Program Analysis

Call Graphs (Part 4)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Winter 2020/2021

Slides adapted from Eric Bodden

Overview

- Introduction
- Single & efficient: CHA, RTA
- Analyzing assignments: VTA, DTA
- Call graphs and points-to analysis: Spark

Spark: Idea

- **RTA, DTA, and VTA: Instances of one single unifying framework**
- **General recipe**
  - First, built pointer-assignment graph (PAG)
  - Propagate information through graph
- **Combine call graph construction with points-to analysis**
  - Reason about objects a variable may refer to

Pointer-Assignment Graph (PAG)

- **Nodes**
  - Allocation
  - Variable
  - Field reference
- **Edges**
  - Allocation
  - Assignment
  - Field store
  - Field load

Pointer-Assignment Graph (PAG)

- **Nodes**
  - Allocation
  - Variable
  - Field reference
- **Edges**
  - Allocation
  - Assignment
  - Field store
  - Field load

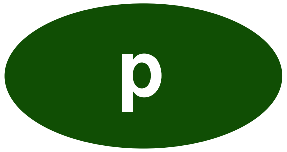
- **One for each `new A()`**
- **Represents a set of objects**
- **Has an associated type, e.g., `A`**

$alloc_1$

Pointer-Assignment Graph (PAG)

- **Nodes**
  - Allocation
  - Variable
  - Field reference
- **Edges**
  - Allocation
  - Assignment
  - Field store
  - Field load

- **One for each local variable, parameter, static field, and thrown exception**
- **Represents a memory location holding pointers to objects**
- **May be typed (depends on setting)**



Pointer-Assignment Graph (PAG)

- **Nodes**
  - Allocation
  - Variable
  - Field reference
- **Edges**
  - Allocation
  - Assignment
  - Field store
  - Field load

- **One for each p.f**
- **Represents a pointer deference**
- **Has a variable node as its base, e.g., p**
- **Also models contents of arrays: a.<elements>**

Pointer-Assignment Graph (PAG)

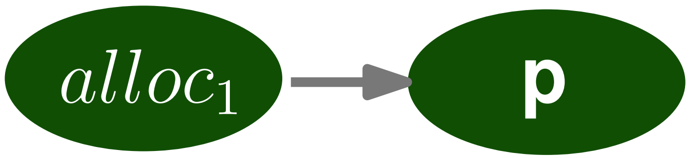
- **Nodes**
  - Allocation
  - Variable
  - Field reference
- **Edges**
  - Allocation
  - Assignment
  - Field store
  - Field load

- **Represents allocation of an object assigned to a variable**
- **E.g., for**

```
p = new HashMap();
```

or

```
s="foo";
```



Pointer-Assignment Graph (PAG)

- **Nodes**
  - Allocation
  - Variable
  - Field reference
- **Edges**
  - Allocation
  - Assignment
  - Field store
  - Field load

- **Represent assignments among variables and fields**
- **E.g., for**

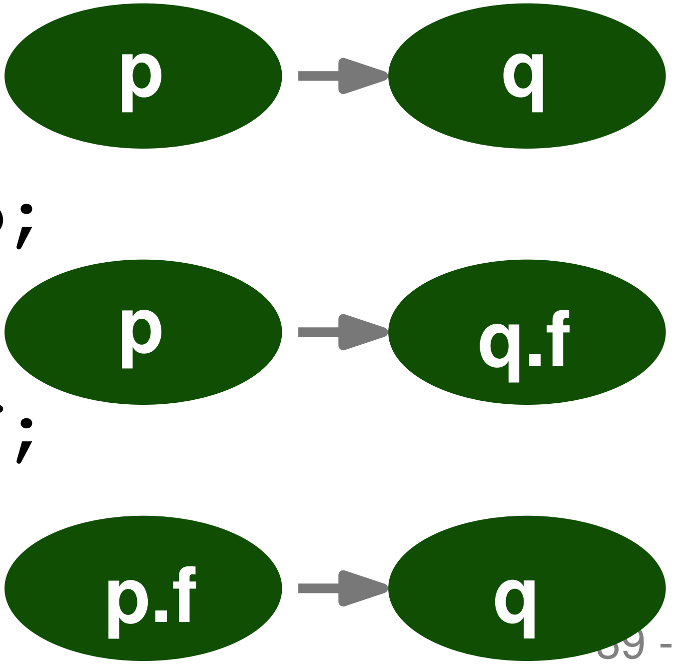
`q = p;`

or

`q.f = p;`

or

`q = p.f;`



Example

```
static void foo() {
  p = new A(); // alloc₁
  q = p;
  r = new B(); // alloc₂
  p.f = r;
  t = bar(q);
  t.m();
}

static C bar(C s) {
  return s.f;
}
```

$$
\begin{tikzcd}
\mathrm{alloc}_1 \arrow[d] & \mathrm{alloc}_2 \arrow[d] \\
p \arrow[d] & r \arrow[d] \\
q \arrow[d] & p.f \\
s & s.f \arrow[d] \\
 & t
\end{tikzcd}
$$

Points-to Sets

- **For each variable, set of objects the variable may refer to**
  - Objects represented as allocation nodes
- **Example:**

```
a = new X(); // alloc1
...
a = new Y(); // alloc2
```

$$pts(a) = \{alloc_1, alloc_2\}$$

Subset-based Analysis

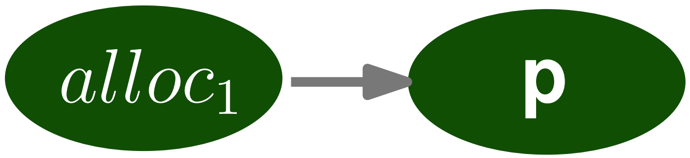
- **Allocation and assignment edges induce subset constraints**
  - Reason: Just because we know that

    ```
    p = new 1;
    ```

    does not mean that later we cannot see

    ```
    p = new 2;
    ```

- **Example:**



induces constraint

$$\{alloc_1\} \subseteq pts(p)$$

Subset-based Analysis

- **Allocation and assignment edges induce subset constraints**
  - Reason: Just because we know that

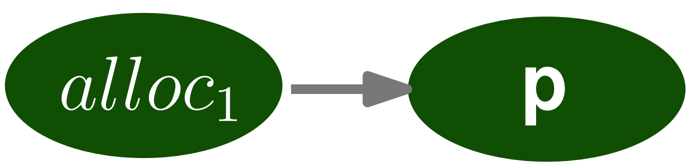
    ```
    p = new 1;
    ```

    does not mean that later we cannot see

    ```
    p = new 2;
    ```

Note: Analysis is flow-insensitive, i.e., values are never assumed to be overwritten

- **Example:**



induces constraint

$$\{alloc_1\} \subseteq pts(p)$$

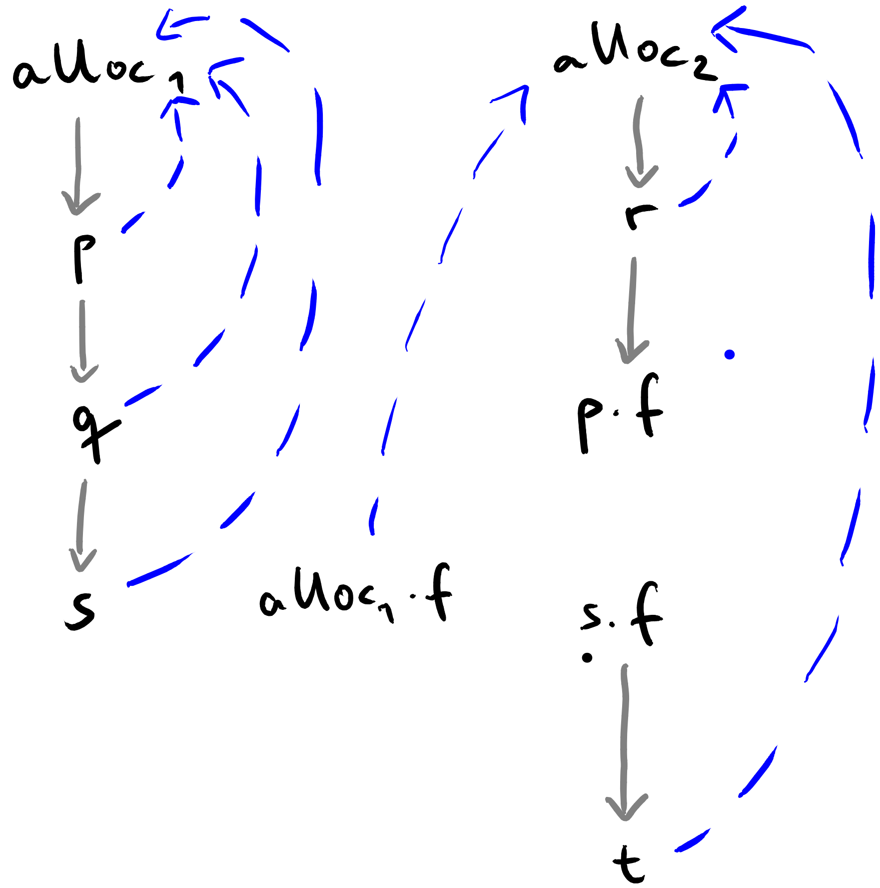
Computing Points-to Sets

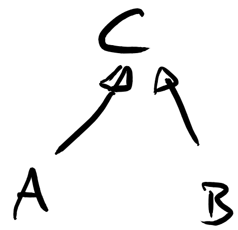
- **New helper node: Concrete fields**
- **Represents all objects pointed to by field `f` of all objects created at allocation site**
  - E.g., $alloc_1.f$

Computing Points-to Sets (2)

Iterative propagation algorithm

- Initialize $pts(v)$ according to allocation edges
- Repeat until no changes
  - Propagate sets along assignment edges $a \rightarrow b$
  - For each load edge $a.f \rightarrow b$:
    - For each $c \in pts(a)$, propagate $pts(c.f)$ to $pts(b)$
  - For each store edge $a \rightarrow b.f$:
    - For each $c \in pts(b)$, propagate $pts(a)$ to $pts(c.f)$





- call goes to B.m()

- - -> points-to

Simpler Variants

- **Spark framework supports many variants**
  - Just one allocation site per type
  - Fields simply represented by their signature
  - Equality instead of subsets for assignments
  - Etc.

Spark

- **Pros**
  - Generic algorithm where precision and efficiency can be tuned
  - Jointly computing call graph and points-to sets increases precision
- **Cons**
  - Still flow-insensitive
  - Can be quite expensive to compute