软件分析

过程内指针分析

熊英飞
北京大学

指向分析

- 每个指针变量可能指向的地址
- 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)）
  - 地址==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
  - $\boldsymbol{p} = \{v, w\}$;
  - $\boldsymbol{q} = ?$
  - $\boldsymbol{o} = ?$

p：变量p的地址
$\boldsymbol{p}$：指针p所指向的集合

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：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
    - $D_{v_1} \sqsupseteq F_{v_1}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
    - $D_{v_2} \sqsupseteq F_{v_2}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
    - …
    - $D_{v_n} \sqsupseteq F_{v_n}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
  - 可以通过转换成如下方程组求解
    - $D_{v_1} = D_{v_1} \sqcup F_{v_1}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
    - $D_{v_2} = D_{v_2} \sqcup F_{v_2}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
    - …
    - $D_{v_n} = D_{v_n} \sqcup F_{v_n}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$

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	$\forall v \in \boldsymbol{a}.\, \boldsymbol{v} \supseteq \boldsymbol{b}$

a：变量a的地址
$\boldsymbol{a}$：指针a所指向的集合

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
  - $\boldsymbol{o} \supseteq \{v\}$
  - $\boldsymbol{q} \supseteq \{p\}$
  - $\forall v \in \boldsymbol{q}. \boldsymbol{p} \supseteq \boldsymbol{v}$
  - $\boldsymbol{p} \supseteq \boldsymbol{o}$
  - $\forall v \in \boldsymbol{q}. \boldsymbol{v} \supseteq \{w\}$
- 约束带全称量词，如何求解？

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

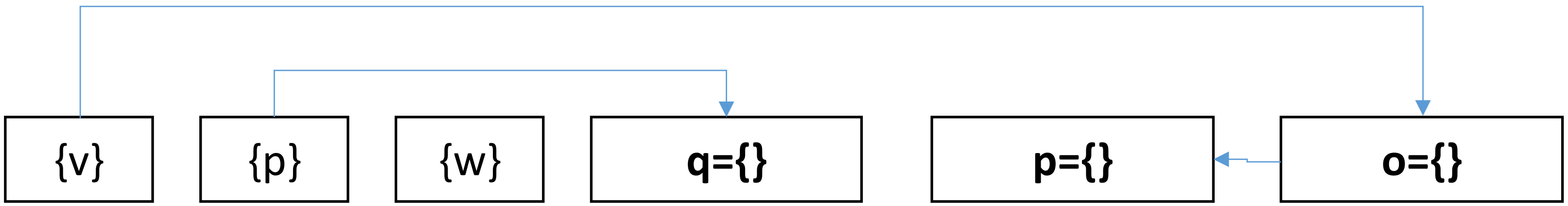
- $\boldsymbol{p} = \boldsymbol{p} \cup \boldsymbol{o} \cup \left(\bigcup_{v\in \boldsymbol{q}} \boldsymbol{v}\right) \cup (p \in \boldsymbol{q}\, ?\, \{w\} : \emptyset)$
- $\boldsymbol{q} = \boldsymbol{q} \cup \{p\} \cup (q \in \boldsymbol{q}\, ?\, \{w\} : \emptyset)$
- $\boldsymbol{o} = \boldsymbol{o}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

$$\begin{aligned}\boldsymbol{p} &= \{\ \ \}\\ \boldsymbol{q} &= \{\ \ \}\\ \boldsymbol{o} &= \{\ \ \}\end{aligned} \quad\Rightarrow\quad \begin{aligned}\boldsymbol{p} &= \{\ \ \}\\ \boldsymbol{q} &= \{p\}\\ \boldsymbol{o} &= \{v\}\end{aligned} \quad\Rightarrow\quad \begin{aligned}\boldsymbol{p} &= \{v, w\}\\ \boldsymbol{q} &= \{p\}\\ \boldsymbol{o} &= \{v\}\end{aligned}$$

但每次都重新计算所有公式效率不高，
某个集合变化后，更新受影响的集合即可。

约束求解方法—直接计算

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

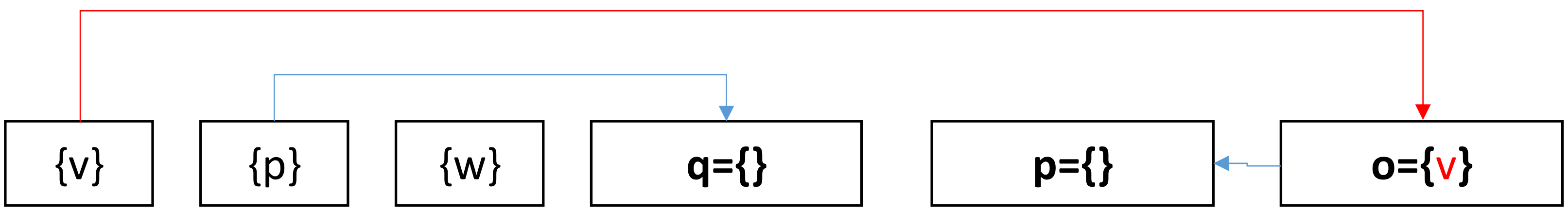


$$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$$

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

约束求解方法—直接计算

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

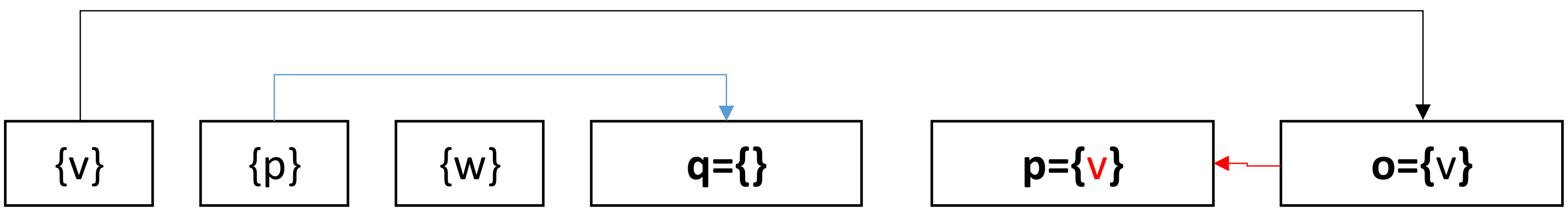


$$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$$

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

约束求解方法—直接计算

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

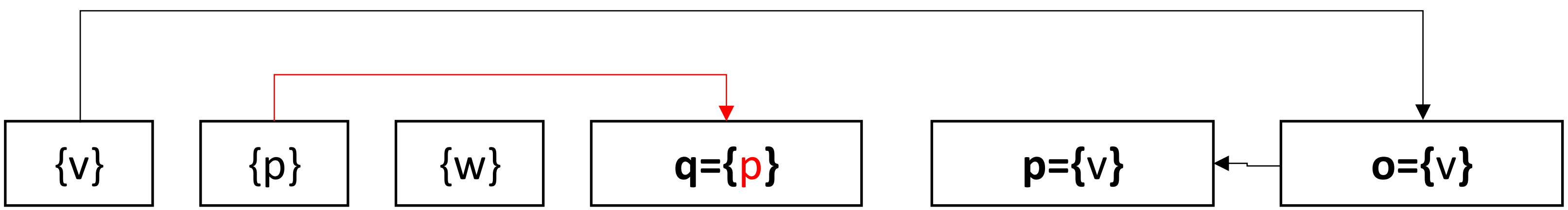


$$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$$

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

约束求解方法—直接计算

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

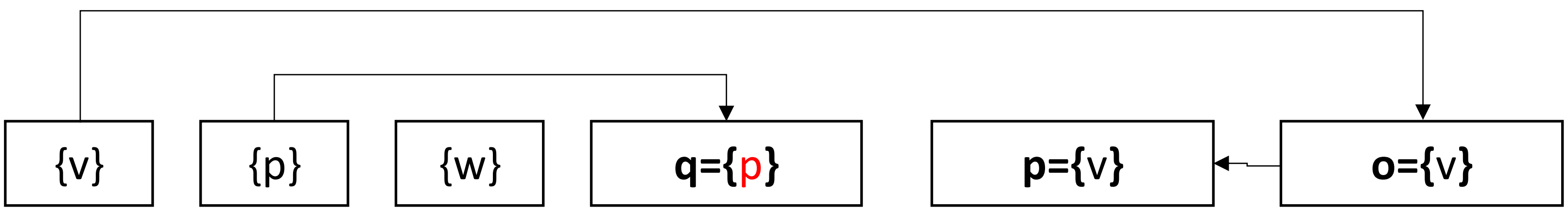


$$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$$

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

约束求解方法—直接计算

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

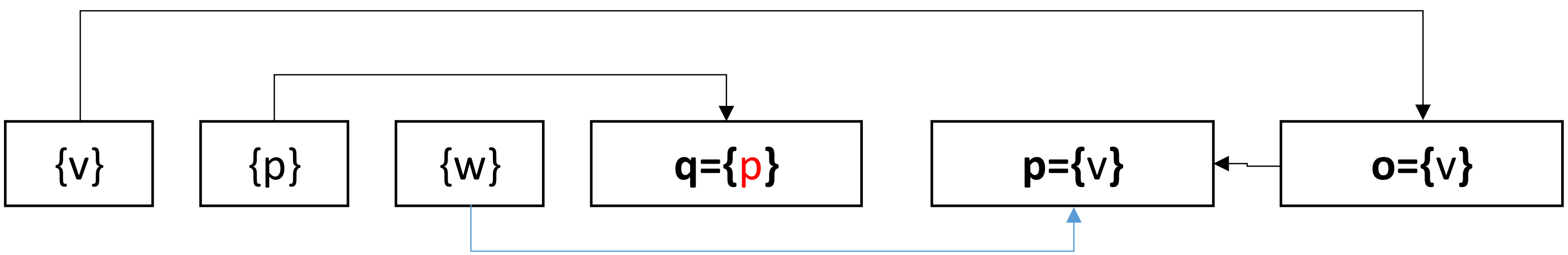


$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$

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

约束求解方法—直接计算

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

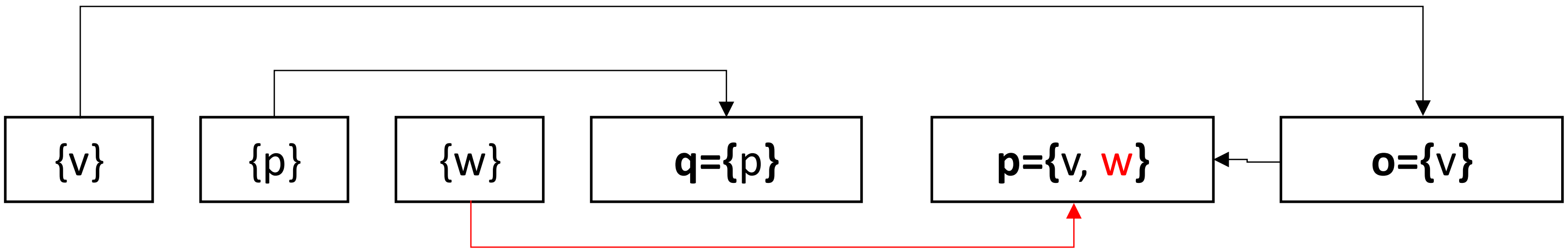


$\forall v \in \boldsymbol{q}.\boldsymbol{p} \supseteq \boldsymbol{v}$

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

约束求解方法—直接计算

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



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

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

抽象解释角度

- 抽象域：从指针变量到变量地址集合的映射
- 抽象值含义：代表执行轨迹的集合，在执行过程中所有指针指向了至少一次任意集合中的变量地址

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

流敏感的指针分析算法

- 如何把Anderson算法转换成数据流分析?
  - 半格集合是什么?
    - 指针变量到地址集合的映射
  - 交汇操作是什么?
    - 对应地址集合取并
  - 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	$f(V) = V[\boldsymbol{a} \mapsto \{b\}]$
a=b	$f(V) = V[\boldsymbol{a} \mapsto \boldsymbol{b}]$
a=*b	$f(V) = V\left[\boldsymbol{a} \mapsto \bigcup_{\forall v \in \boldsymbol{b}} \boldsymbol{v}\right]$
*a=b	**?**

流敏感的指针分析算法

赋值语句	转换函数
a=&b	$f(V) = V[\boldsymbol{a} \mapsto \{b\}]$
a=b	$f(V) = V[\boldsymbol{a} \mapsto \boldsymbol{b}]$
a=*b	$f(V) = V\left[\boldsymbol{a} \mapsto \bigcup_{\forall v \in \boldsymbol{b}} \boldsymbol{v}\right]$
*a=b	$f(V) = \begin{cases} \forall v \in \boldsymbol{a}. V[\boldsymbol{v} \mapsto \boldsymbol{b}] & \lvert\boldsymbol{a}\rvert = 1 \\ \forall v \in \boldsymbol{a}. V[\boldsymbol{v} \mapsto \boldsymbol{v} \cup \boldsymbol{b}] & \lvert\boldsymbol{a}\rvert > 1 \end{cases}$

Strong Update

Weak Update

注：这里不考虑变量没有被初始化的情况

流敏感的指针分析算法

- 流敏感的指针分析算法很慢
- 实践中常采用稀疏分析
  - 后续课程介绍

堆上分配的内存

- a=malloc();
- malloc()语句每次执行创造一个地址
- 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
- 应用抽象的思想
  - 每个malloc()创建一个抽象地址
  - a=malloc(); //1
  - $f(V) = V[\boldsymbol{a} \mapsto \{1\}]$

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类型的地址x，添加两个指针变量
  - x.next
  - x.prev
- 对于任何Node类型的地址x，拆分成四个地址
  - 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

别名分析

- 给定两个变量a, b，判断这两个变量是否指向相同的地址，返回以下结果之一
  - a, b是must aliases：始终指向同样的位置
  - a, b是must-not aliases：始终不指向同样的位置
  - a, b是may aliases: 可能指向同样的位置，也可能不指向
- 别名分析结果可以从指向分析导出
  - 如果**a**=**b**且|**a**|=1，则a和b为must aliases
  - 如果**a**∩**b**=∅，则a和b为must-not aliases
  - 否则a和b为may aliases
- 别名分析实践中常直接从指向分析导出

数组

- 数组粉碎：用一个抽象值描述数组中所有元素
  - 最简单，被广泛用于指针分析
- 数组扩展：对数组的每一维建立一个变量保存抽象值
  - 抽象域必须要推出下标（如区间），否则变量数量无上限
  - 一般只针对最多前k个变量建立，可以加快分析速度，同时处理数组大小未知或上限无限大的情况

数组

- 数组分区：抽象为从索引区间到抽象值的映射
  - 执行int a[100]，初始化a:{[0,99] →⊥}
  - 执行a[i]=5，其中i的抽象值是[2,5]
    - a:{[0,1] →⊥, [2,5] → [5,5], [6,99] →⊥}
  - 执行a[j]=3，其中j的抽象值是[5,6]
    - a:{[0,1] →⊥, [2,4] → [5,5], [5,5] → [3,5], [6,6] → [3,3], [7,99] → ⊥}
    - 注意需要执行弱更新
  - 抽象域必须要推出下标
  - 同时上界可以增加∞处理大小未知的情况

指针运算

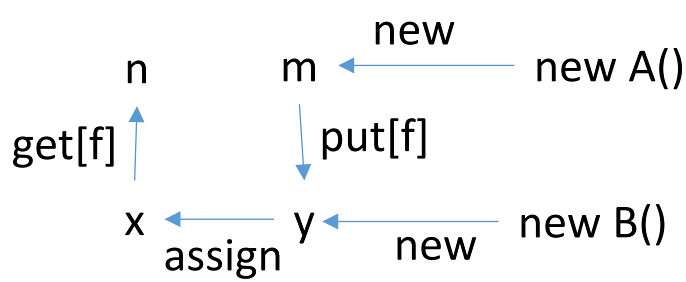
- 大多数实践中的分析工具不支持指针运算
  - 如果有指针运算，则直接返回unsound的结果。比如返回运算中第一个指针所指的值
- 部分分析针对特定形式的指针运算做处理
  - 如果指针运算是*(p+exp)的形式的话，指针运算本质上和数组访问是一回事

基于CFL可达性的域敏感分析

- 域敏感指针分析也可以转换为CFL可达性问题
- 实际分析效果和域敏感的Anderson分析等价，但可以借用高效的CFL求解器来实现，实现可能更简单

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;

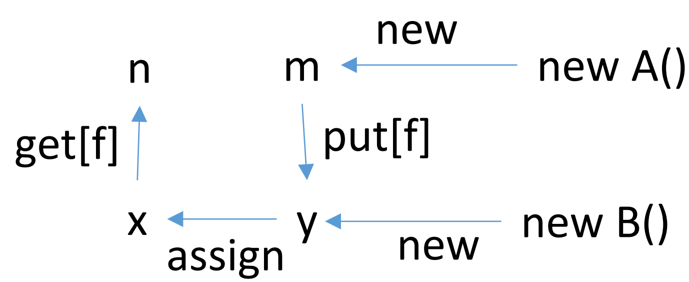


图上的每条边f同时存在反向边f

FlowTo= new (assign | put[f] Alias get[f])*
PointsTo = (assign | get[f] Alias put[f])* new
Alias = PointsTo FlowTo

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;
```



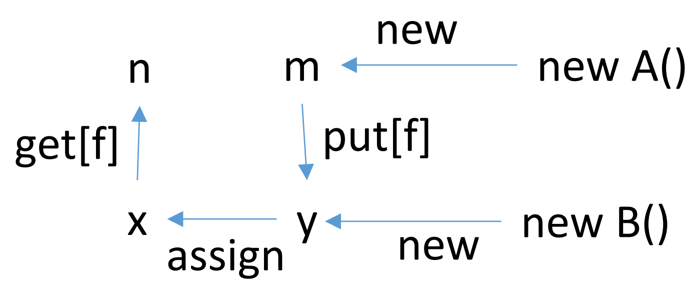
图上的每条边f同时存在反向边f

FlowTo= new (assign | put[f] Alias get[f])*
PointsTo = (assign | get[f] Alias put[f])* new
Alias = PointsTo FlowTo

能否不定义Alias关系？比如：
FlowTo = new FlowTo’
FlowTo’ = put[f] FlowTo’ get[f]
| FlowTo’ FlowTo’ | assign | ϵ

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$\underline{f}$

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

> 能否把assign统一定义在Alias内部？如：
>
> FlowTo= new (Alias | put[f] Alias get[f])*
>
> PointsTo = (Alias | $\underline{\text{get[f]}}$ Alias $\underline{\text{put[f]}}$)* $\underline{\text{new}}$
>
> Alias = PointsTo FlowTo | assign | $\underline{\text{assign}}$

进一步解读

- 刚刚的两个问题说明:
  - 当x和y之间有别名关系的时候
  - x.f和y.f所指向的地址集合就始终相等
  - 但x和y所指向的地址集合不一定相等
- 这也是设计域敏感分析的一个易错点

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

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

作业

- 流敏感的Anderson指针分析满足分配性吗?

参考文献

- Lecture Notes on Pointer Analysis
  - Jonathan Aldrich
  - https://www.cs.cmu.edu/~aldrich/courses/15-819O-13sp/resources/pointer.pdf
- 《编译原理》12章