软件分析

多角度理解程序分析

熊英飞
北京大学

程序分析的分类-敏感性

- 一般而言，抽象过程中考虑的信息越多，程序分析的精度就越高，但分析的速度就越慢
- 程序分析中考虑的信息通常用敏感性来表示
  - 流敏感性flow-sensitivity
  - 路径敏感性path-sensitivity
  - 上下文敏感性context-sensitivity
  - 字段敏感性field-sensitivity
- 注意区别：
  - 敏感性 vs 分析结果的形式
  - 抽象域的值可以进一步映射为想要的分析结果

术语-流敏感(flow-sensitivity)

- 流非敏感分析（flow-insensitive analysis）：如果把程序中语句随意交换位置（即：改变控制流），如果分析结果始终不变，则该分析为流非敏感分析。
- 流敏感分析（flow-sensitive analysis）：其他情况

- 数据流分析通常为流敏感的

流非敏感区间分析举例

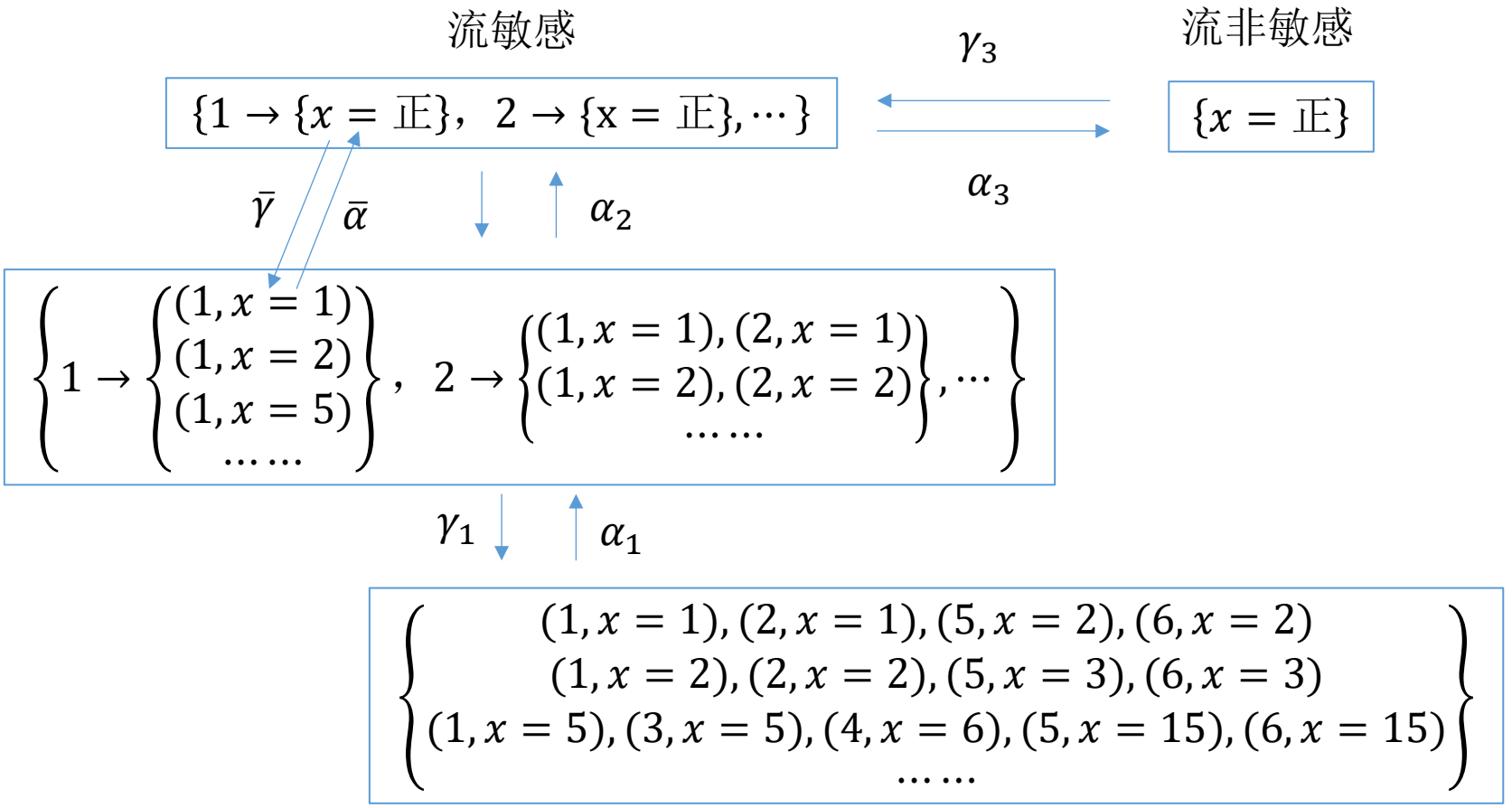
```
If (…)
  x = 0;
  y = x;
else
  x = 1;
  y = x - 1;
```

- 求程序执行过程中x和y所有可能取值的范围
- 流敏感分析：x:[0, 1]，y:[0, 0]
- 流非敏感分析：x:[0, 1]，y:[-1, 1]

流非敏感分析

- 不区分不同节点上的OUT值，我们就得到了流非敏感分析
  - $F_{fi}(OUT) = \sqcup_{v \in V} f_v(OUT)$
- 对比流敏感分析
  - $\mathrm{F}(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n}) =$
    $(\mathrm{f}_{\mathrm{v}_1}(\sqcup_{\mathrm{w} \in \mathrm{pred}(\mathrm{v}_1)} \mathrm{OUT}_{\mathrm{w}}),$
    $\mathrm{f}_{\mathrm{v}_2}(\sqcup_{\mathrm{w} \in \mathrm{pred}(\mathrm{v}_2)} \mathrm{OUT}_{\mathrm{w}}),$
    $\ldots,$
    $\mathrm{f}_{\mathrm{v}_\mathrm{n}}(\sqcup_{\mathrm{w} \in \mathrm{pred}(\mathrm{v}_\mathrm{n})} \mathrm{OUT}_{\mathrm{w}}))$
- 可以定义流非敏感结果和流敏感结果之间的伽罗瓦连接。
  - $\alpha(\mathrm{OUT}_{\mathrm{v}_1}, \mathrm{OUT}_{\mathrm{v}_2}, \ldots, \mathrm{OUT}_{\mathrm{v}_\mathrm{n}}) = \sqcup_{\mathrm{v} \in \mathrm{V}} \mathrm{OUT}_{\mathrm{v}}$
  - $\gamma(\mathrm{OUT}) = (\mathrm{OUT}, \mathrm{OUT}, \ldots, \mathrm{OUT})$
- 容易看出， F_{fi}是F的安全抽象。

流敏感vs流非敏感


流敏感
流非敏感
γ_3
$\{1 \to \{x = 正\},\ 2 \to \{x = 正\}, \cdots\}$
$\{x = 正\}$
α_3
$\bar{\gamma}$
$\bar{\alpha}$
α_2
$\left\{1 \to \begin{Bmatrix}(1, x = 1)\\(1, x = 2)\\(1, x = 5)\\\cdots\cdots\end{Bmatrix},\ 2 \to \begin{Bmatrix}(1, x = 1), (2, x = 1)\\(1, x = 2), (2, x = 2)\\\cdots\cdots\end{Bmatrix}, \cdots\right\}$
γ_1
α_1
$\left\{\begin{matrix}(1, x = 1), (2, x = 1), (5, x = 2), (6, x = 2)\\(1, x = 2), (2, x = 2), (5, x = 3), (6, x = 3)\\(1, x = 5), (3, x = 5), (4, x = 6), (5, x = 15), (6, x = 15)\\\cdots\cdots\end{matrix}\right\}$

流非敏感分析

- 实际中的流非敏感分析通常针对分析进行适当化简

```
a=100;
if(a>0)
  a=a+1;
b=a+1;
```

流非敏感符号分析

$F(a,b)$
$= (a \sqcup 正 \sqcup a + 正,$
$b \sqcup a + 正)$

按变量组织转换函数

流非敏感活跃变量分析

$OUT = OUT \cup \{a\}$

如果某节点的KILL中的变量在任意节点的GEN中，则该变量永远不会被删除，如果不在任意节点的GEN中，则该变量永远不会被添加。所以可以直接忽略KILL。

时间空间复杂度

- 活跃变量分析：语句数为n，程序中变量个数为m，使用bitvector表示集合
- 流非敏感的活跃变量分析：每条语句对应一个并集操作，时间为O(m)，迭代一轮即收敛，因此时间复杂度上界为O(nm)，空间复杂度上界为O(m)
- 流敏感的活跃变量分析：格的高度为O(m)，即每个结点的值最多变化O(m)次。每个结点有最多O(n)个后继节点，即每个结点的值最多被更新O(mn)次。每次有后继结点变化可以只合并变化的结点，因此单个均摊之后结点总更新复杂度$O(nm^2)$，总时间复杂度上界$O(n^2m^2)$，空间复杂度上界为O(nm)

- 对于特定分析，流非敏感分析能到达很快的处理速度和可接受的精度（如基于SSA的指针分析）

路径敏感性

- 路径非敏感分析：假设所有分支都可达，忽略分支循环语句中的条件
- 路径敏感分析：考虑程序中的路径可行性，尽量只分析可能的路径

- 带条件压缩函数的分析就是路径敏感分析

Datalog

- Datalog——逻辑编程语言Prolog的子集
- 一个Datalog程序由如下规则组成：
  - predicate1(Var or constant list) :- predicate2(Var or constant list),predicate3(Var or constant list), …
  - predicate(constant list)

- 如：
  - grandmentor(X, Y) :- mentor(X, Z), mentor(Z, Y)
  - mentor(kongzi, mengzi)
  - mentor(mengzi, xunzi)

- Datalog程序的语义
  - 反复应用规则，直到推出所有的结论——即不动点算法
  - 上述例子得到grandmentor(kongzi, xunzi)

从逻辑编程角度看程序分析

- 一个Datalog编写的正向数据流分析标准型，假设并集
  - out(D, V) :- gen(D, V)
  - out(D, V) :- edge(V’, V), out(D, V’), not_kill(D, V)
  - out(d, entry) // if d ∈ I
  - V表示结点，D表示一个集合中的元素

练习：交集的情况怎么写?

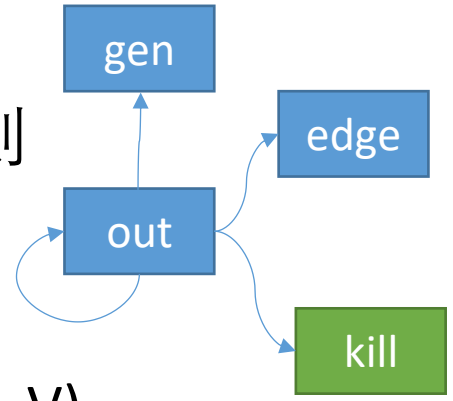
- out(D, V) :- gen(D, V)
- out(D, v) :- out(D, v_1), out(D, v_2), ..., out(D, v_n), not_kill(D, v) //$v_1, v_2, \ldots v_n$是v的前驱结点
- out(d, entry) // if $d \in I$

Datalog¬

- not_kill关系的构造效率较低
- 理想写法：
  - out(D, V) :- edge(V’, V), out(D, V’), not kill(D, V)
- 但是，引入not可能带来矛盾
  - p(x) :- not p(x)
  - 不动点角度理解：单次迭代并非一个单调函数

Datalog¬

- 解决方法：分层(stratified)规则
  - 谓词上的任何环状依赖不能包含否定规则
- 依赖示例
  - out(D, V) :- gen(D, V)
  - out(D, V) :- edge(V', V), out(D, V'), not kill(D, V)
  - out(d, entry)



- 不动点角度理解：否定规则将谓词分成若干层，每层需要计算到不动点，多层之间顺序计算
- 主流Datalog引擎通常支持Datalog¬

Datalog引擎

- Souffle
- LogicBlox
- IRIS
- XSB
- Coral
- 更多参考： https://en.wikipedia.org/wiki/Datalog

历史

- 大量的静态分析都可以通过Datalog简洁实现，但因为逻辑语言的效率，一直没有普及
- 2005年，斯坦福Monica Lam团队开发了高效Datalog解释器bddbddb，使得Datalog执行效率接近专门算法的执行效率
- 之后大量静态分析直接采用Datalog实现

方程求解

- 数据流分析的传递函数和⊓操作定义了一组方程
  - $\mathrm{OUT}_{v_1} = \mathrm{F}_{v_1}(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n})$
  - $\mathrm{OUT}_{v_2} = \mathrm{F}_{v_2}(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n})$
  - …
  - $\mathrm{OUT}_{v_n} = \mathrm{F}_{v_n}(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n})$
- 其中
  - $\mathrm{F}_{\mathrm{entry}}(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n}) = \mathrm{I}$
  - $\mathrm{F}_{v_i}(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n}) = f_{v_i}(\sqcup_{w \in \mathrm{pred}(v_i)} \mathrm{OUT}_w)$
- 数据流分析即为求解该方程的最小解
  - 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
  - 最小解是最精确的解

从不等式到方程组

- 不等式
  - $\mathrm{D}_{\mathrm{v}_1} \sqsupseteq \mathrm{F}_{\mathrm{v}_1}(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \ldots, \mathrm{D}_{\mathrm{v}_n})$
  - $\mathrm{D}_{\mathrm{v}_2} \sqsupseteq \mathrm{F}_{\mathrm{v}_2}(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \ldots, \mathrm{D}_{\mathrm{v}_n})$
  - …
  - $\mathrm{D}_{\mathrm{v}_n} \sqsupseteq \mathrm{F}_{\mathrm{v}_n}(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \ldots, \mathrm{D}_{\mathrm{v}_n})$
- 可以通过转换成如下方程组求解
  - $\mathrm{D}_{\mathrm{v}_1} = \mathrm{D}_{\mathrm{v}_1} \sqcup \mathrm{F}_{\mathrm{v}_1}(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \ldots, \mathrm{D}_{\mathrm{v}_n})$
  - $\mathrm{D}_{\mathrm{v}_2} = \mathrm{D}_{\mathrm{v}_2} \sqcup \mathrm{F}_{\mathrm{v}_2}(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \ldots, \mathrm{D}_{\mathrm{v}_n})$
  - …
  - $\mathrm{D}_{\mathrm{v}_n} = \mathrm{D}_{\mathrm{v}_n} \sqcup \mathrm{F}_{\mathrm{v}_n}(\mathrm{D}_{\mathrm{v}_1}, \mathrm{D}_{\mathrm{v}_2}, \ldots, \mathrm{D}_{\mathrm{v}_n})$

作业：

- 一个用Datalog编写的符号分析，在应用到下面程序上时，产生了一部分规则，请补全剩余的规则，并分析相比之前的符号分析，精度是否一样？如果不一样，请举一个例子。分析规则和结果中不出现踩和⊥。
  - 注：只是将如下程序手动转换成Datalog规则，不用编写针对任意程序通用的分析

```
1. x-=1;
2. y+=1;
3. while(y < z) {
4.   x *= -100;
5.   y += 1;}
```

输入：x为负，y为零，z为正
求输出的符号

```
out(x, entry, 负)
out(y, entry, 零)
out(z, entry, 正)
out(x, exit, ?)
out(y, exit, ?)
out(z, exit, ?)
edge(1, 2), edge(2, 3),
edge (5, 3),edge(3, 4),
edge(4, 5), edge(entry, 1),
edge(3, exit)
out(x, 1, 正) :- in(x, 1, 正)
out(x, 1, 零) :- in(x, 1, 正)
out(x, 1, 负) :- in(x, 1, 负)
out(x, 1, 负) :- in(x, 1, 零)
out(v, 3, 甲) :- in(v, 3, 甲)
```

参考资料

- Datalog Introduction
  - Jan Chomicki
  - https://cse.buffalo.edu/~chomicki/636/datalog-h.pdf
- Datalog引擎列表
  - https://en.wikipedia.org/wiki/Datalog