



软件分析

# 数据流分析：基础

熊英飞  
北京大学



# 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)`

- `p`为特定程序

- 给定某邪恶程序

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

- `Halt(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() {  
    if (!Halt(Evil)) return;  
    else while(1);  
}
```



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

- 忽略所有条件判断中的条件，一律抽象为不确定选择

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



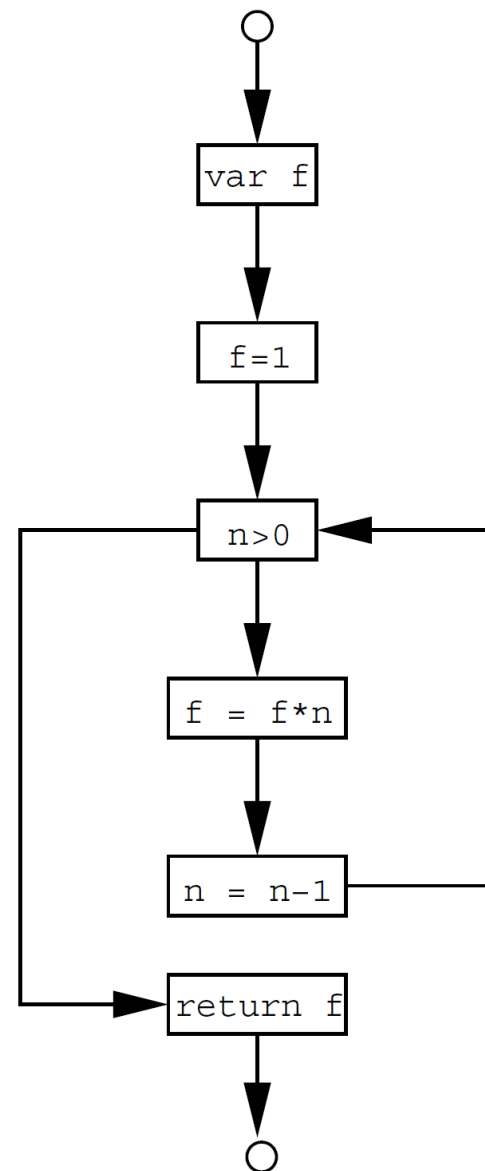
# 停机问题-抽象过程分析

- 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}
  - 糅 = {所有的整数和NaN}
- 运算（列标号●行标号）

+	正	负	零	糅
正	正			
负	糅	负		
零	正	负	零	
糅	糅	糅	糅	糅



-	正	负	零	罣
正	罣	负	负	罣
负	正	罣	正	罣
零	正	负	零	罣
罣	罣	罣	罣	罣

*	正	负	零	罣
正	正			
负	负	正		
零	零	零	零	
罣	罣	罣	罣	罣

/	正	负	零	罣
正	正	负	零	罣
负	负	正	零	罣
零	罣	罣	罣	罣
罣	罣	罣	罣	罣



# 符号分析-示例

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

```
y+=1;
```

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

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

```
    y += 1;
```

```
}
```

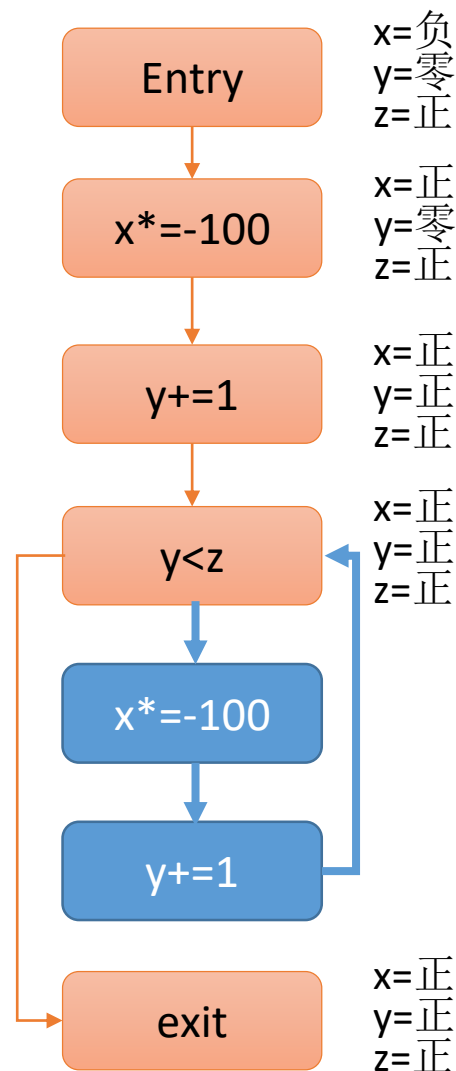
输入：x为负，y为零，z为正

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



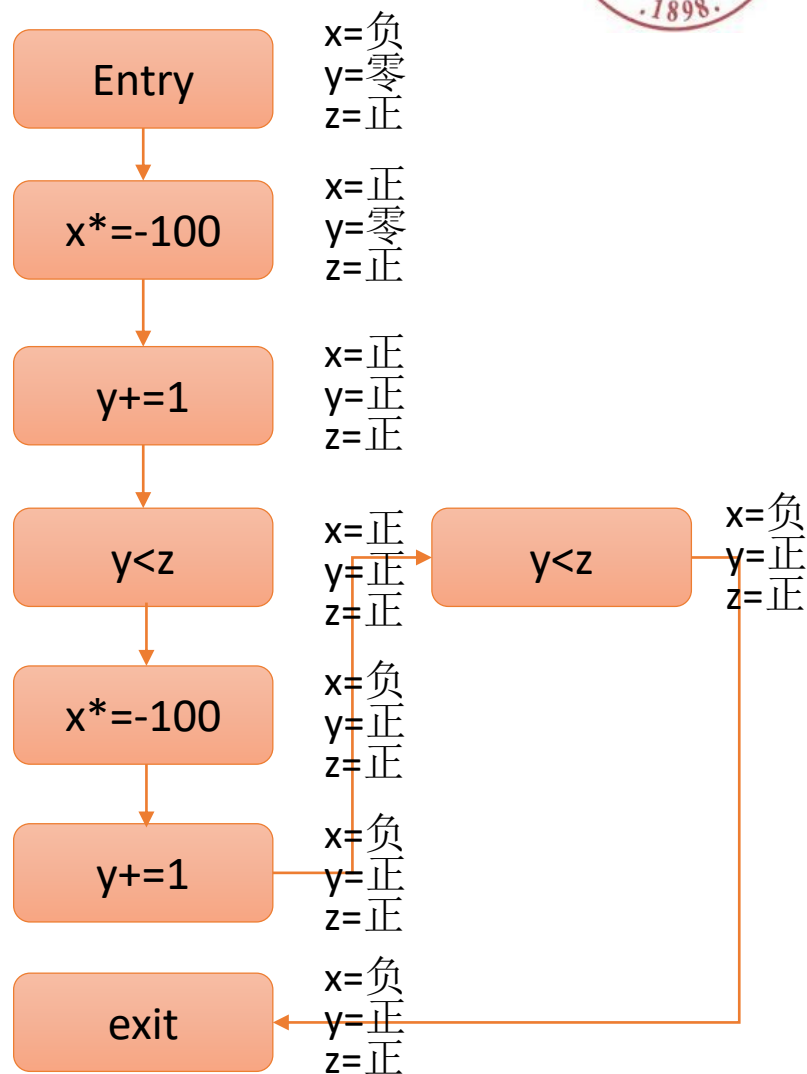


# 符号分析-基本思路

- 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：

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

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





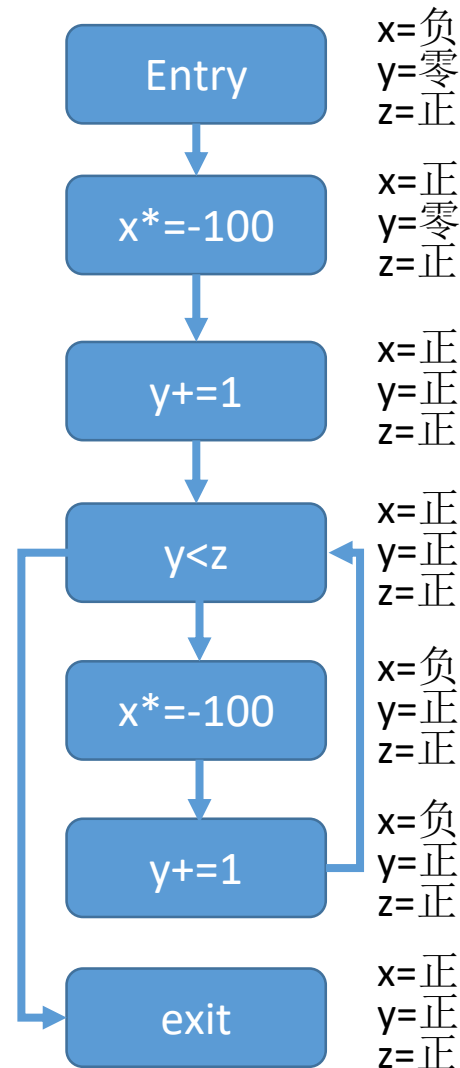
# 符号分析-基本思路

- 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  $\cap (v_1, v_2, \dots)$ 。
- 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;  
}
```

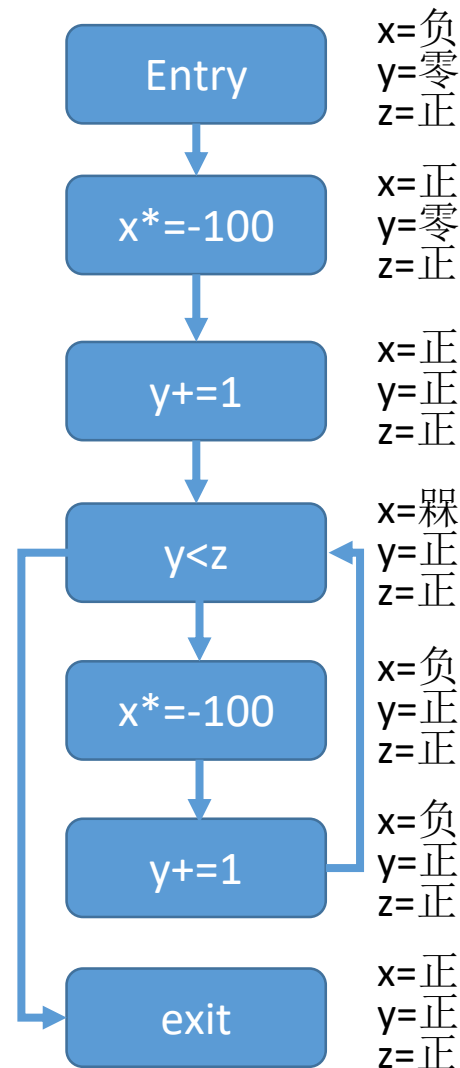






# 符号分析-示例

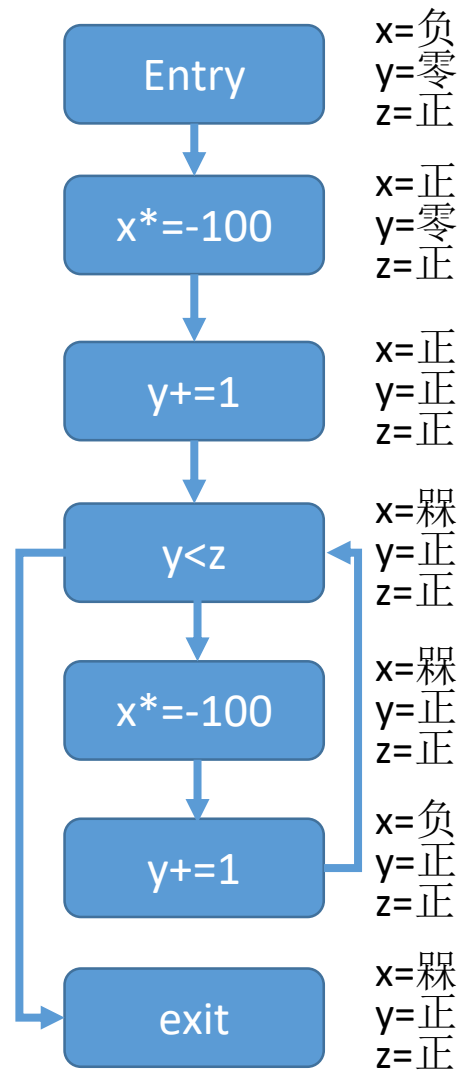
```
x*=-100;  
y+=1;  
while(y < z) {  
    x *= -100;  
    y += 1;  
}
```





# 符号分析-示例

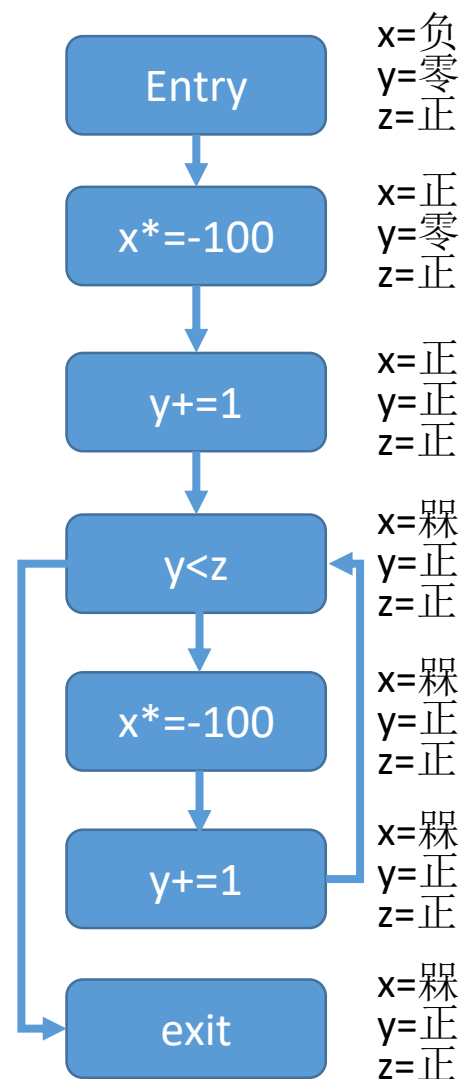
```
x*=-100;  
y+=1;  
while(y < z) {  
    x *= -100;  
    y += 1;  
}
```





# 符号分析-示例

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





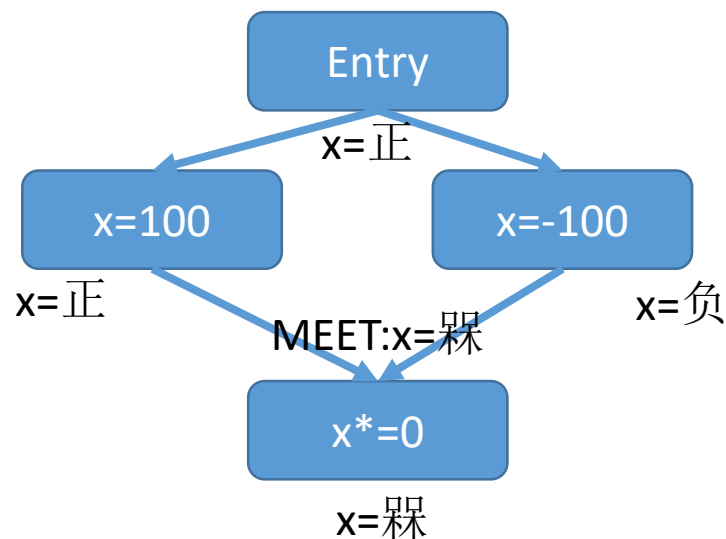
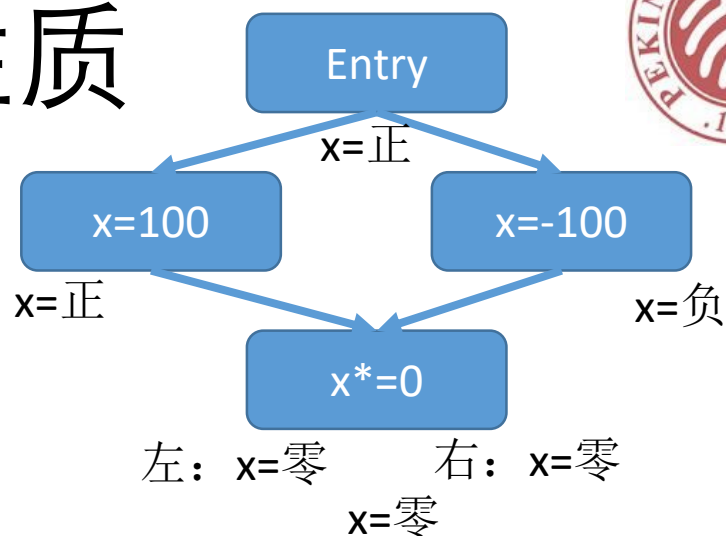
# 符号分析-算法

- 令  $\mathbf{S} = \{(s_x, s_y, s_z) \mid s_x, s_y, s_z \in \{\text{正, 负, 零, 罅, T}\}\}$
- 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  $f_v: \mathbf{S} \rightarrow \mathbf{S}$ 
  - $f_{\text{exit}} = \text{id}$
  - $f_{\text{其他结点}}$  = 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2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)

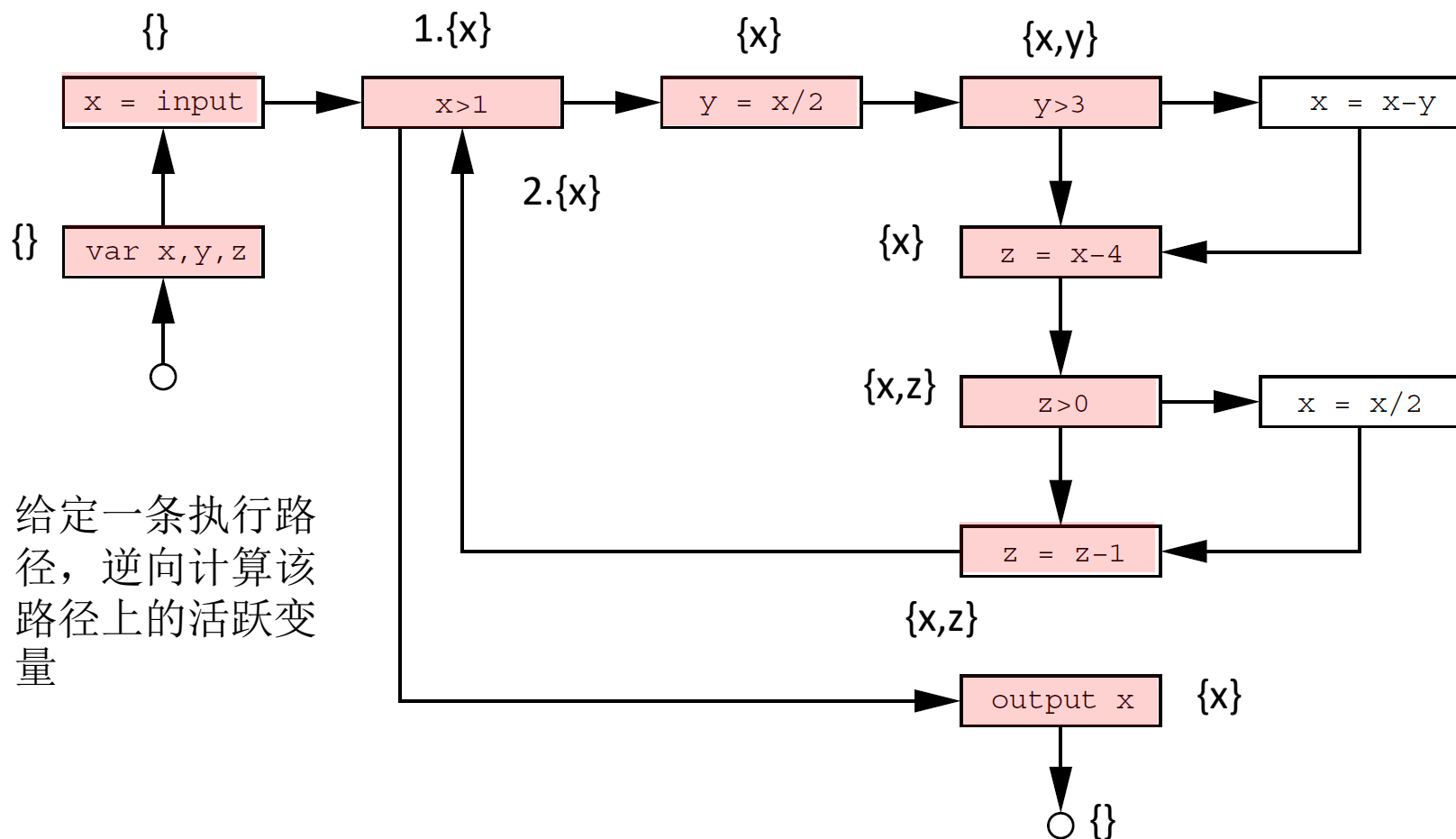


- 活跃变量：给定程序中的某条语句s和变量v，如果在s执行前保存在v中的值在后续执行中还会被读取就被称作活跃变量
- 第四行的y和x是否为活跃变量？
- 第八行的y和z呢？
- 活跃变量分析：返回所有可能的活跃变量
  - may分析

```
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;
```



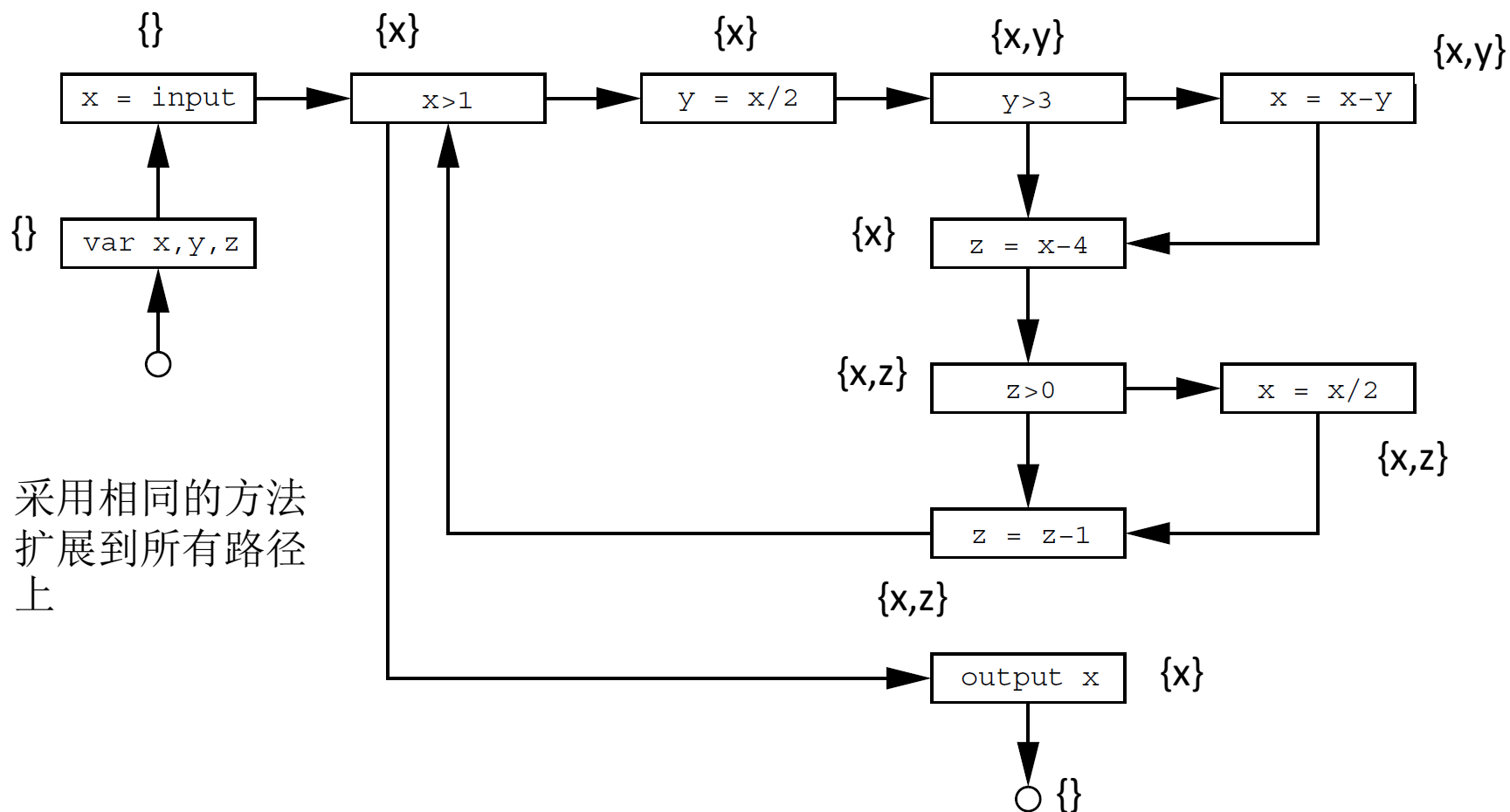
# 活跃变量分析-基本思想







# 活跃变量分析-例子





# 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 $x \sqcap \top = x$



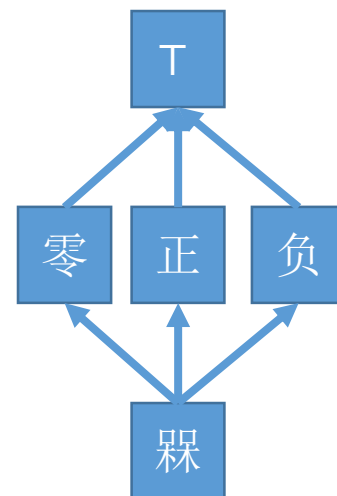
# 偏序 Partial Order

- 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

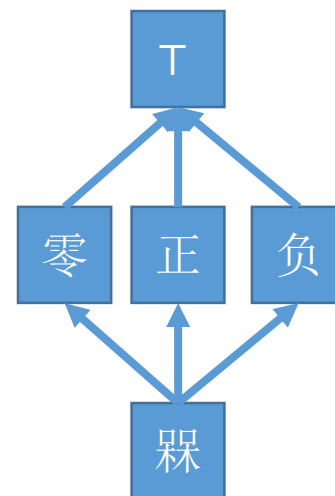
- 抽象符号域的五個元素和交汇操作组成了一个半格
- 半格的笛卡尔乘积( $S \times T, \sqcap_{xy}$ )还是半格
  - $(s_1, t_1) \sqcap_{xy} (s_2, t_2) = (s_1 \sqcap_x s_2, t_1 \sqcap_y t_2)$
- 任意集合和交集操作组成了一个半格
  - 偏序关系为子集关系
  - 顶元素为全集
- 任意集合和并集操作组成了一个半格
  - 偏序关系为超集关系
  - 顶元素为空集





# 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







# 单调（递增）函数

## Monotone (Increasing) 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：单调不等于  $a \sqsubseteq f(a)$
- 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  $I$
- 一组单调函数，对任意  $v \in V - entry$  存在一个单调函数  $f_v$
- 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
 $\forall v \in (V - \text{entry}): \text{DATA}_v \leftarrow \top$ 
ToVisit  $\leftarrow V - \text{entry}$  //可以换成succ(entry)吗?
While(ToVisit.size > 0) {
    v  $\leftarrow$  ToVisit中任意结点
    ToVisit -= v
    MEETv  $\leftarrow \bigcap_{w \in \text{pred}(v)} \text{DATA}_w$ 
    If( $\text{DATA}_v \neq f_v(\text{MEET}_v)$ ) ToVisit  $\cup = \text{succ}(v)$ 
    DATAv  $\leftarrow f_v(\text{MEET}_v)$ 
}
```



# 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 $f(D) = (D - KILL) \cup GEN$ 的形式



# 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  $v = \dots$ 
    - $KILL = \{\text{所有赋值给 } v \text{ 的语句编号}\}$
    - $GEN = \{\text{当前语句编号}\}$
  - 对于其他语句
    - $KILL = GEN = \{\}$



# 作业：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的所有路径都对表达式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



# 参考资料

- 《编译原理》 第9章
- Lecture Notes on Static Analysis
  - <https://cs.au.dk/~amoeller/spa/>