软件分析

数据流分析：基础

熊英飞
北京大学
2018

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

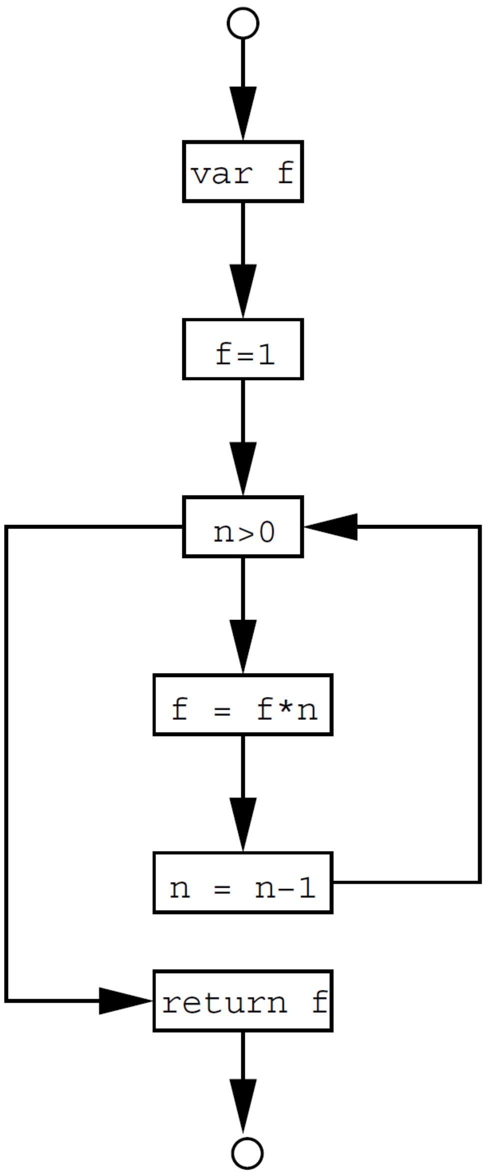
```
void Evil() {
    再来一次:
    向左走 goto 再来一次;
    向右走 return;
}
```

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、槑四种结果

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}
  - 槑={所有的整数和NaN}
- 运算（列标号●行标号）

+	正	负	零	槑
正	正			
负	槑	负		
零	正	负	零	
槑	槑	槑	槑	槑

-	正	负	零	槑
正	槑	负	负	槑
负	正	槑	正	槑
零	正	负	零	槑
槑	槑	槑	槑	槑

*	正	负	零	槑
正	正			
负	负	正		
零	零	零	零	
槑	槑	槑	槑	槑

/	正	负	零	槑
正	正	负	零	槑
负	负	正	零	槑
零	槑	槑	槑	槑
槑	槑	槑	槑	槑

符号分析-示例

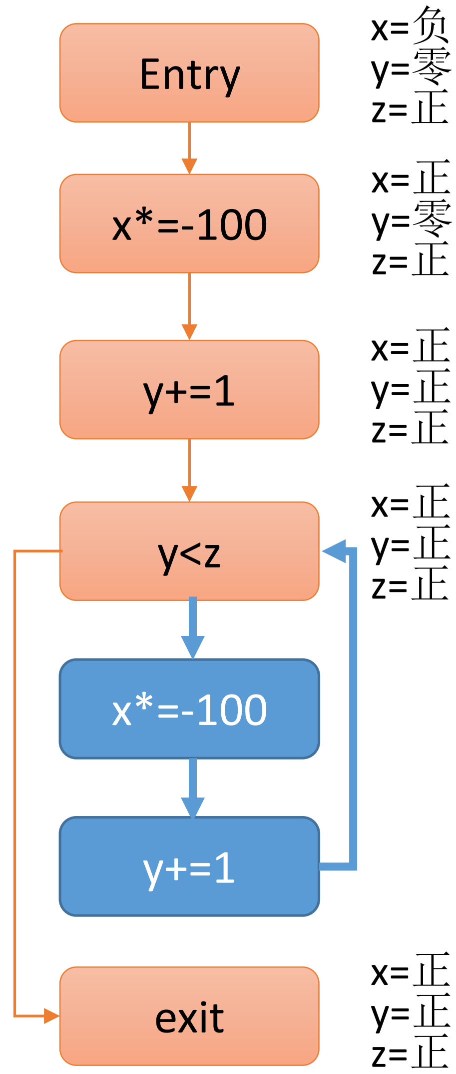
```
x*=-100;
y+=1;
while(y < z) {
    x *= -100;
    y += 1;
}
```

输入：x为负，y为零，z为正

输出：x为槑，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

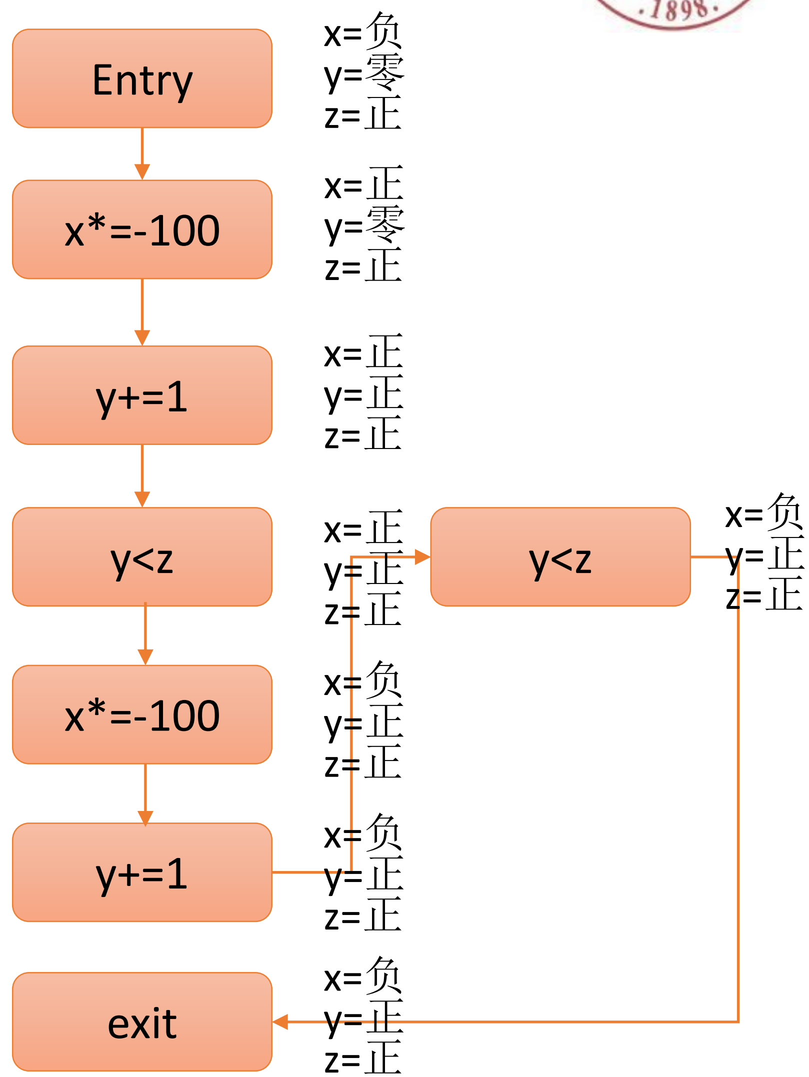


符号分析-基本思路

- 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：
  - $\sqcap(v_1, v_2) =$

$$\begin{cases} v_1 & \text{如果} v_1 = v_2 \\ \text{槑} & \text{其他情况} \end{cases}$$

- ⊓((正正正),(负正正))=(槑正正)

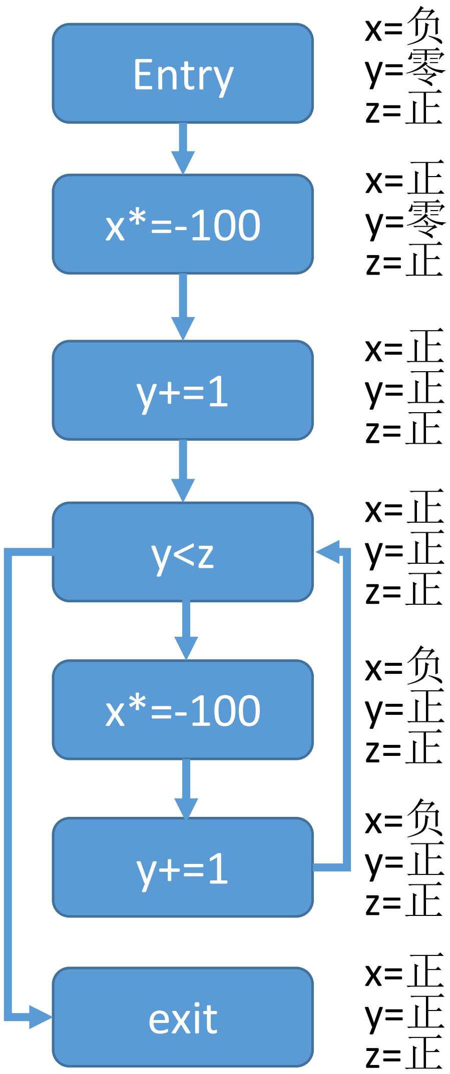


符号分析-基本思路

- 如果我们能知道程序所有可能的路径产生的结果符号$v_1, v_2, \ldots$，我们就知道了程序的最终结果$\sqcap(v_1, v_2, \ldots)$。
- 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

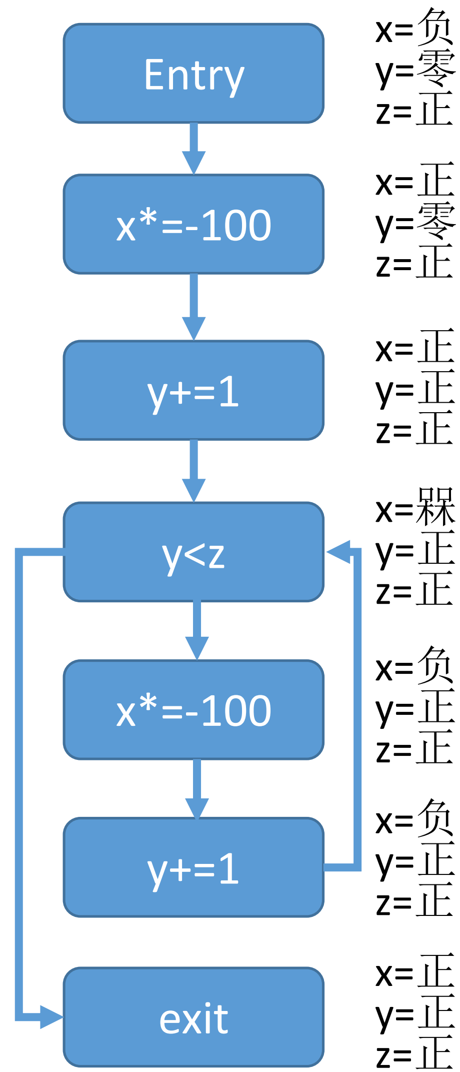
符号分析-示例

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



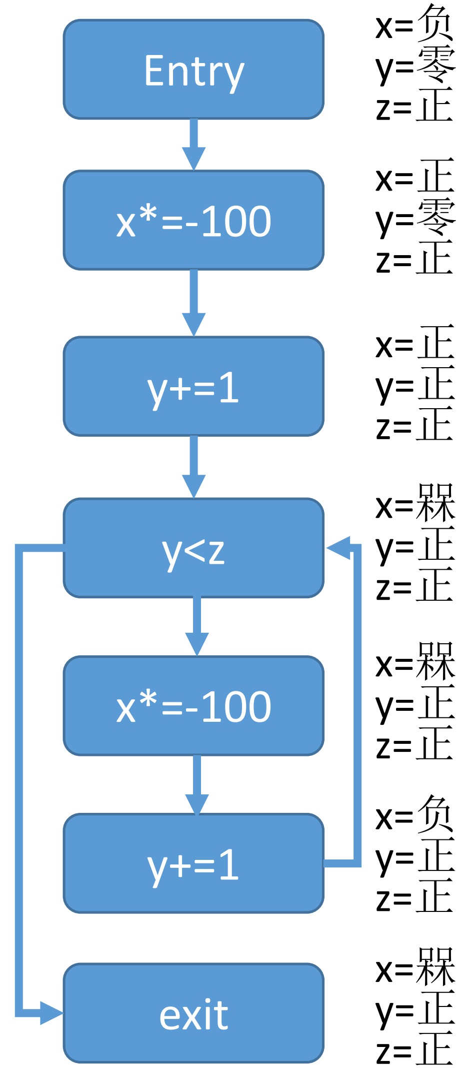
符号分析-示例

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



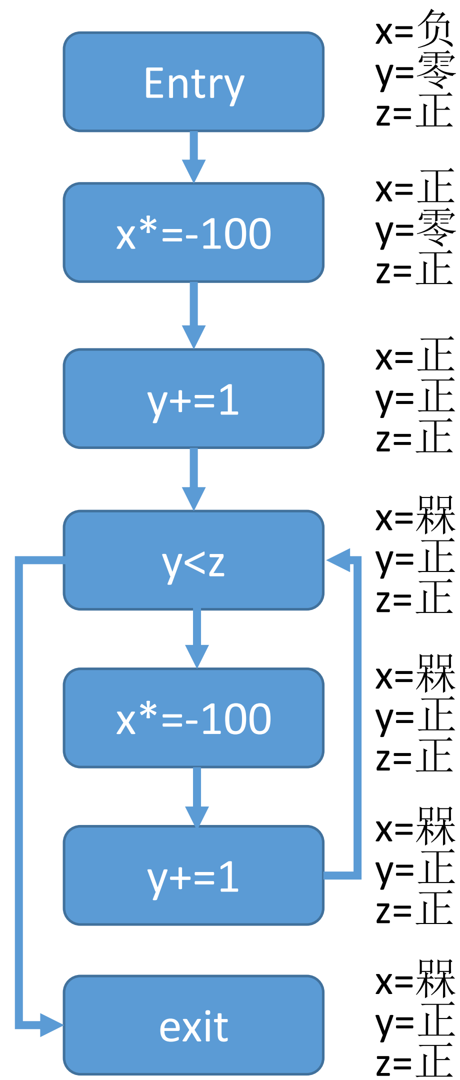
符号分析-示例

```
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;
}
```

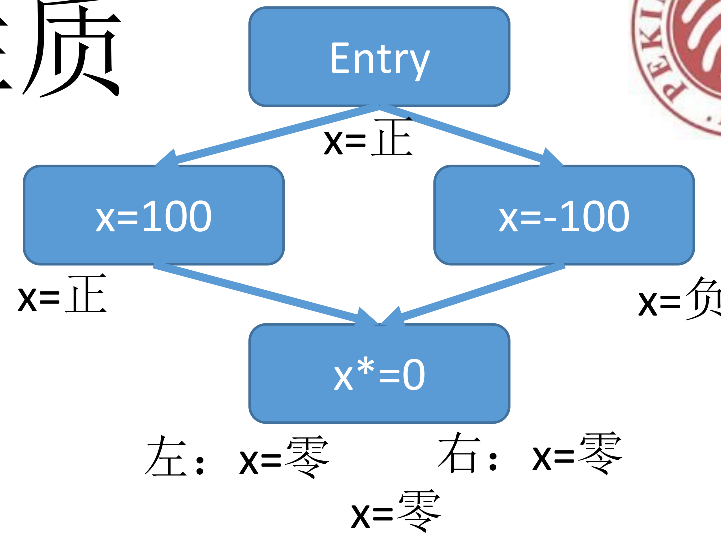


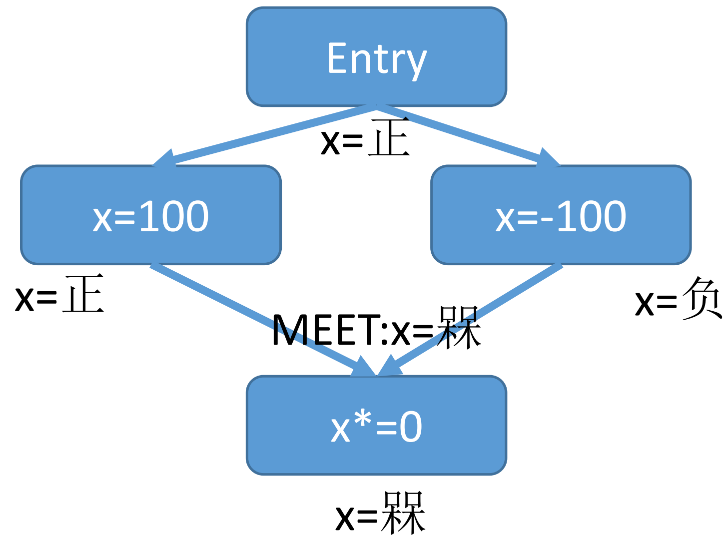
符号分析-算法

- 令$\boldsymbol{S} = \left\{\left(s_x, s_y, s_z\right) \middle| s_x, s_y, s_z \in \{正, 负, 零, 槑, \top\}\right\}$
- 每个结点的值为**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
  - DATA_{entry}=(负,零,正)
  - $\mathrm{DATA}_{其他结点}$=(T, T, T)
- 结点转换函数f_v: $\boldsymbol{S} \rightarrow \boldsymbol{S}$
  - $f_{exit} = id$
  - $f_{其他结点}$=根据相应语句进行计算
- 交汇运算$MEET_v$ =$\sqcap_{w \in pred(v)} \mathrm{DATA}_w$，⊓操作扩展到T：$x \sqcap \top = x$
- 结点更新运算$S_v = f_v(MEET_v)$
- 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

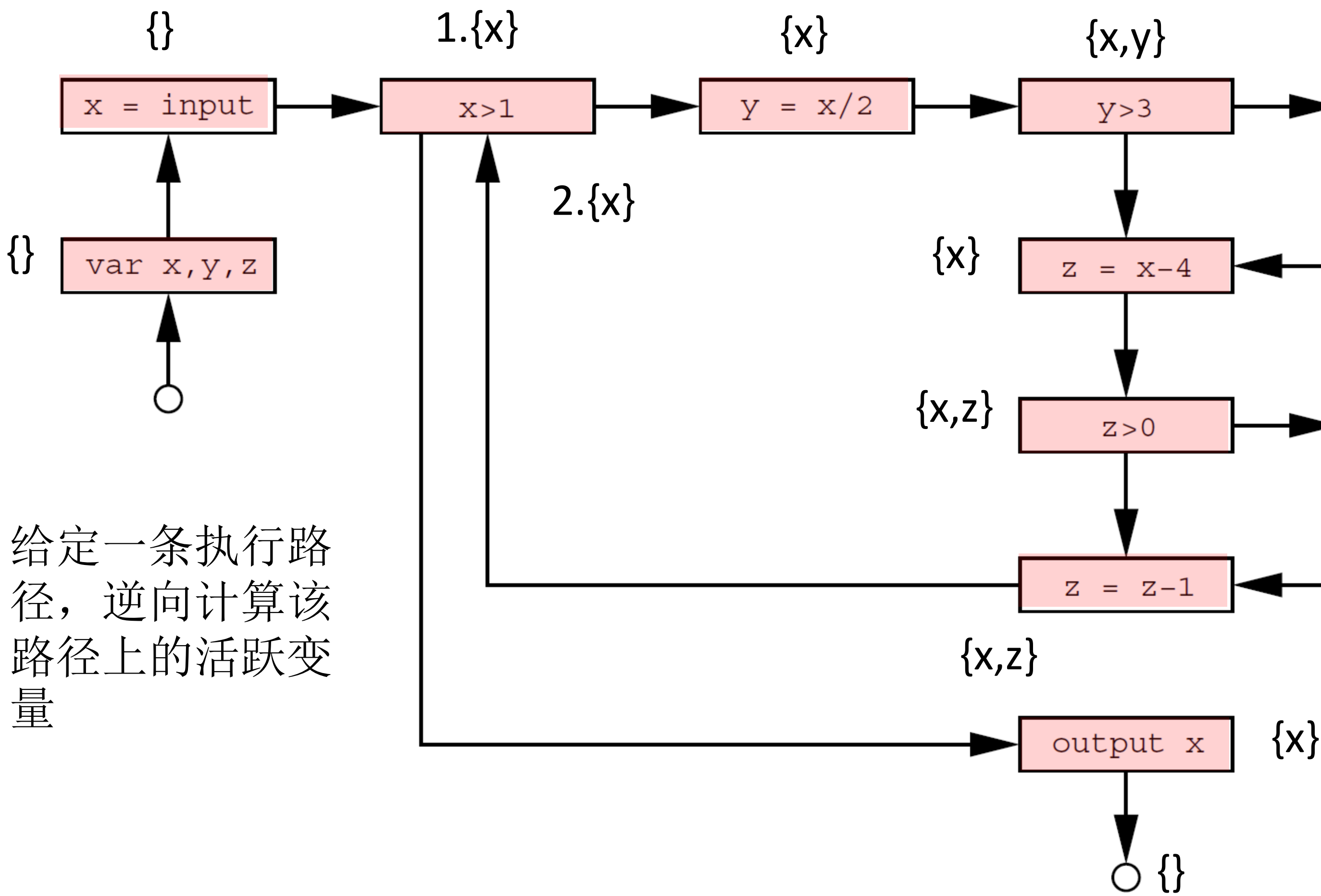
- 近似方案1：忽略掉程序的条件判断，认为所有分支都有可能到达
- 近似方案2：不在路径末尾做合并，在控制流汇合的所有位置提前做合并

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

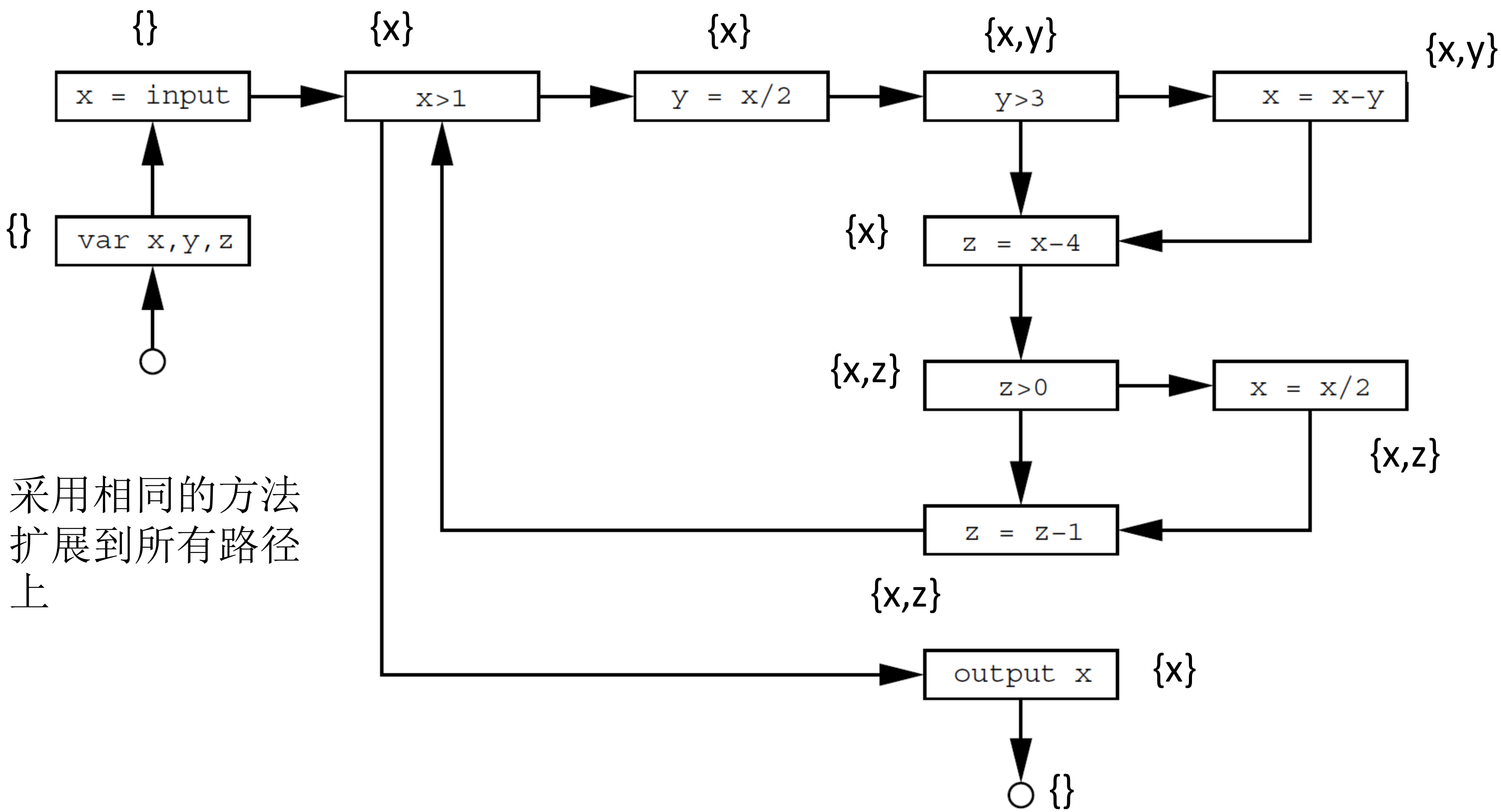
```
1.  var x,y,z;
2.  x = input;
3.  while (x>1) {
4.    y = x/2;
5.    if (y>3) x = x-y;
6.    z = x-4;
7.    if (z>0) x = x/2;
8.    z = z-1;
9.  }
10. output x;
```

活跃变量分析-基本思想



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：$\mathrm{DATA}_V = \{\}$
- 结点转换函数：$f_v(L) = (L \backslash \mathrm{KILL}_v) \cup GEN_v$
  - $GEN_v = vars(v)$
  - $KILL_v = \begin{cases} \{x\} & v \coloneqq \mathrm{x} = \mathrm{exp}; \\ \{x\} & v \coloneqq \mathrm{int\ x}; \\ \{\} & otherwise \end{cases}$
- 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）

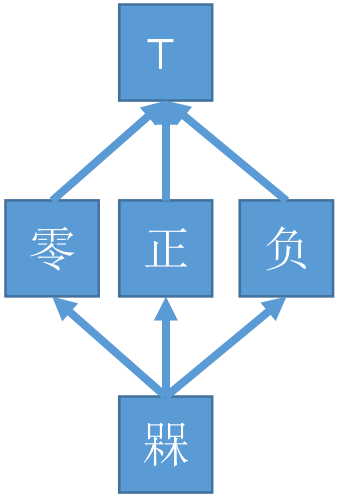
- 半格是一个二元组(S,⊓)，其中S是一个集合，⊓是一个交汇运算，并且任意$x, y, z \in S$都满足下列条件：
  - 幂等性idempotence: $x \sqcap x = x$
  - 交换性commutativity: $x \sqcap y = y \sqcap x$
  - 结合性associativity：$(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
  - 存在一个最大元⊤，使得$x \sqcap \top = x$

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$x \sqcap y = x$

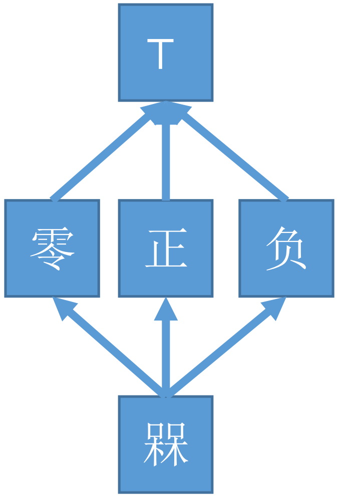
半格示例

- 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
  - $(s_1, t_1) \sqcap_{xy} (s_2, t_2) = (s_1 \sqcap_x s_2, t_1 \sqcap_y t_2)$
- 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$\sqcap_S$
- 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（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

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

  $\mathrm{If}\big(\mathrm{DATA}_{\mathrm{V}} \neq \mathrm{f}_{\mathrm{v}}(\mathrm{MEET}_{\mathrm{v}})\big)$ ToVisit ∪= 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