



软件分析

数据流分析

熊英飞
北京大学
2014



复习

- 大多数程序分析问题都是不可判定问题
 - 莱斯定理
- 解决途径是对程序做抽象
 - `must`分析/下近似
 - `may`分析/上近似



复习 – 停机问题的证明方法

- 假设存在停机问题判断算法: `bool Halt(p, i)`

- `p`为特定程序, `i`为输入

- 给定某邪恶程序

```
void Evil(p) {  
    if (!Halt(p, p)) return;  
    else while(1);  
}
```

- `Halt(Evil, Evil)`的返回值是什么?

- 如果为真, 则`Evil`不停机, 矛盾
- 如果为假, 则`Evil`停机, 矛盾



停机问题-抽象方法

- 邪恶程序存在的关键在于程序中有if存在
- 不如忽略掉所有程序的if条件部分

```
void Evil(p) {  
    if (!Halt(p, p)) return;  
    else while(1);  
}
```



```
void Evil(p) {  
    向左走 return;  
    向右走 while(1);  
}
```

- 语义：“向左走/向右走”为非确定性选择，程序随机从“向左走”和“向右走”后面的语句中选择一条执行。



停机问题-抽象方法

- 邪恶程序仍然可以用循环写出

```
void Evil(p) {  
    while (Halt(p, p));  
}
```

- 忽略所有条件判断中的条件，一律抽象为不确定选择

```
void Evil(p) {  
    再来一次:  
    向左走 goto 再来一次;  
    向右走 return;  
}
```



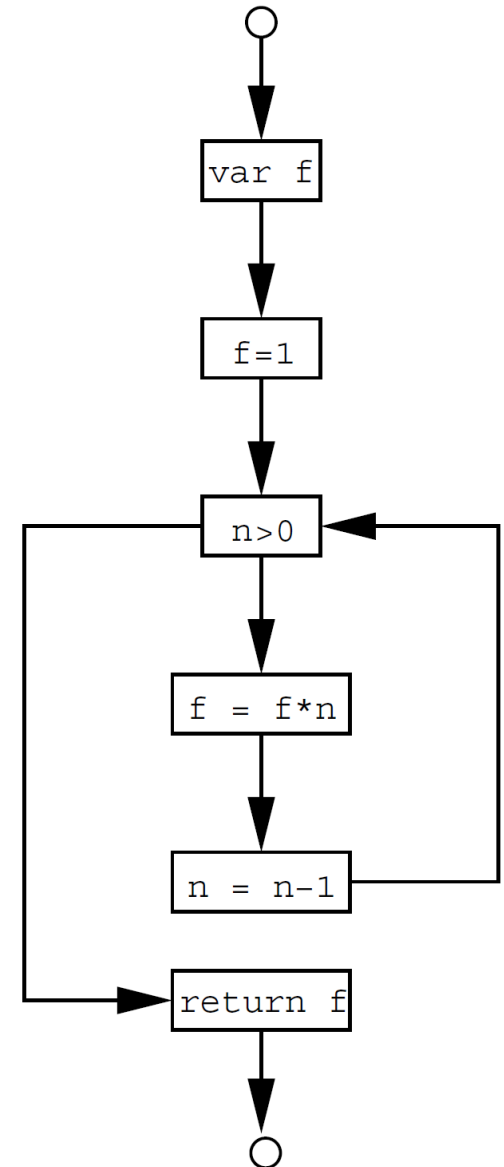
停机问题-抽象过程分析

- 针对给定输入
 - 原始程序只有一条执行路径，抽象程序上有多条执行路径
 - 原始程序的执行路径一定包含在抽象程序的执行路径中
- 停机问题
 - 原始程序停机：存在自然数 n ，程序的执行路径长度小于 n
 - 抽象程序停机：存在自然数 n ，程序中所有执行路径的长度都小于 n



停机问题-判定方法

- 判断方法：绘制控制流图
 - 控制流图：结点为程序语句，边为语句间的转移
- 如果控制流图上有环，则可能不终止，否则一定终止





数据流分析-小结1

- 近似方案1：忽略掉程序的条件判断，认为所有分支都有可能到达
- 数据流分析：程序可以看成是状态（数据）和状态之间的转移（控制）两部分，因为状态转移的条件都被忽略了，核心分析的部分是状态数据在转移过程中的变化，所以叫做数据流分析。



符号分析

- 给定一个只包含浮点数变量和常量的程序，已知输入的符号，求输出的符号
- 采用上节课讲到的抽象域，输出正、零、负、躲四种结果



复习：符号分析的抽象

- 抽象符号
 - 正 = {所有的正数}
 - 零 = {0}
 - 负 = {所有的负数}
 - 靛 = {所有的整数和NaN}
- 运算（列标号 ● 行标号）

+	正	负	零	靛
正	正			
负	靛	负		
零	正	负	零	
靛	靛	靛	靛	靛



-	正	负	零	躲
正	躲	负	负	躲
负	正	躲	正	躲
零	正	负	零	躲
躲	躲	躲	躲	躲

*	正	负	零	躲
正	正			
负	负	正		
零	零	零	零	
躲	躲	躲	躲	躲

/	正	负	零	躲
正	正	负	零	躲
负	负	正	零	躲
零	躲	躲	躲	躲
躲	躲	躲	躲	躲



符号分析-示例

```
x*=-100;
```

```
y+=1;
```

```
while(y < z) {
```

```
    x *= -100;
```

```
    y += 1;
```

```
}
```

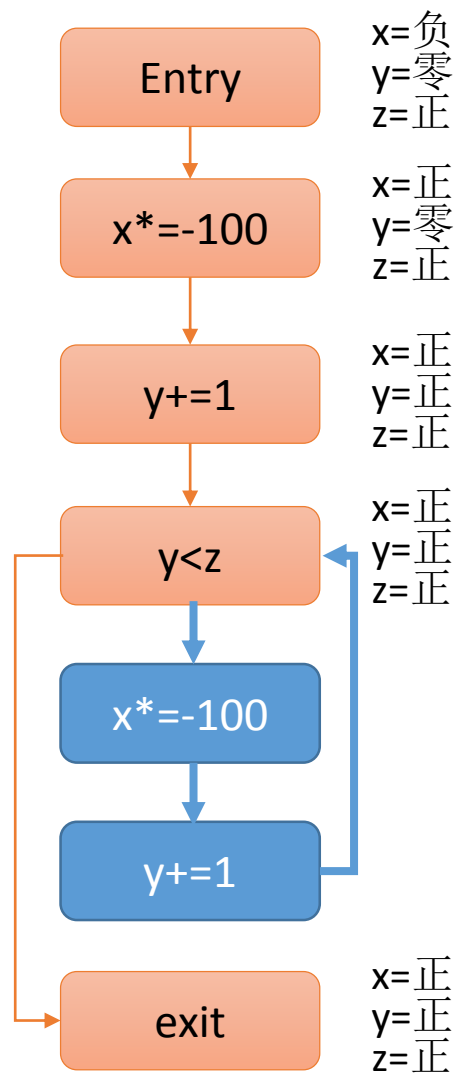
输入：x为负，y为零，z为正

输出：x为负，y为正，z为正



符号分析-基本思路

- 给定程序的一条执行路径，我们能推出结果符号的抽象取值



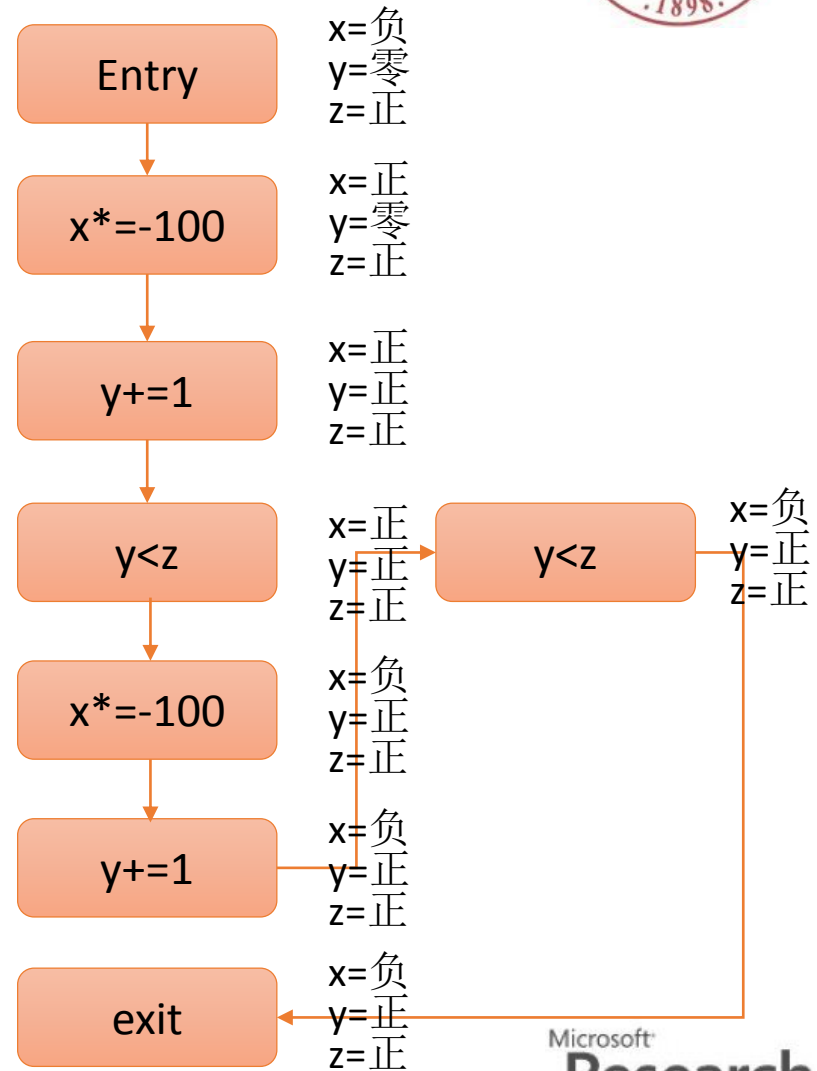


符号分析-基本思路

- 给定程序的两条执行路径，我们得到两个结果符号的抽象取值 v_1, v_2 ，我们可以用如下的操作来合并这两个值：

- $$\sqcap(v_1, v_2) = \begin{cases} v_1 & \text{如果 } v_1 = v_2 \\ \text{躲} & \text{其他情况} \end{cases}$$

- $\sqcap((\text{正正正}), (\text{负正正})) = (\text{躲正正})$





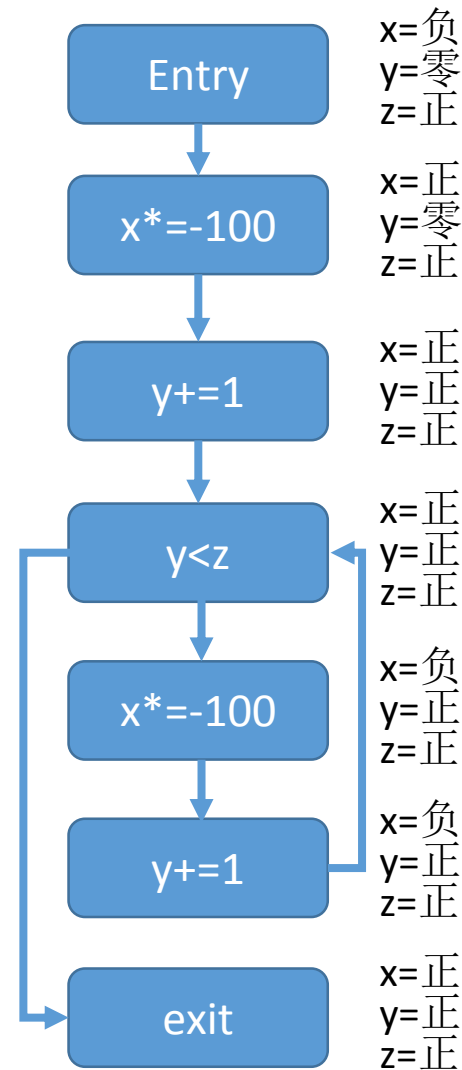
符号分析-基本思路

- 如果我们能知道程序所有可能的路径产生的结果符号 v_1, v_2, \dots ，我们就知道了程序的最终结果 $\sqcap (v_1, v_2, \dots)$ 。
- 如何知道程序有哪些可能的路径？
 - 近似方案1：忽略掉程序的条件判断，认为所有分支都有可能到达
- 如何能遍历所有可能的路径？
 - 近似方案2：不在路径末尾做合并，在控制流汇合的所有位置提前做合并



符号分析-示例

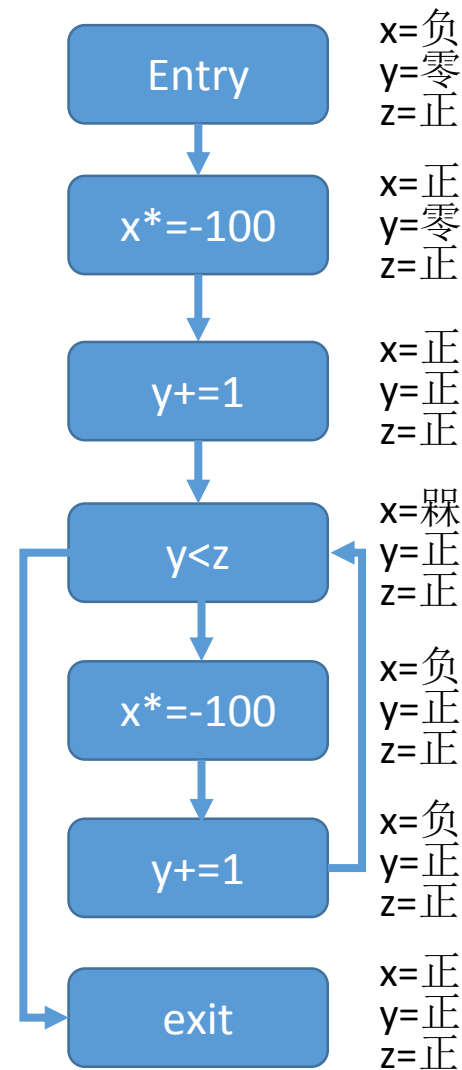
```
x*=-100;  
y+=1;  
while(y < z) {  
    x *= -100;  
    y += 1;  
}
```





符号分析-示例

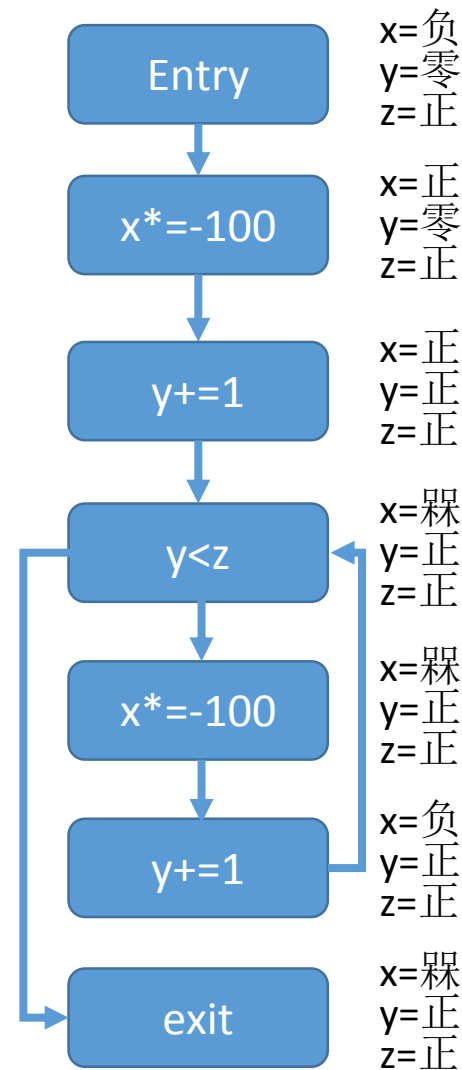
```
x*=-100;  
y+=1;  
while(y < z) {  
    x *= -100;  
    y += 1;  
}
```





符号分析-示例

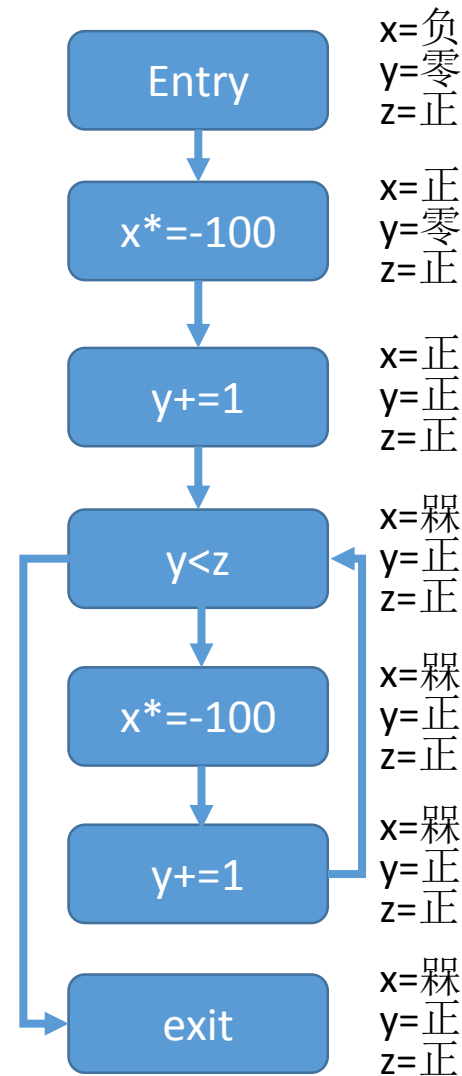
```
x*=-100;  
y+=1;  
while(y < z) {  
    x *= -100;  
    y += 1;  
}
```





符号分析-示例

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





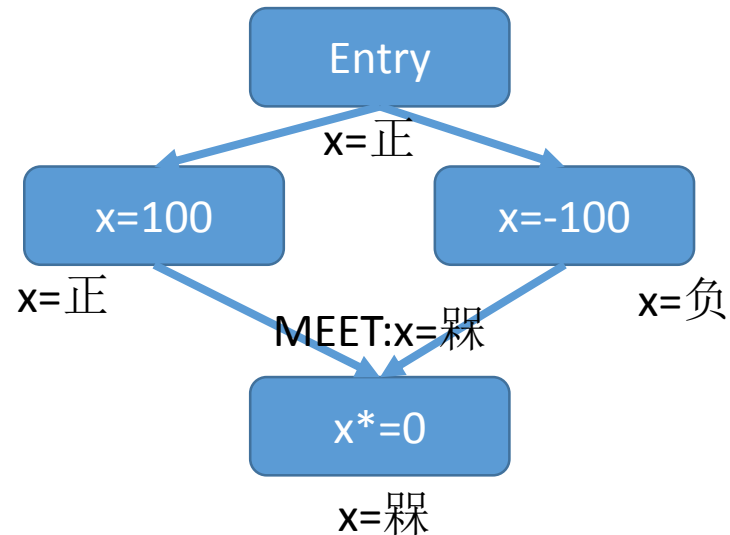
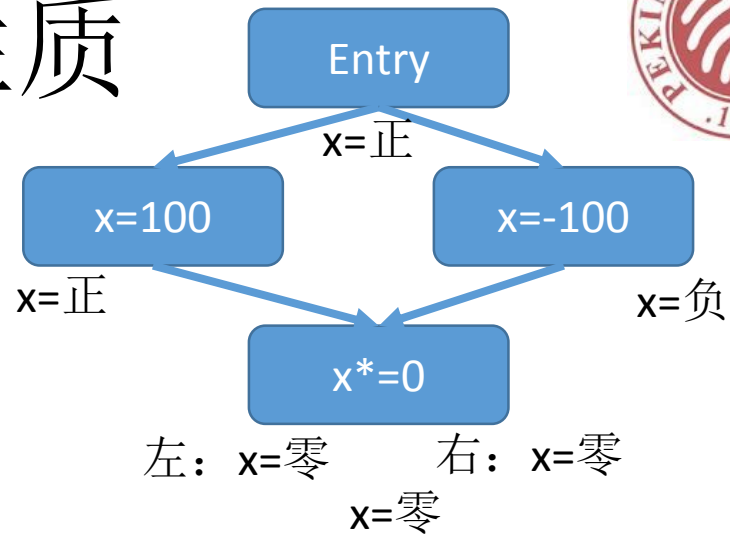
符号分析-算法

- 令 $\mathbf{S} = \{(s_x, s_y, s_z) \mid s_x, s_y, s_z \in \{\text{正, 负, 零, 糶, T}\}\}$
- 每个结点的值为 \mathbf{S} 的一个元素，代表对应语句执行之后的变量符号，用 DATA 表示
- 初始值
 - $\text{DATA}_{\text{entry}} = (\text{负}, \text{零}, \text{正})$
 - $\text{DATA}_{\text{其他结点}} = (\text{T}, \text{T}, \text{T})$
- 结点转换函数 $f_v: \mathbf{S} \rightarrow \mathbf{S}$
 - $f_{\text{exit}} = \text{id}$
 - $f_{\text{其他结点}} = \text{根据相应语句进行计算}$
- 交汇运算 $\text{MEET}_v = \sqcap_{w \in \text{pred}(v)} \text{DATA}_w$ ， \sqcap 操作扩展到 T : $x \sqcap \text{T} = x$
- 结点更新运算 $S_v = f_v(\text{MEET}_v)$
- 如果某个结点的前驱结点发生了变化，则使用结点更新运算更新该结点的附加值
- 如果没有任何结点的值发生变化，则程序终止。



符号分析-算法性质

- 该算法是安全的吗？
 - 近似方案并非等价变换，那么该近似方案是安全的吗？
- 该算法保证终止(Terminating)吗？
 - 路径上有环的时候，是否会一直循环？
- 该算法一定合流(Confluent)吗？
 - 有多个结点可更新的时候，是否无论先更新哪个结点最后都会到达同样的结果？
- 终止+合流=收敛 (Convergence)
- 以上问题的答案将在数据流分析框架部分统一回答





数据流分析-小结2

- 给出一条程序路径上的分析方案，和不同路径上的结果合并方案
- 近似方案1：忽略掉程序的条件判断，认为所有分支都有可能到达
- 近似方案2：不在路径末尾做合并，在控制流汇合的所有位置提前做合并



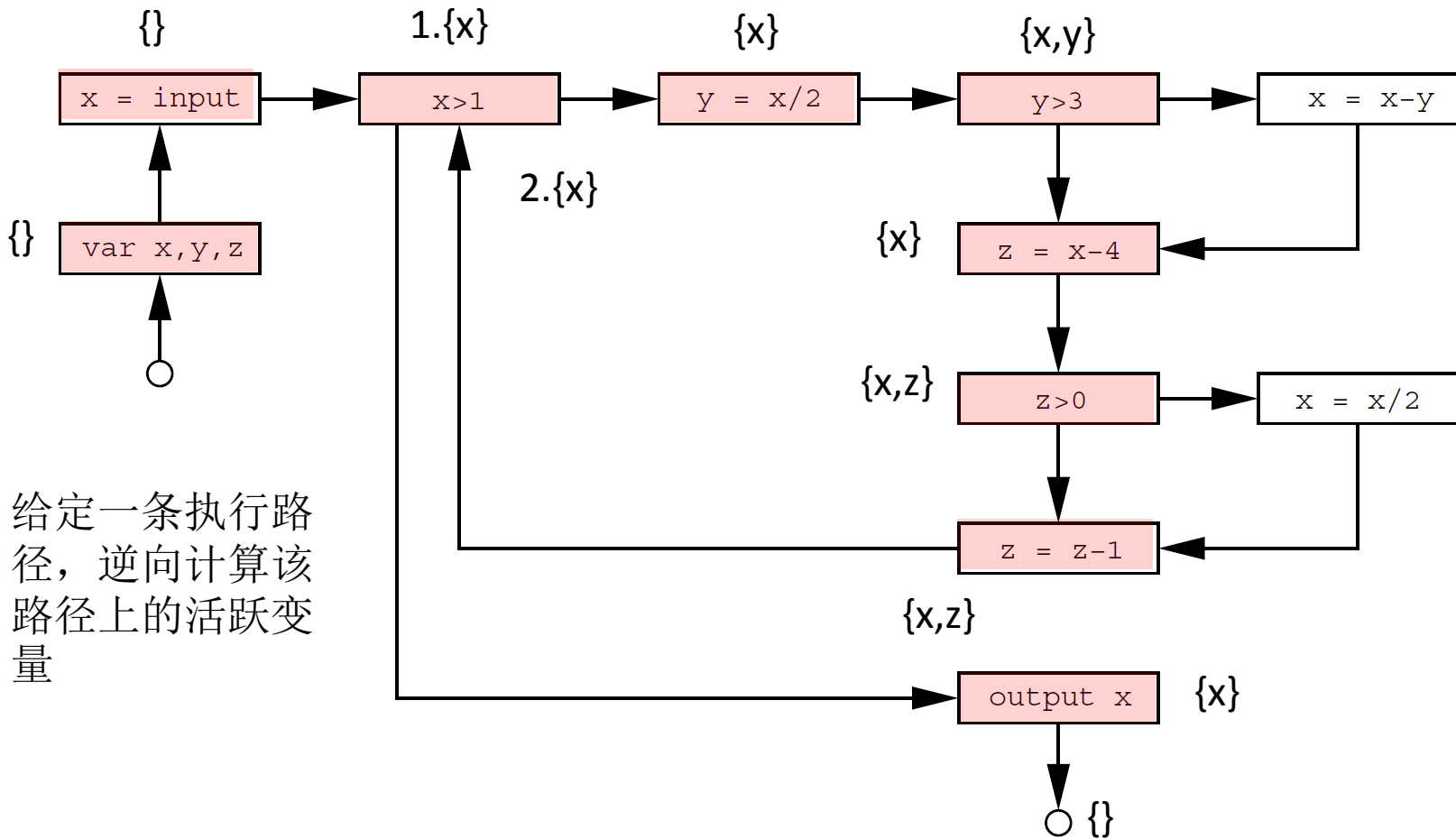
数据流分析-活跃变量分析 (Liveness Analysis)

- 活跃变量: 给定程序中
的某条语句s和变量v,
如果在s执行前保存在v
中的值在后续执行中还
会被读取就被称作活跃
变量
- 第四行的y和x是否为活
跃变量?
- 第八行的y和z呢?
- 活跃变量分析: 返回所
有可能的活跃变量
 - may分析

```
1. var x,y,z;  
2. x = input;  
3. while (x>1) {  
4.   y = x/2;  
5.   if (y>3) x = x-y;  
6.   z = x-4;  
7.   if (z>0) x = x/2;  
8.   z = z-1;  
9. }  
10. output x;
```



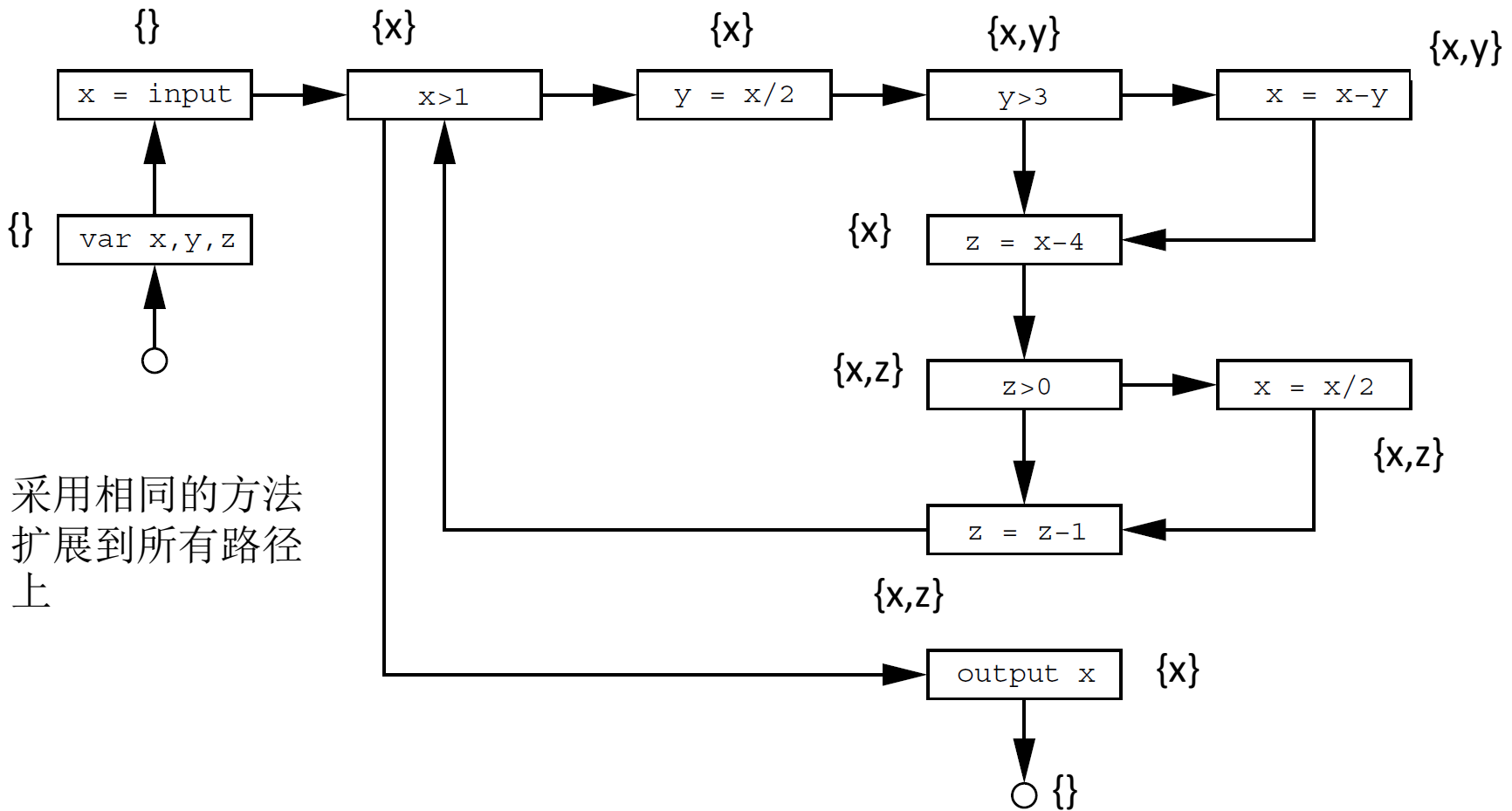
活跃变量分析-基本思想



给定一条执行路径，逆向计算该路径上的活跃变量



活跃变量分析-例子



采用相同的方法
扩展到所有路径
上



活跃变量分析-算法

- 初始值: $DATA_v = \{\}$
- 结点转换函数: $f_v(L) = (L \setminus KILL_v) \cup GEN_v$
 - $GEN_v = vars(v)$
 - $KILL_v = \begin{cases} \{x\} & v := x = \text{exp}; \\ \{x\} & v := \text{int } x; \\ \{\} & \text{otherwise} \end{cases}$
- 交汇运算 $MEET_v = \bigcup_{w \in succ(v)} DATA_w$
- 结点更新运算 $L_v = f_v(MEET_v)$
- 如果某个结点的后继结点发生了变化, 则使用结点更新运算更新该结点的附加值
- 如果没有任何结点的值发生变化, 则程序终止。



活跃变量分析-算法性质

- 该算法是安全的吗？
 - 安全性：每个节点对应的L集合包括了所有的活跃变量
 - 对于单条路径，该性质可以归纳证明
 - 如何证明对所有路径的安全性？
- 该算法保证收敛吗？



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, \sqcap) ，其中 S 是一个集合， \sqcap 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 \top ，使得 $x \sqcap \top = x$



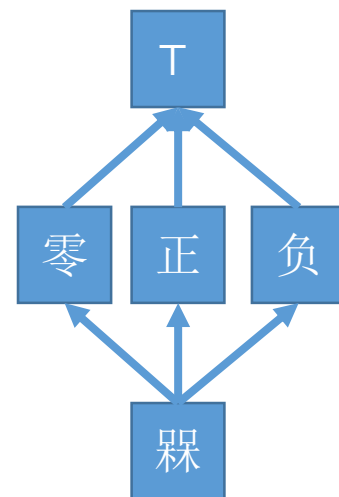
偏序 Partial Order

- 偏序是一个二元组 (S, \sqsubseteq) ，其中 S 是一个集合， \sqsubseteq 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

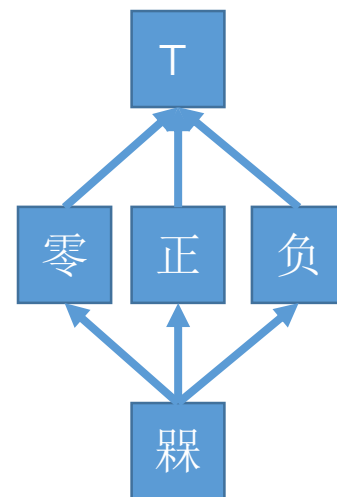
- 抽象符号域的五元素和交汇操作组成了一个半格
- 半格的笛卡尔乘积($S \times T, \sqcap_{xy}$)还是半格
 - $(s_1, t_1) \sqcap_{xy} (s_2, t_2) = (s_1 \sqcap_x s_2, t_1 \sqcap_y t_2)$
- 任意集合和交集操作组成了一个半格
 - 偏序关系为子集关系
 - 顶元素为全集
- 任意集合和并集操作组成了一个半格
 - 偏序关系为超集关系
 - 顶元素为空集





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 τ_S
- 引理： $\bigcap_{s \in S} s$ 是 S 的最大下界
 - 证明：
 - 根据幂等性、交换性和结合性，我们有 $\forall v \in S: (\bigcap_{s \in S} s) \sqcap v = \bigcap_{s \in S} s$ ，所以 $\bigcap_{s \in S} s$ 是 S 的下界
 - 给定另一个下界 u ，我们有 $\forall s \in S: s \sqcap u = u$ ， $(\bigcap_{s \in S} s) \sqcap u = \bigcap_{s \in S} (s \sqcap u) = u$ ，所以 $\bigcap_{s \in S} s$ 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 $f(S) = (S - KILL) \cup GEN$ 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 $v \in V - \text{entry}$ 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

```
DATAentry = I
∀v ∈ (V - entry): DATAv ← Tv
ToVisit ← V - entry //可以换成succ(entry)吗?
While(ToVisit.size > 0) {
  v ← ToVisit中任意结点
  ToVisit -= v
  MEETv ← ∏w ∈ pred(v) DATAw
  If(DATAv ≠ fv(MEETv)) ToVisit ∪= succ(v)
  DATAv ← fv(MEETv)
}
```



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 $DATA_{v_i} \sqsubseteq \bigcap_{v_1 v_2 v_3 \dots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \dots \circ f_{v_1}(I_{entry})$
 - 示例：符号分析的偏序关系中 \perp 比较小， \top 比较大，结果是下近似
 - 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 $v_1 v_2 v_3 \dots v_i$, $DATA_{v_i}$ 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, $DATA_{v_i}$ 的值一定小于等于 $f_{v_i} \circ f_{v_{i-1}} \circ \dots \circ f_{v_1}(I_{entry})$ 。由于原路径的任意性, $DATA_{v_i}$ 是一个下界。
- 再根据前面的引理, $\bigcap_{v_1 v_2 v_3 \dots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \dots \circ f_{v_1}(I_{entry})$ 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
 - $\forall v \in V, x, y \in S: f_v(x) \sqcap f_v(y) = f_v(x \sqcap y)$
- 示例
 - 符号分析中的结点转换函数不满足分配性
 - 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 $f(DATA) = (DATA - KILL) \cup GEN$ 满足分配性
- 当数据流分析满足分配性的时候， $DATA_{v_i} = \sqcap_{v_1 v_2 v_3 \dots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \dots \circ f_{v_1}(I_{entry})$
 - 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, \sqsubseteq) 和一个单调函数 f ，链 $T_s, f(T_s), f(f(T_s)), \dots$ 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 $f^n(T_s)$ 是 f 的最大不动点。
 - 证明：
 - 收敛于 f 的不动点
 - $f(T_s) \sqsubseteq T_s$ ，两边应用 f ，得 $f(f(T_s)) \sqsubseteq f(T_s)$ ，
 - 应用 f ，可得 $f(f(f(T_s))) \sqsubseteq f(f(T_s))$
 - 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 $u \sqsubseteq T_s$ ，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
 - $(DATA_{v_1}, DATA_{v_2}, \dots, DATA_{v_n}) := f(DATA_{v_1}, DATA_{v_2}, \dots, DATA_{v_n})$
- 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



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
 - 交汇函数形成一个格
 - 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 $f(D) = (D - KILL) \cup GEN$ 的形式



练习：可达定值(Reaching Definition)分析

- 对程序中任意语句，分析程序运行到该语句时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May分析
 - 例：
 1. `a=100;`
 2. `if (...)`
 3. `a = 200;`
 4. `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1
 - 运行到4的时候a的定值是1, 3



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Must分析
 - 例：
 1. $a = a + (b + 10);$
 2. $if (...)$
 3. $a = a + 10;$
 4. $return a;$
 - 1运行结束的时候可用表达式是 $b + 10$ 、 $a + (b + 10)$
 - 2运行结束的时候可用表达式是 $b + 10$ 、 $a + (b + 10)$
 - 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 $b + 10$



程序分析的分类-敏感性

- 一般而言，抽象过程中考虑的信息越多，程序分析的精度就越高，但分析的速度就越慢
- 程序分析中考虑的信息通常用敏感性来表示
 - 流敏感性flow-sensitivity
 - 路径敏感性path-sensitivity
 - 上下文敏感性context-sensitivity
 - 字段敏感性field-sensitivity

术语-流敏感(flow-sensitivity)



- 流不敏感分析（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，则该分析为流不敏感分析。
- 流敏感分析（flow-sensitive analysis）：其他情况
- 数据流分析通常为流敏感的



流不敏感分析示例

- 流不敏感的符号分析
 - 收集程序中的所有赋值语句
 - 使用一个变量符号值的集合
 - 反复计算所有赋值语句更新该集合
- 流不敏感的活跃变量分析
 - 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
- 流不敏感的活跃变量：每条语句的操作时间为 $O(m)$ ，因此时间复杂度上界为 $O(nm)$ ，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，流不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）

术语-路径敏感(path-sensitivity)



- 路径不敏感分析：忽略掉程序中条件判断的分析
- 路径敏感分析：考虑程序中条件判断的分析



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);