软件分析

数据流分析：框架

熊英飞
北京大学
2017

数据流分析单调框架

- 数据流分析单调框架：对前面所述算法以及所有同类算法的一个通用框架
- 目标：通过配置框架的参数，可以导出各种类型的算法，并保证算法的安全性、终止性、收敛性
- 需要抽象的内容
  - 不同算法在结点上附加的值的类型不同，需要有一个统一接口
  - 不同算法给出的结点转换函数不同，需要有一个统一接口

半格（semilattice）

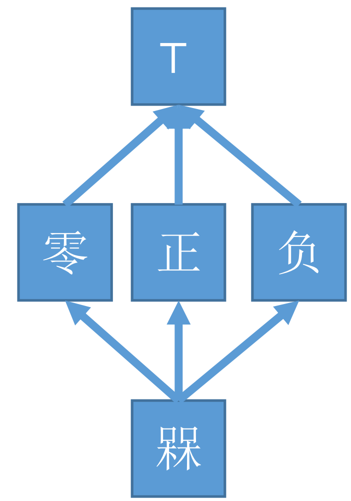
- 半格是一个二元组(S,⊓)，其中S是一个集合，⊓是一个交汇运算，并且任意$x, y, z \in S$都满足下列条件：
  - 幂等性idempotence: $x \sqcap x = x$
  - 交换性commutativity: $x \sqcap y = y \sqcap x$
  - 结合性associativity：$(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
  - 存在一个最大元⊤，使得$x \sqcap \top = x$

偏序Partial Order

- 偏序是一个二元组(S, ⊑)，其中S是一个集合，⊑是一个定义在S上的二元关系，并且满足如下性质：
  - 自反性：$\forall a \in S: a \sqsubseteq a$
  - 传递性：$\forall x, y, z \in S: x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z$
  - 非对称性：$x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$
- 每个半格都定义了一个偏序关系
  - $x \sqsubseteq y$当且仅当$x \sqcap y = x$

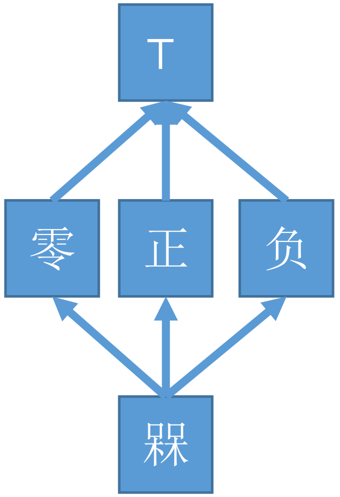
半格示例

- 抽象符号域的五个元素和交汇操作组成了一个半格
- 半格的笛卡尔乘积$(S \times T, \sqcap_{xy})$还是半格
  - $(s_1, t_1) \sqcap_{xy} (s_2, t_2) = \left(s_1 \sqcap_x s_2, t_1 \sqcap_y t_2\right)$
- 任意集合和交集操作组成了一个半格
  - 偏序关系为子集关系
  - 顶元素为全集
- 任意集合和并集操作组成了一个半格
  - 偏序关系为超集关系
  - 顶元素为空集



半格的高度

- 半格的偏序图中任意两个结点的最大距离+1
- 示例：
  - 抽象符号域的半格高度为3
  - 集合和交集/并集组成的半格高度为集合大小+1
    - 活跃变量分析中半格高度为变量总数+1



集合的最大下界

- 下界：给定集合S，如果满足$\forall s \in S: u \sqsubseteq s$，则称u是S的一个下界
- 最大下界：设u是集合S的下界，给定任意下界u'，如果满足$u' \sqsubseteq u$，则称u是S的最大下界，记为$\sqcap_S$
- 引理：$\sqcap_{s\in S} s$是S的最大下界
  - 证明：
    - 根据幂等性、交换性和结合性，我们有$\forall v \in S$: $(\sqcap_{s\in S} s) \sqcap v = \sqcap_{s\in S} s$，所以$\sqcap_{s\in S} s$是S的下界
    - 给定另一个下界u，我们有$\forall s \in \mathrm{S} : s \sqcap u = u$，$(\sqcap_{s\in S} s) \sqcap u = (\sqcap_{s\in S}(s \sqcap u)) = \mathrm{u}$，所以$\sqcap_{s\in S} s$是最大下界
- 推论：半格的任意子集都有最大下界

单调函数Monotone Function

- 给定一个偏序关系$(S, \sqsubseteq)$，称一个定义在S上的函数f为单调函数，当且仅当对任意$a, b \in S$满足
  - $a \sqsubseteq b \Rightarrow f(a) \sqsubseteq f(b)$
- 注意：单调函数与递增函数不同

- 单调函数示例
  - 在符号分析的半格中，固定任一输入参数，抽象符号的四个操作均为单调函数
  - 在集合和交/并操作构成的半格中，给定任意两个集合GEN, KILL，函数$f(S) = (S - KILL) \cup \mathrm{GEN}$为单调函数

数据流分析单调框架

- 一个控制流图(V, E)
- 一个有限高度的半格$(S,\sqcap)$
- 一个entry的初值I
- 一组结点转换函数，对任意$v \in V - entry$存在一个结点转换函数f_v

- 注意：对于逆向分析，变换控制流图方向再应用单调框架即可

数据流分析实现算法

$\mathrm{DATA}_{\mathrm{entry}} = \mathrm{I}$

$\forall \mathrm{v} \in (\mathrm{V} - \mathrm{entry}): \mathrm{DATA}_{\mathrm{v}} \leftarrow \top$

$\mathrm{ToVisit} \leftarrow \mathrm{V} - \mathrm{entry}$ //可以换成succ(entry)吗?

While(ToVisit.size > 0) {

  v ← ToVisit中任意结点

  ToVisit -= v

  $\mathrm{MEET}_{\mathrm{v}} \leftarrow \sqcap_{\mathrm{w} \in \mathrm{pred(v)}} \mathrm{DATA}_{\mathrm{w}}$

  $\mathrm{If}\left(\mathrm{DATA}_{\mathrm{V}} \neq \mathrm{f}_{\mathrm{v}}(\mathrm{MEET}_{\mathrm{v}})\right) \mathrm{ToVisit} \cup= \mathrm{succ(v)}$

  $\mathrm{DATA}_{\mathrm{v}} \leftarrow \mathrm{f}_{\mathrm{v}}(\mathrm{MEET}_{\mathrm{v}})$

}

数据流分析小结

- 应用单调框架设计一个数据流分析包含如下内容
  - 设计每个结点附加值的定义域
  - 设计交汇函数
  - 设计从语句导出结点变换函数的方法
  - 入口结点的初值
- 需要证明如下内容
  - 在单条路径上，变换函数保证安全性
  - 交汇函数对多条路径的合并方式保证安全性
  - 交汇函数形成一个半格
  - 半格的高度有限
    - 通常通过结点附加值的定义域为有限集合证明
  - 变换函数均为单调函数
    - 通常定义为$f(D) = (D - KILL) \cup \mathrm{GEN}$的形式

数据流分析的安全性-定义

- 安全性：对控制流图上任意结点v_i和所有从entry到v_i的路径集合P，满足$\mathrm{DATA}_{\mathrm{v_i}} \sqsubseteq \sqcap_{v_1 v_2 v_3 \dots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \cdots \circ f_{v_1}(I_{entry})$
  - 示例：符号分析的偏序关系中槑比较小，⊤比较大，结果是上近似
  - 示例：活跃变量分析的偏序关系为超集关系，所以数据流分析产生相等或者较大集合，是上近似

数据流分析的安全性-证明

- 给定任意路径的$v_1 v_2 v_3 \ldots v_i$，$\mathrm{DATA}_{\mathrm{v_i}}$的计算相当于在每两个相邻转换函数$f_{v_i} \circ f_{v_{i-1}}$之间加入了MEET交汇计算，根据幂等性，任意交汇计算的结果一定在偏序上小于等于原始结果。再根据转换函数的单调性，$\mathrm{DATA}_{\mathrm{v_i}}$的值一定小于等于$f_{v_i} \circ f_{v_{i-1}} \circ \cdots \circ f_{v_1}(I_{entry})$。由于原路径的任意性，$\mathrm{DATA}_{\mathrm{v_i}}$是一个下界。
- 再根据前面的引理，$\sqcap_{v_1 v_2 v_3 \ldots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \cdots \circ f_{v_1}(I_{entry})$是最大下界，所以原命题成立。

数据流分析的分配性

- 一个数据流分析满足分配性，如果
  - $\forall v \in V, x, y \in S: f_v(x) \sqcap f_v(y) = f_v(x \sqcap y)$
- 例：符号分析中的结点转换函数不满足分配性
  - 为什么？
  - 令f_v等于“乘以零”，$f_v(\text{正}) \sqcap f_v(\text{负})$
- 例：在集合和交/并操作构成的半格中，给定任意两个集合GEN, KILL，函数$f(DATA) = (DATA -$

数据流分析的分配性

- 一个数据流分析满足分配性，如果
  - $\forall v \in V, x, y \in S: f_v(x) \sqcap f_v(y) = f_v(x \sqcap y)$
- 当数据流分析满足分配性的时候，$\mathrm{DATA}_{\mathrm{v_i}} = \sqcap_{v_1 v_2 v_3 \ldots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \cdots \circ f_{v_1}(I_{entry})$
  - 也就是说，此时近似方案2不是近似，而是等价变换
  - 但是，数据流分析本身还可能是近似
    - 近似方案1是近似
    - 结点转换函数有可能是近似

数据流分析收敛性

- 不动点：给定一个函数$f: S \rightarrow S$，如果$f(x) = x$，则称x是f的一个不动点
- 不动点定理：给定高度有限的半格$(S, \sqcap)$和一个单调函数f，链$\top_s, f(\top_s), f(f(\top_s)), \ldots$必定在有限步之内收敛于f的最大不动点，即存在非负整数n，使得$f^n(\top_s)$是f的最大不动点。
  - 证明：
    - 收敛于f的不动点
      - $f(\top_s) \sqsubseteq \top_s$，两边应用f，得$f(f(\top_s)) \sqsubseteq f(\top_s)$，
      - 应用f，可得$f\left(f\left(f(\top_s)\right)\right) \sqsubseteq f\left(f(\top_s)\right)$
      - 因此，原链是一个递减链。因为该格高度有限，所以必然存在某个位置前后元素相等，即，到达不动点。
    - 收敛于最大不动点
      - 假设有另一不动点u，则$u \sqsubseteq \top_s$，两边反复应用f可证

数据流分析收敛性

- 给定固定的结点选择策略，原算法可以看做是反复应用一个函数
  - $(DATA_{v_1}, DATA_{v_2}, \ldots, DATA_{v_n}) := f(DATA_{v_1}, DATA_{v_2}, \ldots, DATA_{v_n})$
    - 为什么没有$DATA_{entry}$
- 根据不动点定理，原算法在有限步内终止，并且收敛于最大不动点

练习：可达定值(Reaching Definition)分析

- 对程序中任意语句，分析运行该语句后每个变量的值可能是由哪些语句赋值的，给出语句标号
  - 假设程序中没有指针、引用、复合结构
  - 要求上近似
  - 例：

```
1.  a=100;
2.  if (…)
3.    a = 200;
4.    b = a;
5.  return a;
```

    - 运行到2的时候a的定值是1
    - 运行到3的时候a的定值是3
    - 运行到4的时候a的定值是3， b的定值是4
    - 运行到5的时候a的定值是1，3， b的定值是4

答案：可达定值(Reaching Definition)分析

- 正向分析
- 半格元素：一个集合的序列，每个序列位置代表一个变量，每个位置的集合代表该变量的定值语句序号
- 交汇操作：对应位置的并
- 变换函数：
  - 对于赋值语句v=…
    - KILL={所有赋值给v的语句编号}
    - GEN={当前语句编号}
  - 对于其他语句
    - KILL=GEN={}

课后作业（截止日期：9月30日）

- 给定程序语言如下。其中use使用某个指针（可能读写该指针指向的地址），其余语句的语义和C语言相同。

  program := main(*vars*) { *stmts* }
  vars := *var*, *vars* | *var* |
  stmts := *stmt* ; *stmts* |
  stmt := *var* = *expr*
  | if (*bExp*) { *stmts* } else { *stmts* }
  | free(*var*)
  | use(var)
  | return
  expr := malloc() | *var*
  bExp := *var* == *var*
  var := [a-zA-Z_]+

- 基于数据流分析设计算法，尽可能多的查找并修复程序中的内存泄露。修复方式为在代码中插入free(var)语句。要求修复的安全性，即在所有通过free(var)的语句的路径中：
  - 在执行free(var)之前，var中保存了由某个malloc返回的对象
  - 在执行free(var)之后，不会再有任何use语句使用该指针指向的对象
  - 在该路径上没有别的free语句释放同一个对象
- 假设没有多重指针，即var不能指向另一个var
- 提示：可能需要多次调用数据流分析

课后作业：例

```
1.  Main (b, c) {
2.    a=malloc();
3.    if (a==b) {
4.      return;
5.    } else {}
6.    b = a;
7.    free(b);
8.  }
```

- 修复方法：在第4句前插入free(a);

课后作业：假设条件

- 假设别名分析已经提供，即
  - 给定位于两个程序点的两个变量，别名分析返回
    - Must Alias: 在所有执行中，这两个变量是否一定指向同一个对象
    - Must-not Alias: 在所有执行中，这两个变量是否一定不指向同一个对象
    - May Alias: 不属于以上情况