软件分析

常见抽象域和符号抽象

熊英飞
北京大学
2018

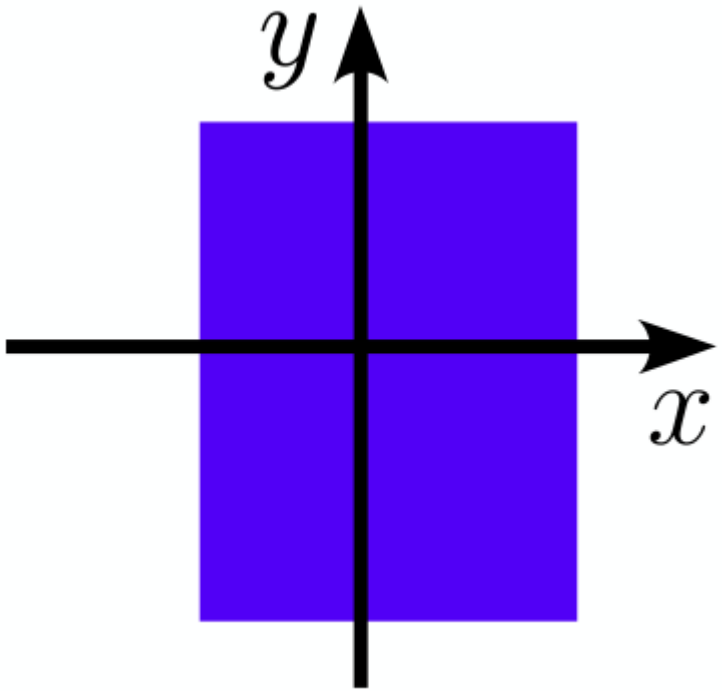
关系抽象

- 区间分析将每个变量的可能取值抽象成一个区间，不考虑变量之间的关系。
- 这类不考虑变量之间关系的抽象称为非关系抽象。

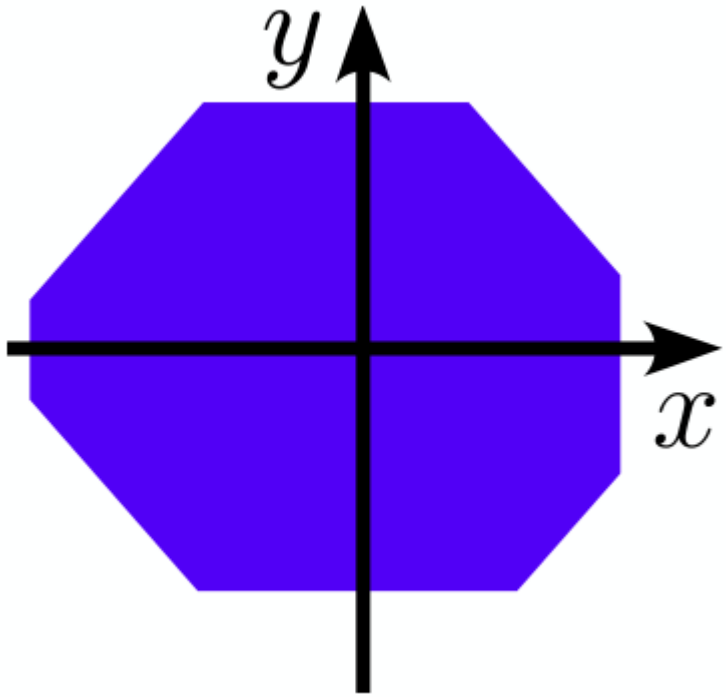
- 考虑变量之间关系的抽象称为关系抽象。

关系抽象举例：八边形

假设程序中只有x和y两个变量



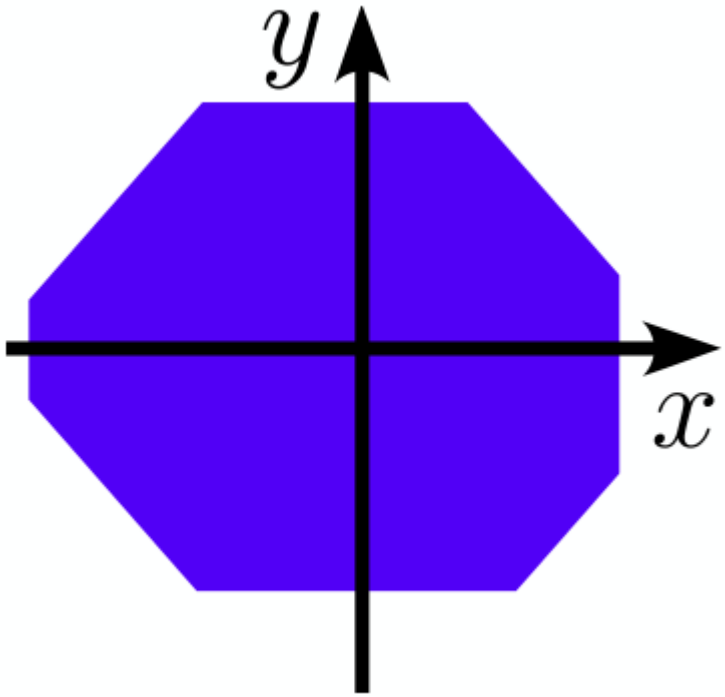
区间抽象形成一个矩形



加上4条45度的线来形成八边形

关系抽象举例：八边形

假设程序中只有x和y两个变量

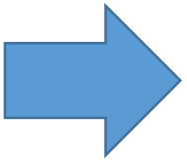


加上4条45度的线来形成八边形

x的上界a_1：$x \leq a_1$
x的下界a_2：$x \geq a_2$
y的上界a_3：$y \leq a_3$
y的下界a_4：$y \geq a_4$
x+y的上界a_5：$x + y \leq a_5$
x+y的下界a_6：$x + y \geq a_6$
x-y的上界a_7：$x - y \leq a_7$
x-y的下界a_8：$x - y \geq a_8$

关系抽象举例：八边形

x的上界a_1：$x \le a_1$
x的下界a_2：$x \ge a_2$
y的上界a_3：$y \le a_3$
y的下界a_4：$y \ge a_4$
x+y的上界a_5：$x + y \le a_5$
x+y的下界a_6：$x + y \ge a_6$
x-y的上界a_7：$x - y \le a_7$
x-y的下界a_8：$x - y \ge a_8$



x的上界$\frac{1}{2}a_1$：$+x + x \le a_1$
x的下界$-\frac{1}{2}a_2$：$-x - x \le a_2$
y的上界$\frac{1}{2}a_3$：$+y + y \le a_3$
y的下界$-\frac{1}{2}a_4$：$-y - y \le a_4$
x+y的上界a_5：$+x + y \le a_5$
x+y的下界$-a_6$：$-x - y \le a_6$
x-y的上界a_7：$+x - y \le a_7$
x-y的下界$-a_8$：$-x + y \le a_8$

即$\pm v_1 \pm v_2 \le a$，其中$v_1, v_2 \in \{x, y\}$

对多个变量进行抽象

- 对任意两个变量记录八边形
- 即$\pm v_1 \pm v_2 \leq a$，其中v_1, v_2为程序上的任意变量
- 可用矩阵表示

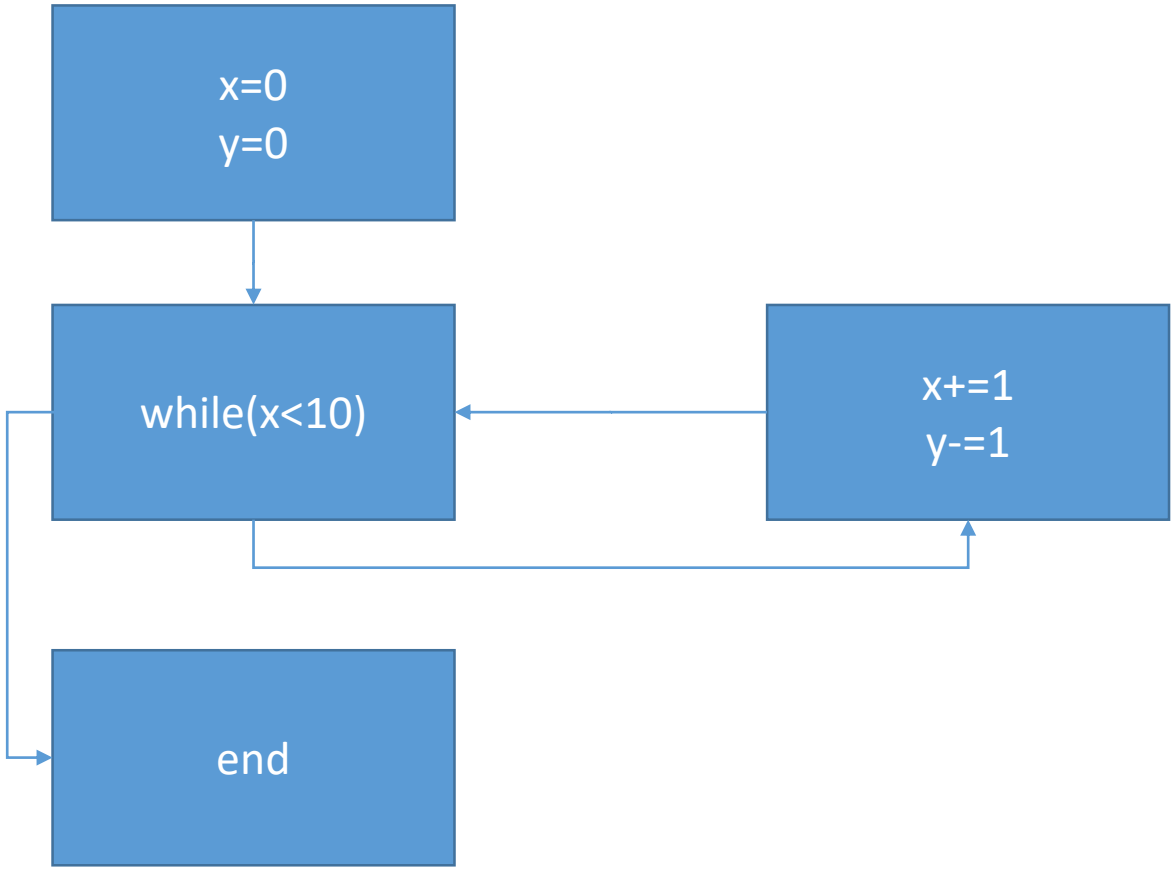
	+x	-x	+y	-y
+x	10	-	-	-
-x	-	0	-	-
+y	10	5	20	-
-y	-2	5	-	-10

2个变量的矩阵。更多变量需要更大的矩阵。

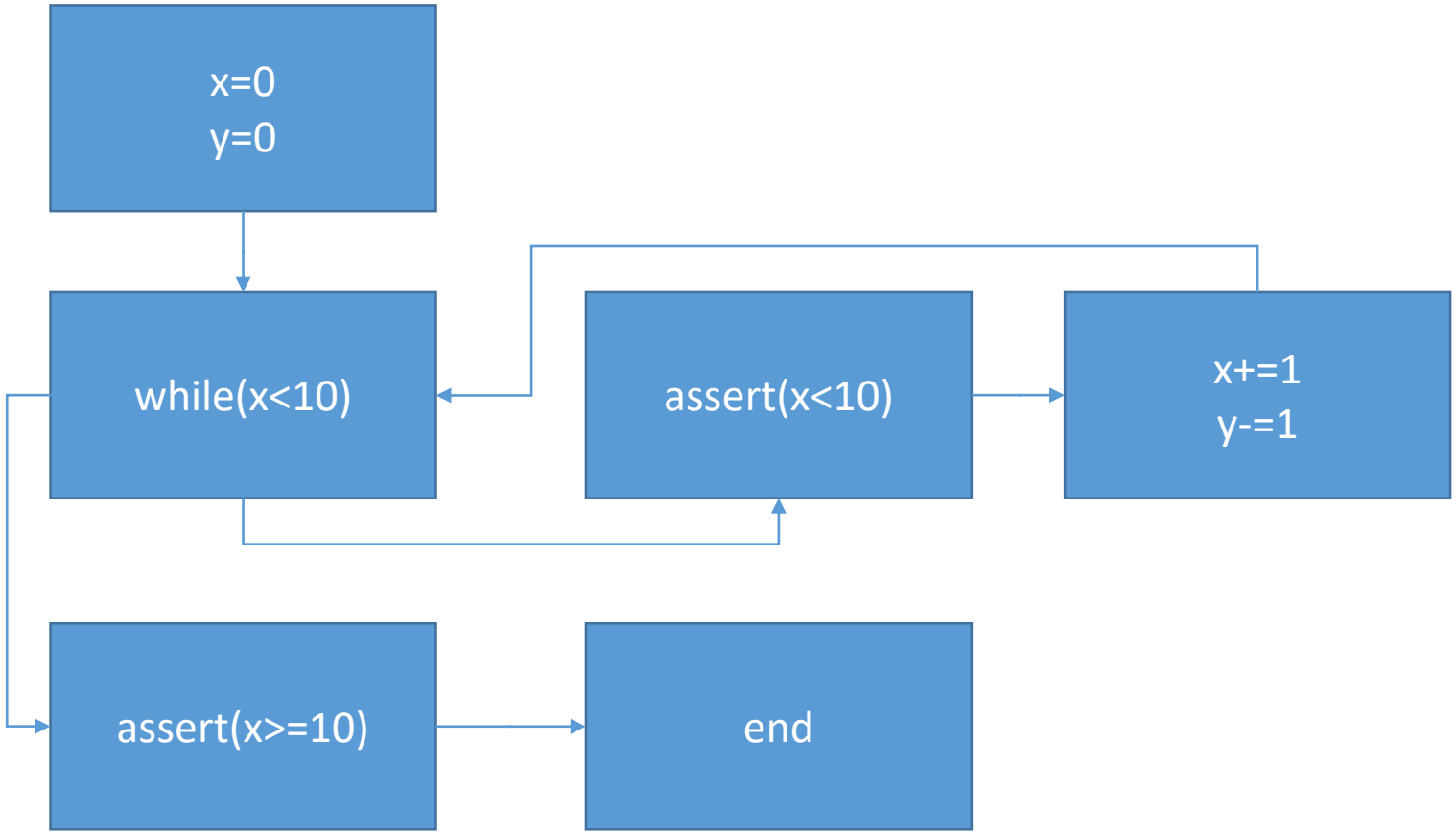
八边形上的计算

- x=x+1
  - 将x有关的八边形沿x轴移动1个单位
- $z = x \cup y$
  - 对于任意变量v，令<z,v>的八边形为包住<x,v>和<y,v>的最小八边形
- $z = x \cap y$
  - 对于任意变量v，令<z,v>的八边形为包住<x,v>和<y,v>公共部分的最小八边形
- 更多计算方法参考原始论文：
  - Miné A. The octagon abstract domain[J]. Higher-Order and Symbolic Computation, 2006, 19(1):31-100.

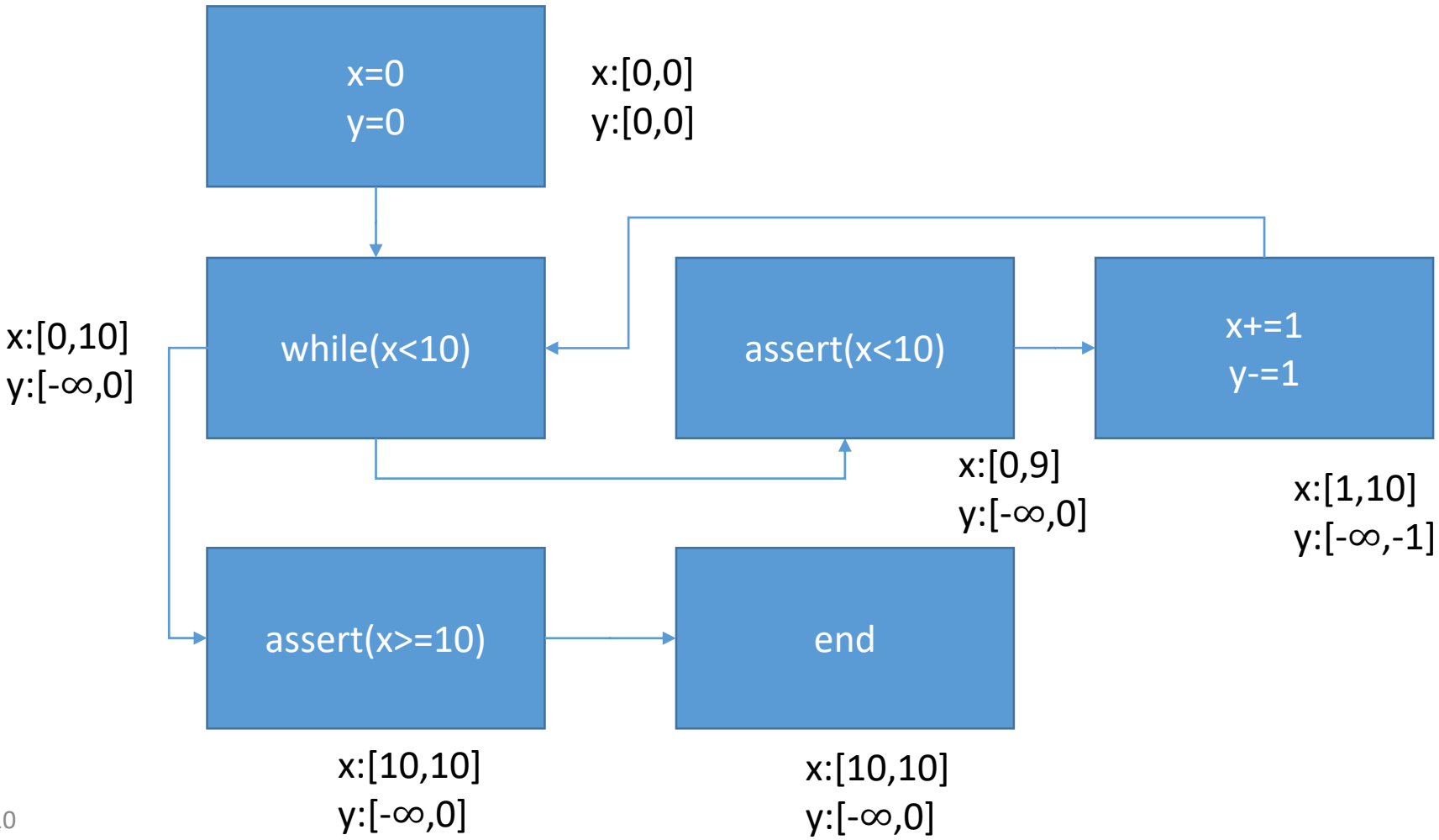
八边形计算举例



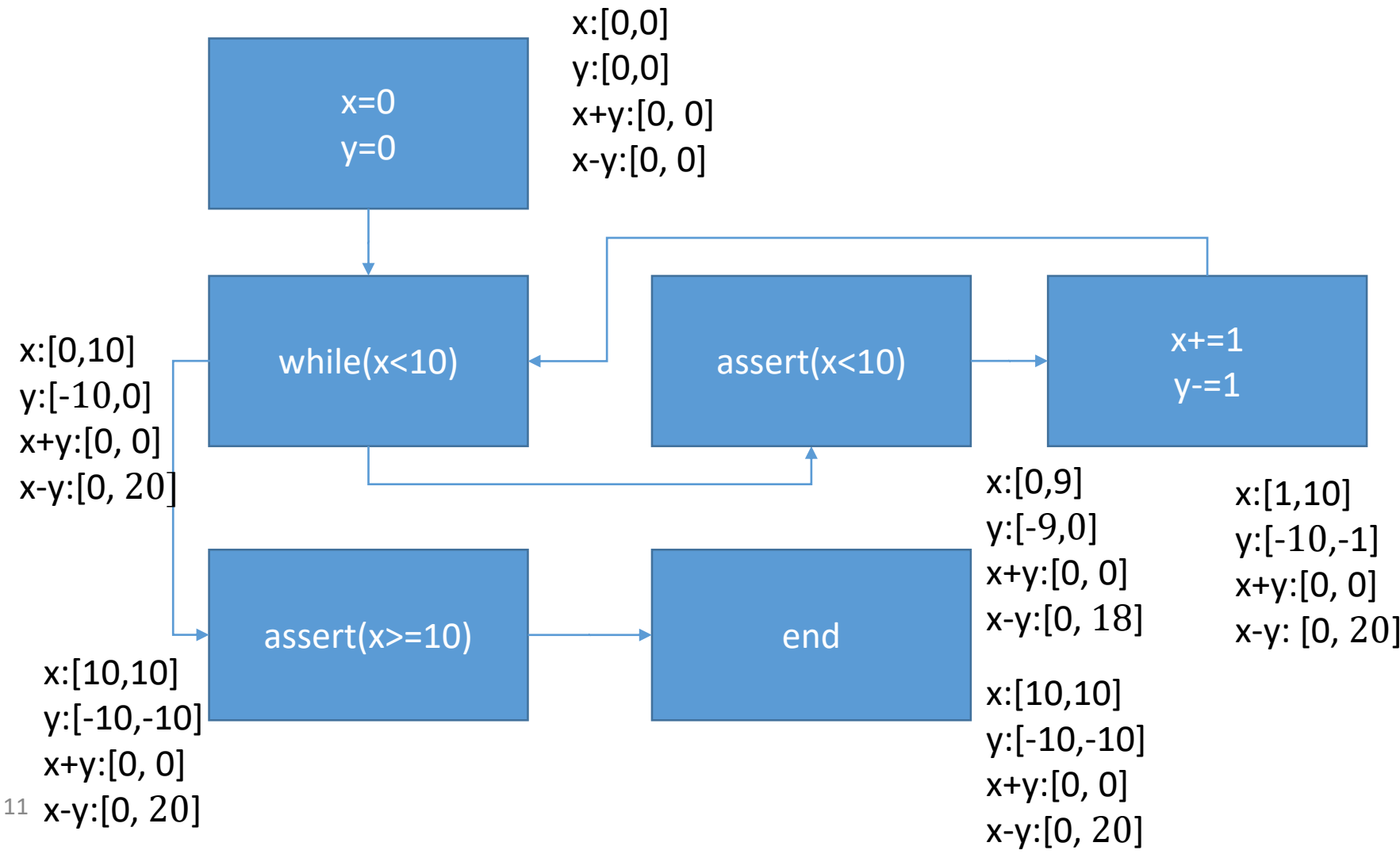
八边形计算举例



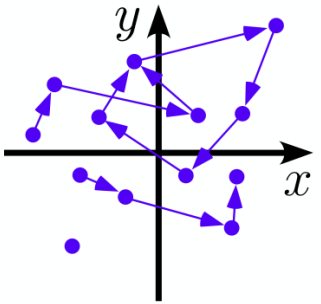
区间计算结果


x=0
y=0
x:[0,0]
y:[0,0]
while(x<10)
x:[0,10]
y:[-∞,0]
assert(x<10)
x:[0,9]
y:[-∞,0]
x+=1
y-=1
x:[1,10]
y:[-∞,-1]
assert(x>=10)
x:[10,10]
y:[-∞,0]
end
x:[10,10]
y:[-∞,0]

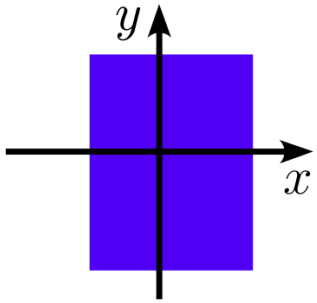
八边形计算结果


x=0
y=0
x:[0,0]
y:[0,0]
x+y:[0, 0]
x-y:[0, 0]
while(x<10)
assert(x<10)
x+=1
y-=1
x:[0,10]
y:[-10,0]
x+y:[0, 0]
x-y:[0, 20]
x:[0,9]
y:[-9,0]
x+y:[0, 0]
x-y:[0, 18]
x:[1,10]
y:[-10,-1]
x+y:[0, 0]
x-y: [0, 20]
assert(x>=10)
end
x:[10,10]
y:[-10,-10]
x+y:[0, 0]
x-y:[0, 20]
x:[10,10]
y:[-10,-10]
x+y:[0, 0]
x-y:[0, 20]

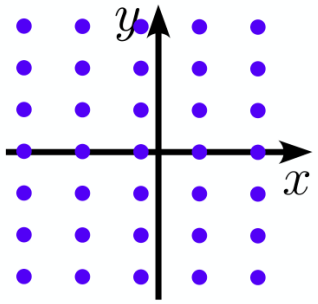
其他数值常用抽象



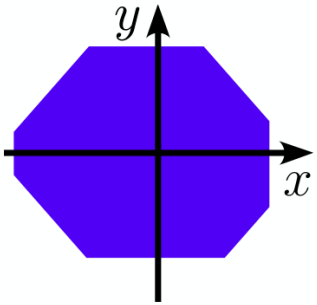
Collecting semantics:
partial traces



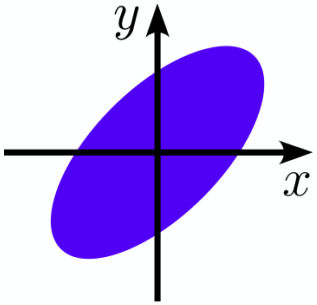
Intervals.
$\mathtt{x} \in [a, b]$



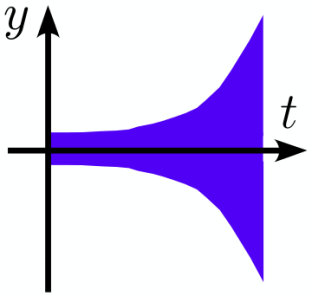
Simple congruences:
$\mathtt{x} \equiv a[b]$



Octagons:
$\pm\mathtt{x} \pm \mathtt{y} \leqslant a$



Ellipses.
$\mathtt{x}^2 + b\mathtt{y}^2 - a\mathtt{xy} \leqslant d$



Exponentials:
$-a^{bt} \leqslant \mathtt{y}(t) \leqslant a^{bt}$

谓词抽象

- 用一系列布尔表达式的值作为抽象域
- 其他很多抽象形式可以看做谓词抽象的一种
- 需要针对谓词设计转换函数
- 如，符号分析可以用谓词抽象表达
  - 对任意变量x，有如下谓词
    - $x > 0, x < 0, x = 0$

在线抽象解释工具

- 示例：Interproc
  - http://pop-art.inrialpes.fr/interproc/interprocweb.cgi
  - 开源工具
    - 用于展示开源抽象域库APRON的静态分析工具
    - 支持整型、浮点型等运算的分析
    - 支持过程间分析（包括递归函数）
    - 不支持数组、结构体等复杂数据结构、也不支持动态内存分配等

抽象解释是非组合式的

- 程序设计语言的语义通常用组合的方式定义
  - $\mu(x * y + y) = \mu(x * y) + \mu(y) = \mu(x) * \mu(y) + \mu(y)$

- 抽象解释的组合会丢失精度
  - $\sigma(x - x) = \sigma(x) - \sigma(x)$
- 假设x为1，则
  - $\sigma(1) =$ 正
  - 正 − 正 =踝
- 但实际上执行x-x的结果恒为0

将表达式作为整体抽象

- 我们希望得到表达式整体最精确的抽象
- $\sigma(x - x) =$ 零

- 如何得到这样的整体抽象？
  - 可能的表达式种类无限，无法一一定义

符号抽象Symbolic Abstraction

- 2004年由Tom Reps等人提出
- 利用SMT Solver的求解能力，自动找到函数的整体精确抽象

抽象域计算问题

- 给定程序和抽象域上的输入，求抽象域上最精确的输出

- 如，给定
  - x=负
  - 求x-x在抽象域上的计算结果
- 答案：零

逻辑与集合

- 任何逻辑表达式定义了一个集合：满足该表达式的值的集合
  - $\varphi: x > 0$
  - 定义了
  - $[\![\varphi]\!]: \{x \mid x > 0\}$
- γ可以写成从抽象值到逻辑表达式的映射
- 子集关系也就对应了逻辑蕴含关系
  - $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!] \Leftrightarrow \varphi_1 \rightarrow \varphi_2$

RSY算法

- Tom Reps等人2004年的论文提出RSY算法
- 假设抽象域和具体域上定义了如下操作和特殊值
  - $\dot{\gamma}$从抽象值到SMT表达式的映射
  - $\dot{\mu}$从程序到SMT表达式的映射
  - β从具体值到最小抽象值的映射，即以下式子成立
    - $x \in \gamma(\beta(x)) \wedge \forall$甲: $(x \in \gamma(\text{甲})) \rightarrow (\gamma(\beta(x)) \subseteq \gamma(\text{甲}))$
    - β为α的特例：$\beta(x) = \alpha(\{x\})$
  - 甲⊔乙：抽象值的并，即以下式子成立
    - $\gamma(\text{甲}) \subseteq \gamma(\text{甲} \sqcup \text{乙})$
    - $\gamma(\text{乙}) \subseteq \gamma(\text{甲} \sqcup \text{乙})$
    - $\forall$丙: $(\gamma(\text{甲}) \subseteq \gamma(\text{丙}) \wedge \gamma(\text{乙}) \subseteq \gamma(\text{丙})) \rightarrow (\gamma(\text{甲} \sqcup \text{乙}) \subseteq \gamma(\text{丙}))$
  - 最小抽象值⊥ 使得$\gamma(\bot) = \emptyset$
- 注意以上操作都是计算机可表示的

RSY算法

- 定理：假设对具体值的任意集合都存在最小抽象。给定具体值的集合S，该集合的最小抽象$\hat{\alpha}(S)$满足
  - $\hat{\alpha}(S) = \sqcup \{\beta(x) \mid x \in S\}$
- 证明：
  - 容易证明$S \subseteq \gamma(\sqcup \{\beta(x) \mid x \in S\})$
  - 接下来用反证法证明$\sqcup \{\beta(x) \mid x \in S\}$是最小抽象
    - 假设存在另一个抽象值甲，满足$S \subseteq \gamma(甲)$，且$\gamma(甲) \subset \gamma(\sqcup \{\beta(x) \mid x \in S\})$
    - 那么一定存在$x \in S$，使得$\neg(\beta(x) \subseteq \gamma(甲))$
    - 即$\beta(x)$不是x的最小抽象，形成矛盾

RSY算法

- 抽象域计算问题：
  - 给定程序p和抽象域上的输入甲，求抽象域上最精确的输出
  - 即：寻找在输入集合γ(甲)下，p的输出集合的最小抽象

- $\hat{\alpha}(S) = \sqcup \{ \beta(x) \mid x \in S \}$

- 基本原理：不断调用SMT Solver寻找S中的元素x，然后将$\beta(x)$并入集合

RSY算法

- 输入：程序p
- 输入：p的抽象输入 甲

result =⊥

While(sat($\dot{\mu}$(p) ∧ $\dot{\gamma}$(甲) ∧ ¬$\dot{\gamma}$(result)))

    y=get-model()

    result=result ⊔ β(y)

return result

示例

- 程序：x-x
- 输入：x=正

- 运行过程：
  - result =⊥，$r = x - x \wedge x > 0 \wedge \neg(\text{false})$可满足，r=0
  - result=零，$r = x - x \wedge x > 0 \wedge \neg(r = 0)$不可满足
  - 程序结束，返回零

示例

- 程序：x+y
- 输入：x=正，y=负

- 运行过程：
  - result =⊥，
    $r = x + y \wedge x > 0 \wedge y < 0 \wedge \neg(\text{false})$可满足，r=0
  - result=零，$r = x + y \wedge x > 0 \wedge y < 0 \wedge \neg(r = 0)$可满足，r=1
  - result=踝，$r = x + y \wedge x > 0 \wedge y < 0 \wedge \neg(true)$不可满足
  - 程序结束，返回踝

从值的抽象到程序的抽象

- RSY算法可以解决抽象域计算问题
- 如何得到程序的整体精确抽象？
  - 方案1：在每次需要在抽象域上根据给定输入执行程序的时候，调用RSY算法
    - 需要反复多次调用SMT求解器，开销较大
  - 方案2：直接采用RSY算法计算程序的抽象

程序的抽象

- 一段程序是从输入到输出的函数
  - 即由输入、输出对组成的集合
- 程序的抽象的记录：可直接记录所有输入对应的输出

f(x)=x+5

f虚

输入	输出
⊥	⊥
正	正
负	零
零	正
叕	叕

f(x,y)=x*y

f虚

	正	负	零	叕	⊥
正	正				
负	负	正			
零	零	零	零		
叕	叕	叕	叕	叕	
⊥	⊥	⊥	⊥	⊥	⊥

程序（函数）抽象的语义

- $\gamma(f^{虚})$为$f^{虚}$上所有输入输出对在具体域上对应的二元组的集合

输入	输出
⊥	⊥
正	正
负	鞅
零	正
鞅	鞅

$$\gamma\left(f^{虚}\right) =$$
$$\gamma(\bot) \times \gamma(\bot) \cup$$
$$\gamma(正) \times \gamma(正) \cup$$
$$\gamma(负) \times \gamma(鞅) \cup$$
$$\gamma(零) \times \gamma(正) \cup$$
$$\gamma(鞅) \times \gamma(鞅)$$

定义RSY算法需要的操作

- 函数抽象合并
  - $\left(f_1^{虚} \sqcup f_2^{虚}\right)(甲) = f_1^{虚}(甲) \sqcup f_2^{虚}(甲)$
  - 即合并对应输入上的输出
- 最小函数抽象
  - $f_{\perp}^{虚}(\_) = \perp$
- β在函数上的扩展
  - $\beta([x,y])(甲) = \begin{cases} \perp, & \neg(x \in \gamma(甲)) \\ \beta(y), & x \in \gamma(甲) \end{cases}$
- $\dot{\gamma}$在函数上的扩展：
  - 依次翻译输入输出对
  - $[正, 负], \ldots$ 翻译为
  - $x > 0 \rightarrow r < 0 \wedge \cdots$

用RSY算法抽象程序

- 输入：程序p

result = $f_{\perp}^{\text{虚}}$

While(sat($\dot{\mu}(\mathrm{p}) \wedge \neg\dot{\gamma}(\mathrm{result})$))

    y=get-model()

    result=result $\sqcup\ \beta(\mathrm{y})$

return result

示例

- 程序：x-x

- 运行过程：
  - result =$f_{\perp}^{\text{虚}}$，
    $r = x - x \wedge \neg(x > 0 \rightarrow false \wedge \cdots)$可满足，[x,r]=[1, 0]
  - result ={[正, 零], [负, ⊥], [零, ⊥], [鞣, 零]},
    $r = x - x \wedge \neg(x > 0 \rightarrow r = 0 \wedge (true \rightarrow r = 0) \ldots)$
    可满足，[x,r]=[-1, 0]
  - result ={[正, 零], [负, 零], [零, ⊥], [鞣, 零]},
    $r = x - x \wedge \neg(\ldots)$可满足，[x,r]=[0, 0]
  - result ={[正, 零], [负, 零], [零, 零], [鞣, 零]},
    $r = x - x \wedge \neg(\ldots)$不可满足

符号抽象问题

- 抽象域计算问题和程序抽象问题可以统一成如下符号抽象问题

- 给定逻辑公式φ，抽象域虚，寻找抽象域中关于公式φ的最精确抽象甲，即满足
  - $[\![\varphi]\!] \subseteq \gamma(\text{甲}) \wedge$
  - $\forall$乙：$[\![\varphi]\!] \subseteq \gamma(\text{乙}) \rightarrow \gamma(\text{甲}) \subseteq \gamma(\text{乙})$

参考资料

- Miné A. The octagon abstract domain[J]. Higher-Order and Symbolic Computation, 2006, 19(1):31-100.
- Thomas W. Reps, Aditya V. Thakur: Automating Abstract Interpretation. VMCAI 2016. 3-40
- MIT抽象解释课程：http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www