ ∅ do
5   p ← popFromWorklist()
6   foreach o ∈ pts(p) do
7     foreach store *p = q do // Store rule
8       if q → o ∉ E then
9         E ← E ∪ {q → o} // Add copy edge
10        pushIntoWorklist(q)
11     foreach load r = *p do // Load rule
12       if o → r ∉ E then
13         E ← E ∪ {o → r} // Add copy edge
14        pushIntoWorklist(o)
15  foreach p → x ∈ E do // Copy rule
16    pts(x) ← pts(x) ∪ 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  
}
```

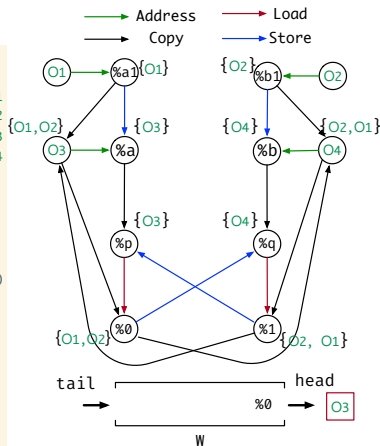


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15  foreach p → x ∈ E do // Copy rule  
16    pts(x) ← pts(x) ∪ 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  
}
```

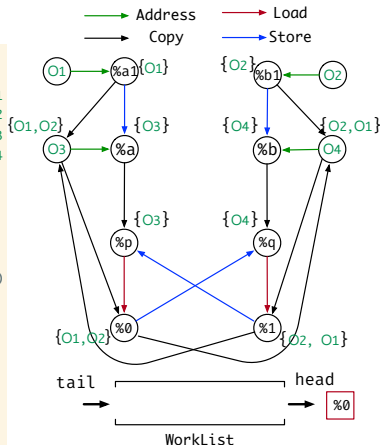


```
G = < V, E > // Constraint Graph  
V: a set of nodes in graph  
E: a set of edges in graph  
WorkList: a vector of nodes  
1 foreach address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11     foreach load r = *p do // Load rule  
12       if o → r ∉ E then  
13         E ← E ∪ {o → r} // Add copy edge  
14        pushIntoWorklist(o)  
15 foreach p → x ∈ E do // Copy rule  
16   pts(x) ← pts(x) ∪ 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 address p = &o do // Address rule  
2   pts(p) = {o}  
3   pushIntoWorklist(p)  
4 while WorkList ≠ ∅ do  
5   p ← popFromWorklist()  
6   foreach o ∈ pts(p) do  
7     foreach store *p = q do // Store rule  
8       if q → o ∉ E then  
9         E ← E ∪ {q → o} // Add copy edge  
10        pushIntoWorklist(q)  
11      foreach load r = *p do // Load rule  
12        if o → r ∉ E then  
13          E ← E ∪ {o → r} // Add copy edge  
14          pushIntoWorklist(o)  
15      foreach p → x ∈ E do // Copy rule  
16        pts(x) ← pts(x) ∪ pts(p)  
17        if pts(x) changed then  
18          pushIntoWorklist(x)
```