

Pointer Analysis

Rupesh Nasre.

CS6843 Program Analysis
IIT Madras
Jan 2018

Outline

- Introduction
- Pointer analysis as a DFA problem
- Design decisions
- Andersen's analysis, Steensgaard's analysis
- Pointer analysis as a graph problem
 - Optimizations
- Pointer analysis as graph rewrite rules
- Applications
- Parallelization
 - Constraint based
 - Replication based

Points-to Analysis as a Graph Problem

Each pointer as a node, directed edge $q \rightarrow p$ indicates points-to set of q is a subset of that of p .

Input: set C of points-to constraints

Process address-of constraints

Add edges to constraint graph G using copy constraints

repeat

 Propagate points-to information in G

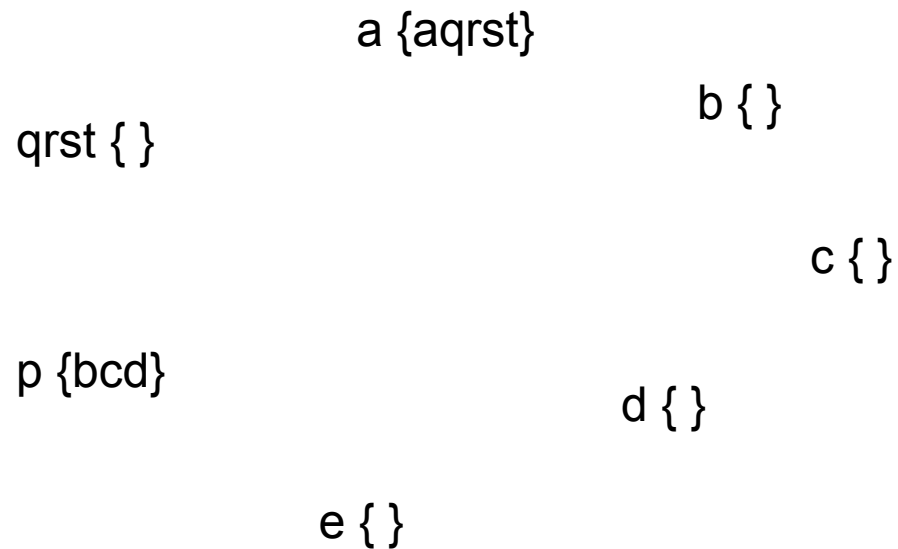
 Add edges to G using load and store constraints

until fixpoint

Points-to Analysis as a Graph Problem

$*e = c, c = *a, e = d, b = a, *a = p$

Initially, $a \rightarrow \{a, q, r, s, t\}, p \rightarrow \{b, c, d\}$

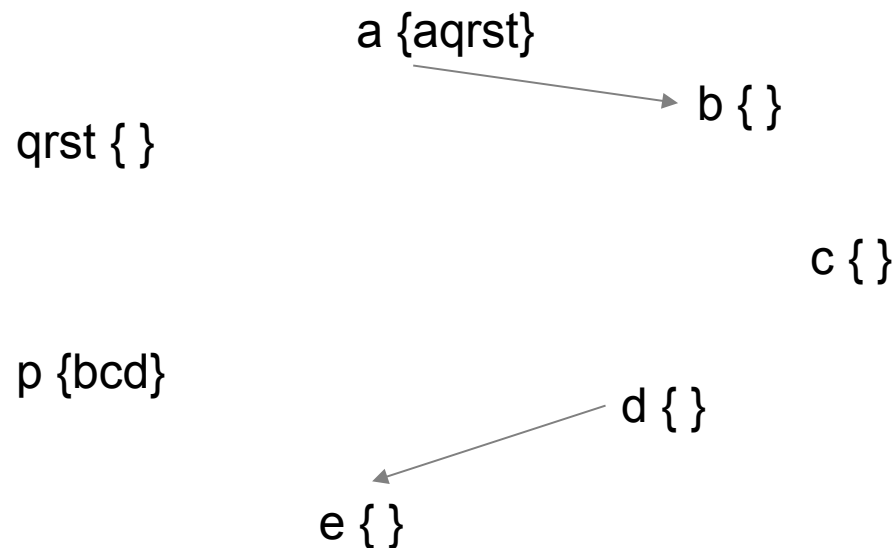


Points-to Analysis as a Graph Problem

$*e = c, c = *a, e = d, b = a, *a = p$

Initially, $a \rightarrow \{a, q, r, s, t\}, p \rightarrow \{b, c, d\}$

Iteration 0



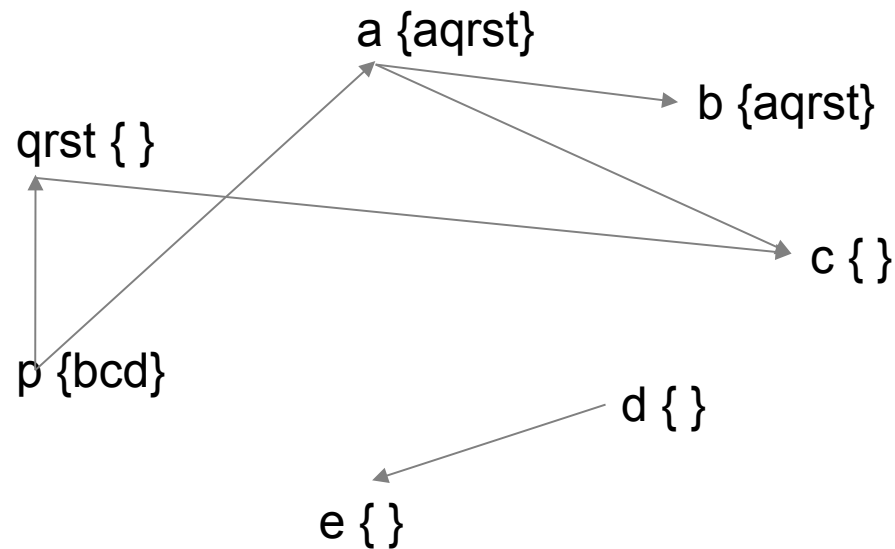
$$\begin{array}{l} e = d \\ b = a \\ \hline *e = c \\ c = *a \\ *a = p \end{array}$$

Points-to Analysis as a Graph Problem

$*e = c, c = *a, e = d, b = a, *a = p$

Initially, $a \rightarrow \{a, q, r, s, t\}, p \rightarrow \{b, c, d\}$

Iteration 1



$e = d$
 $b = a$

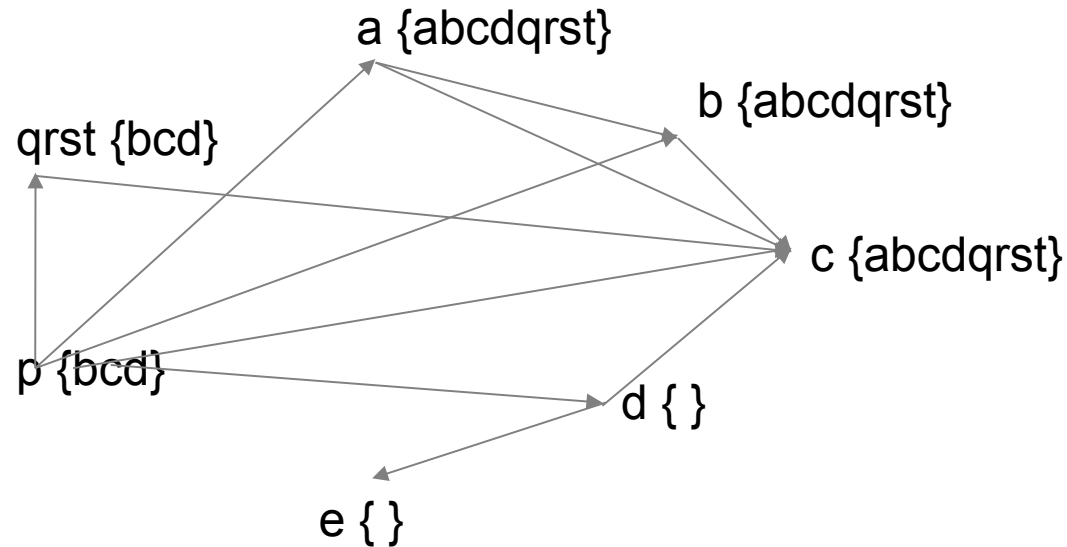
 $*e = c$
 $c = *a$
 $*a = p$

Points-to Analysis as a Graph Problem

$*e = c, c = *a, e = d, b = a, *a = p$

Initially, $a \rightarrow \{a, q, r, s, t\}, p \rightarrow \{b, c, d\}$

Iteration 2



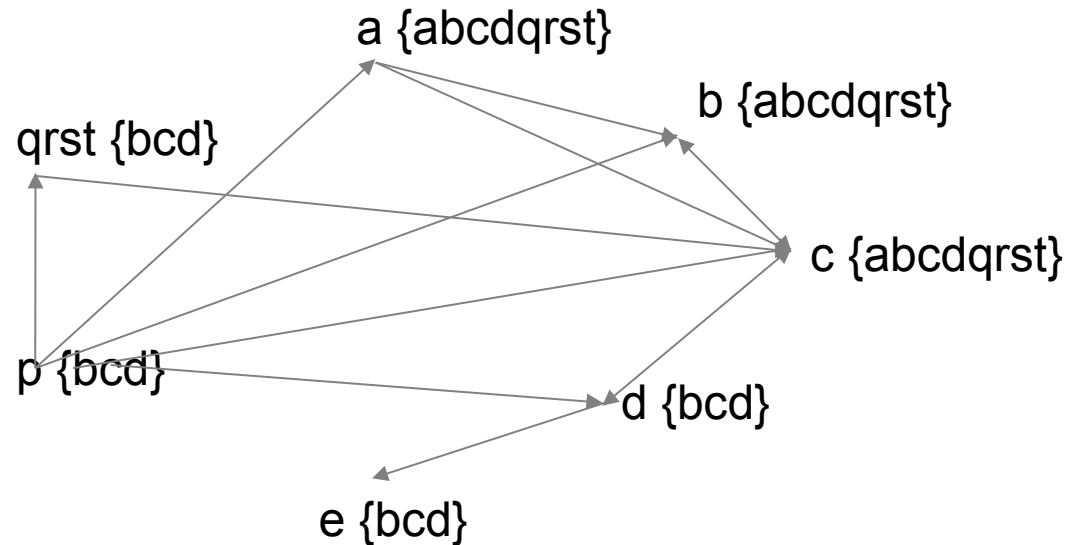
$$\begin{array}{l} e = d \\ b = a \\ \hline *e = c \\ c = *a \\ *a = p \end{array}$$

Points-to Analysis as a Graph Problem

$*e = c, c = *a, e = d, b = a, *a = p$

Initially, $a \rightarrow \{a, q, r, s, t\}, p \rightarrow \{b, c, d\}$

Iteration 3



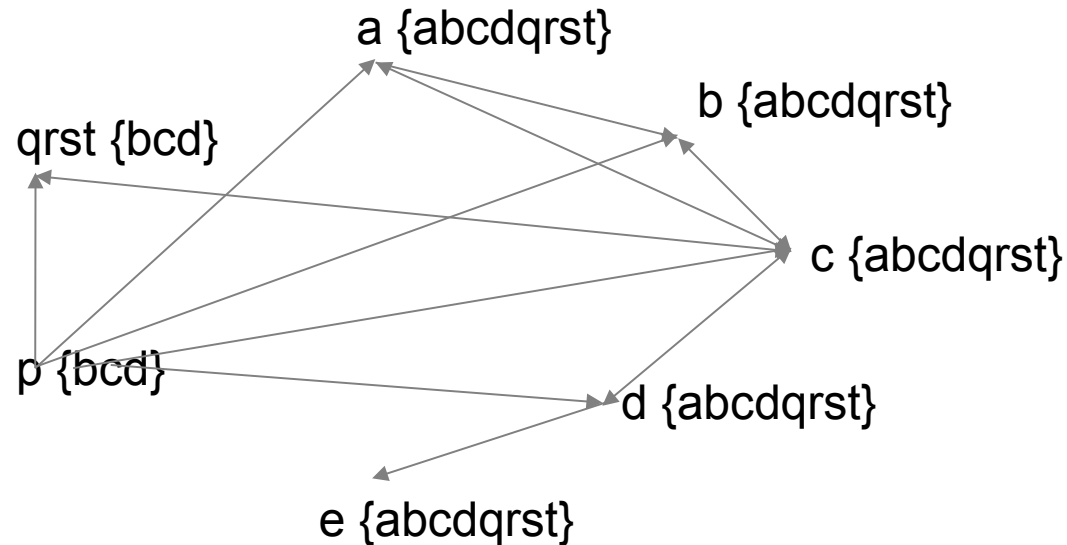
$$\begin{array}{l} e = d \\ b = a \\ \hline *e = c \\ c = *a \\ *a = p \end{array}$$

Points-to Analysis as a Graph Problem

$*e = c, c = *a, e = d, b = a, *a = p$

Initially, $a \rightarrow \{a, q, r, s, t\}, p \rightarrow \{b, c, d\}$

Iteration 4



$e = d$
 $b = a$

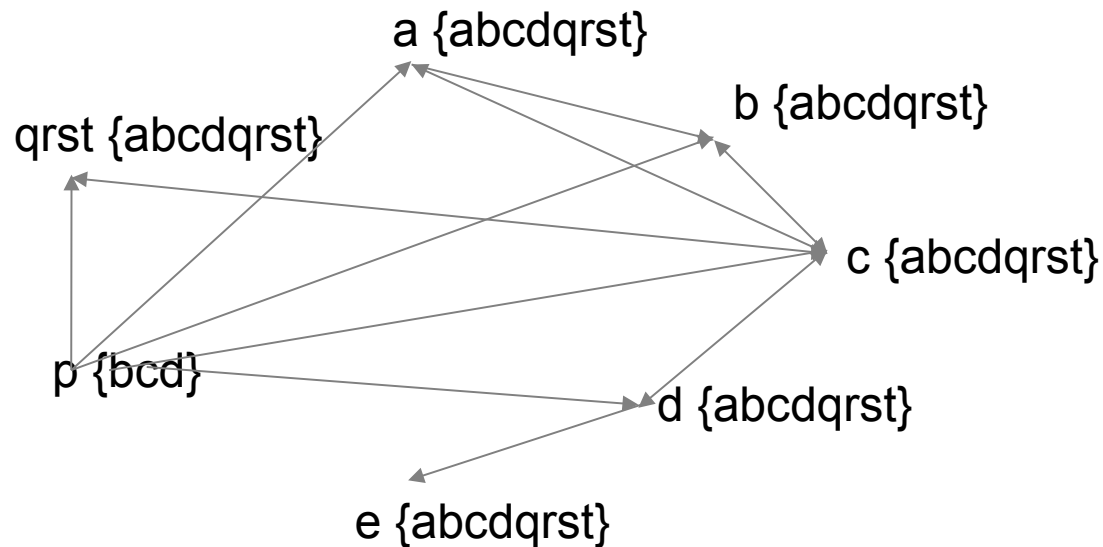
 $*e = c$
 $c = *a$
 $*a = p$

Points-to Analysis as a Graph Problem

$*e = c, c = *a, e = d, b = a, *a = p$

Initially, $a \rightarrow \{a, q, r, s, t\}, p \rightarrow \{b, c, d\}$

Iteration 5: fixed-point



$e = d$
 $b = a$

 $*e = c$
 $c = *a$
 $*a = p$

Why a Graph Formulation?

- A naïve formulation offers no benefits over the constraint-based formulation.
- We need to exploit structural properties of the constraint graph for efficient execution.
 - Online cycle detection
 - Online dominator detection
 - Propagation order: Topological sort, Depth first

Classwork

- Compute points-to information using constraint graph.

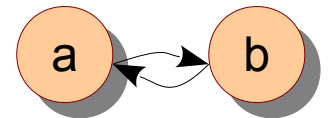
```
a = &x  
b = a  
*p = b  
p = &a  
*q = p  
b = &y  
a = &b  
q = b
```

Pointer Equivalence

- Two pointers are equivalent if they have the same points-to sets. Simple.
- If we identify such pointers *before* computing their points-to information, we can reduce the number of pointers tracked during the analysis.
- Now let's go back to the constraint graph.

Why a Graph Formulation?

- If the program contains statements $a = b$, $b = a$, what can you say about the points-to sets of a and b at the fixed-point?
- How does the constraint graph look like?
- How about $a = b$, $b = c$, $c = a$?
- How about $a = c$, $b = *p$, $c = b$?

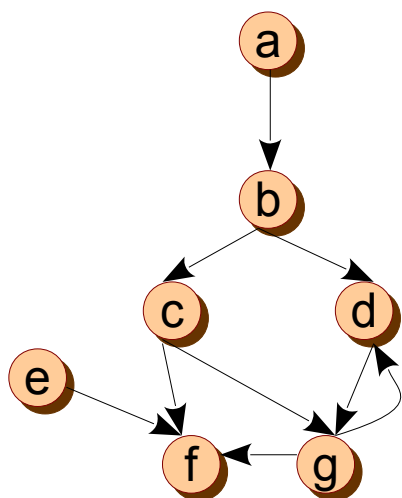


Online Cycle Detection

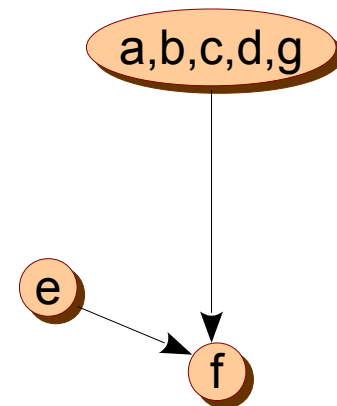
- Edges get added to the graph dynamically.
- So, cycle detection is performed online.
- Cycles are collapsed – usually replaced with a representative.
- Can use union-find.

Online Dominator Detection

- If two nodes in a constraint graph have the same dominator, they are pointer equivalent.
- A dominator and its dominees are pointer equivalent.
- **doms** is a transitive relation.



$b \text{ doms } g$
 $\neg(b \text{ doms } f)$
 $a \text{ doms } b$
By transitivity, $a \text{ doms } g$



Offline Variable Substitution

- But some constraints were easy to check for equivalence without running the analysis.
 - $a = b, b = a$
 - $a = *p, *p = a$
 - $a = b, c = a, c = b$ and no other incoming edge to c .
- OVS is performed before running pointer analysis.

Propagation Order

- A topological ordering is beneficial for propagating points-to information (wave propagation)
- The information may also be propagated in depth-first manner (deep propagation)
- DP is helpful to reuse the difference in points-to information

How About Constraint Order?

- Given a set of constraints, find an optimal way of evaluating them
- Like most CS problems, this is NP-Complete
- Reducible from Set Cover

Reduction from Set Cover

- Given an instance of Set Cover $SC(U, S, K)$
 - U : universe of elements
 - S : set of subsets S_i
 - K : some number

$S = \{1, 4\}, \{2, 5\}, \{2, 4, 5\}, \{3\}$
Solution Two: $\{1, 4\}, \{2, 4, 5\}, \{3\}$
Solution One: $\{1, 4\}, \{2, 5\}, \{3\}$

whether there exists a set of K subsets covering U

- Reduce to $PTA(C, S, K)$ where
 - C is a set of copy constraints
 - S is a variable of interest w.r.t. fixed-point
 - K is the number of steps in which the fixed-point is reached

SC $\cong$ PTA

- SC(U, S, K) $\cong$ PTA(C, S, K)
- Linear time reduction
 - for each $s \in S_i$ add s to $\text{ptsto}(S_i)$
 - for each set S_i create a copy statement $S = S_i$
- A solution to PTA $\Rightarrow$ A solution to SC
- A solution to PTA $\Leftarrow$ A solution to SC
- Poly-time verification

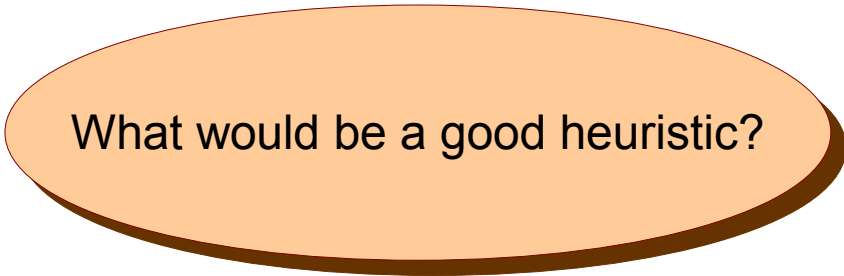
NP-Hard

NPC

NP

How About Constraint Order?

- Given a set of constraints, find an optimal way of evaluating them
- Like most CS problems, this is NP-Complete
- Reducible from Set Cover
- Need to depend upon heuristics



What would be a good heuristic?

Constraint Priority

- Priority of a constraint in iteration i is the amount of **new points-to information** it adds in iteration $(i - 1)$.
- Constraints are **grouped** in different priority levels which are ordered based on their priority.
- A constraint may **jump** across multiple priority levels during the analysis.

Bucketization

Iteration 1

Iteration 2

Iteration 3

...

Iteration n

Level 5

Level 4

Level 3

Level 2

Level 1

Level 0

C1

C2

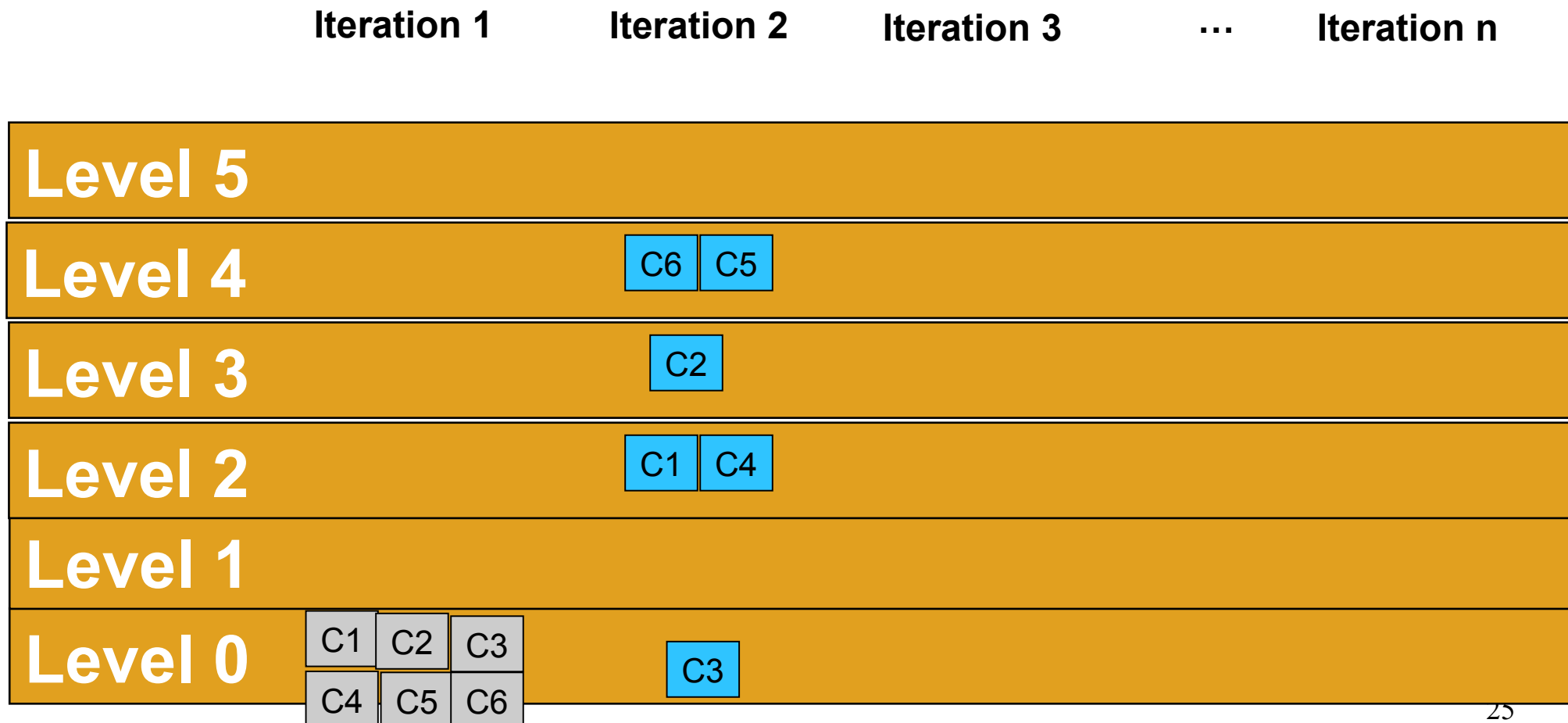
C3

C4

C5

C6

Bucketization



Bucketization

Iteration 1

Iteration 2

Iteration 3

...

Iteration n

Level 5				C5			
Level 4				C6 C5		C1	
Level 3				C2		C2	
Level 2				C1 C4		C6 C4	
Level 1							
Level 0		C1	C2	C3	C3		
	C4	C5	C6	C3	C3		

Bucketization

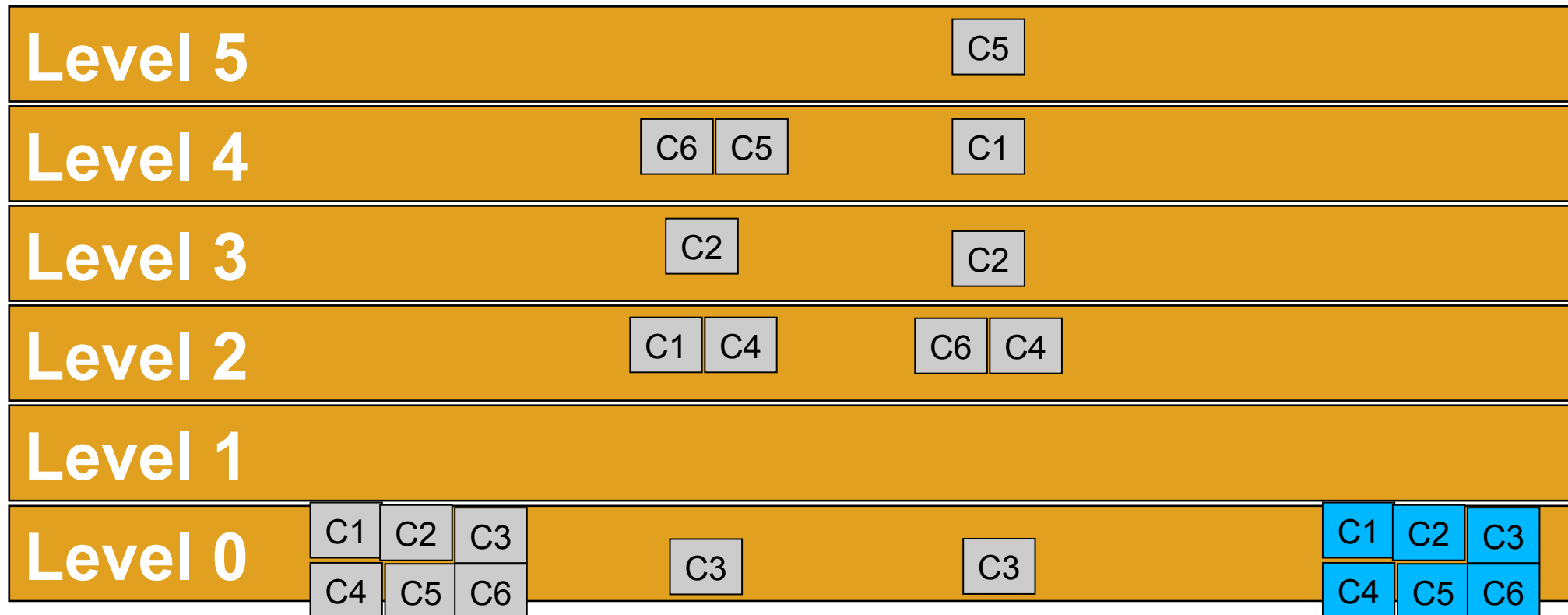
Iteration 1

Iteration 2

Iteration 3

...

Iteration n



Skewed Evaluation

Iteration 1

Iteration 2

Iteration 3

...

Iteration n

Level 5

Level 4

C6

C5

Level 3

C2

Level 2

C1

C4

Level 1

Level 0

C1

C2

C3

C4

C5

C6

C3

Skewed Evaluation

Iteration 1

Iteration 2

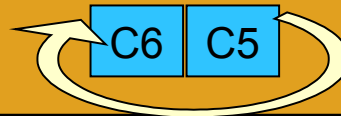
Iteration 3

...

Iteration n

Level 5

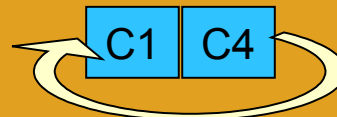
Level 4



Level 3

C2

Level 2



Level 1

Level 0



C3

Prioritized Points-to Analysis

Processing order

*a = p (18)

c = *a (8)

*e = c (0)

Prioritized Points-to Analysis

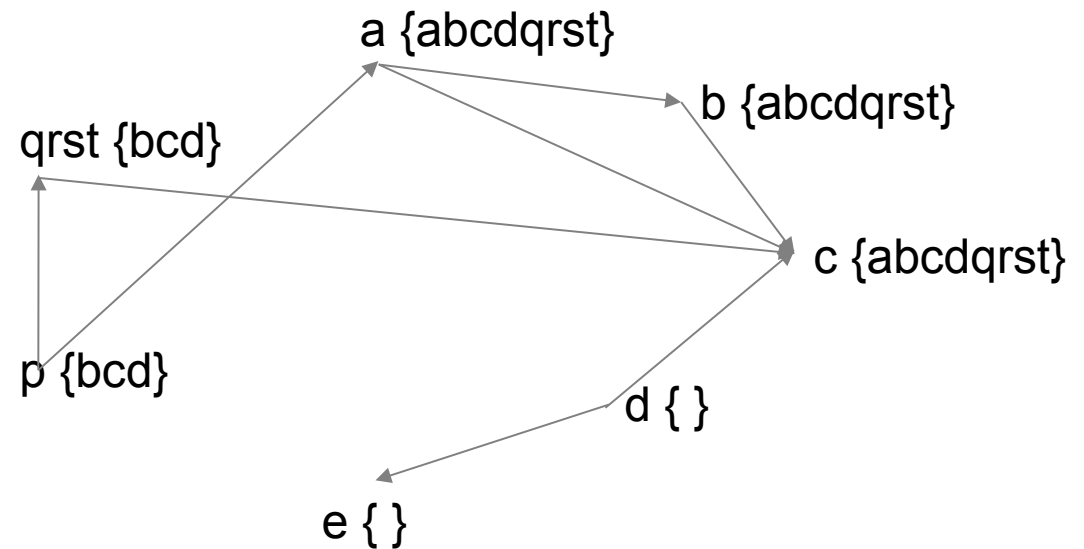
Processing order

*a = p (18)

c = *a (8)

*e = c (0)

Priority: Iteration 1

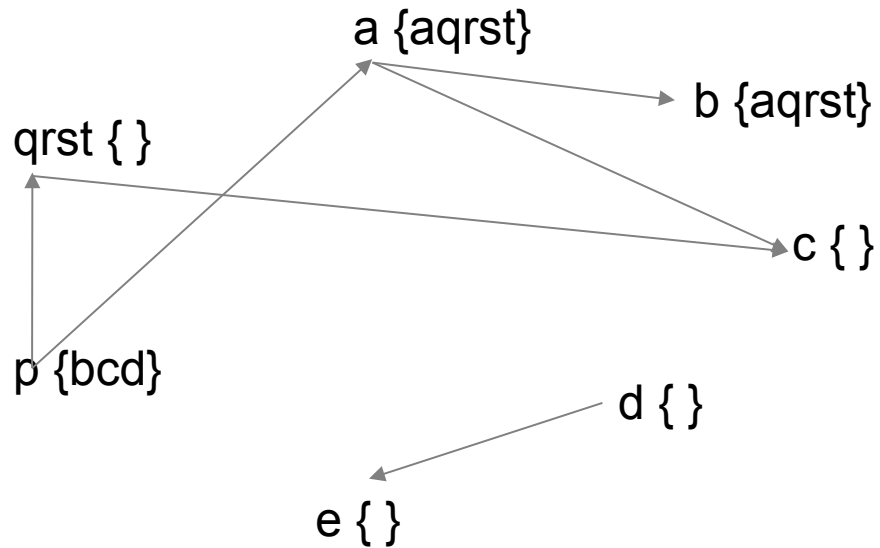


Prioritized Points-to Analysis

Fixed Processing order

```
*e = c  
c = *a  
*a = p
```

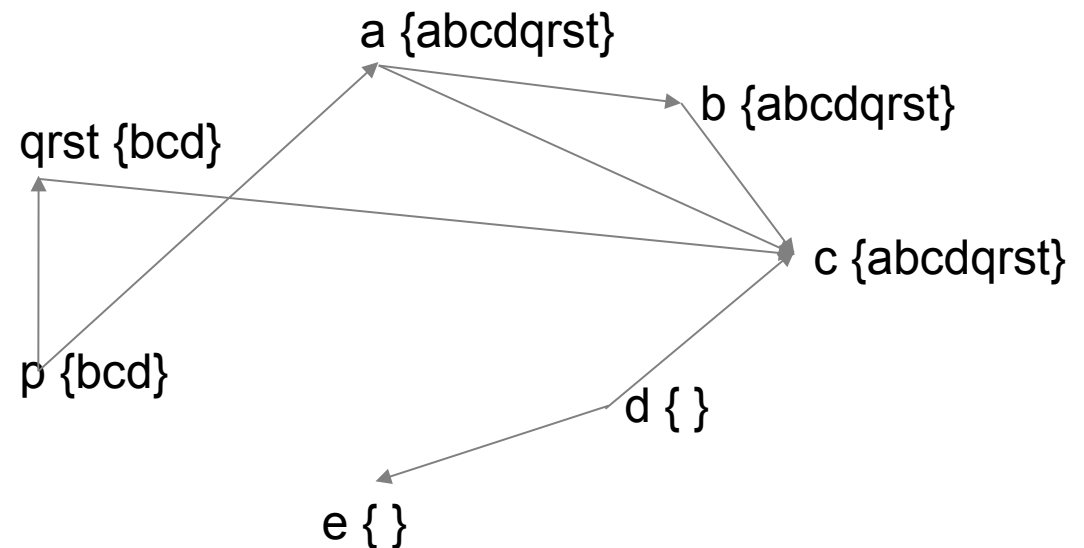
Andersen: Iteration 1



Processing order

```
*a = p (18)  
c = *a (8)  
*e = c (0)
```

Priority: Iteration 1

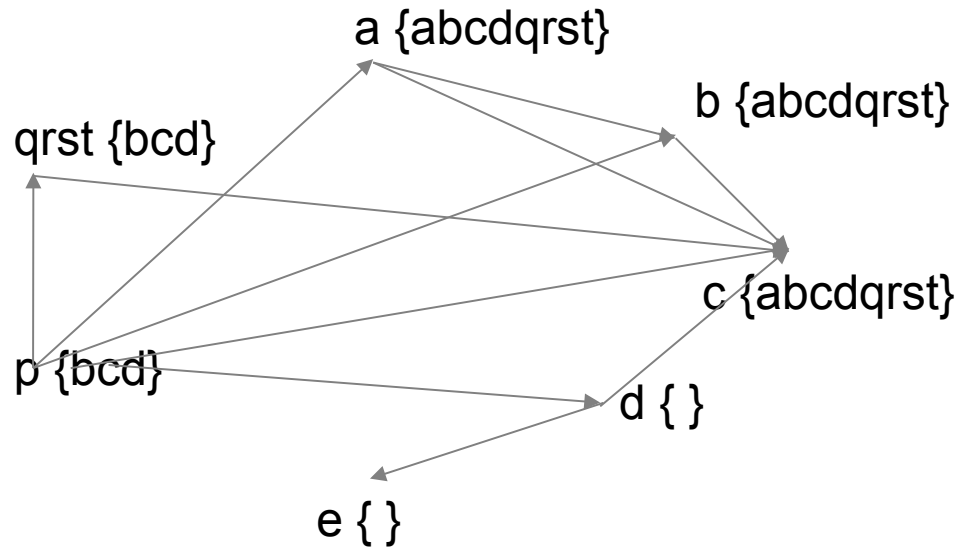


Prioritized Points-to Analysis

Fixed Processing order

$*e = c$
 $c = *a$
 $*a = p$

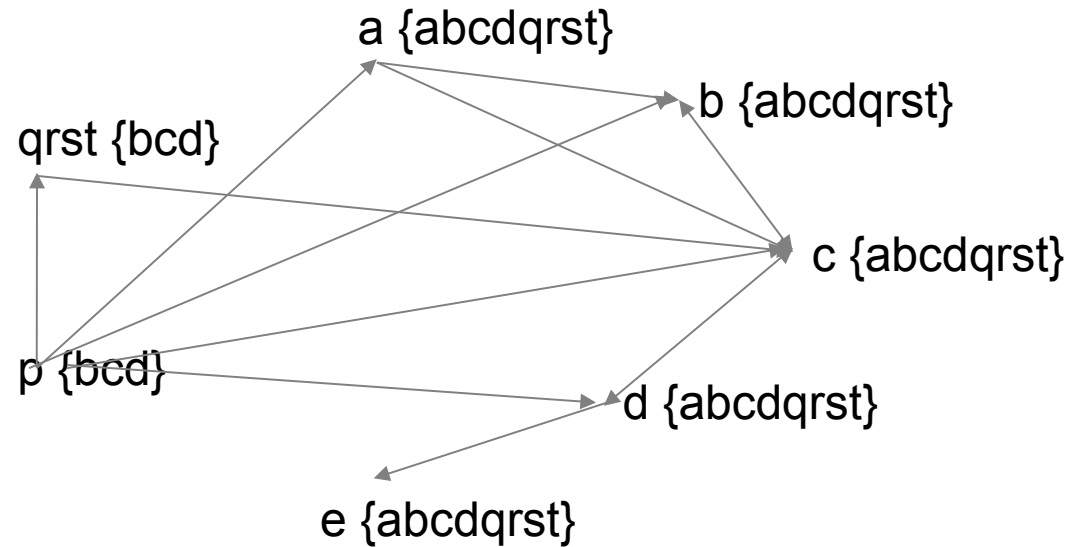
Andersen: Iteration 2



Processing order

$*a = p$ (6)
 $c = *a$ (0)
 $*e = c$ (10)

Priority: Iteration 2

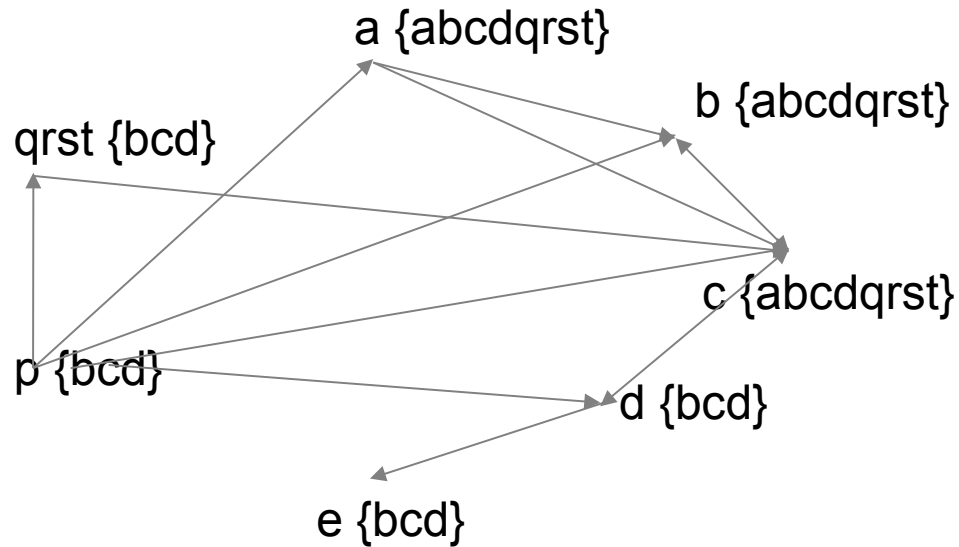


Prioritized Points-to Analysis

Fixed Processing order

```
*e = c  
c = *a  
*a = p
```

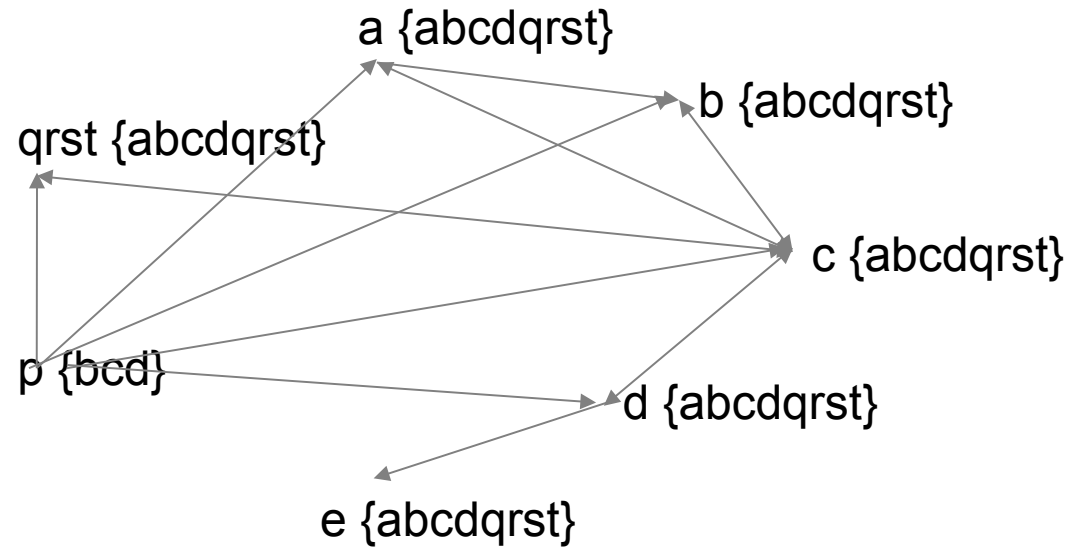
Andersen: Iteration 3



Processing order

```
*e = c (20)  
*a = p (0)  
c = *a (0)
```

Priority: Iteration 3

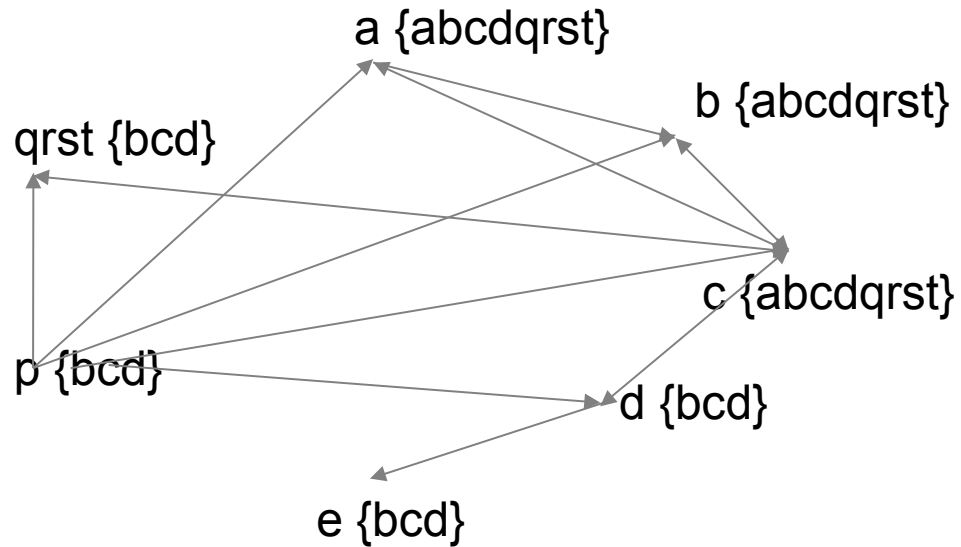


Prioritized Points-to Analysis

Fixed Processing order

```
*e = c  
c = *a  
*a = p
```

Andersen: Iteration 4



Processing order

```
*e = c (0)  
*a = p (0)  
c = *a (0)
```

Priority: fixed-point

