软件分析

数据流分析：示例

熊英飞
北京大学
2017

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

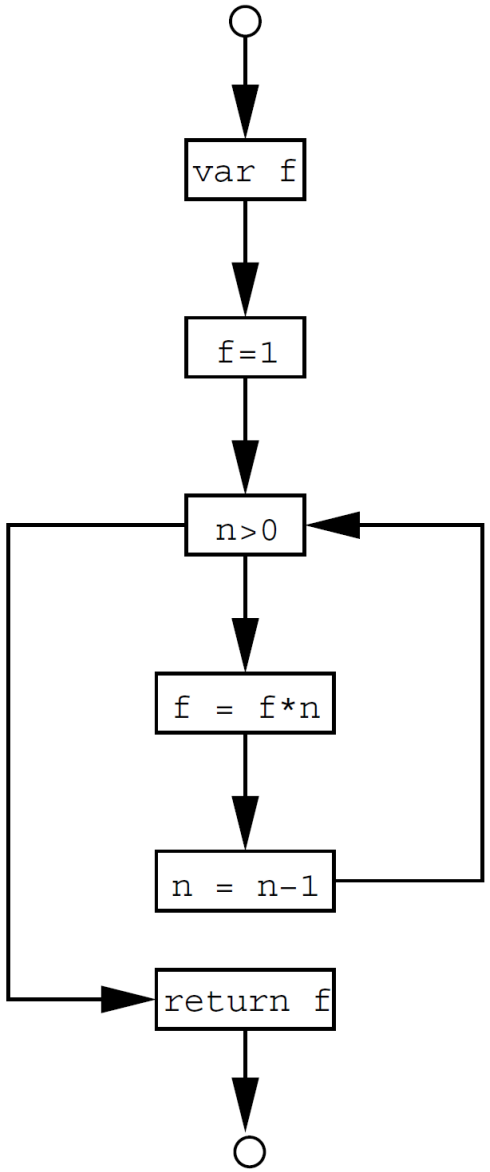
```
void Evil() {
    再来一次：
    向左走 goto 再来一次;
    向右走 return;
}
```

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、踩四种结果

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}
  - 踝={所有的整数和NaN}
- 运算（列标号●行标号）

+	正	负	零	踝
正	正			
负	踝	负		
零	正	负	零	
踝	踝	踝	踝	踝

-	正	负	零	踝
正	踝	负	负	踝
负	正	踝	正	踝
零	正	负	零	踝
踝	踝	踝	踝	踝

*	正	负	零	踝
正	正			
负	负	正		
零	零	零	零	
踝	踝	踝	踝	踝

/	正	负	零	踝
正	正	负	零	踝
负	负	正	零	踝
零	踝	踝	踝	踝
踝	踝	踝	踝	踝

符号分析-示例

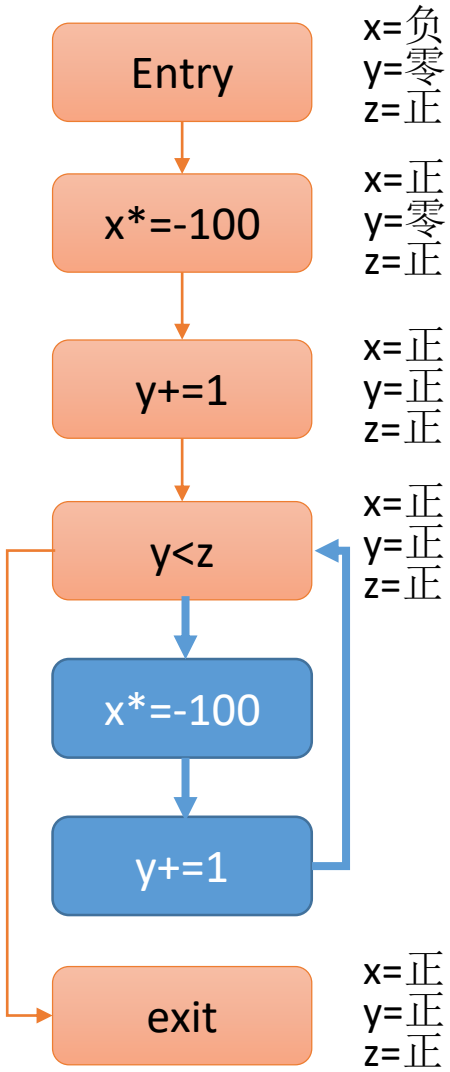
```
x*=-100;
y+=1;
while(y < z) {
    x *= -100;
    y += 1;
}
```

输入：x为负，y为零，z为正

输出：x为踩，y为正，z为正

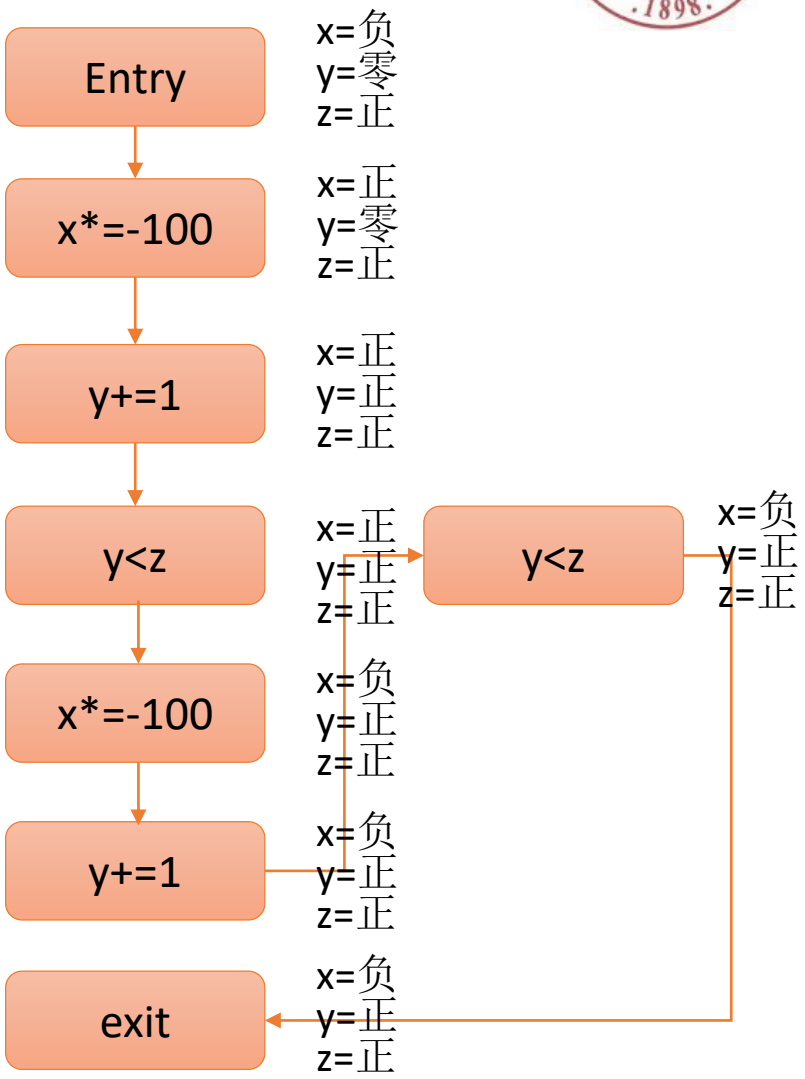
符号分析-基本思路

- 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：
  - $\sqcap(v_1, v_2) = \begin{cases} v_1 & \text{如果} v_1 = v_2 \\ \text{踩} & \text{其他情况} \end{cases}$
- ⊓((正正正),(负正正))=(踩正正)

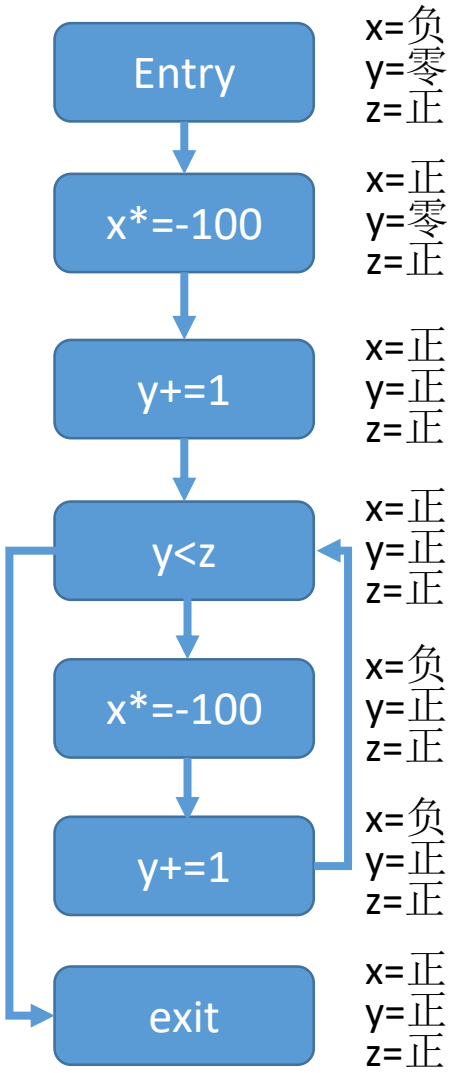


符号分析-基本思路

- 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

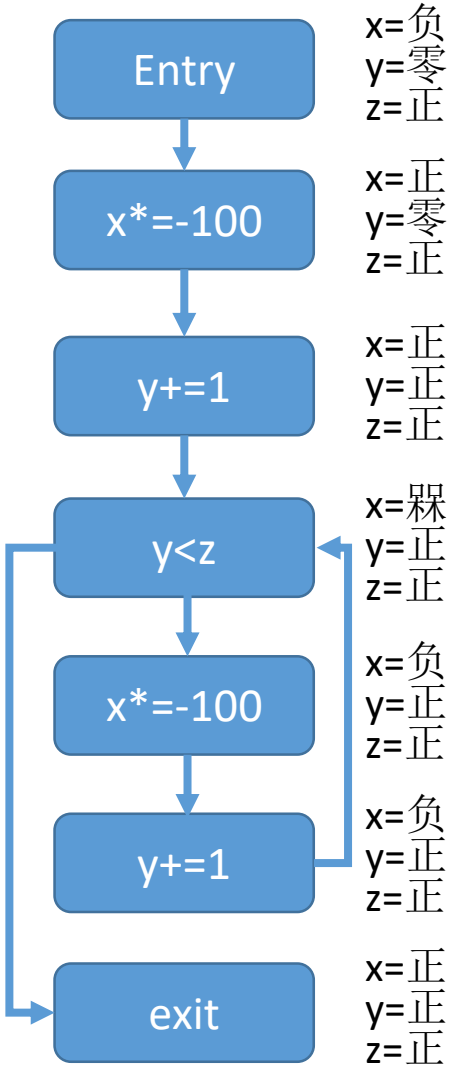
符号分析-示例

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



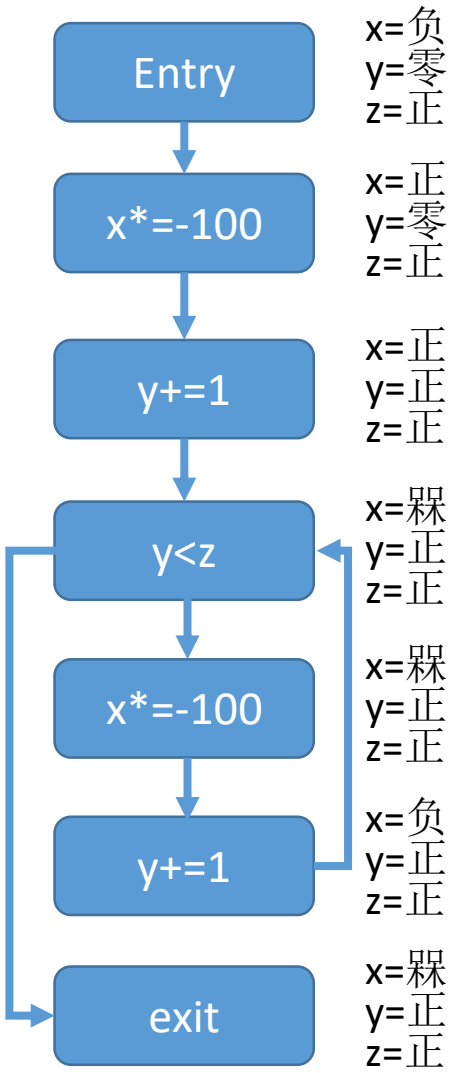
符号分析-示例

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



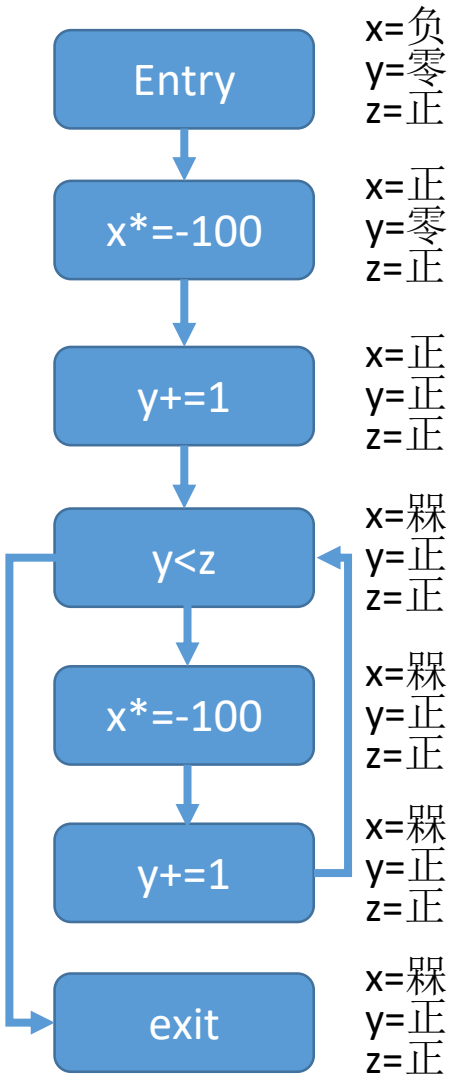
符号分析-示例

```
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;
}
```

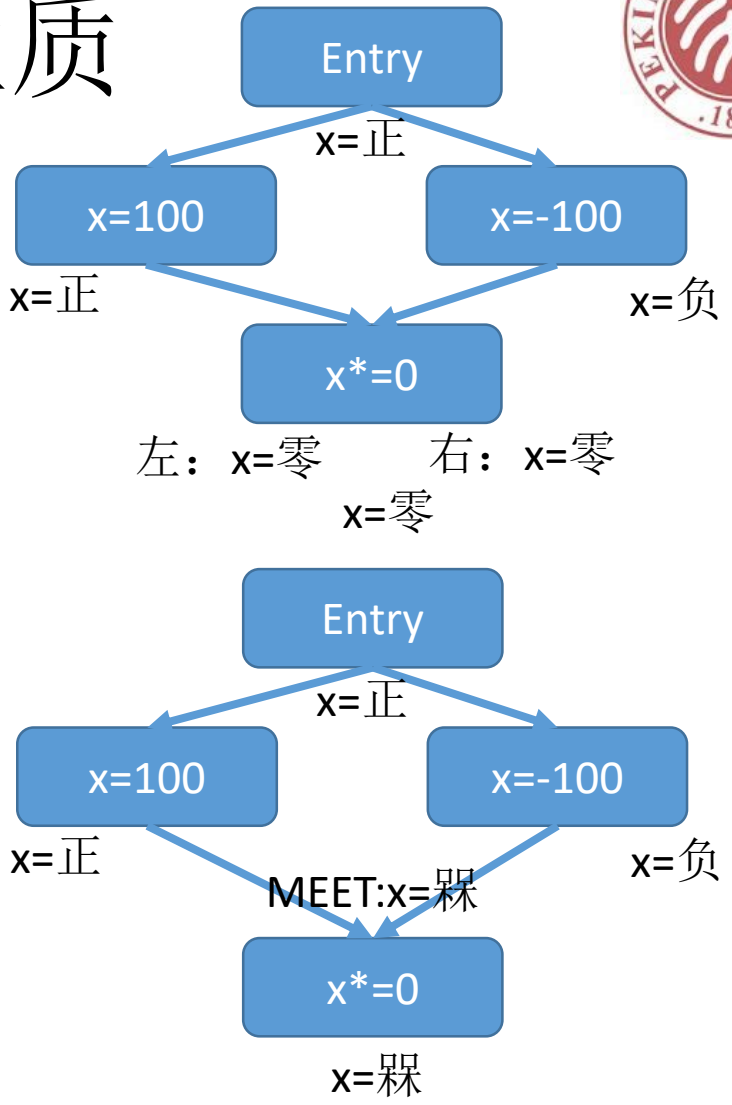


符号分析-算法

- 令$\mathbf{S} = \{(s_x, s_y, s_z) | s_x, s_y, s_z \in \{正, 负, 零, 鞣, \top\}\}$
- 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)
  - $\mathrm{DATA}_{其他结点}$=(⊤, ⊤, ⊤)
- 结点转换函数$f_v: \boldsymbol{S} \to \boldsymbol{S}$
  - $f_{exit} = id$
  - $f_{其他结点}$=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⊤：$x \sqcap \top = x$
- 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

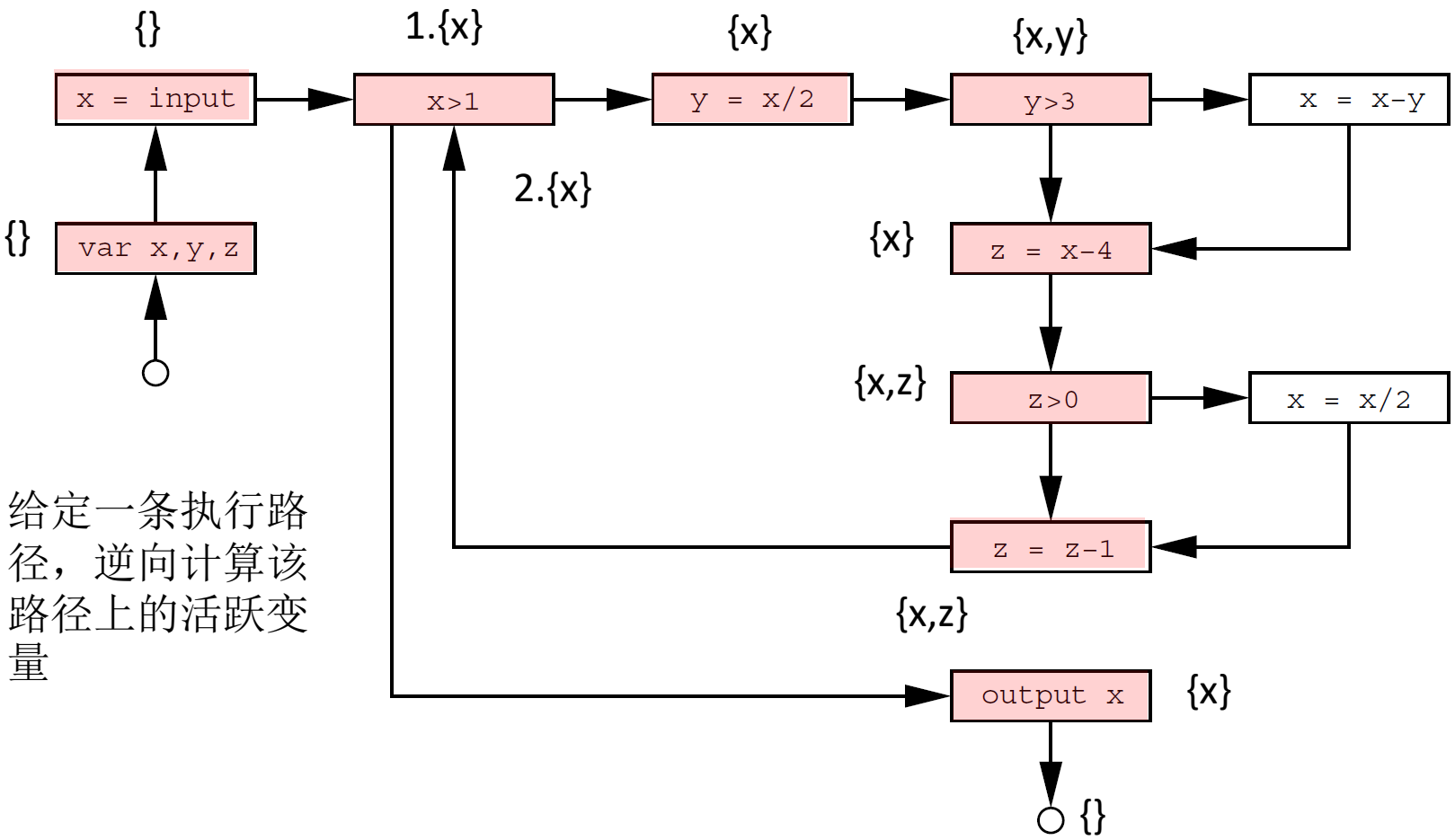
- 近似方案1：忽略掉程序的条件判断，认为所有分支都有可能到达
- 近似方案2：不在路径末尾做合并，在控制流汇合的所有位置提前做合并

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

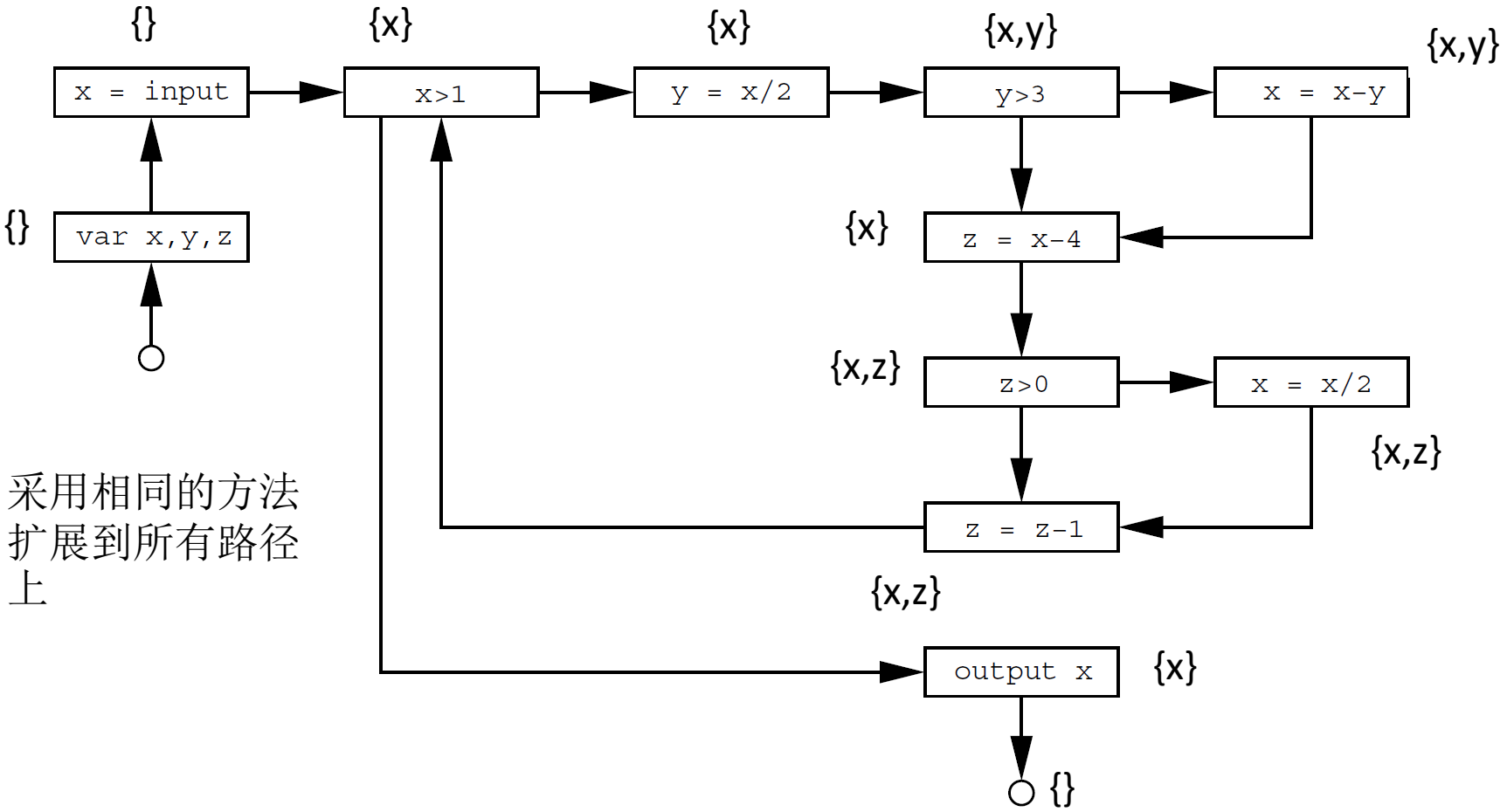
```
1.  var x,y,z;
2.  x = input;
3.  while (x>1) {
4.    y = x/2;
5.    if (y>3) x = x-y;
6.    z = x-4;
7.    if (z>0) x = x/2;
8.    z = z-1;
9.  }
10. output x;
```

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：$f_v(L) = (L \backslash \mathrm{KILL}_v) \cup GEN_v$
  - $GEN_v = vars(v)$
  - $KILL_v = \begin{cases} \{x\} & v \coloneqq \mathrm{x} = \exp; \\ \{x\} & v \coloneqq \mathrm{int\ x}; \\ \{\} & otherwise \end{cases}$
- 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？