



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
 - $\text{out}(D, V) :- \text{gen}(D, V)$
 - $\text{out}(D, V) :- \text{edge}(V', V), \text{out}(D, V'), \text{not_kill}(D, V)$
 - $\text{out}(d, \text{entry}) // \text{if } d \in I$
 - 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？

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



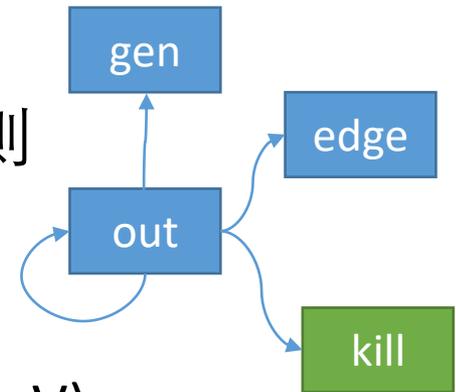
Datalog \neg

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



Datalog \neg

- 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 \neg





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bddbdb，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 \sqcap 操作定义了一组方程
 - $OUT_{v_1} = F_{v_1}(OUT_{v_1}, OUT_{v_2}, \dots, OUT_{v_n})$
 - $OUT_{v_2} = F_{v_2}(OUT_{v_1}, OUT_{v_2}, \dots, OUT_{v_n})$
 - ...
 - $OUT_{v_n} = F_{v_n}(OUT_{v_1}, OUT_{v_2}, \dots, OUT_{v_n})$
- 其中
 - $F_{entry}(OUT_{v_1}, OUT_{v_2}, \dots, OUT_{v_n}) = I$
 - $F_{v_i}(OUT_{v_1}, OUT_{v_2}, \dots, OUT_{v_n}) = f_{v_i}(\sqcup_{w \in pred(v_i)} OUT_w)$
- 数据流分析即为求解该方程的最小解
 - 传递函数和 \sqcup 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



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_\(computer_science\)](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
 - $D_{v_1} \supseteq F_{v_1}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
 - $D_{v_2} \supseteq F_{v_2}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
 - ...
 - $D_{v_n} \supseteq F_{v_n}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
 - 可以通过转换成如下方程组求解
 - $D_{v_1} = D_{v_1} \sqcup F_{v_1}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
 - $D_{v_2} = D_{v_2} \sqcup F_{v_2}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$
 - ...
 - $D_{v_n} = D_{v_n} \sqcup F_{v_n}(D_{v_1}, D_{v_2}, \dots, D_{v_n})$



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

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