软件分析

数据流分析：扩展

熊英飞
北京大学
2017

练习：可用表达式（available expression）分析

- 给定程序中某个位置p，如果从入口到p的所有结点都对表达式exp求值，并且最后一次求值后该表达式的所有变量都没有被修改，则exp称作p的一个可用表达式。给出分析寻找可用表达式。
  - 假设程序中没有指针、数据、引用、复合结构
  - 要求下近似
  - 例：

```
1.  a=c+(b+10);
2.  if (…)
3.    c = a+10;
4.  return a;
```

  - 1运行结束的时候可用表达式是b+10、c+(b+10)
  - 2运行结束的时候可用表达式是b+10、c+(b+10)
  - 3运行结束的时候可用表达式是b+10、a+10
  - 4运行结束的时候可用表达式是b+10

答案：可用表达式（available expression）分析

- 正向分析
- 半格元素：程序中任意表达式的集合
- 交汇操作：交集操作
- 变换函数：
  - 对于赋值语句v=...
    - KILL={所有包含v的表达式}
    - GEN={当前语句中求值的不含v的表达式}
  - 对于其他语句
    - KILL={}
    - GEN={当前语句中求值的表达式}

练习：区间（Internval）分析

- 求结果的上界和下界
  - 要求上近似
  - 假设程序中的运算只含有加减运算
  - 例：

```
1.  a=0;
2.  for(int i=0; i<b; i++)
3.    a=a+1;
4.  return a;
```

  - 结果为a:[0,+∞]

区间（Internval）分析

- 正向分析
- 半格元素：程序中每个变量的区间
- 交汇操作：区间的并
- 变换函数：
  - 在区间上执行对应的加减操作

- 不满足单调框架条件：半格不是有限的
  - 分析可能会不终止

Widening & Narrowing

Widening

- 从无限的空间中选择一些代表元素组成有限空间
- 定义单调函数w把原始空间映射到有限空间上
  - 应满足：$w(x) \sqsubseteq x$

- 定义有限集合{-∞, 10, 20, 50, 100, +∞}
- 定义映射函数

$$w([l,h]) = [max\{i \in B \mid i \leq l\}, min\{i \in B \mid h \leq i\}]$$

- 如：
  - $w([15,75]) = [10,100]$

Widening

- 原始转换函数f
- 新转换函数$w \circ f$

- 安全性讨论
  - 新转换仍然单调
  - 新转换结果小于等于原结果，意味着$DATA_V$的结果小于等于原始结果

Widening的问题

- Widening牺牲精确度来保证收敛性，有时该牺牲很大。
- 令有限集合为 $\{-\infty, 0, 1, 7, +\infty\}$

```
y = 0; x = 7; x = x+1;
while (input) {
  x = 7;
  x = x+1;
  y = y+1;
}
```

- while(input)处的结果变化

$$[\mathbf{x} \mapsto \bot, \mathbf{y} \mapsto \bot]$$
$$[\mathbf{x} \mapsto [8,8], \mathbf{y} \mapsto [0,1]]$$
$$[\mathbf{x} \mapsto [8,8], \mathbf{y} \mapsto [0,2]]$$
$$[\mathbf{x} \mapsto [8,8], \mathbf{y} \mapsto [0,3]]$$
$$\vdots$$

不使用Widening，不收敛

$$[\mathbf{x} \mapsto \bot, \mathbf{y} \mapsto \bot]$$
$$[\mathbf{x} \mapsto [7,\infty], \mathbf{y} \mapsto [0,1]]$$
$$[\mathbf{x} \mapsto [7,\infty], \mathbf{y} \mapsto [0,7]]$$
$$[\mathbf{x} \mapsto [7,\infty], \mathbf{y} \mapsto [0,\infty]]$$

使用Widening，不精确

Narrowing

- 通过再次应用原始转换函数对Widening的结果进行简单修正

```
y = 0; x = 7; x = x+1;
while (input) {
  x = 7;
  x = x+1;
  y = y+1;
}
```

可以得到结果

$$[\mathtt{x} \mapsto [8, 8], \mathtt{y} \mapsto [0, \infty]]$$

由于不能保证Narrowing的收敛性，通常应用有限次原始转换函数

从不同角度理解数据流分析

方程求解

- 数据流分析的传递函数和⊓操作定义了一组方程
  - $\mathrm{D_{v_1}} = \mathrm{F_{v_1}}(\mathrm{D_{v_1}}, \mathrm{D_{v_2}}, \ldots, \mathrm{D_{v_n}})$
  - $\mathrm{D_{v_2}} = \mathrm{F_{v_2}}(\mathrm{D_{v_1}}, \mathrm{D_{v_2}}, \ldots, \mathrm{D_{v_n}})$
  - …
  - $\mathrm{D_{v_n}} = \mathrm{F_{v_n}}(\mathrm{D_{v_1}}, \mathrm{D_{v_2}}, \ldots, \mathrm{D_{v_n}})$
- 其中
  - $F_{v_1}(\mathrm{D_{v_1}}, \mathrm{D_{v_2}}, \ldots, \mathrm{D_{v_n}}) = f_{v_1}(I)$
  - $F_{v_i}(\mathrm{D_{v_1}}, \mathrm{D_{v_2}}, \ldots, \mathrm{D_{v_n}}) = f_{v_i}\left(\sqcap_{j \in pred(i)} D_{v_j}\right)$
- 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

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
  - 从$(I, \top, \top, \ldots, \top)$开始反复应用$F_{v_1}$到$F_{v_n}$，直到达到不动点
  - 增量优化：每次只执行受到影响的F_{v_i}

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

流非敏感分析

- 转换成同样的方程组，并用不动点算法求解

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

流非敏感符号分析

$a = 正 \sqcap a + 正$

$b = a + 正$

不考虑位置，用所有赋值语句更新所有变量

流非敏感活跃变量分析

$\mathrm{DATA} = \mathrm{DATA} \cup \{a\}$

对于整个程序产生一个集合，只要程序中有读取变量v的语句，就将其加入集合

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：每条语句的操作时间为O(m)，因此时间复杂度上界为O(m^2)，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)，转移函数、交汇运算和比较运算都是O(m)，时间复杂度上界为O(nm^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）

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, constant1) :- predicate2(Var, constant2), predicate2(Var2, constant3)
  - predicate(constant)

- 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)

逻辑规则视角

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

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

Datalog¬

- not_kill关系的构造效率较低
- 理想写法：
  - data(D, V) :- edge(V', V), live(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
- 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
  - 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¬

小结

- 数据流分析的安全性可以通过最大下界的性质来证明
- 数据流分析的收敛性可以通过不动点定理来分析
- 数据流分析也可以看做是一个方程求解或者逻辑式程序运行的过程
- 可以通过Widening和Narrowing来处理无限半格的情况
  - 也可以用于加速收敛较慢的有限半格上的分析