
软件分析

# 数据流分析:性质与扩展

熊英飞
北京大学
2015

Microsoft Research
微软亚洲研究院

# 复习：什么是半格？

- 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$x \sqcap \top = x$

# 复习：如何从半格定义偏序？

- $x \sqsubseteq y$当且仅当$x \sqcap y = x$

# 复习：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

# 复习：数据流分析实现算法

$DATA_{entry} = I$

$\forall v \in (V - entry): DATA_v \leftarrow \top_V$

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

While(ToVisit.size $> 0$) {

 $v \leftarrow$ ToVisit中任意结点

 ToVisit -= v

 $MEET_v \leftarrow \sqcap_{w \in pred(v)} DATA_w$

 $If\left(DATA_V \neq f_v(MEET_v)\right)$ ToVisit $\cup=$ succ(v)

 $DATA_v \leftarrow f_v(MEET_v)$

}

# 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$\mathrm{DATA}_{\mathrm{v_i}} \sqsubseteq \sqcap_{v_1 v_2 v_3 \ldots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \cdots \circ f_{v_1}(I_{entry})$
  - 示例：符号分析的偏序关系中⊥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

# 复习：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$\top_S$
- 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$\forall s \in S: s \sqcap u = u$，$(\sqcap_{s \in S} s) \sqcap u = (\sqcap_{s \in S} (s \sqcap u)) = u$，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

# 数据流分析的安全性-证明

- 给定任意路径的$v_1 v_2 v_3 \dots v_i$，$\mathrm{DATA}_{\mathrm{v_i}}$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，$\sqcap_{v_1 v_2 v_3 \dots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \cdots \circ f_{v_1}(I_{entry})$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”，$f_v$(正) $\sqcap$ $f_v$(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，$\mathrm{DATA}_{\mathbf{v_i}} = \sqcap_{v_1 v_2 v_3 \ldots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \cdots \circ f_{v_1}(I_{entry})$
  - 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$f: S \to S$，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$f\left(f(f(\top_s))\right) \sqsubseteq f(f(\top_s))$
      - 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
  - $(DATA_{v_1}, DATA_{v_2}, \ldots, DATA_{v_n}) := f(DATA_{v_1}, DATA_{v_2}, \ldots, DATA_{v_n})$
- 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={}

# 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

- 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

# 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的过程
- 可以通过Widening和Narrowing来处理无限半格的情况

# 下节课内容


- 下节课邀请高庆带领大家进行LLVM开发实践
- 要求：
  - 自带笔记本电脑
  - 按照下页步骤预先安装好LLVM

# LLVM安装方法


- 方法一（推荐）：从http://yun.baidu.com/s/1gdGtrLD下载虚拟机，使用VMWare加载。虚拟机使用方法：
  - VMWare官方网站上可以下载免费的VMWare Player
  - 使用VMWare打开虚拟机（名称为Ubuntu12-64）
  - 使用root登录，密码123
  - LLVM已经编译好，源码在/home/llvm，二进制文件在/home/build中
- 方法二：
  - 操作系统：Linux或Mac OS操作系统，或相应虚拟机
  - 源码下载：按照http://clang.llvm.org/get_started.html中，Building Clang and Working with the Code的On Unix-like Systems1~6条的说明进行。Mac OS下命令可能稍有不同。
  - 安装：按照http://llvm.org/docs/BuildingLLVMWithAutotools.html#a-quick-summary中，quick summary的说明完成安装，注意设置../llvm/configure --enable-optimized这个参数以加速编译。编译过程大概一两个小时。

# 课后作业（截止日期：10月8日）


- 给定程序语言：

  *program* := *main( vars ) { stmts }*
  *vars* := *var, vars | var |*
  *stmts* := *stmt* ; *stmts* |
  *stmt* := *var* = *expr*
  | if (*bExp*) { *stmts* } else { *stmts* }
  | free(*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，不会再有任何语句使用该对象
  - 在该路径上没有别的free语句释放同一个对象
- 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