Lab: Andersen's Points-To Analysis on Constraint Graph

(Week 2)

Yulei Sui, Mohamad Barbar, Hanyuan Li, Angela He

School of Computer Science and Engineering

University of New South Wales, Australia

- Constraint Graph
  - Modeling the relations between variables (e.g., pointers and memory allocations/objects)
  - Solving constraints is important to reason about program semantics (e.g., points-to relations)
- Points-to Analysis on Constraint Graph
  - What does the pointer *p* point to? Does *p* point to the same thing as *q*? Does *p* point to object *o*?
  - **And much more you may want to ask about pointers and what they will point to at runtime**
  - More about Andersen's analysis: `https://github.com/SVF-tools/Software-Security-Analysis/blob/slides/1.lab-andersen.pdf`
  - Implementation hints on the wiki: `https://github.com/SVF-tools/Software-Security-Analysis/wiki/Lab-Exercise-1#5-implementation-hints`
  - **This lab** covers Andersen's analysis on **self-constructed graphs**. **Week 3** will introduce constraint solving and Andersen **on LLVM and SVF IR**.

Scope

ADDR instructions: $p = \&o$

$$o \xrightarrow{Addr} p$$

Scope

ADDR instructions: $p = \&o$

$$o \xrightarrow{Addr} p$$

COPY instructions: $p = q$

$$q \xrightarrow{Copy} p$$

Scope

ADDR instructions: $p = \&o$

$$o \xrightarrow{Addr} p$$

STORE instructions: $*p = q$

$$q \xrightarrow{Store} p$$

COPY instructions: $p = q$

$$q \xrightarrow{Copy} p$$

Scope

ADDR instructions: $p = \&o$

$o \xrightarrow{Addr} p$

STORE instructions: $*p = q$

$q \xrightarrow{Store} p$

COPY instructions: $p = q$

$q \xrightarrow{Copy} p$

LOAD instructions: $p = *q$

$q \xrightarrow{Load} p$

Goal

A points-to set *pts* for every pointer and object.

For example, $pts(p) = \{o_1, o_2\}$, $pts(q) = \{o_1\}$, $pts(o_1) = \{\ \}$, ...

Constraint solving

- Our instructions form constraints
- We solve these constraints by manipulating points-to sets (all unions!)
- Keep “solving” these until we reach our goal
- We are done when we reach a fixpoint

Initial state

An empty points-to set for every pointer and every object.

Solving ADDR instructions

$$p = \&o$$

$$\hookrightarrow$$

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

Solving ADDR instructions

$p = \&o$

$\hookrightarrow$

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

$o \xrightarrow{ADDR} p$

$\hookrightarrow$

$o \xrightarrow{ADDR} p$

Solving COPY instructions

$$p = q$$

$$\hookrightarrow$$

$$pts(p) = pts(p) \cup pts(q)$$

Solving COPY instructions

$$p = q \quad \hookrightarrow \quad pts(p) = pts(p) \cup pts(q)$$

$$q \xrightarrow{COPY} p \quad \hookrightarrow \quad q \xrightarrow{COPY} p$$

Solving LOAD instructions

$$p = *q$$

$$\hookrightarrow$$

$$pts(p) = pts(p) \cup pts(*q)$$

Solving LOAD instructions

$$p = *q$$

$$\hookrightarrow$$

$$pts(p) = pts(p) \cup pts(*q)$$

$$\forall o \in pts(q).\ pts(p) = pts(p) \cup pts(o)$$

Solving LOAD instructions

$p = *q$

$\hookrightarrow$

~~$pts(p) = pts(p) \cup pts(*q)$~~

$\forall o \in pts(q).\ pts(p) = pts(p) \cup pts(o)$

o

$q \xrightarrow{Load} p$

$\hookrightarrow$ (assume $pts(q) = \{o\}$)

o

$\downarrow$ *Copy*

$q \xrightarrow{Load} p$

Solving STORE instructions

$$*p = q$$

$$\hookrightarrow$$

$$pts(*p) = pts(*p) \cup pts(q)$$

Solving STORE instructions

$*p = q$

$\hookrightarrow$

~~$pts(*p) = pts(*p) \cup pts(q)$~~

$\forall o \in pts(p).\ pts(o) = pts(o) \cup pts(q)$

Solving STORE instructions

$*p = q$

$\hookrightarrow$

~~$pts(*p) = pts(*p) \cup pts(q)$~~

$\forall o \in pts(p).\ pts(o) = pts(o) \cup pts(q)$

o

$q \xrightarrow{Store} p$

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

Algorithmically solving the constraints

We'll be using a worklist...

If a node's points-to set is ever changed, push it onto the worklist.

If a node is ever given a new outgoing COPY edge, push it onto the worklist.

Algorithmically solving the constraints

We'll be using a worklist...

If a node's points-to set is ever changed, push it onto the worklist.
If a node is ever given a new outgoing COPY edge, push it onto the worklist.

1. Initialise the graph with all the instructions.
2. Solve every ADDR instruction.
3. As long as the worklist is not empty...
   - 3.1 Pop node *n* from the worklist.
   - 3.2 Solve each incoming STORE edge and each outgoing LOAD edge of *n*.
   - 3.3 Solve each outgoing COPY edge of *n*.
4. We're done! We've reached a fixpoint!

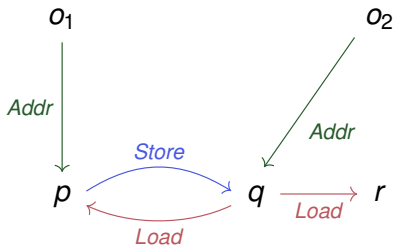
Example – initialise the graph

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

$pt(p) = \{\ \}$
$pt(q) = \{\ \}$
$pt(r) = \{\ \}$

$pt(o_1) = \{\ \}$
$pt(o_2) = \{\ \}$

Worklist: –

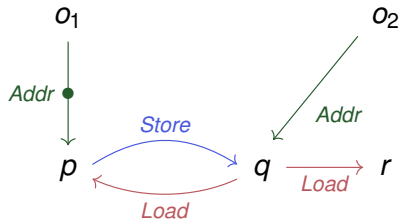


Example – solve ADDR instructions (1)

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1 ⟸
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: –

$pt(p) = \{\ \}$ $pt(o_1) = \{\ \}$
$pt(q) = \{\ \}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

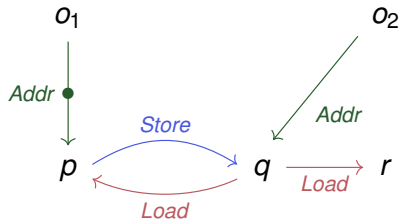


Example – solve ADDR instructions (1)

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1 ⟸
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: –

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{\ \}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

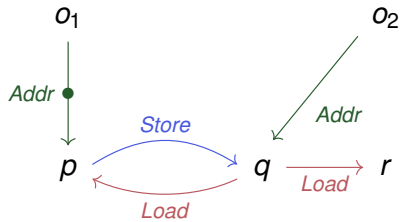


Example – solve ADDR instructions (1)

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1 ⟸
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: p

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{\ \}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

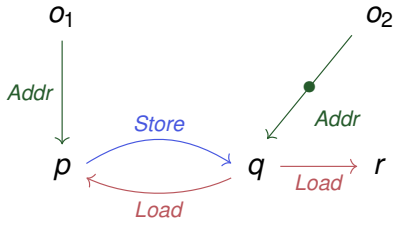


Example – solve ADDR instructions (2)

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2 ⟸
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: p

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{\ \}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

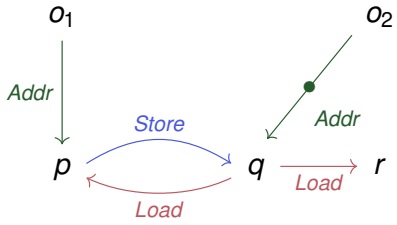


Example – solve ADDR instructions (2)

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2 ⟸
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: p

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

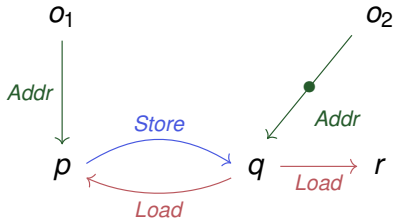


Example – solve ADDR instructions (2)

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2 ⟸
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: p, q

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

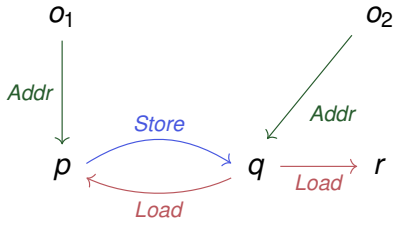


Example – worklist (1): p's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{p}$, q

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

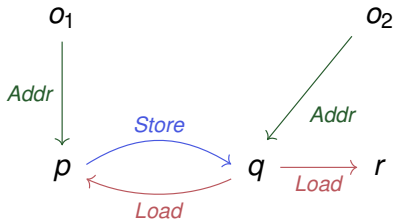


Example – worklist (1): p's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{p}$, q

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

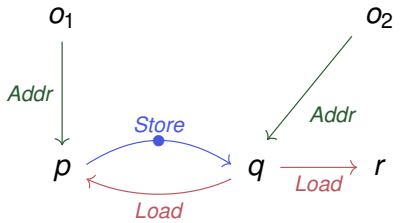


Example – worklist (2): q's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE ⟸
  p = *q;          // LOAD
}
```

Worklist: q

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

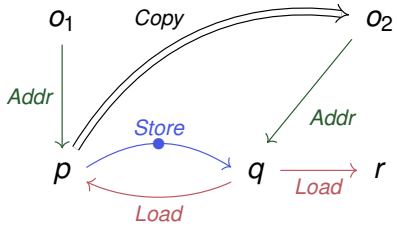


Example – worklist (2): q's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE ⟸
  p = *q;          // LOAD
}
```

Worklist: q

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

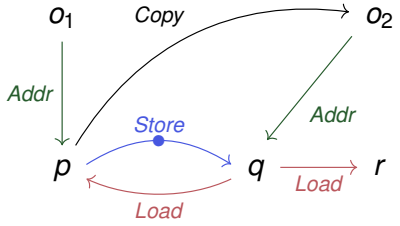


Example – worklist (2): q's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE ⟸
  p = *q;          // LOAD
}
```

Worklist: $\boxed{q}$, p

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

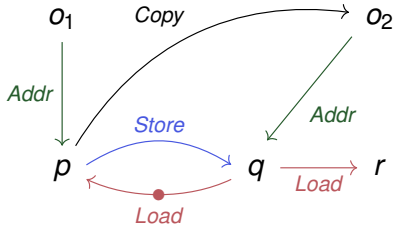


Example – worklist (2): q's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD ⟸
}
```

Worklist: $\boxed{q}$, p

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

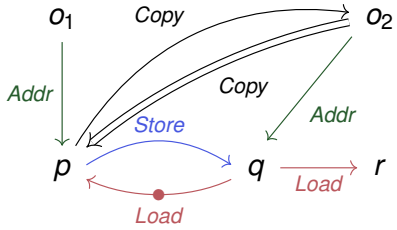


Example – worklist (2): q's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD ⟸
}
```

Worklist: $\boxed{q}$, p

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

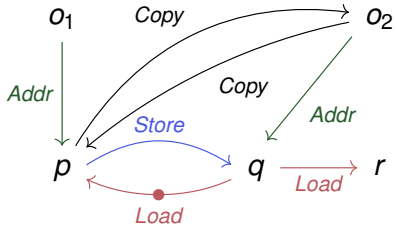


Example – worklist (2): q's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD ⟸
}
```

Worklist: $\boxed{q}$, p, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

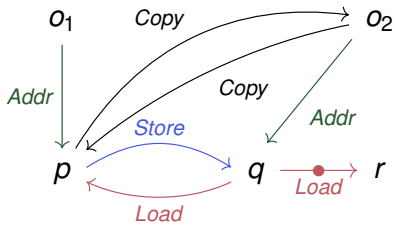


Example – worklist (2): q's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD ⟸
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{q}$, p, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

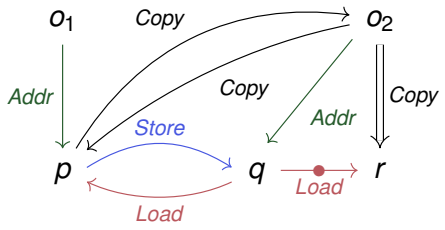


Example – worklist (2): q's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD ⟸
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{q}$, p, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

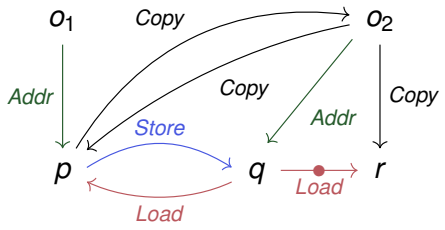


Example – worklist (2): q's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD ⟸
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{q}$, p, o_2, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

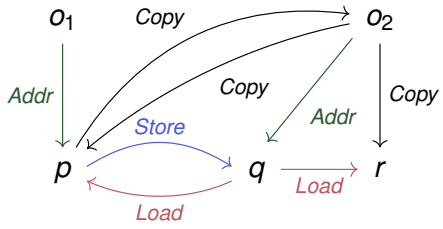


Example – worklist (2): q's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{q}$, p, o_2, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

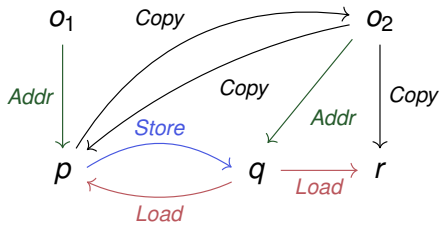


Example – worklist (3): p's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

Worklist: $\boxed{p}$, o_2, o_2

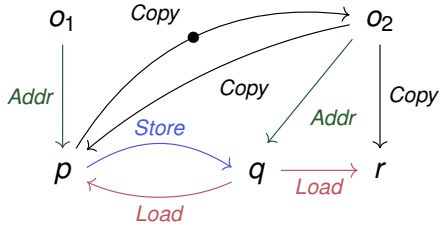


Example – worklist (3): p's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{p}$, o_2, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{\ \}$

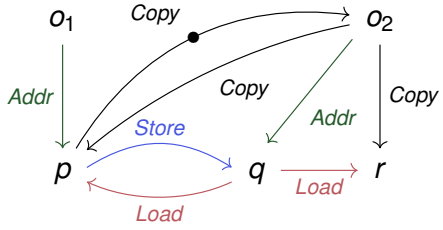


Example – worklist (3): p's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{p}$, o_2, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{\ \}$

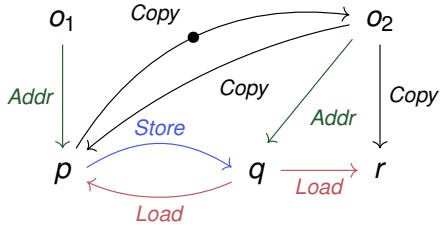


Example – worklist (3): p's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{p}$, o_2, o_2, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{\ \}$

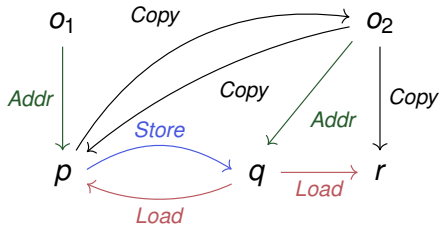


Example – worklist (4): o_2's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, o_2, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{\ \}$

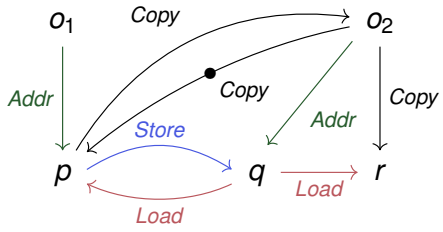


Example – worklist (4): o_2's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, o_2, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{\ \}$

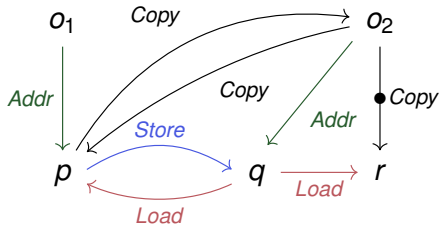


Example – worklist (4): o_2's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, o_2, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{\ \}$

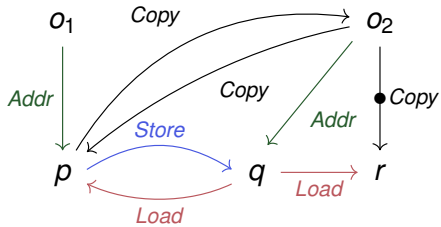


Example – worklist (4): o_2's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, o_2, o_2

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$

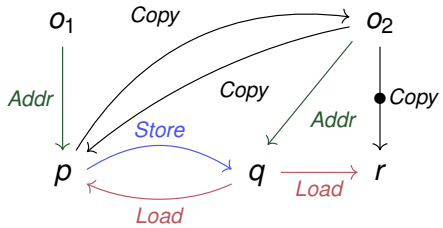


Example – worklist (4): o_2's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, o_2, o_2, r

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$

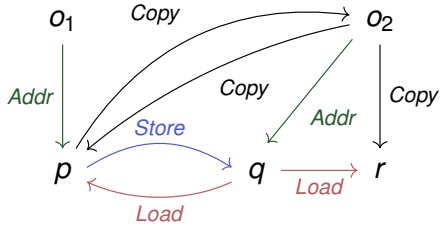


Example – worklist (5): o_2's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, o_2, r

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$

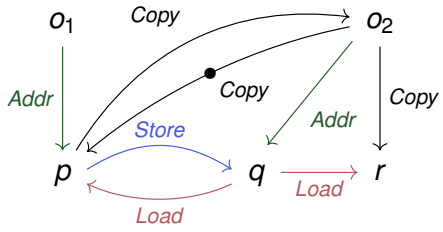


Example – worklist (5): o_2's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, o_2, r

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$

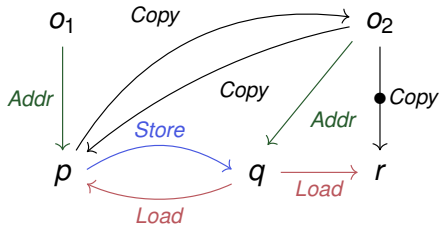


Example – worklist (5): o_2's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, o_2, r

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$

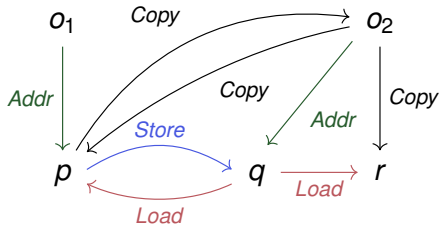


Example – worklist (6): o_2's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, r

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$

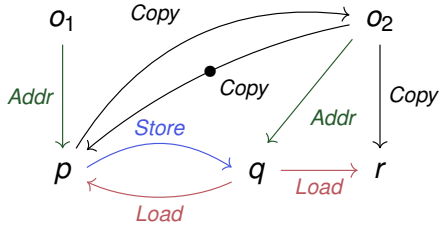


Example – worklist (6): o_2's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, r

$$pt(p) = \{o_1\} \qquad pt(o_1) = \{\ \}$$
$$pt(q) = \{o_2\} \qquad pt(o_2) = \{o_1\}$$
$$pt(r) = \{o_1\}$$

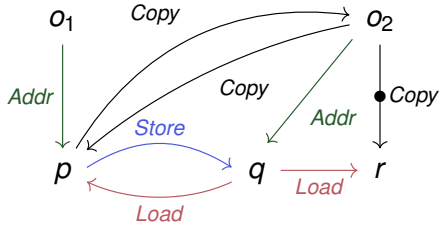


Example – worklist (6): o_2's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: $\boxed{o_2}$, r

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$

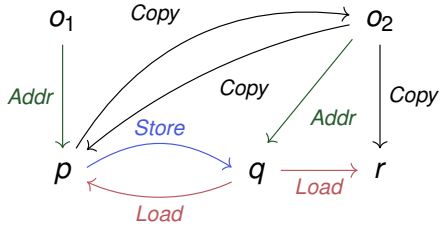


Example – worklist (7): r's incoming STOREs/outgoing LOADs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: r

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$

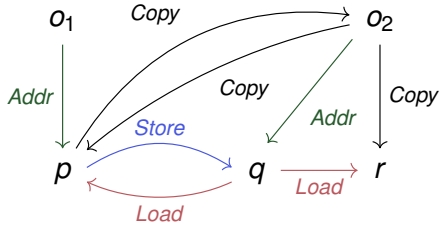


Example – worklist (7): r's outgoing COPYs

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: r

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$

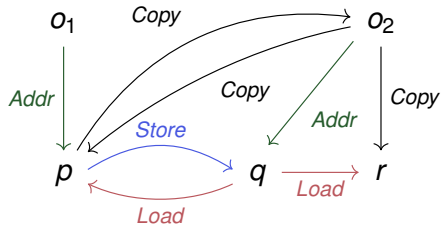


Example – fixpoint!

```
int main(void) {
  int *p, **q, *r;
  p = malloc(...); // ADDR o1
  q = malloc(...); // ADDR o2
  r = *q;          // LOAD
  *q = p;          // STORE
  p = *q;          // LOAD
}
```

Worklist: –

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{\ \}$
$pt(r) = \{o_1\}$



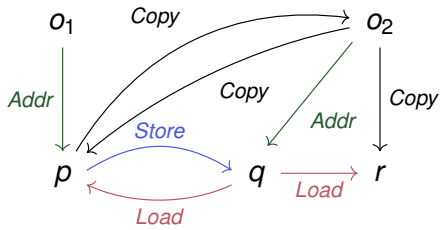
Example – tainted flows

```
int main(void) {
  int *p, **q, *r;
  p = sensitive(...); // ADDR o1
  q = malloc(...);    // ADDR o2
  r = *q;             // LOAD
  *q = p;             // STORE
  p = *q;             // LOAD
  broadcast(p);  // Safe?
  broadcast(q);  // Safe?
  broadcast(r);  // Safe?
}
```

$pt(p) = \{o_1\}$ $\quad pt(o_1) = \{\ \}$

$pt(q) = \{o_2\}$ $\quad pt(o_2) = \{o_1\}$

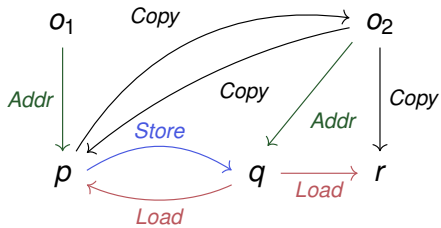
$pt(r) = \{o_1\}$



Example – tainted flows

```
int main(void) {
  int *p, **q, *r;
  p = sensitive(...); // ADDR o1
  q = malloc(...);    // ADDR o2
  r = *q;             // LOAD
  *q = p;             // STORE
  p = *q;             // LOAD
  broadcast(p);  // Safe?  NO
  broadcast(q);  // Safe?
  broadcast(r);  // Safe?
}
```

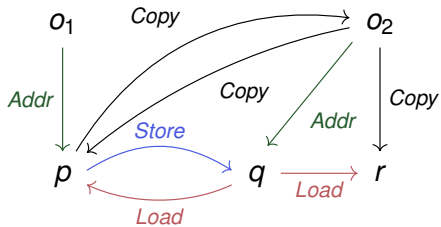
$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$



Example – tainted flows

```
int main(void) {
  int *p, **q, *r;
  p = sensitive(...); // ADDR o1
  q = malloc(...);    // ADDR o2
  r = *q;             // LOAD
  *q = p;             // STORE
  p = *q;             // LOAD
  broadcast(p);  // Safe?  NO
  broadcast(q);  // Safe?  YES
  broadcast(r);  // Safe?
}
```

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$



Example – tainted flows

```
int main(void) {
  int *p, **q, *r;
  p = sensitive(...); // ADDR o1
  q = malloc(...);    // ADDR o2
  r = *q;             // LOAD
  *q = p;             // STORE
  p = *q;             // LOAD
  broadcast(p);  // Safe?  NO
  broadcast(q);  // Safe?  YES
  broadcast(r);  // Safe?  NO
}
```

$pt(p) = \{o_1\}$ $pt(o_1) = \{\ \}$
$pt(q) = \{o_2\}$ $pt(o_2) = \{o_1\}$
$pt(r) = \{o_1\}$