软件分析

数据流分析：性质和扩展

熊英飞
北京大学
2018

复习：数据流分析

- 数据流分析中采用了哪两种近似方案？

- 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

复习：半格

- 已知半格 $(S,\sqcap_S)$ 和半格 $(T,\sqcap_T)$ 的高度分别是x和y，求半格$(S\times T,\sqcap_{ST})$的高度
  - $(s_1,t_1)\sqcap_{ST}(s_2,t_2)=(s_1\sqcap_S s_2,t_1\sqcap_T t_2)$

- 答案：x+y-1

复习：单调（递增）函数

- 以下函数是否是单调的：
  - $f(x) = x - 1$
  - 导数各处不为0的函数
  - 求集合x的补集
  - $f(x) = g \circ h(x)$，已知g和h是单调的
  - $f(X) = g(X) \cup h(X)$，已知f,h,g是定义在集合上的函数，g和h单调
  - $f(X) = g(X) - h(X)$，已知f,h,g是定义在集合上的函数，g和h单调

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

$\mathrm{DATA}_{\mathrm{entry}} = \mathrm{I}$

$\forall \mathrm{v} \in (\mathrm{V} - \mathrm{entry}): \mathrm{DATA}_{\mathrm{v}} \leftarrow \top$

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

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

  $\mathrm{If}\left(\mathrm{DATA}_{\mathrm{V}} \neq \mathrm{f}_{\mathrm{v}}(\mathrm{MEET}_{\mathrm{v}})\right)$ ToVisit $\cup$= 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

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槑比较小，$\top$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

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”，$f_v(\text{正}) \sqcap f_v(\text{负})$
- 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，$\mathrm{DATA}_{\mathrm{v_i}} = \sqcap_{v_1 v_2 v_3 \ldots v_i \in P} f_{v_i} \circ f_{v_{i-1}} \circ \cdots \circ f_{v_1}(I_{entry})$
  - 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$f\left(f\left(f(\top_s)\right)\right) \sqsubseteq f\left(f(\top_s)\right)$
      - 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
  - $(DATA_{v_1}, DATA_{v_2}, \ldots, DATA_{v_n}) := \mathrm{F}(DATA_{v_1}, DATA_{v_2}, \ldots, DATA_{v_n})$
    - 为什么没有$DATA_{entry}$
- 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：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
  - $[a, b] \sqcap [c, d] = [\min(a, c), \max(b, d)]$
- 变换函数：
  - 在区间上执行对应的加减操作
  - $[a, b] + [c, d] = [a + c, b + d]$
  - $[a, b] - [c, d] = [a - d, b - c]$

- 不满足单调框架条件：半格不是有限的
  - 分析可能会不终止

区间分析改进

- 程序中的数字都是有上下界的，假设超过上下界会导致程序崩溃
  - $[a,b]+[c,d]=$
$$\begin{cases} \emptyset & a+c>int\_max \\ (a+c,\min(b+d,int\_max)) & a+c\le int\_max \end{cases}$$

- 原分析终止，但需要int_max步才能收敛

Widening & Narrowing

基础Widening

- 定义单调函数w把结果进一步抽象来加速收敛
  - 原始转换函数f
  - 新转换函数$w \circ f$

- 定义有限集合B={-∞, 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：
  - $w([15,75]) = [10,100]$

基础Widening的安全性

- 如果$w(x) \sqsubseteq x$，则分析结果保证安全

- 安全性讨论
  - 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

基础Widening的收敛性

- 如果w是单调函数，则基础Widening收敛
  - 因为$w \circ f$仍然是单调函数

一般Widening

- 更一般的widening同时参考更新前和更新后的值。
  - 原数据流分析算法更新语句：
    - $\mathrm{DATA_v} \leftarrow \mathrm{f_v}(\mathrm{MEET_v})$
  - 引入widening算子∇：
    - $\mathrm{DATA_v} \leftarrow \mathrm{DATA_v}\ \nabla\ \mathrm{f_v}(\mathrm{MEET_v})$

- 用更一般的widening可以实现更快速的收敛，如
  - $[a, b]\nabla[c, d] = [x, y]$ where
    - $x = \begin{cases} c & c \geq a \\ -\infty & c < a \end{cases}$
    - $y = \begin{cases} d & d \leq b \\ +\infty & d > b \end{cases}$

一般Widening的性质

- 如果$x\nabla y \sqsubseteq y$，则一般Widening的分析结果保证安全性

- Widening算子必须保证结果是收敛的
- 注意：Widening算子本身通常不保证g(x)单调递增
  - 假设$f(x)$对[1,1]和[1,2]都返回[1,2]
  - $g([1,1]) = [1,1]\nabla[1,2] = [1,\infty]$
  - $g([1,2]) = [1,2]\nabla[1,2] = [1,2]$

Widening的问题

- Widening牺牲精确度来保证收敛性，有时该牺牲很大。
- 令基础widening的有限集合为$\{-\infty, 0, 1, 7, +\infty\}$

```
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

$[x \mapsto \perp, y \mapsto \perp]$
$[x \mapsto [8,8], y \mapsto [0,1]]$
$[x \mapsto [8,8], y \mapsto [0,2]]$
$[x \mapsto [8,8], y \mapsto [0,3]]$
…

不使用Widening，收敛慢或不收敛

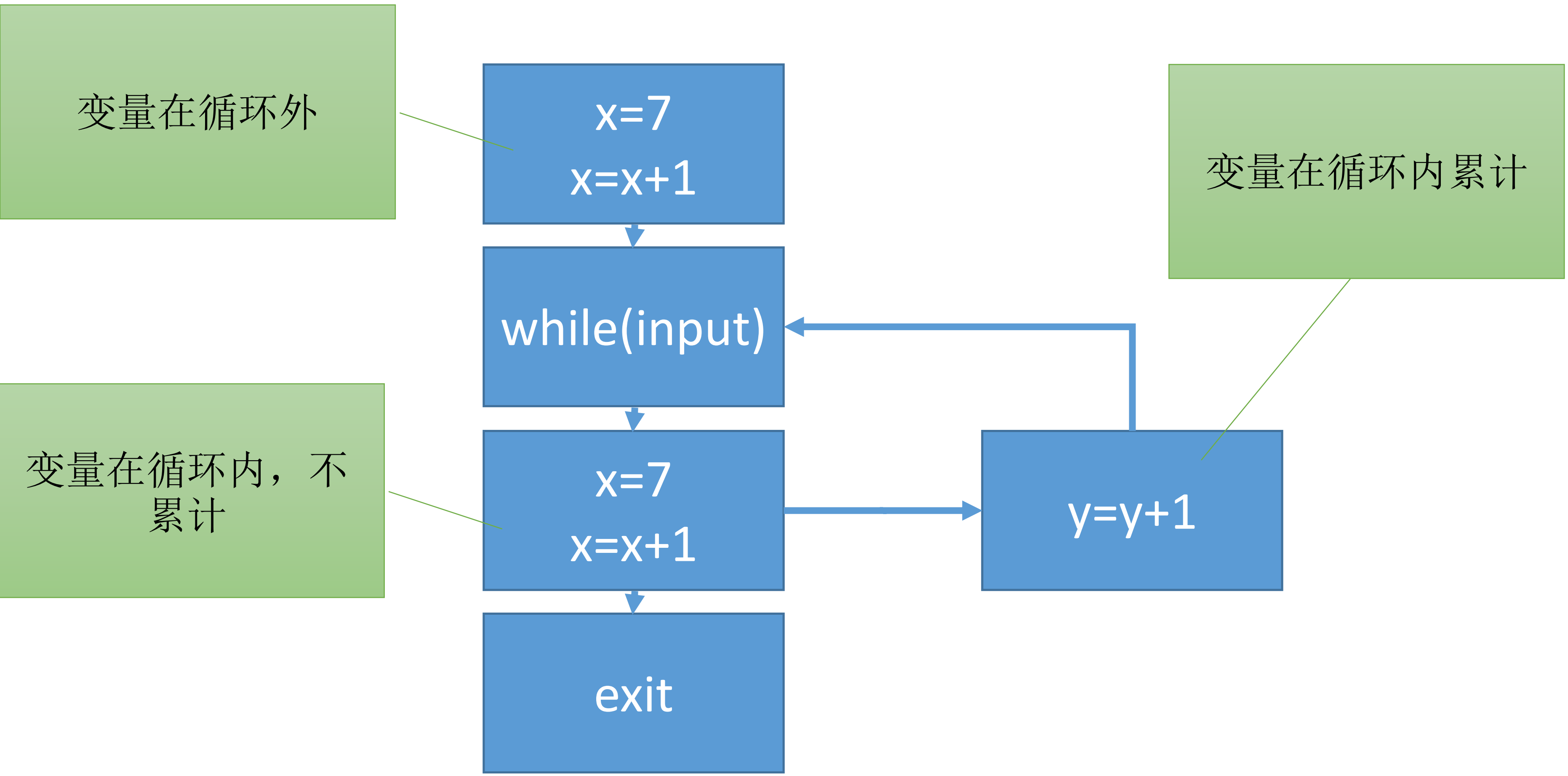
$[x \mapsto \perp, y \mapsto \perp]$
$[x \mapsto [7,\infty], y \mapsto [0,0]]$
$[x \mapsto [7,\infty], y \mapsto [0,7]]$
$[x \mapsto [7,\infty], y \mapsto [0,\infty]]$

使用基础Widening不精确

$[x \mapsto \perp, y \mapsto \perp]$
$[x \mapsto [7,\infty], y \mapsto [0,0]]$
$[x \mapsto [7,\infty], y \mapsto [0,\infty]]$

使用基础Widening收敛更快，不精确

分析一般Widening的例子



经过有限次迭代能收敛的情况丢失精度

Narrowing

- 通过再次应用原始转换函数对Widening的结果进行修正

```
y = 0; x = 7; x = x+1;
while (input) {
  x = 7;
  x = x+1;
  y = y+1;
}
```

可以得到结果

$$[\mathbf{x} \mapsto [8,8], \mathbf{y} \mapsto [0,\infty]]$$

Narrowing的安全性

- 分析数据流分析收敛性的时候，我们说过整体数据流分析可以看做一个函数F
- 令
  - 原数据流分析的函数为F，收敛于I_F
  - 经过Widening的函数为G，收敛于I_G
- 那么有
  - 因为 $I_F \sqsupseteq I_G$
  - 所以 $I_F = F(I_F) \sqsupseteq F(I_G) \sqsupseteq G(I_G) = I_G$
  - 即 $I_F \sqsupseteq F(I_G) \sqsupseteq I_G$
- 类似可以得到
  - $I_F \sqsupseteq F^k(I_G) \sqsupseteq I_G$
- 即Narrowing保证安全性

Narrowing的收敛性

- Narrowing不保证收敛
- 收敛的情况下也不保证快速收敛

- 解决方案：应用widening技术到narrowing过程中

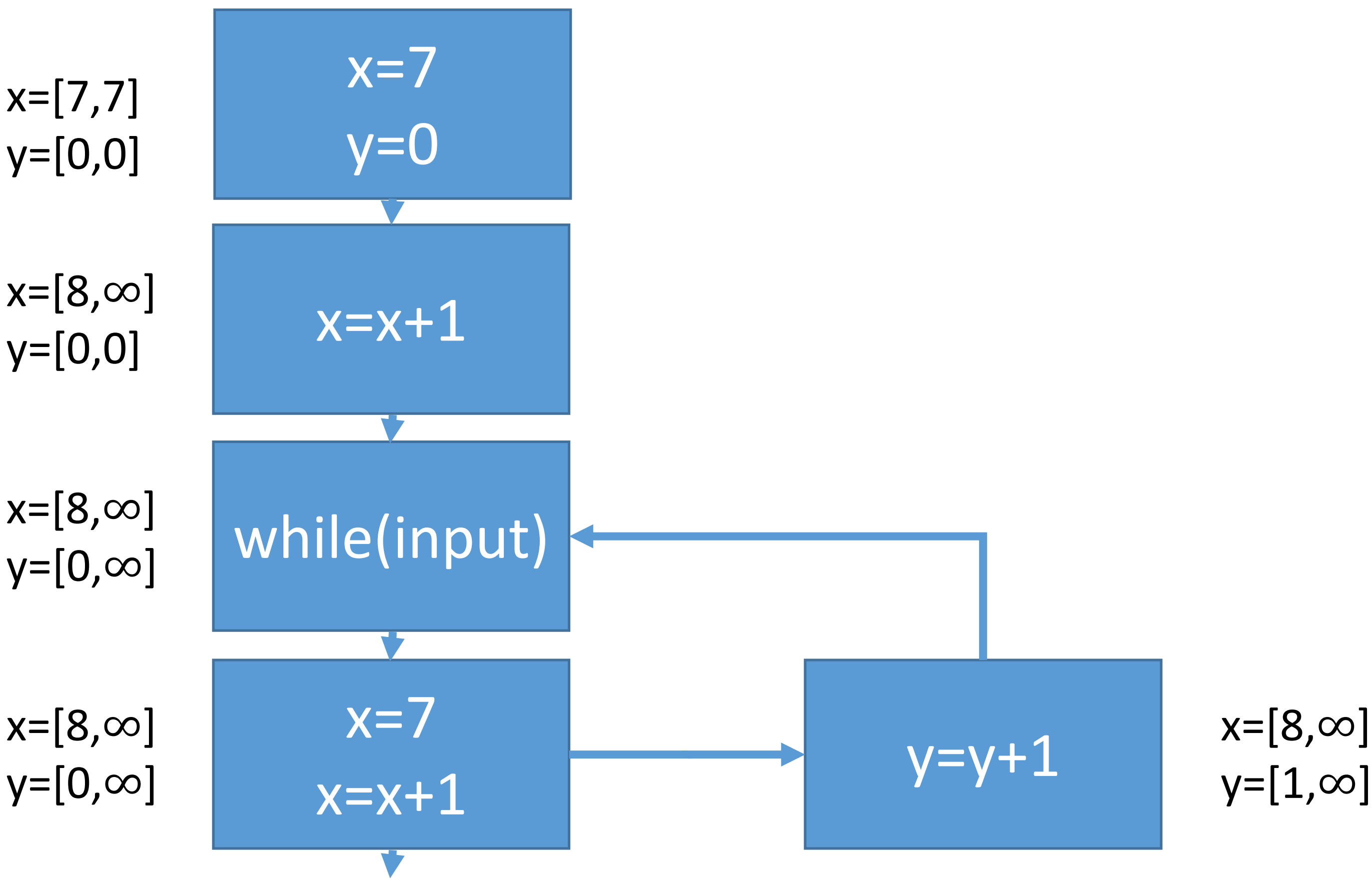
Narrowing算子

- 引入narrowing算子Δ：

$$\mathrm{DATA_v} \leftarrow \mathrm{DATA_v}\ \Delta\ \mathrm{f_v}(\mathrm{MEET_v})$$

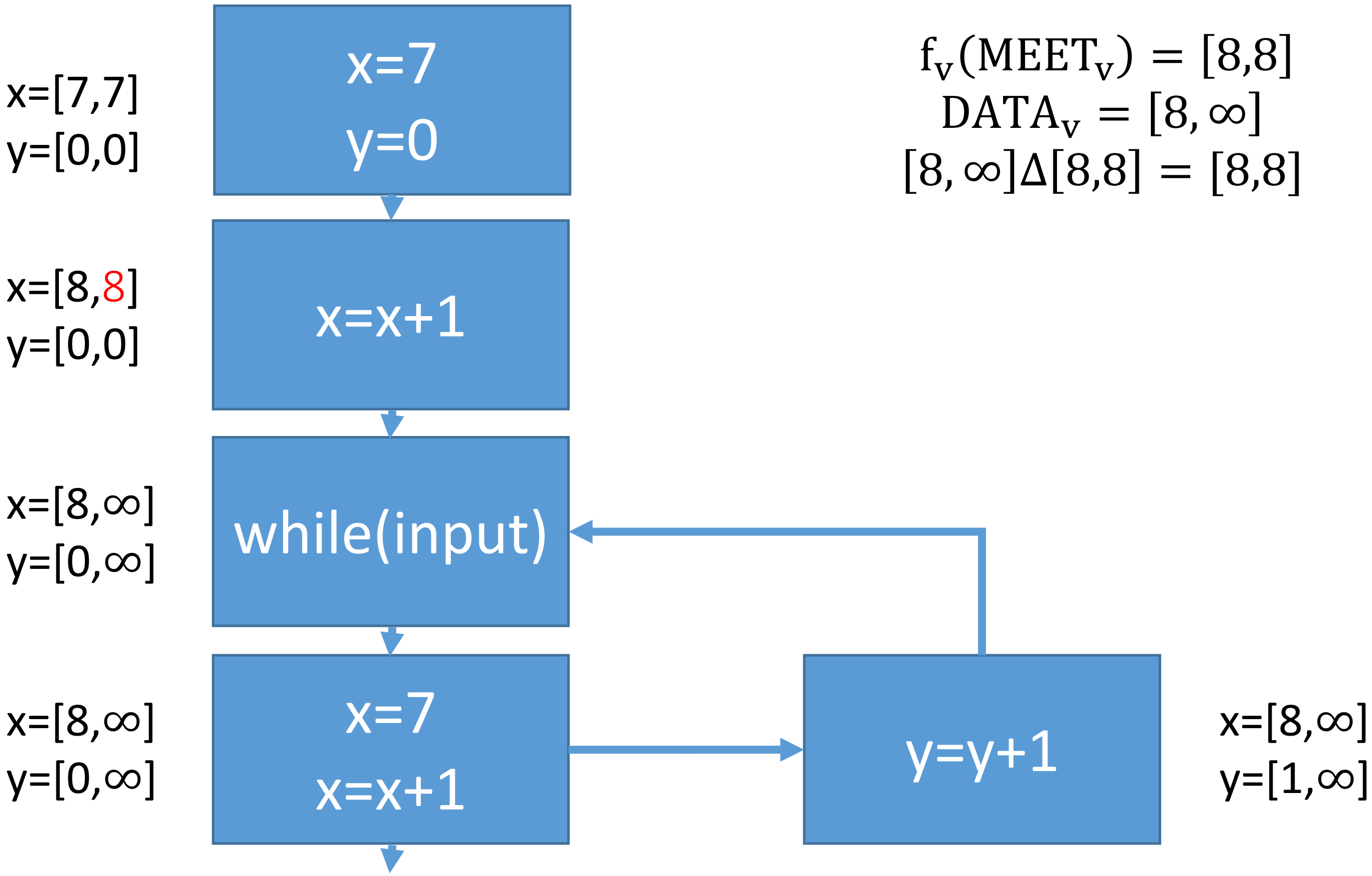
- 如
  - $[a,b]\Delta[c,d]=[x,y]$, where
    - $x=\begin{cases} a & a\neq -\infty \\ c & a=-\infty \end{cases}$
    - $y=\begin{cases} b & b\neq +\infty \\ d & b=+\infty \end{cases}$
  - 即：已经收敛到的整数不改动，只重新计算被widening扩展到的无穷大

Narrowing例子

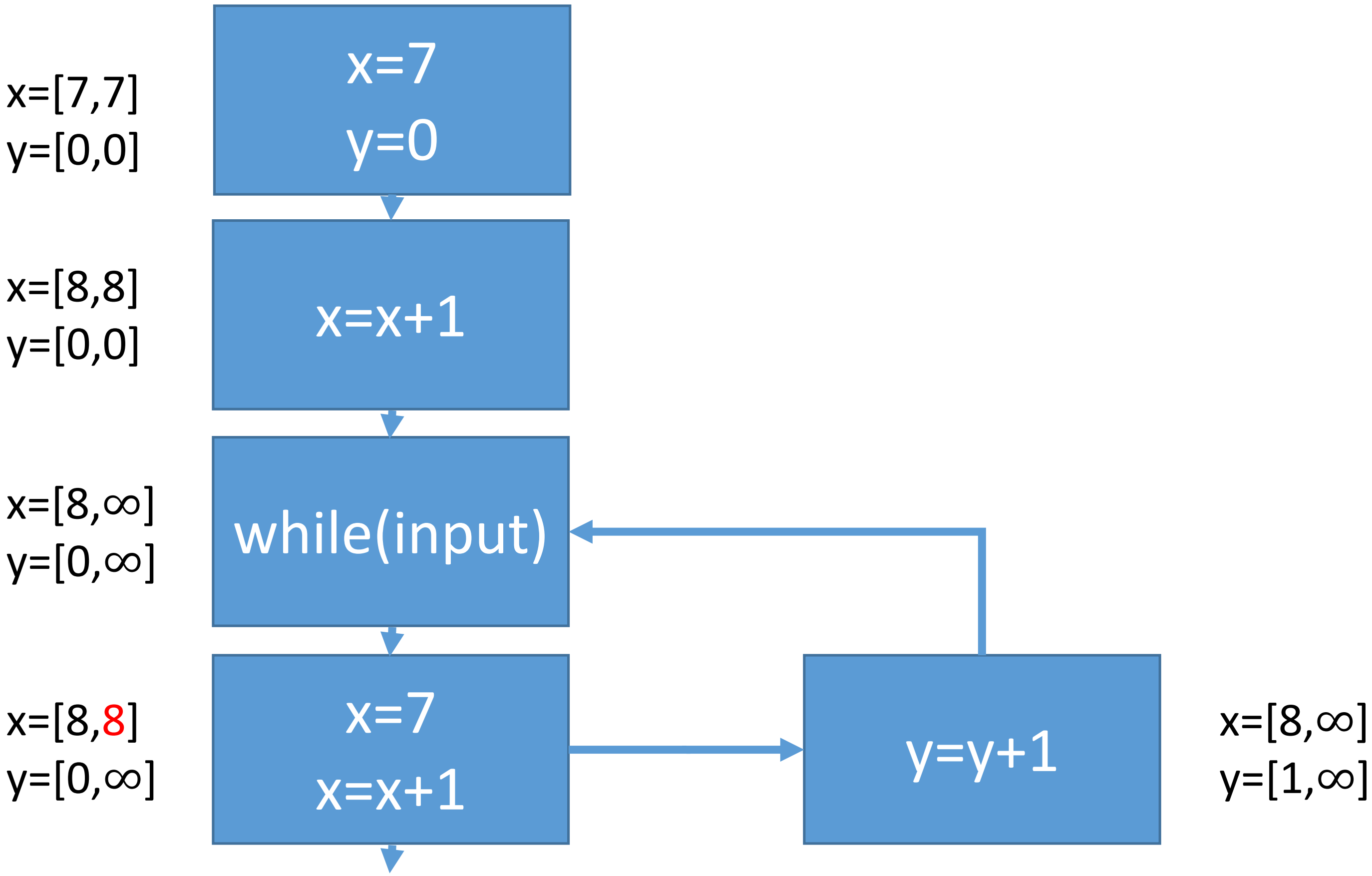


Narrowing例子

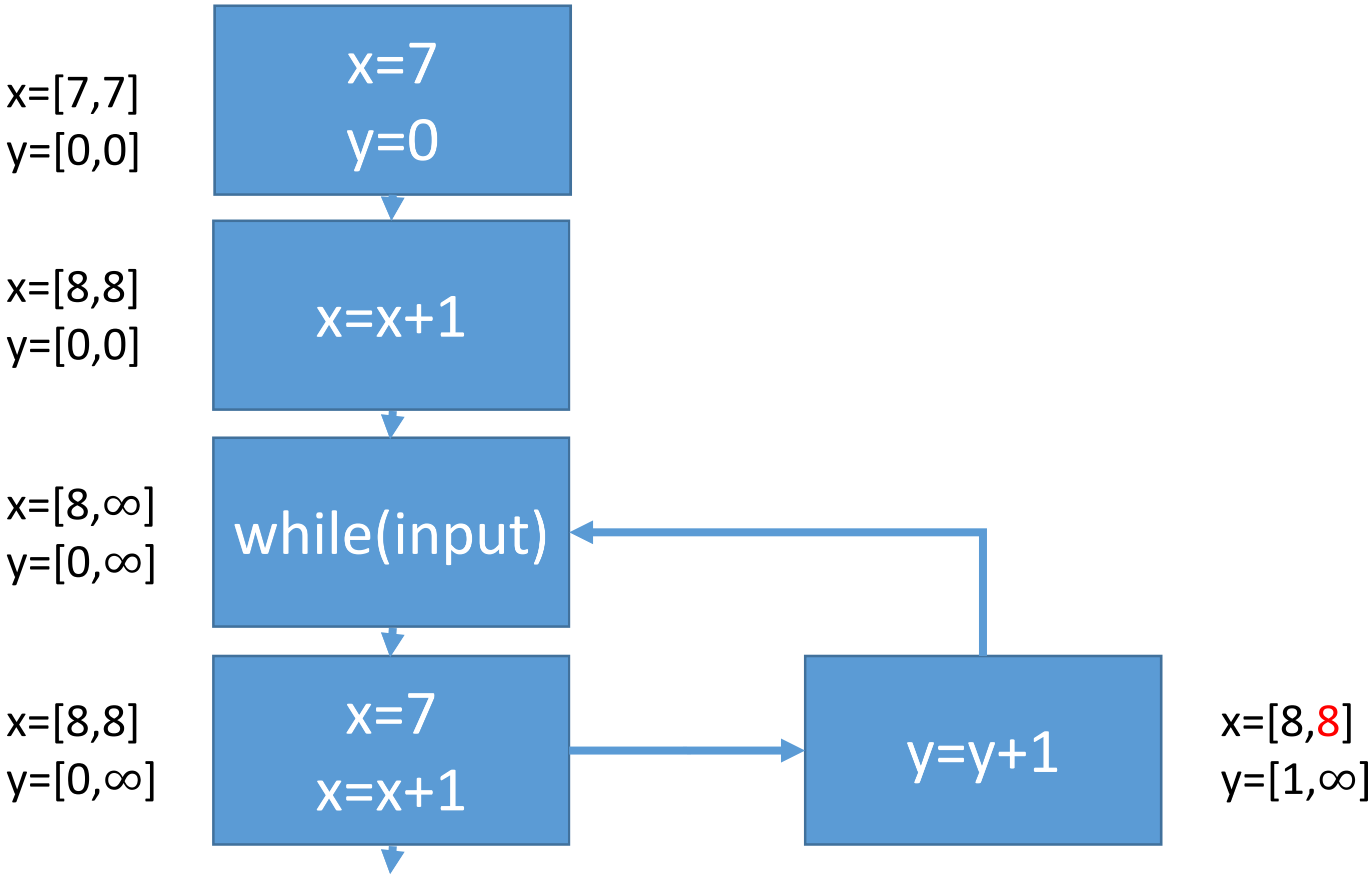
$$f_v(\mathrm{MEET}_v) = [8,8]$$
$$\mathrm{DATA}_v = [8,\infty]$$
$$[8,\infty]\Delta[8,8] = [8,8]$$



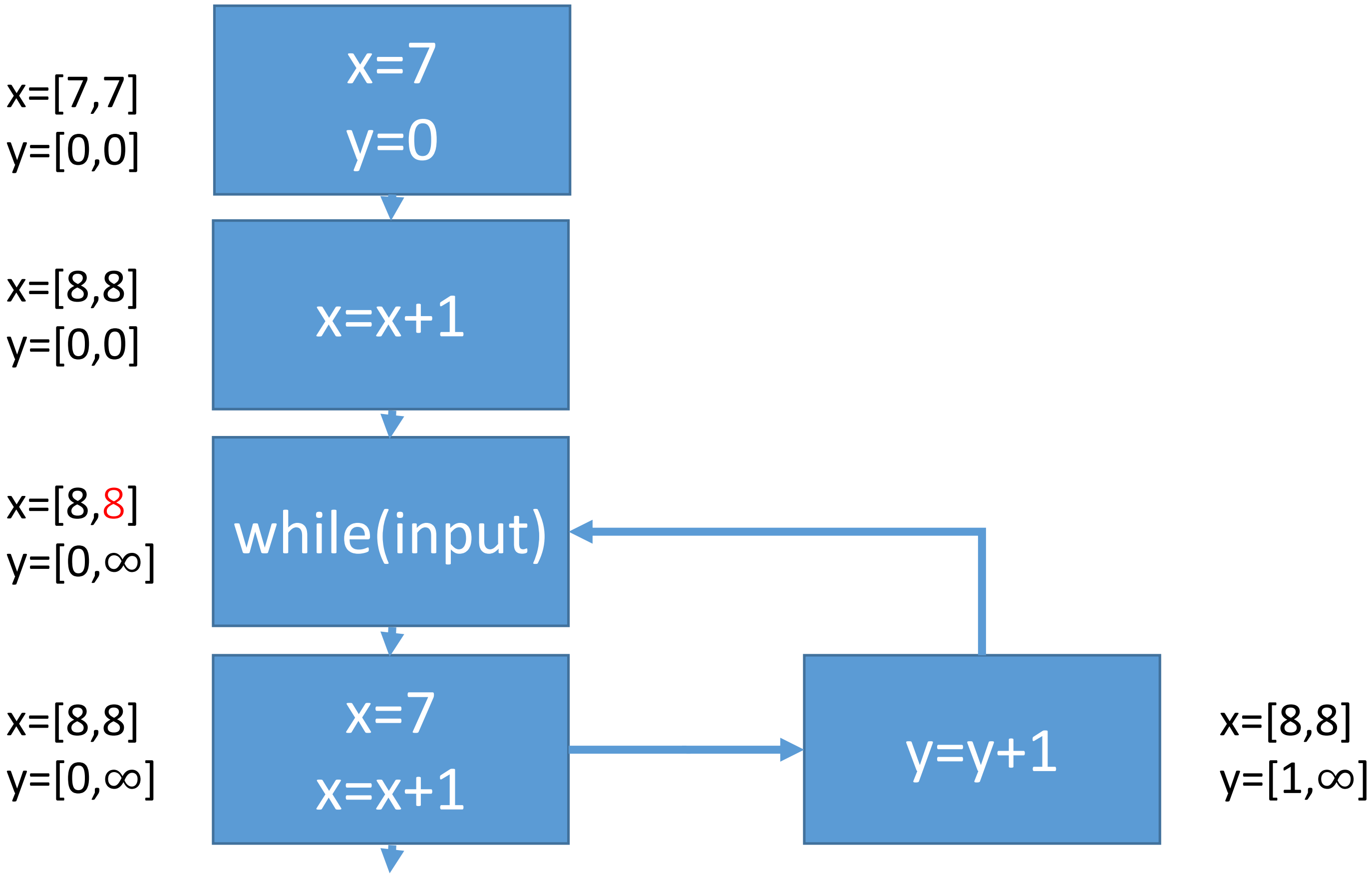
Narrowing例子



Narrowing例子



Narrowing例子



Narrowing算子的性质

- 同widening的情形类似：

- 如果$x \Delta y \sqsubseteq y$，则narrowing保证安全
- Narrowing算子需要保证收敛

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