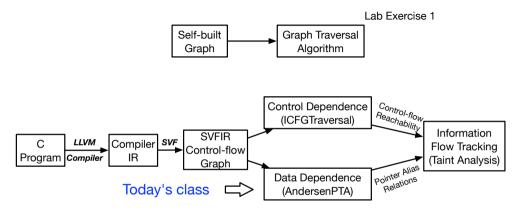
Data-Flow and Taint Analysis

(Week 3)

Yulei Sui

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

Today's class



Revisiting Andersen's Analysis

- Top-level variables, whose addresses are not taken (ValPN in SVF)
 - Including stack virtual registers (symbols starting with "%") and global variables (symbols starting with "@") are explicit, i.e., directly accessed.

- Top-level variables, whose addresses are not taken (ValPN in SVF)
 - Including stack virtual registers (symbols starting with "%") and global variables (symbols starting with "@") are explicit, i.e., directly accessed.
 - Def-use for top-level variables are directly available from LLVM's SSA form.
 - For example, def-use for %a1 from Instruction-1 to Instruction-2.
 - Instruction-1: %a1 = alloca i8, align 1;
 - Instruction-2: store ptr %a1, ptr %a, align 8

- Top-level variables, whose addresses are not taken (ValPN in SVF)
 - Including stack virtual registers (symbols starting with "%") and global variables (symbols starting with "@") are explicit, i.e., directly accessed.
 - Def-use for top-level variables are directly available from LLVM's SSA form.
 - For example, def-use for %a1 from Instruction-1 to Instruction-2.
 - Instruction-1: %a1 = alloca i8, align 1;
 - Instruction-2: store ptr %a1, ptr %a, align 8
- Address-taken variables (abstract objects), accessed indirectly at load or store instructions via top-level variables (ObjPN in SVF)
 - A stack object created at an LLVM's 'alloca' instruction or a heap object created via (e.g., 'malloc' callsite) or a global object.

- Top-level variables, whose addresses are not taken (ValPN in SVF)
 - Including stack virtual registers (symbols starting with "%") and global variables (symbols starting with "@") are explicit, i.e., directly accessed.
 - Def-use for top-level variables are directly available from LLVM's SSA form.
 - For example, def-use for %a1 from Instruction-1 to Instruction-2.
 - Instruction-1: %a1 = alloca i8, align 1;
 - Instruction-2: store ptr %a1, ptr %a, align 8
- Address-taken variables (abstract objects), accessed indirectly at load or store instructions via top-level variables (ObjPN in SVF)
 - A stack object created at an LLVM's 'alloca' instruction or a heap object created via (e.g., 'malloc' callsite) or a global object.
 - Def-use for address-taken variables are computed via pointer analysis.
 - For example, there is a def-use for object o from Instruction-1 to Instruction-2 if pointers %a and %b both point to o.
 - Instruction-1: store ptr %a1, ptr %a, align 8
 - Instruction-2: %c = load ptr %b, align 8

Pointer Analysis (Revisit Andersen's Analysis in Lab-Exercise-1)

A typical data-flow analysis

- Points-to Analysis: aims to statically determine the possible runtime values
 of a pointer at compile-time.
 - Compute the points-to set (a set of address-taken variables) of each pointer (top-level variable)
 - For example, p = &a; q = p;
 - The resulting points-to sets of p and q are: $pts(p) = pts(q) = \{a\}$

Pointer Analysis (Revisit Andersen's Analysis in Lab-Exercise-1)

A typical data-flow analysis

- Points-to Analysis: aims to statically determine the possible runtime values of a pointer at compile-time.
 - Compute the points-to set (a set of address-taken variables) of each pointer (top-level variable)
 - For example, p = &a; q = p;
 - The resulting points-to sets of p and q are: pts(p) = pts(q) = {a}
- Alias Analysis: determines whether two pointer dereferences refer to the same memory location.
 - If the points-to sets of two pointers p and q have overlapping elements (i.e., pts(p) ∩ pts(q) ≠ ∅) then p and q are aliases. The derereferences of p and q may refer to the same memory location.

Why shall we learn pointer analysis?

 Essential for building data-dependence relations between variables (memory objects).

- Essential for building data-dependence relations between variables (memory objects).
- Understanding aliases through different memory accesses

- Essential for building data-dependence relations between variables (memory objects).
- Understanding aliases through different memory accesses
 - p = &a; q = p; *p = x; y = *q; y has the same value as x since *p and *q both always refer to a.

- Essential for building data-dependence relations between variables (memory objects).
- Understanding aliases through different memory accesses

```
    p = &a; q = p; *p = x; y = *q;
    y has the same value as x since *p and *q both always refer to a.
```

- Compiler optimizations and bug detection
 - Constant propagation
 - *p = 1; x = *q;
 x is a constant value and equals 1, if p and q are must-aliases (always point to the same memory location w.r.t every execution path).
 - *p = 1; *q = r; x = *p;
 x is a constant value and equals 1, if p and q do not alias with each other.

- Essential for building data-dependence relations between variables (memory objects).
- Understanding aliases through different memory accesses

```
    p = &a; q = p; *p = x; y = *q;
    y has the same value as x since *p and *q both always refer to a.
```

- Compiler optimizations and bug detection
 - Constant propagation
 - *p = 1; x = *q;
 x is a constant value and equals 1, if p and q are must-aliases (always point to the same memory location w.r.t every execution path).
 - *p = 1; *q = r; x = *p;
 x is a constant value and equals 1, if p and q do not alias with each other.
 - Taint analysis
 - *p = taintedInput; x = *q;
 x is tainted if p and q are aliases.

Precision Dimensions

Can be generally classified into the following precision dimensions at different levels of abstractions.

Flow-insensitive analysis:

- Ignores program execution order
- A single solution across whole program

Context-insensitive analysis:

 Merges all calling contexts when analysing a program method

Path-insensitive analysis:

 Merges all incoming path information at the join points of the control-flow graph

Flow-sensitive analysis:

- Respects the program execution order
- Separate solution at each program point

Context-sensitive analysis:

 Distinguishes between different calling contexts of a program method

Path-sensitive analysis:

Computes a solution per (abstract) program path.

Precision Dimensions

Levels of Abstractions

Assume x is a tainted value

$$p = x$$

$$p = y$$

flow-sensitivity

at which program point p is tainted?

C

Assume A is a tainted v

context-sensitivity

under which calling context p is tainted?

p = xelse p = y

path-sensitivity

along which program path p is tainted?

Andersen's Pointer Analysis

A flow-insensitive, context-insensitive and path-insensitive points-to analysis to determine points-to set of a pointer by analyzing the Constraint Graph or Program Assignment Graph (PAG) of a program.

Andersen's Pointer Analysis

A flow-insensitive, context-insensitive and path-insensitive points-to analysis to determine points-to set of a pointer by analyzing the Constraint Graph or Program Assignment Graph (PAG) of a program.

- Also known as inclusion-based points-to analysis, the most popular and widely used pointer analysis.
- Solving constraint edges between ConstraintNodes (SVFVars, which are either pointer types or objects).
- The analysis requires iterative solving of the ConstraintGraph by (1) propagating points-to sets among graph nodes, and (2) adding new edges until a fixed point is reached, i.e., no new edges are added and no points-to sets change. (Lab-Exercise-1)

Andersen, L. O. (1994). Program analysis and specialization for the C programming language (Doctoral dissertation, University of Cophenhagen).

The Ant and the Grasshopper: Fast and Accurate Pointer Analysis for Millions of Lines of Code, PLDI 2007

The analysis operating upon a program's constraint graph which is a subgraph of PAG (program assignment graph).

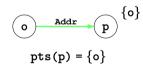
- ConstraintNode represents
 - A pointer (ValVar): (top-level variable) or
 - An object (ObjVar): (address-taken objects, i.e., heap/stack/global/function objs)
- ConstraintEdge represents a constraint between two nodes

The analysis operating upon a program's constraint graph which is a subgraph of PAG (program assignment graph).

- ConstraintNode represents
 - A pointer (ValVar): (top-level variable) or
 - An object (ObjVar): (address-taken objects, i.e., heap/stack/global/function objs)
- ConstraintEdge represents a constraint between two nodes

Constraint Edge	LLVM IR	C code	Constraint rules
$p \xleftarrow{\text{Addr}} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	%q = bitcast %p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\texttt{Load}} p$	%q = load %p	q = *p	$\forall o \in \mathtt{pts}(\mathtt{p}) : add \; edge \; q \xleftarrow{\mathtt{Copy}} o$
$p \xleftarrow{\mathtt{Store}} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(p) : add \; edge \; o \xleftarrow{\mathtt{Copy}} \; q$
$q \xleftarrow{Gep,fld} p$	%q = gep %p, fld	$\mathtt{q} = \mathtt{\&p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$

Constraint Edge	LLVM IR	C code	Constraint rules
$p \xleftarrow{\texttt{Addr}} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	%q = bitcast %p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\text{Load}} p$	%q = load %p	q = *p	$\forall o \in \mathtt{pts}(p) : add \; edge \; q \xleftarrow{Copy} \; o$
$p \xleftarrow{\text{Store}} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(p) : add \; edge \; o \xleftarrow{Copy} q$
$q \xleftarrow{\texttt{Gep,fld}} p$	%q = gep %p, fld	$\mathtt{q} = \&\mathtt{p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$



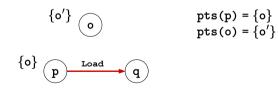
Constraint Edge	LLVM IR	C code	Constraint rules
$p \stackrel{\mathtt{Addr}}{\longleftarrow} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	%q = bitcast %p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\text{Load}} p$	%q = load %p	$\mathtt{q}=\ast\mathtt{p}$	$\forall o \in \mathtt{pts}(p) : add \; edge \; q \xleftarrow{copy} \; o$
$p \xleftarrow{\mathtt{Store}} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(p) : add \; edge \; o \xleftarrow{Copy} q$
$q \xleftarrow{\texttt{Gep}, \texttt{fld}} p$	%q = gep %p, fld	$\mathtt{q} = \&\mathtt{p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$

$$\begin{cases}
o \\
p
\end{cases}
\xrightarrow{\text{copy}} q$$

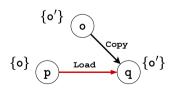
$$pts(p) = \{o\}$$

Constraint Edge	LLVM IR	C code	Constraint rules
$p \stackrel{\mathtt{Addr}}{\longleftarrow} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	%q = bitcast %p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\text{Load}} p$	%q = load %p	$\mathtt{q}=\ast\mathtt{p}$	$\forall o \in \mathtt{pts}(p) : add \; edge \; q \xleftarrow{copy} \; o$
$p \xleftarrow{\mathtt{Store}} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(p) : add \; edge \; o \xleftarrow{Copy} q$
$q \xleftarrow{\texttt{Gep}, \texttt{fld}} p$	%q = gep %p, fld	$\mathtt{q} = \&\mathtt{p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$

Constraint Edge	LLVM IR	C code	Constraint rules
$p \stackrel{\mathtt{Addr}}{\longleftarrow} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	%q = bitcast %p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\texttt{Load}} p$	%q = load %p	q = *p	$\forall o \in \mathtt{pts}(\mathtt{p}) : add \; edge \; q \xleftarrow{Copy} o$
$p \stackrel{\mathtt{Store}}{\longleftarrow} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(p) : add \; edge \; o \xleftarrow{Copy} \; q$
$q \xleftarrow{\texttt{Gep,fld}} p$	%q = gep %p, fld	$\mathtt{q} = \&\mathtt{p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$

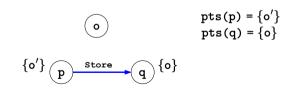


Constraint Edge	LLVM IR	C code	Constraint rules
$p \stackrel{\mathtt{Addr}}{\longleftarrow} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	% q = bitcast % p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\texttt{Load}} p$	%q = load %p	q = *p	$\forall o \in \mathtt{pts}(\mathtt{p}) : add \; edge \; q \xleftarrow{Copy} o$
$p \stackrel{\mathtt{Store}}{\longleftarrow} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(p) : add \; edge \; o \xleftarrow{Copy} q$
$q \xleftarrow{\texttt{Gep}, \texttt{fld}} p$	%q = gep %p, fld	$\mathtt{q} = \&\mathtt{p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$

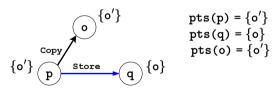


$$pts(p) = \{o\}$$
$$pts(o) = \{o'\}$$
$$pts(q) = \{o'\}$$

Constraint Edge	LLVM IR	C code	Constraint rules
$p \stackrel{\mathtt{Addr}}{\longleftarrow} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	% q = bitcast % p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\texttt{Load}} p$	%q = load %p	$\mathtt{q}=\ast\mathtt{p}$	$\forall o \in \mathtt{pts}(\mathtt{p}) : add \; edge \; q \xleftarrow{\mathtt{Copy}} o$
$p \xleftarrow{\mathtt{Store}} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(\mathtt{p}) : add \; edge \; o \xleftarrow{\mathtt{Copy}} \mathtt{q}$
$q \xleftarrow{\texttt{Gep}, \texttt{fld}} p$	%q = gep %p, fld	$\mathtt{q} = \&\mathtt{p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$



Constraint Edge	LLVM IR	C code	Constraint rules
$p \stackrel{\mathtt{Addr}}{\longleftarrow} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	%q = bitcast %p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\texttt{Load}} p$	%q = load %p	$\mathtt{q}=\ast\mathtt{p}$	$\forall o \in \mathtt{pts}(p) : add \; edge \; q \xleftarrow{Copy} \; o$
$p \stackrel{\text{Store}}{\longleftarrow} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(p) : add \; edge \; o \xleftarrow{Copy} q$
$q \xleftarrow{\texttt{Gep,fld}} p$	%q = gep %p, fld	$\mathtt{q} = \mathtt{\&p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$



$$pts(p) = \{o'\}$$

 $pts(q) = \{o\}$
 $pts(o) = \{o'\}$

Constraint Edge	LLVM IR	C code	Constraint rules
$p \stackrel{\mathtt{Addr}}{\longleftarrow} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	%q = bitcast %p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\text{Load}} p$	%q = load %p	q = *p	$\forall o \in \mathtt{pts}(\mathtt{p}) : add \; edge \; \mathtt{q} \xleftarrow{\mathtt{Copy}} o$
$p \stackrel{\mathtt{Store}}{\longleftarrow} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(\mathtt{p}) : add \; edge \; o \xleftarrow{\mathtt{Copy}} \mathtt{q}$
$q \xleftarrow{\texttt{Gep,fld}} p$	%q = gep %p, fld	$\mathtt{q} = \mathtt{\&p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$

$$\begin{cases}
o \\
p
\end{cases}
\xrightarrow{\text{Gep,fld}} q$$

$$pts(p) = \{o\}$$

Constraint Edge	LLVM IR	C code	Constraint rules
$p \stackrel{\mathtt{Addr}}{\longleftarrow} o$	%p = alloca _o	p = &o	$\mathtt{pts}(\mathtt{p}) = \mathtt{pts}(\mathtt{p}) \cup \{\mathtt{o}\}$
$q \xleftarrow{\texttt{Copy}} p$	%q = bitcast %p	q = p	$\mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \mathtt{pts}(\mathtt{p})$
$q \xleftarrow{\text{Load}} p$	%q = load %p	q = *p	$\forall o \in \mathtt{pts}(p) : add \; edge \; q \xleftarrow{Copy} o$
$p \stackrel{\mathtt{Store}}{\longleftarrow} q$	store %q, %p	*p = q	$\forall o \in \mathtt{pts}(\mathtt{p}) : add \; edge \; o \xleftarrow{Copy} \; q$
$q \xleftarrow{\texttt{Gep,fld}} p$	%q = gep %p, fld	$\mathtt{q} = \mathtt{\&p} \!\to\! \mathtt{fld}$	$\forall \mathtt{o} \in \mathtt{pts}(\mathtt{p}) : \mathtt{pts}(\mathtt{q}) = \mathtt{pts}(\mathtt{q}) \cup \{\mathtt{o.fld}\}$

Constraint Solving Algorithm for Andersen's Analysis

- A worklist holds a list of constraint graph nodes for iterative processing
 - Initialize the points-to set of the destination node of each address edge. Initialize the worklist with nodes that have incoming address edges.
 - Pop a node p from the worklist.
 - Handle each incoming store edge and each outgoing load edge of node p by adding copy edges.
 - Handle each outgoing copy edge of p by propagating points-to information.
 - A node is pushed into the worklist if (1) its points-to set changes or (2) it is a source node of a new copy edge added to the graph.
- Any new copy edge added needs to be resolved and performs points-to propagation. New points-to sets discovered may trigger introducing new copy edges via load and store edges. The constraint solving should converge to a fixed point, where no new edges are added, and no points-to sets change.

Compiling a C Program to Its LLVM IR

```
void swap(char ***p, char ***q){
   char* t = *p;
   *p = *q;
   *q = t;
}
int main(){
   char a1;
   char *a;
   char b1;
   char *b;
   a = &a1;
   b = &b1;
   swap(&a,&b);
}
Compile
```

swap.c

```
define void @swap(ptr %p, ptr %g) #0 {
entry:
 %0 = load ptr. ptr %p, align 8
 %1 = load ptr, ptr %q, align 8
 store ptr %1, ptr %p, align 8
 store ptr %0, ptr %g, align 8
  ret void
define i32 @main() #0 {
entry:
 %a1 = alloca i8, align 1
 %a = alloca ptr. align 8
 %b1 = alloca i8, align 1
 %b = alloca ptr, align 8
 store ptr %al. ptr %a. align 8
 store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
```

swap.ll

*https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVFIR#2-llvm-ir-generation

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

https://github.com/svf-tools/SVF/wiki/Analyze-a-Simple-C-Program#5-pag

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

^{*}https://github.com/svf-tools/SVF/wiki/Analyze-a-Simple-C-Program#5-pag

→ Load

Address

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

https://github.com/svf-tools/SVF/wiki/Analyze-a-Simple-C-Program#5-pag

```
Address
                                                                             ► I oad
                                                                            → Store
                                                             Copy
define i32 @main() #0 {
                                                                        {02}
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                        {04}
                                                            {O3}
                                      // O3
  %b1 = alloca i8, align 1
                                                                                     (04)
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
  store ptr %0, ptr %g, align 8
  ret void
```

```
Algorithm 1: 1 Andersen's Pointer Analysis
  Input: G =< V, E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                              // Address rule
      pts(p) = o:
      pushIntoWorklist(p):
s while WorkList + @ do
      p := popFromWorklist();
      foreach o ∈ nts(n) do
          foreach ostoren e F do
                                                // Store rule
8
              if q <sup>Copy</sup> o ∉ E then
                  E := E \sqcup \{a \xrightarrow{Copy} a\}:
10
                                            // Add copy edge
                  nushIntoWorklist(a):
11
          foreach p<sup>Load</sup>r ∈ F do
                                                 // Load rule
12
              if o copy r ∉ E then
                  E := E \cup \{o \xrightarrow{Copy} r\}:
                                            // Add copy edge
14
                  pushIntoWorklist(o):
15
      foreach p Copy x ∈ F do
16
                                                 // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
          if pts(x) changed then
18
             pushIntoWorklist(x);
      foreach p \xrightarrow{Gep,f1d} x \in E do
                                                  // Gep rule
20
21
          foreach o E pts(p) do
            pts(x) := pts(x) \cup \{o.fld\};
23
          if nto(x) changed then
              pushIntoWorklist(x):
24
```

https://github.com/svf-tools/SVF/wiki/Analyze-a-Simple-C-Program#5-pag

Andersen's Pointer Analysis

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

```
Algorithm 2: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o → p do
                                                // Address rule
       pts(p) := pts(p) \cup \{o\}:
      pushIntoWorklist(p):
5 while WorkList ≠ Ø do
       p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if a copy o ∉ E then
                   E := E \cup \{q \xrightarrow{Copy} o\}:
                                               // Add copy edge
10
                   pushIntoWorklist(a):
11
          foreach p \xrightarrow{Load} r \in E do
                                                    // Load rule
12
              if o copy r # E then
13
                   E := E \cup \{o \xrightarrow{Copy} r\};
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in F do
16
                                                    // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
21
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

Andersen's Pointer Analysis

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

```
Algorithm 3: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address n do
                                                // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p):
5 while WorkList ≠ Ø do
       p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if a copy o ∉ E then
                   E := E \cup \{q \xrightarrow{Copy} o\}:
                                               // Add copy edge
10
                   pushIntoWorklist(a):
11
          foreach p \xrightarrow{Load} r \in E do
                                                    // Load rule
12
              if o copy r # E then
13
                   E := E \cup \{o \xrightarrow{Copy} r\};
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in F do
16
                                                    // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

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

```
Algorithm 4: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address n do
                                                // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p):
5 while WorkList ≠ Ø do
       p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if a copy o ∉ E then
                   E := E \cup \{q \xrightarrow{Copy} o\}:
                                               // Add copy edge
10
                   pushIntoWorklist(a):
11
          foreach p \xrightarrow{Load} r \in E do
                                                    // Load rule
12
              if o copy r # E then
13
                   E := E \cup \{o \xrightarrow{Copy} r\};
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in F do
16
                                                    // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

```
Address
                                                                          Load
                                                           Copy
                                                                          Store
                                                                     {O2}
define i32 @main() #0 {
entry:
  %a1 = alloca i8, align 1
                                                         {O3}
                                                                      {04}
  %a = alloca ptr. align 8
                                     // 03
  %b1 = alloca i8, align 1
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
  ret i32 0
define void @swap(ptr %p, ptr %a) #0 {
entry:
  %0 = load ptr, ptr %p, align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                head
  store ptr %0, ptr %g, align 8
  ret void
                                                              World ist
```

```
Algorithm 5: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address n do
                                                // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p):
5 while WorkList ≠ Ø do
       p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach g \xrightarrow{\text{store}} p \in E do
                                                  // Store rule
              if a copy o ∉ E then
9
                   E := E \cup \{q \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(q);
11
          foreach p \xrightarrow{Load} r \in E do
                                                    // Load rule
12
              if o copy r # E then
13
                   E := E \cup \{o \xrightarrow{Copy} r\};
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in F do
16
                                                    // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

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

```
Algorithm 6: 1 Andersen's Pointer Analysis
   Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
          E: a set of edges in graph
 1 WorkList := an empty vector of nodes:
 2 foreach o Address p do
                                               // Address rule
      pts(p) := pts(p) \sqcup \{o\}:
      pushIntoWorklist(p);
 s while WorkList # @ do
       n := nonFromWorklist():
       foreach o e pts(p) do
          foreach getore p ∈ F do
                                                 // Store rule
               if q \xrightarrow{Copy} o \notin E then
                   E := E \sqcup \{a \xrightarrow{Copy} o\}:
                                             // Add copy edge
10
                   pushIntoWorklist(g):
           foreach p \xrightarrow{Load} r \in E do
12
                                                   // Load rule
              if o Copy r & E then
                   E := E \cup \{o \xrightarrow{Copy} r\};
                                             // Add copy edge
14
15
                   pushIntoWorklist(o):
       foreach p \xrightarrow{Copy} x \in E do
                                                  // Copy rule
16
           pts(x) := pts(x) \cup pts(p):
           if pts(x) changed then
18
           | pushIntoWorklist(x);
19
       foreach p^{Gep,fld} x \in F do
                                                   // Gep rule
20
21
          foreach o e pts(p) do
22
            | pts(x) := pts(x) ∪ {o.fld};
           if pts(x) changed then
23
              pushIntoWorklist(x):
```

```
Address
                                                                          Load
                                                           Copy
                                                                          → Store
define i32 @main() #0 {
entry:
                                      // 01
  %a1 = alloca i8, align 1
                                                         {O3}
                                                                      {04}
                                      // O(201)
  %a = alloca ptr. align 8
                                      // O3
  %b1 = alloca i8, align 1
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
                                                          {O3}
                                                                      {04}
  call void @swap(ptr %a, ptr %b)
  ret i32 0
define void @swap(ptr %p, ptr %a) #0 {
entry:
  %0 = load ptr, ptr %p, align 8
  %1 = load ptr. ptr %q, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                head
  store ptr %0, ptr %g, align 8
                                                              O3 %a %b1 %p
  ret void
                                                              Worklist
```

```
Algorithm 7: 1 Andersen's Pointer Analysis
 Input: G =< V.E >: Constraint Graph
         V: a set of nodes in graph
         E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address n do
                                              // Address rule
     pts(p) := pts(p) U {o}:
     pushIntoWorklist(p):
5 while WorkList ≠ Ø do
      p := popFromWorklist():
      foreach o ∈ pts(p) do
         foreach a \xrightarrow{\text{Store}} p \in E do
                                                // Store rule
             if a copy o ∉ E then
                  E := E \cup \{q \xrightarrow{Copy} o\}:
                                            // Add copy edge
                 pushIntoWorklist(a):
         foreach p \xrightarrow{Load} r \in E do
                                                 // Load rule
             if o copy r # E then
                  E := E \cup \{o \xrightarrow{Copy} r\}:
                                            // Add copy edge
                 pushIntoWorklist(o);
      foreach p \xrightarrow{Copy} x \in F do
                                                 // Copy rule
         pts(x) := pts(x) \cup pts(p);
          if pts(x) changed then
             pushIntoWorklist(x):
      foreach p Gep,fld y ∈ F do
                                                  // Gep rule
         foreach o ∈ pts(p) do
             pts(x) := pts(x) \cup \{o.fld\};
         if pts(x) changed then
             pushIntoWorklist(x);
```

10

11

12

13

14

15

16

17

18

19

20

22

23

24

```
Address
                                                                              Load
                                                            Copy
                                                                            Store
define i32 @main() #0 {
                                                                       {O2}
                                              (01)
entry:
  %a1 = alloca i8, align 1
                                             {01}
  %a = alloca ptr. align 8
                                                          {O3}
                                                                       (04)
                                     // O3
  %b1 = alloca i8, align 1
                                     // 04
                                              (03
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                           {O3}
                                                                       {04}
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                       (%0)
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
                                                               %1 O3 %g %h
  ret void
                                                                   W
```

```
Algorithm 8: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p):
5 while WorkList ≠ Ø do
       p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach g \xrightarrow{\text{store}} p \in E do
                                                  // Store rule
              if a copy o ∉ E then
9
                   E := E \cup \{q \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                   pushIntoWorklist(q);
11
          foreach p \xrightarrow{Load} r \in E do
                                                    // Load rule
12
              if o copy r # E then
13
                   E := E \cup \{o \xrightarrow{Copy} r\};
                                              // Add copy edge
14
                   pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in F do
16
                                                    // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                     // Gep rule
20
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

```
Address
                                                                             Load
                                                            Copy
                                                                           Store
define i32 @main() #0 {
                                                                      {O2}
                                             (01
entry:
  %a1 = alloca i8, align 1
                                     // O2,
  %a = alloca ptr. align 8
                                                                       (04)
                                    // 03{01}
                                                          {O3}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                          {O3}
                                                                       {04}
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                       (%n)
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                head
  store ptr %0, ptr %q, align 8
                                                           O3 %1 O3 %g %b
  ret void
                                                                  W
```

```
Algorithm 9: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                               // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p):
5 while WorkList ≠ Ø do
      p := popFromWorklist():
      foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if a copy o ∉ E then
                  E := E \cup \{q \xrightarrow{Copy} o\}:
                                              // Add copy edge
10
                  pushIntoWorklist(q);
11
          foreach p \xrightarrow{Load} r \in E do
                                                   // Load rule
12
              if o copy r & E then
13
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                             // Add copy edge
14
                  pushIntoWorklist(o);
15
      foreach p Copy x ∈ F do
16
                                                   // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
      foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                    // Gep rule
20
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

```
Address
                                                                             Load
                                                            Copy
                                                                           Store
define i32 @main() #0 {
                                                                      {O2}
                                              (01
entry:
  %a1 = alloca i8, align 1
                                     // O2,
  %a = alloca ptr. align 8
                                                                       (04)
                                    // 03{01}
                                                          {O3}
                                                                                     {O2}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                          {O3}
                                                                       {04}
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                       (%n)
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                head
  store ptr %0, ptr %q, align 8
                                                            O4 O3 %1 O3 %a
  ret void
                                                                  W
```

```
Algorithm 10: 1 Andersen's Pointer Analysis
 Input: G =< V.E >: Constraint Graph
         V: a set of nodes in graph
         E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                              // Address rule
     pts(p) := pts(p) U {o}:
     pushIntoWorklist(p);
5 while WorkList ≠ Ø do
      p := popFromWorklist():
      foreach o ∈ pts(p) do
         foreach a \xrightarrow{\text{Store}} p \in E do
                                                // Store rule
             if a copy o ∉ E then
                 E := E \cup \{q \xrightarrow{Copy} o\}:
                                            // Add copy edge
                 pushIntoWorklist(a):
         foreach p \xrightarrow{Load} r \in E do
                                                  // Load rule
             if o copy r # E then
                  E := E \sqcup \{o \xrightarrow{Copy} r\}:
                                            // Add copy edge
                 pushIntoWorklist(o);
      foreach p \xrightarrow{Copy} x \in F do
                                                 // Copy rule
         pts(x) := pts(x) \cup pts(p);
          if pts(x) changed then
             pushIntoWorklist(x):
      foreach p Gep,fld y ∈ F do
                                                  // Gep rule
         foreach o ∈ pts(p) do
             pts(x) := pts(x) \cup \{o.fld\};
         if pts(x) changed then
             pushIntoWorklist(x);
```

10

11

12

13

14

15

16

17

18

19

20

22

23

24

```
Address
                                                                              Load
                                                            Copy
                                                                           Store
define i32 @main() #0 {
                                                                      {O2}
                                              (01)
entry:
  %a1 = alloca i8, align 1
                                     // O2,
  %a = alloca ptr. align 8
                                                                       (04)
                                    // 03{01}
                                                          {O3}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                          {O3}
                                                                       {04}
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
  %0 = load ptr. ptr %p. align 8
                                                       (%0)
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                head
  store ptr %0, ptr %q, align 8
                                                       O4%0 O4 O3 %1 O3
  ret void
                                                                  W
```

```
Algorithm 11: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                               // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p);
5 while WorkList ≠ Ø do
      p := popFromWorklist():
      foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if a copy o ∉ E then
                   E := E \cup \{q \xrightarrow{Copy} o\};
                                             // Add copy edge
10
                  pushIntoWorklist(q);
11
          foreach p \xrightarrow{Load} r \in E do
                                                   // Load rule
12
              if o copy r & E then
13
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                             // Add copy edge
14
                  pushIntoWorklist(o);
15
      foreach p Copy x ∈ F do
16
                                                   // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
      foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                    // Gep rule
20
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

```
Address
                                                                              Load
                                                            Copy
                                                                           Store
define i32 @main() #0 {
                                                                      {O2}
                                              (01)
entry:
  %a1 = alloca i8, align 1
                                     // O2,
  %a = alloca ptr. align 8
                                                                       (04)
                                     // 03{01}
                                                          {O3}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                           {O3}
                                                                       {04}
  ret i32 0
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                       (%0) {O1}
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                head
  store ptr %0, ptr %q, align 8
                                                         %0 O4%0 O4 O3 %1
  ret void
                                                                   W
```

```
Algorithm 12: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                              // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p):
5 while WorkList ≠ Ø do
      p := popFromWorklist():
      foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                 // Store rule
              if a copy o ∉ E then
                  E := E \cup \{q \xrightarrow{Copy} o\}:
                                             // Add copy edge
10
                  pushIntoWorklist(a):
11
          foreach p \xrightarrow{Load} r \in E do
                                                  // Load rule
12
              if o copy r # E then
13
                  E := E \sqcup \{o \xrightarrow{Copy} r\}:
                                             // Add copy edge
14
                  pushIntoWorklist(o);
15
      foreach p \xrightarrow{Copy} x \in F do
16
                                                  // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
      foreach p Gep,fld y ∈ F do
                                                   // Gep rule
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

```
Address
                                                                              Load
                                                            Copy
                                                                            Store
define i32 @main() #0 {
                                                                       {O2}
                                              (01)
entry:
  %a1 = alloca i8, align 1
                                     // O2,
  %a = alloca ptr. align 8
                                                                       (04)
                                     // 03{01}
                                                          {O3}
                                                                                      {O2}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                           {O3}
                                                                       {04}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                       (%0) {O1}
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
                                                             %0 O4%0 O4 O3
  ret void
                                                                   W
```

```
Algorithm 13: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p);
5 while WorkList ≠ Ø do
      p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if a copy o ∉ E then
                  E := E \cup \{q \xrightarrow{Copy} o\}:
                                               // Add copy edge
10
                  pushIntoWorklist(a):
11
          foreach p \xrightarrow{Load} r \in E do
                                                   // Load rule
12
              if o copy r # E then
13
                  E := E \cup \{o \xrightarrow{Copy} r\};
                                              // Add copy edge
14
                  pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in F do
16
                                                   // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                     // Gep rule
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

```
Address
                                                                              Load
                                                            Copy
                                                                            Store
define i32 @main() #0 {
                                                                       {O2}
                                              (01)
entry:
  %a1 = alloca i8, align 1
                                     // O2,
  %a = alloca ptr. align 8
                                                                       (04)
                                     // 03{01}
                                                           {O3}
                                                                                      {O2}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                           {O3}
                                                                       {04}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                       (%0) {O1}
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                 tail
                                                                                 head
  store ptr %0, ptr %q, align 8
                                                                 %0 O4%0 O4
  ret void
                                                                   W
```

```
Algorithm 14: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p);
5 while WorkList ≠ Ø do
      p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                  // Store rule
              if a copy o ∉ E then
                  E := E \cup \{q \xrightarrow{Copy} o\}:
                                               // Add copy edge
10
                  pushIntoWorklist(a):
11
          foreach p \xrightarrow{Load} r \in E do
                                                   // Load rule
12
              if o copy r # E then
13
                  E := E \cup \{o \xrightarrow{Copy} r\};
                                              // Add copy edge
14
                  pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in F do
16
                                                   // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
       foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                     // Gep rule
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

```
Address
                                                                              Load
                                                            Copy
                                                                            Store
define i32 @main() #0 {
                                                                       {O2}
                                              (01)
entry:
  %a1 = alloca i8, align 1
                                     // O2,
  %a = alloca ptr. align 8
                                                                       (04)
                                     // 03{01}
                                                          {O3}
                                                                                      {O2}
  %b1 = alloca i8, align 1
                                     // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                           {O3}
                                                                       {04}
  ret i32 0
                                                        %p
define void @swap(ptr %p, ptr %q) #0 {
entry:
                                                       (%0){O1}
                                                                      {02} (%
  %0 = load ptr. ptr %p. align 8
  %1 = load ptr. ptr %g, align 8
  store ptr %1, ptr %p, align 8
                                                tail
                                                                                 head
  store ptr %0, ptr %q, align 8
                                                                %1 %0 O4%0
  ret void
                                                                   W
```

```
Algorithm 15: 1 Andersen's Pointer Analysis
  Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                              // Address rule
      pts(p) := pts(p) U {o}:
      pushIntoWorklist(p):
5 while WorkList ≠ Ø do
      p := popFromWorklist():
      foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                // Store rule
              if a copy o ∉ E then
                  E := E \cup \{q \xrightarrow{Copy} o\}:
                                             // Add copy edge
10
                  pushIntoWorklist(a):
11
          foreach p \xrightarrow{Load} r \in E do
                                                  // Load rule
12
              if o copy r # E then
13
                  E := E \sqcup \{o \xrightarrow{Copy} r\}:
                                             // Add copy edge
14
                  pushIntoWorklist(o);
15
      foreach p \xrightarrow{Copy} x \in F do
                                                  // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
18
          if pts(x) changed then
19
              pushIntoWorklist(x):
      foreach p Gep,fld y ∈ F do
                                                   // Gep rule
          foreach o ∈ pts(p) do
22
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

16

```
Address
                                                                                Load
                                                              Copy
                                                                              Store
define i32 @main() #0 {
                                                                         {O2}
                                               (01)
entry:
  %a1 = alloca i8, align 1
                                      // O2,
  %a = alloca ptr. align 8
                                                                         (04)
                                      // 03{01}
                                                            {O3}
  %b1 = alloca i8, align 1
                                                                                      (04
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                                                               10
                                                            {O3}
                                                                         {04}
                                                                                               11
  ret i32 0
                                                         %p
                                                                                               12
                                                                                               13
define void @swap(ptr %p, ptr %q) #0 {
                                                                                               14
                                                                                               15
entry:
                                                         (%0){O1}
                                                                        {02} (%
  %0 = load ptr. ptr %p. align 8
                                                                                               16
  %1 = load ptr. ptr %g, align 8
                                                                                               17
  store ptr %1, ptr %p, align 8
                                                                                               18
                                                  tail
                                                                                   head
                                                                                               19
  store ptr %0, ptr %q, align 8
                                                                   O4 %1 %0 O4
  ret void
                                                                     W
                                                                                               22
                                                                                               23
                                                                                               24
```

```
Algorithm 16: 1 Andersen's Pointer Analysis
              Input: G =< V.E >: Constraint Graph
                      V: a set of nodes in graph
                      E: a set of edges in graph
            1 WorkList := an empty vector of nodes:
            2 foreach o Address p do
                                                          // Address rule
                  pts(p) := pts(p) U {o}:
                  pushIntoWorklist(p):
\{O2,O1\}_{5} while WorkList \neq \emptyset do
                  p := popFromWorklist():
                  foreach o ∈ pts(p) do
                      foreach a \xrightarrow{\text{Store}} p \in E do
                                                             // Store rule
                          if a copy o ∉ E then
                              E := E \cup \{q \xrightarrow{Copy} o\}:
                                                         // Add copy edge
                              pushIntoWorklist(a):
                      foreach p \xrightarrow{Load} r \in E do
                                                              // Load rule
                          if o copy r # E then
                              E := E \sqcup \{o \xrightarrow{Copy} r\}:
                                                         // Add copy edge
                              pushIntoWorklist(o);
                  foreach p \xrightarrow{Copy} x \in F do
                                                              // Copy rule
                      pts(x) := pts(x) \cup pts(p);
                      if pts(x) changed then
                          pushIntoWorklist(x):
                  foreach p Gep,fld y ∈ F do
                                                               // Gep rule
                      foreach o ∈ pts(p) do
                          pts(x) := pts(x) \cup \{o.fld\};
                      if pts(x) changed then
                          pushIntoWorklist(x);
```

```
Address
                                                                                Load
                                                              Copy
                                                                              Store
define i32 @main() #0 {
                                                                         {O2}
                                               (01)
entry:
  %a1 = alloca i8, align 1
                                      // O2,
  %a = alloca ptr. align 8
                                                                         (04)
                                      // 03{01}
                                                            {O3}
  %b1 = alloca i8, align 1
                                                                                      100
                                      // 04
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                                                               10
                                                            {O3}
                                                                         {04}
                                                                                               11
  ret i32 0
                                                         %p
                                                                                               12
                                                                                               13
define void @swap(ptr %p, ptr %q) #0 {
                                                                                               14
                                                                                               15
entry:
                                                         (%0){O1}
                                                                              %1
  %0 = load ptr. ptr %p. align 8
                                                                                      01}
                                                                                               16
  %1 = load ptr. ptr %g, align 8
                                                                                               17
  store ptr %1, ptr %p, align 8
                                                                                               18
                                                  tail
                                                                                   head
                                                                                               19
  store ptr %0, ptr %q, align 8
  ret void
                                                                                               20
                                                                     W
                                                                                               22
                                                                                               23
                                                                                               24
```

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

```
Address
                                                                                Load
                                                              Copy
                                                                              Store
define i32 @main() #0 {
                                                                         {O2}
                                               (01)
entry:
  %a1 = alloca i8, align 1
  %a = alloca ptr. align 8
                                                                         (04)
                                                            {O3}
  %b1 = alloca i8, align 1
                                      // 04
                                               (03
  %b = alloca ptr. align 8
  store ptr %a1, ptr %a, align 8
  store ptr %b1, ptr %b, align 8
  call void @swap(ptr %a, ptr %b)
                                                                                               10
                                                             {O3}
                                                                         {04}
                                                                                               11
  ret i32 0
                                                         %p
                                                                                               12
                                                                                               13
define void @swap(ptr %p, ptr %q) #0 {
                                                                                               14
                                                                                               15
entry:
                                                         (%0){O1}
                                                                             (%1
  %0 = load ptr. ptr %p. align 8
                                                                                      01}
                                                                                               16
  %1 = load ptr. ptr %g, align 8
                                                                                               17
  store ptr %1, ptr %p, align 8
                                                                                               18
                                                  tail
                                                                                   head
                                                                                               19
  store ptr %0, ptr %q, align 8
                                                                    O3 %1 O4
  ret void
                                                                                               20
                                                                     W
                                                                                               22
                                                                                               23
                                                                                               24
```

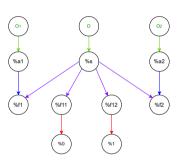
```
Algorithm 18: 1 Andersen's Pointer Analysis
              Input: G =< V.E >: Constraint Graph
                      V: a set of nodes in graph
                      E: a set of edges in graph
            1 WorkList := an empty vector of nodes:
            2 foreach o Address p do
               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

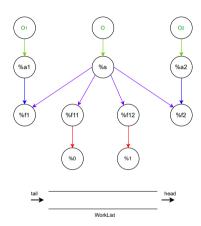
```
define i32 @main() #0 {
entry:
%a1 = alloca i8. alian 1
                               // O1
                              // O2 {O1,O2}
%b1 = alloca i8, alian 1
%a = alloca i8*, alian 8
                               // 04
%b = alloca i8*, alian 8
store i8* %a1, i8** %a, alian 8
store i8* %b1, i8** %b, alian 8
call void @swap(i8** %a, i8** %b)
ret i32 0
define void @swap(i8** %p. i8** %a)
#0 {
entry:
\%0 = load i8** \%p, alian 8
%1 = load i8** %q, align 8
store i8* %1, i8** %p, alian 8
store i8* %0, i8** %a, alian 8
ret void
```

```
Load
        Address
                   — Store
          Copy
                  {O2}
                  {04}
        {03}
        {03}
                  {04}
      %р
      %0
{01.62}
tail
                          head
                              %0
            WorkList
```

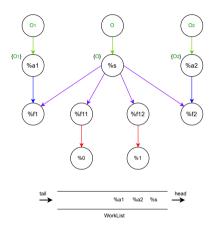
```
Algorithm 19: 1 Andersen's Pointer Analysis
              Input: G =< V.E >: Constraint Graph
                       V: a set of nodes in graph
                       E: a set of edges in graph
             1 WorkList := an empty vector of nodes:
             2 foreach o Address p do
                                                             // Address rule
                   pts(p) := pts(p) U {o}:
                   pushIntoWorklist(p):
\{O2,O1\}_5 while WorkList \neq \emptyset do
                   p := popFromWorklist():
                   foreach o ∈ pts(p) do
                       foreach a \xrightarrow{\text{Store}} p \in E do
                                                               // Store rule
                           if a copy o ∉ E then
                                E := E \sqcup \{a \xrightarrow{Copy} a\}:
                                                           // Add copy edge
            10
                               pushIntoWorklist(a):
            11
                       foreach p \xrightarrow{Load} r \in E do
                                                                // Load rule
            12
                           if o copy r # E then
            13
                               E := E \cup \{o \xrightarrow{Copy} r\};
                                                           // Add copy edge
            14
                               pushIntoWorklist(o);
            15
                   foreach p \xrightarrow{Copy} x \in F do
            16
                                                                // Copy rule
                       pts(x) := pts(x) \cup pts(p);
            18
                       if pts(x) changed then
            19
                           pushIntoWorklist(x):
                   foreach p \stackrel{\text{Gep,fld}}{\longrightarrow} x \in F do
                                                                  // Gep rule
                       foreach o ∈ pts(p) do
            22
                           pts(x) := pts(x) \cup \{o.fld\};
            23
                       if pts(x) changed then
                           pushIntoWorklist(x);
            24
```

```
struct Sf
     int* f1;
     int* f2:
5 int main(){
      struct S s:
     int a1, a2;
     s.f1 = &a1:
     s.f2 = &a2:
     int*p = s.f1;
11
      int* q = s.f2:
12 }
   define i32 @main() #0 {
   entry:
     %s = alloca %struct.S. align 8
     %a1 = alloca i32, align 4
     %a2 = alloca i32, align 4
     %f1 = getelementptr inbounds %struct.S, ptr %s, i32 0, i32 0
     store ptr %a1, ptr %f1, align 8
     %f2 = getelementptr inbounds %struct.S, ptr %s, i32 0, i32 1
     store ptr %a2, ptr %f2, align 8
     %f11 = getelementptr inbounds %struct.S, ptr %s, i32 0, i32 0
11
     %0 = load ptr, ptr %f11, align 8
     %f22 = getelementptr inbounds %struct.S. ptr %s, i32 0, i32 1
13
     %1 = load ptr. ptr %f22, align 8
14
     ret i32 0
15 }
```

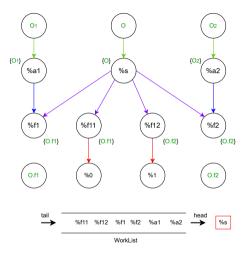




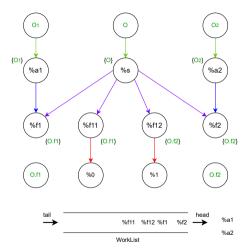
```
Algorithm 20: 1 Andersen's Pointer Analysis
   Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                    // Address rule
      pts(p) = o:
      pushIntoWorklist(p);
5 while WorkList ≠ Ø do
      p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                      // Store rule
               if a copy o ∉ E then
                   E := E \cup \{q \xrightarrow{Copy} o\};
                                                   // Add copy edge
10
                  pushIntoWorklist(q);
11
           foreach p \xrightarrow{Load} r \in E do
                                                        // Load rule
12
               if o copy r ∉ E then
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                                   // Add copy edge
14
                  pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
16
                                                        // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
          if pts(x) changed then
18
              pushIntoWorklist(x):
       foreach p \xrightarrow{Gep,fld} x \in E do
                                                         // Gep rule
20
          foreach o ∈ pts(p) do
21
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```



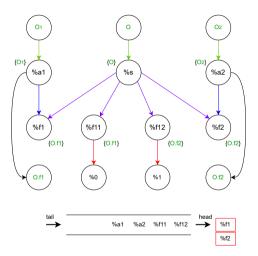
```
Algorithm 21: 1 Andersen's Pointer Analysis
   Input: G =< V E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                   // Address rule
       pts(p) = o:
      pushIntoWorklist(p);
5 while WorkList ≠ Ø do
       p := popFromWorklist();
       foreach o ∈ pts(p) do
          foreach q \xrightarrow{\text{Store}} p \in E do
я
                                                      // Store rule
              if a <sup>Copy</sup> o ∉ E then
9
                  E := E \cup \{a \xrightarrow{Copy} o\}:
                                                  // Add copy edge
10
11
                  pushIntoWorklist(q);
          foreach phoadr ∈ F do
                                                       // Load rule
12
              if o Copy r ∉ E then
13
                  E := E \sqcup \{o \xrightarrow{Copy} r\}:
                                                  // Add copy edge
14
15
                  pushIntoWorklist(o);
       foreach p Gopy x ∈ F do
                                                      // Copy rule
          pts(x) := pts(x) \cup pts(p):
17
           if pts(x) changed then
18
              pushIntoWorklist(x);
19
       foreach p \xrightarrow{Gep,fld} x \in E do
                                                        // Gep rule
20
          foreach o ∈ pts(p) do
21
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
              pushIntoWorklist(x);
```



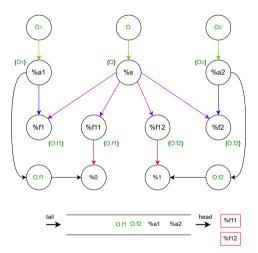
```
Algorithm 22: 1 Andersen's Pointer Analysis
   Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address n do
                                                    // Address rule
       pts(p) = o:
      pushIntoWorklist(p):
5 while WorkList ≠ Ø do
      p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                      // Store rule
              if a Copy o ∉ E then
                  E := E \cup \{q \xrightarrow{Copy} o\};
10
                                                  // Add copy edge
                  pushIntoWorklist(a):
11
          foreach p^{Load}r \in E do
12
                                                       // Load rule
              if o copy r ∉ E then
13
                  E := E \cup \{o \xrightarrow{Copy} r\}:
14
                                                  // Add copy edge
                  pushIntoWorklist(o);
       foreach p \xrightarrow{Copy} x \in F do
                                                       // Copy rule
          pts(x) := pts(x) \cup pts(p);
          if pts(x) changed then
18
              pushIntoWorklist(x):
19
       foreach p \xrightarrow{Gep,fld} x \in E do
20
                                                        // Gep rule
          foreach o e pts(p) do
21
22
              pts(x) := pts(x) \cup \{o.fld\};
           if pts(x) changed then
23
              pushIntoWorklist(x);
24
```



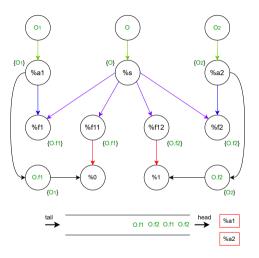
```
Algorithm 23: 1 Andersen's Pointer Analysis
   Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                    // Address rule
      pts(p) = o:
      pushIntoWorklist(p);
5 while WorkList ≠ Ø do
      p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                      // Store rule
               if a copy o ∉ E then
                   E := E \cup \{q \xrightarrow{Copy} o\};
                                                   // Add copy edge
10
                  pushIntoWorklist(q);
11
           foreach p \xrightarrow{Load} r \in E do
                                                        // Load rule
12
               if o copy r ∉ E then
13
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                                   // Add copy edge
14
                  pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
16
                                                        // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
          if pts(x) changed then
18
19
              pushIntoWorklist(x):
       foreach p \xrightarrow{Gep,fld} x \in E do
                                                         // Gep rule
20
          foreach o ∈ pts(p) do
21
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```



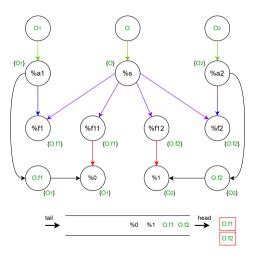
```
Algorithm 24: 1 Andersen's Pointer Analysis
   Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                   // Addrose rulo
      pts(p) = 0
      pushIntoWorklist(p);
s while WorkList ≠ Ø do
      p := popFromWorklist();
       foreach o ∈ pts(p) do
          foreach q \xrightarrow{\text{Store}} p \in E do
                                                     // Store rule
я
              if a copy o ∉ E then
9
                  E := E \cup \{a \xrightarrow{Copy} o\}:
                                                  // Add copy edge
10
11
                  pushIntoWorklist(q);
          foreach p \xrightarrow{Load} r \in E do
12
                                                       // Load rule
              if o copy r ∉ E then
13
                  E := E \sqcup \{o \xrightarrow{Copy} r\}:
                                                  // Add copy edge
14
                  pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
                                                       // Copy rule
16
          pts(x) := pts(x) \cup pts(p);
17
           if pts(x) changed then
              pushIntoWorklist(x):
19
       foreach p Gep,fld x ∈ E do
20
                                                        // Gep rule
          foreach o ∈ pts(p) do
21
           pts(x) := pts(x) \cup \{o.fld\};
22
          if pts(x) changed then
23
              pushIntoWorklist(x);
24
```



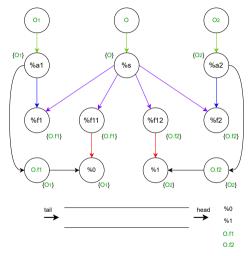
```
Algorithm 25: 1 Andersen's Pointer Analysis
   Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                  // Addrose rulo
      pts(p) = 0
      pushIntoWorklist(p);
s while WorkList ≠ Ø do
      p := popFromWorklist();
       foreach o ∈ pts(p) do
          foreach q \xrightarrow{\text{Store}} p \in E do
                                                    // Store rule
              if a copy o ∉ E then
9
                  E := E \cup \{a \xrightarrow{Copy} o\}:
10
                                                 // Add copy edge
                  pushIntoWorklist(g):
11
          foreach p \xrightarrow{Load} r \in E do
12
                                                      // Load rule
              if o copy r ∉ E then
13
                  E := E \cup \{o \xrightarrow{Copy} r\};
                                                 // Add copy edge
14
                  pushIntoWorklist(o);
15
       foreach p Copy x ∈ F do
                                                      // Copy rule
16
17
          pts(x) := pts(x) \cup pts(p);
          if pts(x) changed then
              pushIntoWorklist(x):
19
       foreach p Gep,fld x ∈ E do
20
                                                       // Gep rule
          foreach o ∈ pts(p) do
21
           pts(x) := pts(x) \cup \{o.fld\};
22
          if pts(x) changed then
23
              pushIntoWorklist(x);
24
```



```
Algorithm 26: 1 Andersen's Pointer Analysis
   Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
           E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address n do
                                                     // Address rule
       pts(p) = o:
       pushIntoWorklist(p):
5 while WorkList ≠ Ø do
       p := popFromWorklist():
       foreach o ∈ pts(p) do
           foreach \underset{p}{\operatorname{g}} \in E do
                                                        // Store rule
               if q <sup>Copy</sup> o ∉ E then
                   E := E \cup \{a \xrightarrow{Copy} o\}:
10
                                                   // Add copy edge
                  pushIntoWorklist(a):
11
           foreach p^{Load}r \in E do
12
                                                         // Load rule
               if o <sup>Copy</sup> r ∉ E then
13
                   E := E \sqcup \{o \xrightarrow{Copy} r\}
14
                                                   // Add copy edge
                   pushIntoWorklist(o);
       foreach p \xrightarrow{Copy} x \in E do
                                                        // Copy rule
           pts(x) := pts(x) \cup pts(p);
17
           if pts(x) changed then
18
               pushIntoWorklist(x);
       foreach p \xrightarrow{Gep,fld} x \in E do
                                                          // Gep rule
20
           foreach o e pts(p) do
21
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
              pushIntoWorklist(x);
```



```
Algorithm 27: 1 Andersen's Pointer Analysis
   Input: G =< V.E >: Constraint Graph
           V: a set of nodes in graph
           E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address n do
                                                     // Address rule
       pts(p) = o:
       pushIntoWorklist(p):
5 while WorkList ≠ Ø do
       p := popFromWorklist():
       foreach o ∈ pts(p) do
           foreach \underset{p}{\operatorname{g}} \in E do
                                                        // Store rule
               if q <sup>Copy</sup> o ∉ E then
                   E := E \cup \{a \xrightarrow{Copy} o\}:
10
                                                   // Add copy edge
                   pushIntoWorklist(q);
11
           foreach p^{Load}r \in E do
12
                                                         // Load rule
               if o <sup>Copy</sup> r ∉ E then
13
                   E := E \sqcup \{o \xrightarrow{Copy} r\}
14
                                                   // Add copy edge
                   pushIntoWorklist(o);
       foreach p \xrightarrow{Copy} x \in E do
                                                        // Copy rule
           pts(x) := pts(x) \cup pts(p);
           if pts(x) changed then
18
               pushIntoWorklist(x);
       foreach p \xrightarrow{Gep,fld} x \in E do
                                                          // Gep rule
21
           foreach o e pts(p) do
              pts(x) := pts(x) \cup \{o.fld\};
22
           if pts(x) changed then
23
              pushIntoWorklist(x);
```



```
Algorithm 28: 1 Andersen's Pointer Analysis
   Input: G =< V.E >: Constraint Graph
          V: a set of nodes in graph
          E: a set of edges in graph
1 WorkList := an empty vector of nodes:
2 foreach o Address p do
                                                    // Address rule
      pts(p) = o:
      pushIntoWorklist(p);
5 while WorkList ≠ Ø do
      p := popFromWorklist():
       foreach o ∈ pts(p) do
          foreach a \xrightarrow{\text{Store}} p \in E do
                                                      // Store rule
               if a copy o ∉ E then
                   E := E \cup \{q \xrightarrow{Copy} o\};
                                                   // Add copy edge
10
                  pushIntoWorklist(q);
11
           foreach p \xrightarrow{Load} r \in E do
                                                        // Load rule
12
               if o copy r ∉ E then
13
                   E := E \cup \{o \xrightarrow{Copy} r\}:
                                                   // Add copy edge
14
                  pushIntoWorklist(o);
15
       foreach p \xrightarrow{Copy} x \in E do
16
                                                        // Copy rule
          pts(x) := pts(x) \cup pts(p);
17
          if pts(x) changed then
18
              pushIntoWorklist(x):
       foreach p \xrightarrow{Gep,fld} x \in E do
                                                         // Gep rule
20
          foreach o ∈ pts(p) do
21
              pts(x) := pts(x) \cup \{o.fld\};
23
          if pts(x) changed then
              pushIntoWorklist(x);
24
```

APIs for Implementing Andersen's analysis

```
::getPts(NodeID ptr)
                                                                      //get points-to set of ptr
    SVF:: AndersenBase
                                  ::addPts(NodeID ptr, NodeID obi)
                                                                      // add obj to point-to set of object ptr
                                  ::unionPts(NodeID ptr, NodeID ptr)
                                                                      // union two point-to sets
                                  ::pushIntoWorklist(NodeID id)
                                                                      // push the node to worklist
                                  ::popFromWorklist()
                                                                      // non a node from the worklist
                                                                      // return true if the node in the worklist
                                  ::isInWorklist(NodeID id)
                                  ::isWorklistEmptv()
                                                                      // return true if the worklist is empty
     SVF:: AndersenPTA
                                  ::addCopyEdge(NodeID src. NodeID dst) // add a copy edge from src to dst
                                  ::getConstraintNode(nodeId id)
                                                                    //get the node based on its id
SVF::ConstraintGraph
                                  :: dump()
                                                                    // dump the ConsG
                                  ::getStoreInEdge()
                                                                 // get incoming store edges of the node
                                  ::getStoreOutEdge()
                                                                 //get outgoing store edges of the node
SVF::ConstraintNode
                                  ::getDirectOutEdge()
                                                                 // get outgoing copy edges of the node
                                  ::getDirectInEdge()
                                                                 // get incoming copy edges of the node
```

```
https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API#worklist-operations
https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API#onits-to-set-operations
https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API#onits-to-set-operations
https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API#constraintgraph-constraintnode-and-constraintedge
```