My Points-to Analysis

Introduction

Here, this is my algorithm of a inter-procedural context-, flow-, field-sensitive points-to analysis for Java.

I will leverage `Soot` and implement this algorithm in `cflow` to make its taint analysis more precise.

For more information about `cflow`, please see source and my notes.

For more information about static program analysis, please see my notes below:

- Lattice theory
- Interprocedural analysis
- IFDS
- Pointer analysis

Modeling

In `Soot`, we call intra-procedural analysis multiple times to simulate an inter-procedural analysis.

For each procedure, we have the lattice $L = States^n$, where $States = Vars \rightarrow location\ set$ and n is the number of nodes in CFG. The detailed info of L is shown as follows:

- **Element**: `Map<variable, set<abstract location>>` at each node in CFG
- **Order**: element $s_1 \sqsubseteq s_2$ **iff** $\forall$ variable $v, s_1(v) \subseteq s_2(v)$
- **Direction**: forward
- **Meet operator**: For current node n, we denote $JOIN(n)$ to union the points-to set of each variables among each predecessor node m

  $$JOIN(n) = \cup_{m \in pred(n)} [\![m]\!]$$

  where $[\![m]\!]$ is the map at node m.
- **Transfer function**:
  - For allocation statement `i : a = new T` at node n:

    $$[\![n]\!] = JOIN(n) \downarrow a\ \cup \{(a, alloc\_i)\}$$

    where $\sigma \downarrow x$ means killing the original points-to set of x:

    $$\sigma \downarrow x = \{(s,t) \in \sigma \mid s \neq x\}$$
  - For assignment statement `a = b` at node n:

    $$[\![n]\!] = assign(JOIN(n), a, b)$$

where $assign(\sigma, x, y)$ means replacing the points-to set of x with the points-to set of y.

$$assign(\sigma, x, y) = \sigma \downarrow x \ \cup \{(x, t) \mid (y, t) \in \sigma\}$$

- **Initial state**: If there is context, add the point-to set of `this object` and arguments for initialization.

Note that lattice L is not a map lattice(Although its elements are map), but it is a product lattice of each node in CFG.

Data Structures

I set `abstractLoc` to describe the abstract location of an object. `abstractLoc` has field:

- `method`: The method that the allocation site is in
- `callString`: The calling context of that method(k-limiting)
- `allocStmt`: The allocation statement
- `type`: the type of the object.

I build a global map `globalPointsToMap` that maps the variables v at each statement s in method m with context c to its possible points-to set of abstract locations.

```
method, context, statement, variable -> set of abstractLoc
```