软件分析

多角度理解程序分析

熊英飞
北京大学

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

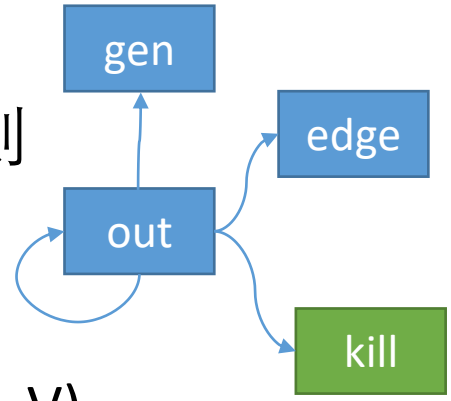
练习：交集的情况怎么写?

- 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
  - $F_{entry}(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n}) = I$
  - $F_{v_i}(\mathrm{OUT}_{v_1}, \mathrm{OUT}_{v_2}, \ldots, \mathrm{OUT}_{v_n}) = f_{v_i}(\sqcup_{w \in \mathrm{pred}(v_i)} \mathrm{OUT}_w)$
- 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

方程组求解算法

- 在数理逻辑学中，该类算法称为Unification算法
  - 参考：http://en.wikipedia.org/wiki/Unification_(computer_science)
- 对于单调函数和有限格，标准的Unification算法就是我们学到的数据流分析算法
  - 轮询算法：一次更新所有的OUT值
  - 工单算法：每次只更新一个受到影响的OUT_{v_i}值

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

作业：

- 使用任意Datalog引擎，用Datalog编写下面程序的符号分析，提交程序和运行截图
  - 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工具

```
x*=-100;
y+=1;
while(y < z) {
        x *= -100;
        y += 1;
}
```

输入：x为负，y为零，z为正
求输出的符号

参考资料

- 《Introduction to Static Analysis》Rival and Yi
- Datalog Introduction
  - Jan Chomicki
  - https://cse.buffalo.edu/~chomicki/636/datalog-h.pdf
- Datalog引擎列表
  - https://en.wikipedia.org/wiki/Datalog