软件分析

过程内指针分析

熊英飞
北京大学
2017

指向分析

- 每个指针变量可能指向的内存位置
- 通常是其他很多分析的基础

- 本节课先考虑流非敏感指向分析
- 先不考虑在堆上分配的内存，不考虑struct、数组等结构，不考虑指针运算（如*(p+1)）
  - 内存位置==局部和全局变量在栈上的地址

指向分析——例子

```
o=&v;
q=&p;
if (a > b) {
  p=*q;
  p=o; }
*q=&w;
```

- 指向分析结果
  - $\boldsymbol{p} = \{v, w\}$;
  - $\boldsymbol{q} =$?
  - $\boldsymbol{o} =$?

指向分析——例子

```
o=&v;
q=&p;
if (a > b) {
  p=*q;
  p=o; }
*q=&w;
```

- 指向分析结果
  - $\boldsymbol{p} = \{v, w\}$;
  - $\boldsymbol{q} = \{p\}$;
  - $\boldsymbol{o} = \{v\}$;
- 问题：如何设计一个指向分析算法？

复习：方程求解

- 数据流分析的传递函数和⊓操作定义了一组方程
  - $D_{v_1} = F_{v_1}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
  - $D_{v_2} = F_{v_2}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
  - …
  - $D_{v_n} = F_{v_n}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
- 其中
  - $F_{v_1}(D_{v_1}, D_{v_2}, \dots, D_{v_n}) = f_{v_1}(I)$
  - $F_{v_i}(D_{v_1}, D_{v_2}, \dots, D_{v_n}) = f_{v_i}\left(\sqcap_{j \in pred(i)} D_{v_j}\right)$
- 数据流分析即为求解该方程的最大解
  - 传递函数和⊓操作表达了该分析的安全性条件，所以该方程的解都是安全的
  - 最大解是最有用的解

从不等式到方程组

- 有一个有用的解不等式的unification算法
  - 不等式
    - $D_{v_1} \sqsubseteq F_{v_1}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
    - $D_{v_2} \sqsubseteq F_{v_2}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
    - …
    - $D_{v_n} \sqsubseteq F_{v_n}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
  - 可以通过转换成如下方程组求解
    - $D_{v_1} = D_{v_1} \sqcap F_{v_1}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
    - $D_{v_2} = D_{v_2} \sqcap F_{v_2}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
    - …
    - $D_{v_n} = D_{v_n} \sqcap F_{v_n}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$

Anderson指向分析算法

赋值语句	约束
a=&b	$\boldsymbol{a} \supseteq \{b\}$
a=b	$\boldsymbol{a} \supseteq \boldsymbol{b}$
a=*b	$\forall v \in \boldsymbol{b}.\, \boldsymbol{a} \supseteq \boldsymbol{v}$
*a=b	$\forall v \in \boldsymbol{a}.\, \boldsymbol{v} \supseteq \boldsymbol{b}$

其他语句可以转换成这四种基本形式

```
*a=**b;
```

```
c=*b;
d=*c;
*a=d;
```

Anderson指向分析算法-例

```
o=&v;
q=&p;
if (a > b) {
  p=*q;
  p=o; }
*q=&w;
```

- 产生约束
  - $\boldsymbol{o} \supseteq \{v\}$
  - $\boldsymbol{q} \supseteq \{p\}$
  - $\forall v \in \boldsymbol{q}. \boldsymbol{p} \supseteq \boldsymbol{v}$
  - $\boldsymbol{p} \supseteq \boldsymbol{o}$
  - $\forall v \in \boldsymbol{q}. \boldsymbol{v} \supseteq \{w\}$

- 如何求解这些约束

约束求解方法—通用框架

- 将约束
  - $\boldsymbol{o} \supseteq \{v\}$
  - $\boldsymbol{q} \supseteq \{p\}$
  - $\forall v \in \boldsymbol{q}. \boldsymbol{p} \supseteq \boldsymbol{v}$
  - $\boldsymbol{p} \supseteq \boldsymbol{o}$
  - $\forall v \in \boldsymbol{q}. \boldsymbol{v} \supseteq \{w\}$
- 转换成标准形式
  - $\boldsymbol{p} = \boldsymbol{p} \cup \boldsymbol{o} \cup \left(\bigcup_{v \in \boldsymbol{q}} \boldsymbol{v}\right) \cup (p \in \boldsymbol{q}\ ?\ \{w\} : \emptyset)$
  - $\boldsymbol{q} = \boldsymbol{q} \cup \{p\} \cup (q \in \boldsymbol{q}\ ?\ \{w\} : \emptyset)$
  - $\boldsymbol{o} = \boldsymbol{o} \cup \{v\} \cup (o \in \boldsymbol{q}\ ?\ \{w\} : \emptyset)$

  - 等号右边都是递增函数

求解方程组

- $\boldsymbol{p} = \boldsymbol{p} \cup \boldsymbol{o} \cup \left(\bigcup_{v \in \boldsymbol{q}} \boldsymbol{v}\right) \cup (p \in \boldsymbol{q}\ ?\ \{w\} : \emptyset)$
- $\boldsymbol{q} = \boldsymbol{q} \cup \{p\} \cup (q \in \boldsymbol{q}\ ?\ \{w\} : \emptyset)$
- $\boldsymbol{o} = \boldsymbol{o} \cup \{v\} \cup (o \in \boldsymbol{q}\ ?\ \{w\} : \emptyset)$

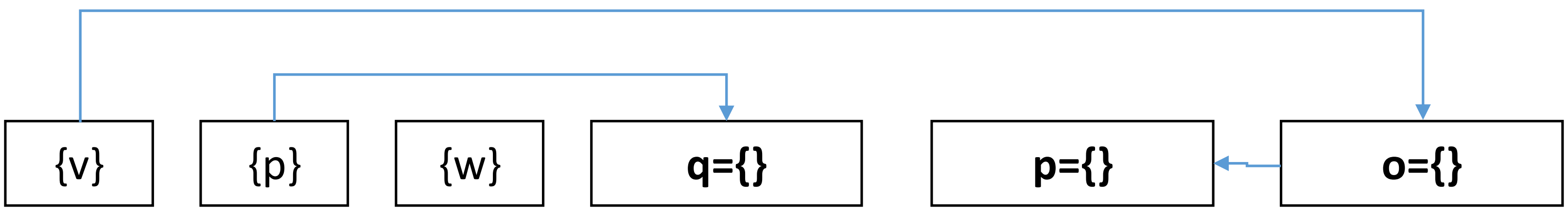
$\boldsymbol{p} = \{\ \ \}$
$\boldsymbol{q} = \{\ \ \}$
$\boldsymbol{o} = \{\ \ \}$

$\boldsymbol{p} = \{\ \ \}$
$\boldsymbol{q} = \{p\}$
$\boldsymbol{o} = \{v\}$

$\boldsymbol{p} = \{v, w\}$
$\boldsymbol{q} = \{p\}$
$\boldsymbol{o} = \{v\}$

约束求解方法—直接计算

- $\boldsymbol{o} \supseteq \{v\}$
- $\boldsymbol{q} \supseteq \{p\}$
- $\forall v \in \boldsymbol{q}.\,\boldsymbol{p} \supseteq \boldsymbol{v}$
- $\boldsymbol{p} \supseteq \boldsymbol{o}$
- $\forall v \in \boldsymbol{q}.\,\boldsymbol{v} \supseteq \{w\}$

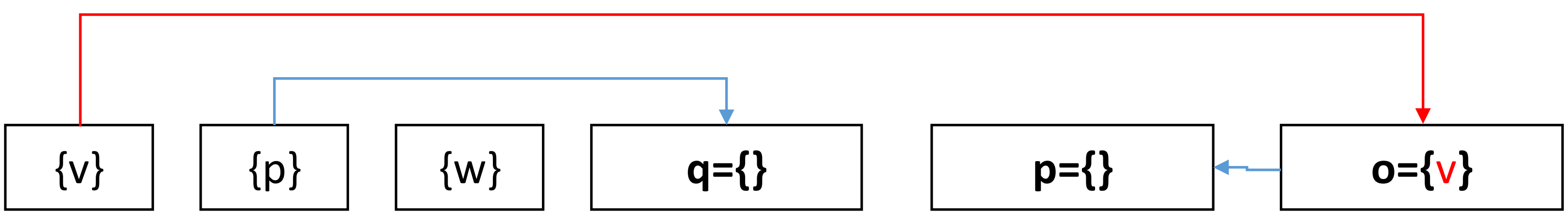


$$\forall v \in \boldsymbol{q}.\,\boldsymbol{p} \supseteq \boldsymbol{v}$$

$$\forall v \in \boldsymbol{q}.\,\boldsymbol{v} \supseteq \{w\}$$

约束求解方法—直接计算

- $\boldsymbol{o} \supseteq \{v\}$
- $\boldsymbol{q} \supseteq \{p\}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$
- $\boldsymbol{p} \supseteq \boldsymbol{o}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$

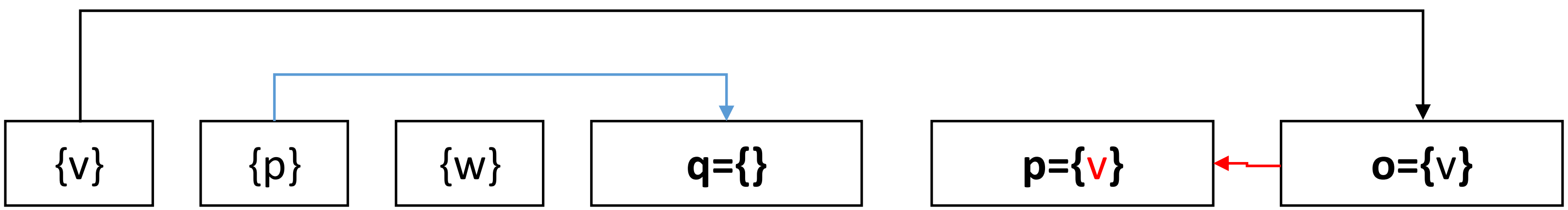


$$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$$

$$\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$$

约束求解方法—直接计算

- $\boldsymbol{o} \supseteq \{v\}$
- $\boldsymbol{q} \supseteq \{p\}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$
- $\boldsymbol{p} \supseteq \boldsymbol{o}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$

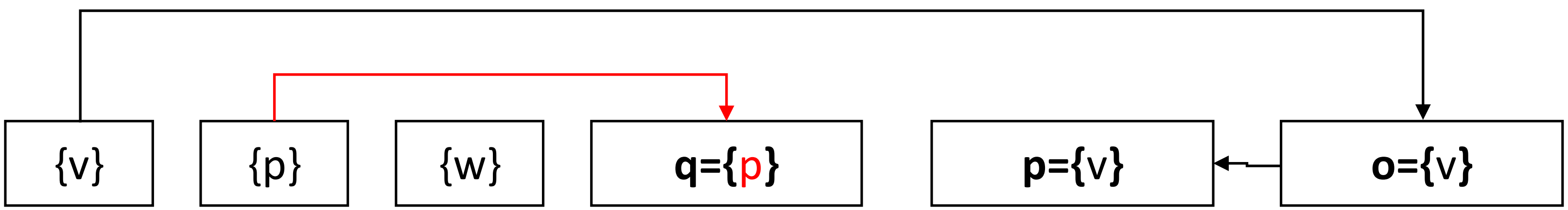


$$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$$

$$\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$$

约束求解方法—直接计算

- $\boldsymbol{o} \supseteq \{v\}$
- $\boldsymbol{q} \supseteq \{p\}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$
- $\boldsymbol{p} \supseteq \boldsymbol{o}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$

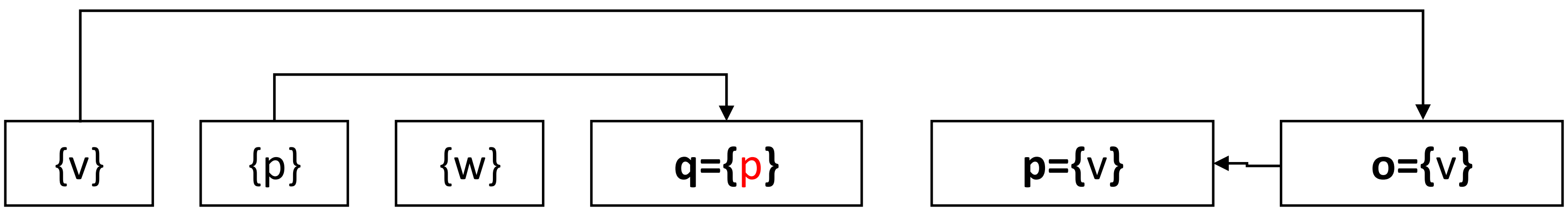


$$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$$

$$\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$$

约束求解方法—直接计算

- $\boldsymbol{o} \supseteq \{v\}$
- $\boldsymbol{q} \supseteq \{p\}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$
- $\boldsymbol{p} \supseteq \boldsymbol{o}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$

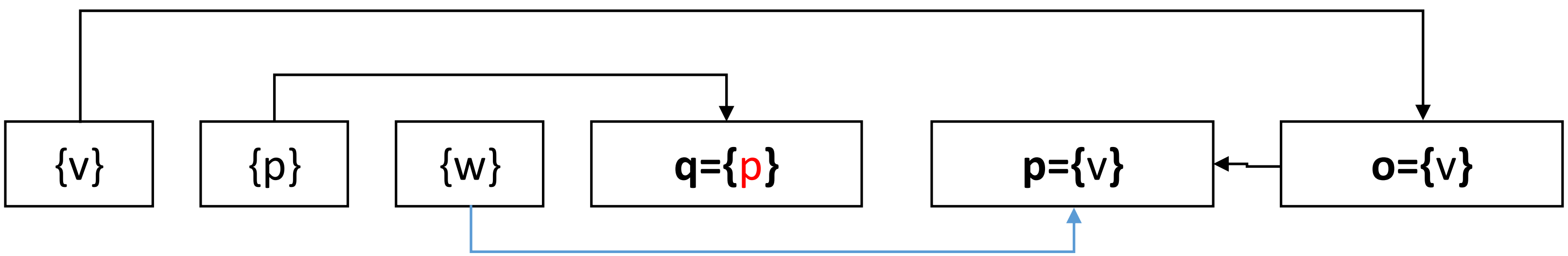


$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$

$\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$

约束求解方法—直接计算

- $\boldsymbol{o} \supseteq \{v\}$
- $\boldsymbol{q} \supseteq \{p\}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$
- $\boldsymbol{p} \supseteq \boldsymbol{o}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$

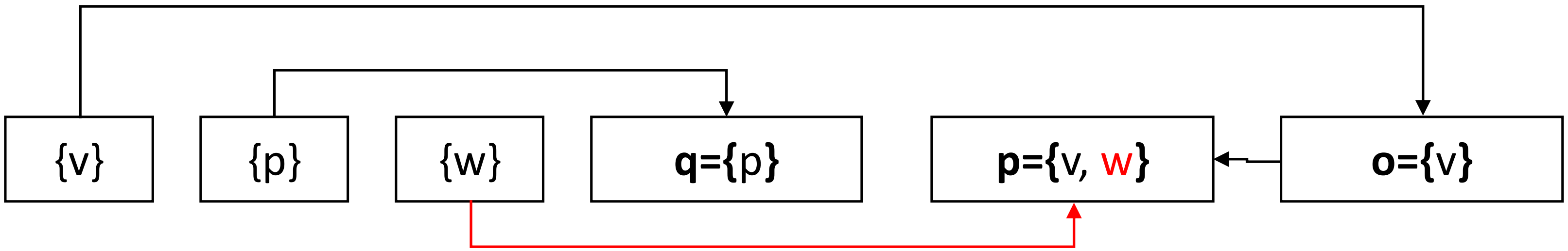


$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$

$\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$

约束求解方法—直接计算

- $\boldsymbol{o} \supseteq \{v\}$
- $\boldsymbol{q} \supseteq \{p\}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$
- $\boldsymbol{p} \supseteq \boldsymbol{o}$
- $\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$



$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$

$\forall v \in \boldsymbol{q}.\boldsymbol{v} \supseteq \{w\}$

复杂度分析

- 对于每条边来说，前驱集合新增元素的时候该边将被激活，激活后执行时间为O(m)，其中m为新增的元素数量
  - 应用均摊分析，每条边传递的总复杂度为O(n)，其中n为结点数量
- 边的数量为$O(n^2)$
- 总复杂度为$O(n^3)$

进一步优化

- 强连通子图中的每个集合必然相等
- 动态检测图中的强连通子图，并且合并成一个集合
- 参考论文：
    - The Ant and the Grasshopper: Fast and Accurate Pointer Analysis for Millions of Lines of Code, Hardekopf and Lin, PLDI 2007

流敏感的指针分析算法

- 能否通过SSA直接将流敏感转换成流非敏感分析？
  - 不能，参见SSA胶片最后部分
- 如何把Anderson算法转换成数据流分析？
  - 半格集合是什么？
    - 指针变量到内存位置集合的映射
  - 交汇操作是什么？
    - 对应内存位置集合取并
  - 四种基本操作对应的转换函数是什么？

流敏感的指针分析算法

赋值语句	转换函数
a=&b	$\boldsymbol{a_{out}} := \{b\}$
a=b	$\boldsymbol{a_{out}} := \boldsymbol{b_{in}}$
a=*b	$\boldsymbol{a_{out}} := \bigcup_{\forall v \in \boldsymbol{b}} \boldsymbol{v_{in}}$
*a=b	**?**

流敏感的指针分析算法

赋值语句	转换函数
a=&b	$\boldsymbol{a}_{\boldsymbol{out}} := \{b\}$
a=b	$\boldsymbol{a}_{\boldsymbol{out}} := \boldsymbol{b}_{\boldsymbol{in}}$
a=*b	$\boldsymbol{a}_{\boldsymbol{out}} := \bigcup_{\forall v \in \boldsymbol{b}} \boldsymbol{v}_{\boldsymbol{in}}$
*a=b	$\begin{cases} \forall v \in \boldsymbol{a}.\boldsymbol{v}_{\boldsymbol{out}} := \boldsymbol{b}_{\boldsymbol{in}} & \lvert \boldsymbol{a} \rvert = 1 \\ \forall v \in \boldsymbol{a}.\boldsymbol{v}_{\boldsymbol{out}} := \boldsymbol{v}_{\boldsymbol{in}} \cup \boldsymbol{b}_{\boldsymbol{in}} & \lvert \boldsymbol{a} \rvert > 1 \end{cases}$

Strong Update

Weak Update

流敏感的指针分析算法

- 传统流敏感的指针分析算法很慢
- 最新工作采用部分SSA来对流敏感进行加速，可以应用到百万量级的代码

- 参考论文：
  - Hardekopf B, Lin C. Flow-sensitive pointer analysis for millions of lines of code. CGO 2011:289-298.

堆上分配的内存

- a=malloc();
- malloc()语句每次执行创造一个内存位置
- 无法静态的知道malloc语句被执行多少次
  - 无法定义出有限半格
- 应用Widening
  - 每个malloc()创建一个抽象内存位置

Struct

```
Struct Node {
  int value;
  Node* next;
  Node* prev;
};
a = malloc();
a->next = b;
a->prev = c;
```

- 如何处理结构体的指针分析？
- 域非敏感Field-Insensitive分析
- 基于域的Field-Based分析
- 域敏感Field-sensitive分析

域非敏感Field-Insensitive分析

```
Struct Node {
  int value;
  Node* next;
  Node* prev;
};
a = malloc();
a->next = b;
a->prev = c;
```

- 把所有struct中的所有fields当成一个对象
- 原程序变为
  - a’=malloc();
  - a’=b;
  - a’=c;
  - 其中a’代表a， a->next， a->prev
- 分析结果
  - a, a->next, a->prev都有可能指向malloc(), b和c

基于域的Field-Based分析

```
Struct Node {
  int value;
  Node* next;
  Node* prev;
};
a = malloc();
a->next = b;
a->prev = c;
b = malloc();
b->next = c;
```

- 把所有对象的特定域当成一个对象
- 原程序变为
  - a=malloc();
  - next=b;
  - prev=c;
  - b=malloc();
  - next = c;
- 分析结果
  - a和a->prev是精确的，但a->next和b->next都指向b和c

域敏感Field-sensitive分析

```
Struct Node {
  int value;
  Node* next;
  Node* prev;
};
a = malloc();
a->next = b;
a->prev = c;
```

- 对于Node类型的内存位置x，添加两个指针变量
  - x.next
  - x.prev
- 对于任何Node类型的内存位置x，拆分成四个内存位置
  - x
  - x.value
  - x.next
  - x.prev
- a->next = b转换成
  - $\forall x \in \boldsymbol{a},\ \ \boldsymbol{x.next} \supseteq \boldsymbol{b}$

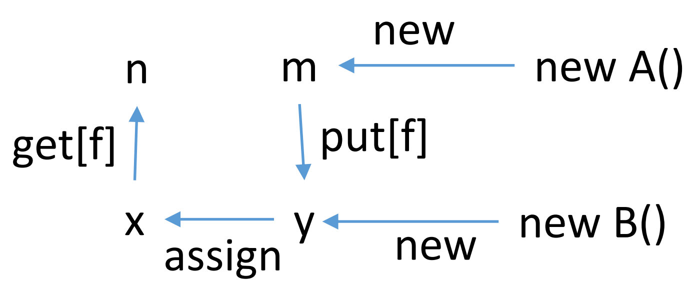
Java上的指向分析

- Java上的指向分析可以看成是C上的子集

Java	C
A a = new A()	A* a = malloc(sizeof(A));
a.next = b	a->next = b
b = a.next	b = a->next

基于CFL可达性的域敏感分析

y = new B();
m=new A();
x=y;
y.f=m;
n=x.f;



图上的每条边f同时存在反向边$\underline{\text{f}}$

FlowTo= new (assign | put[f] Alias get[f])*
PointsTo = ($\underline{\text{assign}}$ | $\underline{\text{get[f]}}$ Alias $\underline{\text{put[f]}}$)* $\underline{\text{new}}$
Alias = PointsTo FlowTo

基于CFL和基于Anderson算法的域敏感分析等价性

基于CFL	基于Anderson算法
$x \xrightarrow{\text{PointsTo}} m$	$m \in \mathbf{x}$
$m \xrightarrow{\text{FlowsTo}} x$	$m \in \mathbf{x}$
$x \xrightarrow{\text{Alias}} y$	$\mathbf{x} \cap \mathbf{y} \neq \emptyset$
$\exists y.\, y \xrightarrow{\text{PointsTo}} m \wedge y \xrightarrow{\text{puts[f] PointsTo}} n$	$n \in \mathbf{m}.\mathbf{f}$

归纳证明 以上各行左右的等价性

- 从左边推出右边：在CFL的路径长度上做归纳
- 从右边推出左边：在集合的元素个数上做归纳

数组和指针运算

- 从本质上来讲都需要区分数组中的元素和分析下标的值
  - p[i], *(p+i)
- 大多数框架提供的指针分析算法不支持数组和指针运算
  - 一个数组被当成一个结点