



软件分析

# 加宽变窄和抽象解释

熊英飞

北京大学

# 练习：区间 (Interval) 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 $a:[0, +\infty]$



# 区间 (Interval) 分析

- 正向分析
- 有界半格元素：程序中每个变量的区间，最小元为空集
- 合并操作：区间的并
  - $[a, b] \sqcup [c, d] = [\min(a, c), \max(b, d)]$
- 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
  - $[a, b] + [c, d] = \begin{cases} \emptyset & a + c > int\_max \\ (a + c, \min(b + d, int\_max)) & a + c \leq int\_max \end{cases}$
- 原分析终止，但需要`int_max`步才能收敛



# 扩展：加宽和变窄



# 加宽Widening

- 区间分析需要很多步才能达到收敛
  - 格的高度太高
- 加宽：通过降低结果的精度来加快收敛速度
  - 简易加宽：降低格的高度
  - 通用加宽：根据变化趋势快速猜测一个结果



# 简易加宽

- 定义单调函数 $w$ 把结果进一步抽象

- 原始转换函数 $f$
- 新转换函数 $w \circ f$

- 定义有限集合 $B = \{-\infty, 10, 20, 50, 100, +\infty\}$

- 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：

- $w([15, 75]) = [10, 100]$



# 简易加宽的例子

- 令简易加宽的有限集合为 $\{-\infty, 0, 1, 7, +\infty\}$

```
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

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

...

不使用加宽，  
收敛慢或不收敛

$[x \mapsto \perp, y \mapsto \perp]$   
 $[x \mapsto [7, \infty], y \mapsto [0, 0]]$   
 $[x \mapsto [7, \infty], y \mapsto [0, 1]]$   
 $[x \mapsto [7, \infty], y \mapsto [0, 7]]$   
 $[x \mapsto [7, \infty], y \mapsto [0, \infty]]$

使用简易加宽  
收敛快，但不精确





# 简易加宽的安全性

- 如果  $x \sqsubseteq w(x)$ ，则分析结果保证安全
- 安全性讨论
  - 新转换结果大于等于原结果，意味着  $\text{OUT}_v$  的结果大于等于原始结果
  - 利用之前类似的方式可以证明最终分析结果一定大于等于原始结果



# 简易加宽的收敛性

- 如果 $w$ 是单调函数，则简易加宽收敛
  - 因为 $w \circ f$ 仍然是单调函数



# 通用加宽

- 更通用的加宽同时参考更新前和更新后的值来猜测最终会收敛的值
  - 原数据流分析算法更新语句：
    - $OUT_v \leftarrow f_v(IN_v)$
  - 引入加宽算子 $\nabla$ :
    - $OUT_v \leftarrow OUT_v \nabla f_v(IN_v)$
- 通用加宽可以实现更快速的收敛，如
  - $[a, b] \nabla \perp = [a, b]$
  - $\perp \nabla [c, d] = [c, d]$
  - $[a, b] \nabla [c, d] = [m, n]$  where
    - $m = \begin{cases} a & c \geq a \\ -\infty & c < a \end{cases}$
    - $n = \begin{cases} b & d \leq b \\ +\infty & d > b \end{cases}$

解读:  $x \in [a, b]$ 意味着两个约束

- $x \geq a$
- $x \leq b$

该算子本质是去掉循环不保持的约束  
这也是一种设计加宽算子的思路



# 通用加宽的例子

- 令简易加宽的有限集合为 $\{-\infty, 0, 1, 7, +\infty\}$

```
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

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

...

不使用加宽，  
收敛慢或不收敛

$[x \mapsto \perp, y \mapsto \perp]$   
 $[x \mapsto [7, \infty], y \mapsto [0, 0]]$   
 $[x \mapsto [7, \infty], y \mapsto [0, 1]]$   
 $[x \mapsto [7, \infty], y \mapsto [0, 7]]$   
 $[x \mapsto [7, \infty], y \mapsto [0, \infty]]$

使用简易加宽  
收敛快，但不精确

$[x \mapsto \perp, y \mapsto \perp]$   
 $[x \mapsto [8, 8], y \mapsto [0, 0]]$   
 $[x \mapsto [8, 8], y \mapsto [0, \infty]]$

使用通用加宽  
收敛更快，  
结果(恰好)精确



# 通用加宽的安全性

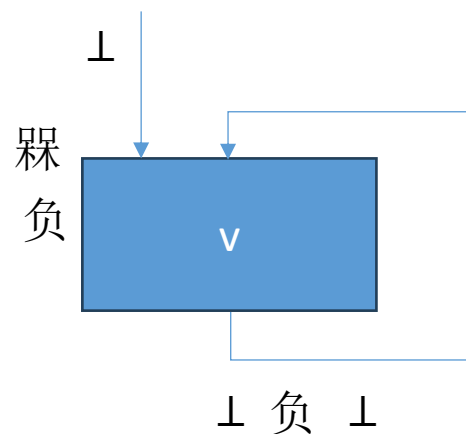
- 如果  $y \sqsubseteq x \nabla y$ ，则通用加宽的分析结果保证安全性
  - 其他教材上还要求  $x \sqsubseteq x \nabla y$ ，但我感觉不需要
- 定理：加上加宽算子之后，如果分析收敛，则分析结果大于等于轮询函数  $F$  的分析结果
  - 定义函数  $G(X) = X \nabla F(X)$
  - 则原始分析是在  $\perp$  上不断应用  $F$ ，而新分析是不断应用  $G$
  - 现在证明对任意  $i$ ， $F^i(\perp) \sqsubseteq G^i(\perp)$ 
    - $i = 0$  时，显然成立
    - 假设  $i = k$  时成立，因为  $F$  的单调性，那么有  $F^{k+1}(\perp) \sqsubseteq F(G^k(\perp))$
    - 又因为  $\nabla$  的性质，我们有  $F(G^k(\perp)) \sqsubseteq G^k(\perp) \nabla F(G^k(\perp)) = G^{k+1}(\perp)$
    - 即  $i = k + 1$  时也成立
  - 给定足够大的  $i$ ，使  $F$  和  $G$  都到不动点，仍有  $F^i(\perp) \sqsubseteq G^i(\perp)$



# 讨论： $x \sqsubseteq x \nabla y$ 的作用？

- 如果没有  $x \sqsubseteq x \nabla y$  的性质，则意味着下一轮的值可以比上一轮更小，可能导致震荡不终止
- 考虑符号抽象域和如下加宽算子

$$\begin{aligned} \bullet \quad x \nabla y &= \begin{cases} \text{罙} & y = \perp \\ y & \text{otherwise} \end{cases} \\ \bullet \quad f_v(\text{甲}) &= \begin{cases} \text{负} & \text{甲} = \text{罙} \\ \perp & \text{otherwise} \end{cases} \end{aligned}$$



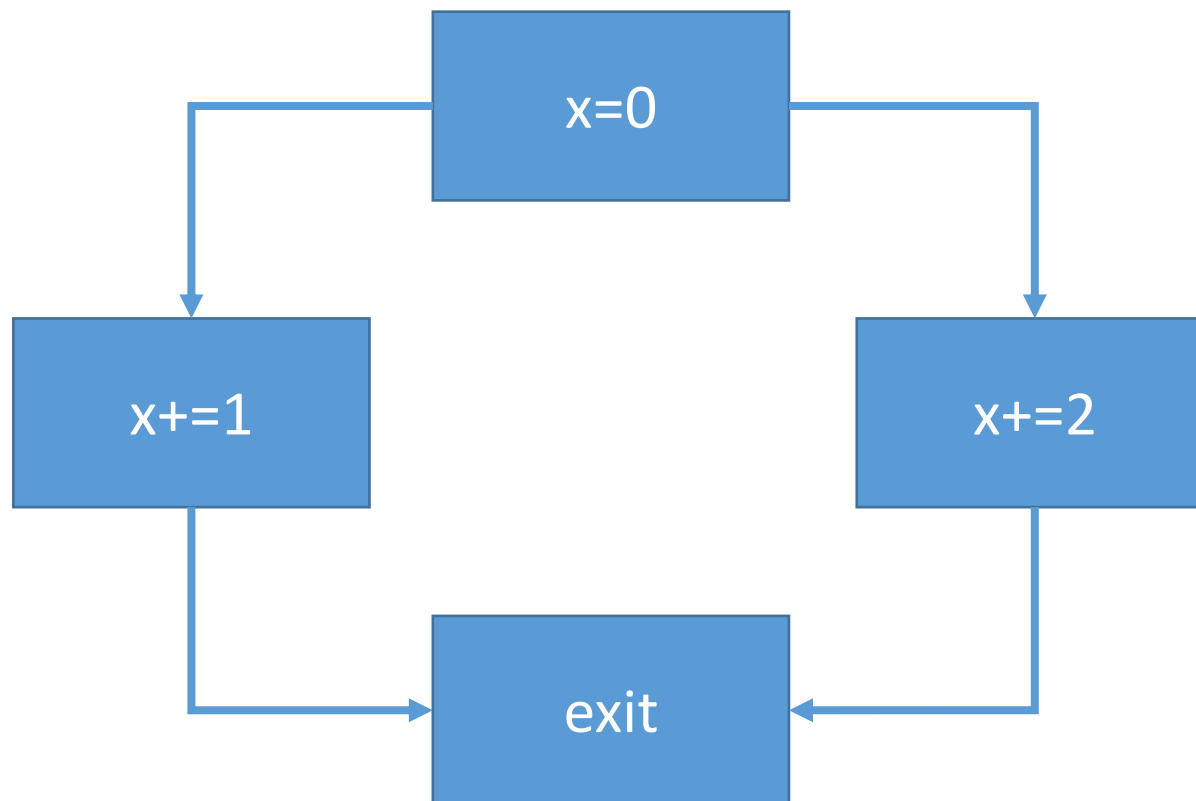


# 通用加宽的收敛性

- 目前没有找到容易判断的属性来证明通用加宽的收敛性
  - 加宽算子本身通常不保证变换函数单调递增
  - $[1,1] \nabla [1,2] = [1,\infty]$
  - $[1,2] \nabla [1,2] = [1,2]$
- 能否给出一个区间分析上不收敛的例子？



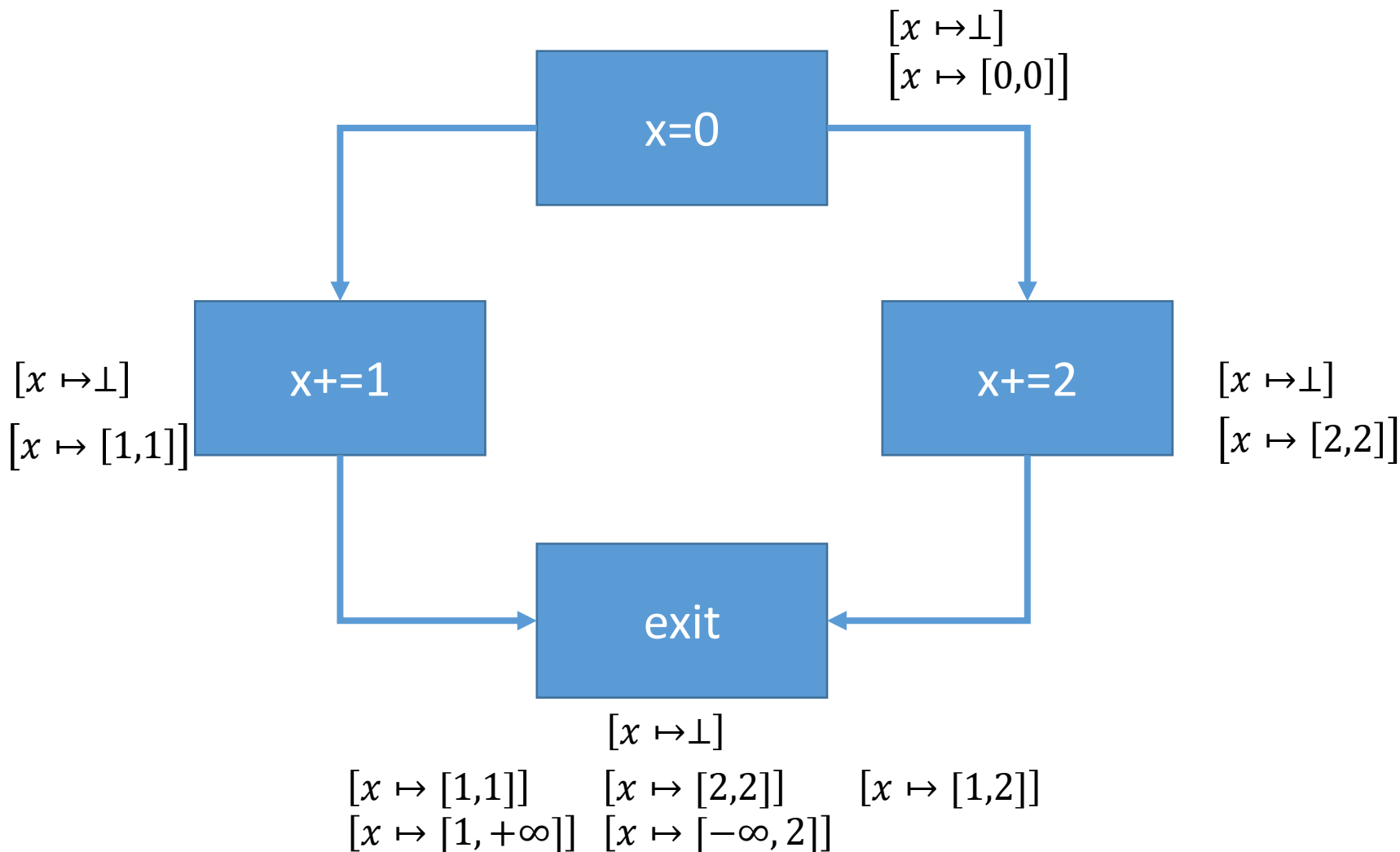
# 加宽不收敛的例子







# 加宽不收敛的例子





# 通用加宽的终止性

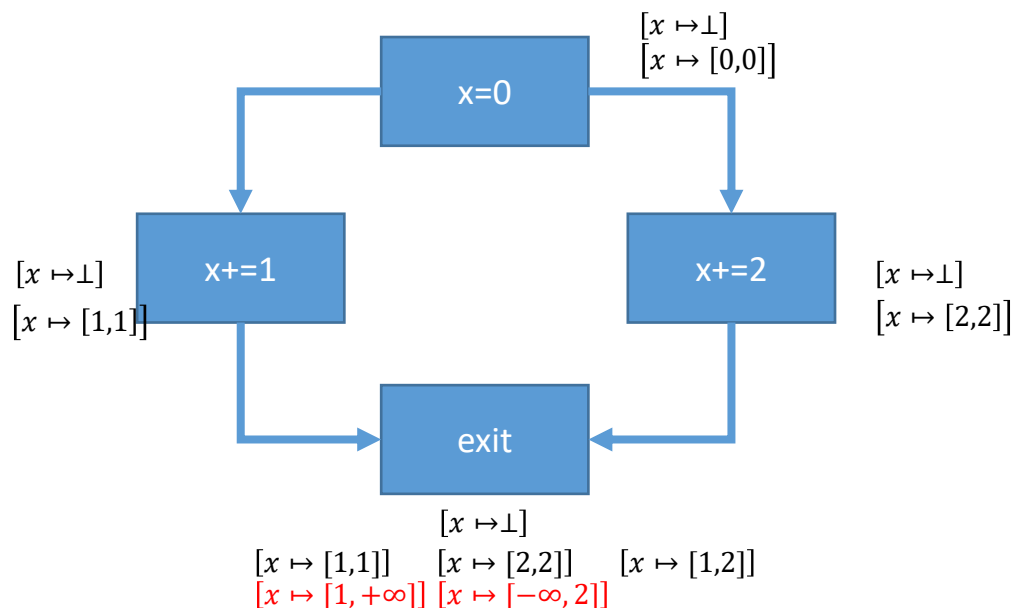
- 相对合流，终止性更加重要
- 但同样，目前也没有找到容易判断的属性来证明通用加宽的终止性
- 多数教材要求下面的属性来保证终止
  - 对任意抽象域上的无穷序列  $x_0, x_1, \dots$
  - 如下序列将在有限长度内稳定，即存在  $k$ ，令  $y_k = y_{k+1}$ 
    - $y_0 = x_0$
    - $y_{k+1} = y_k \nabla x_k$
- 显然该性质保证终止性，但形式和终止性定义一致，要求却强了很多，基本只会增加证明难度



# 如何拯救加宽带来的不精确?

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

$[x \mapsto \perp, y \mapsto \perp]$   
 $[x \mapsto [7, \infty], y \mapsto [0, 0]]$   
 $[x \mapsto [7, \infty], y \mapsto [0, 1]]$   
 $[x \mapsto [7, \infty], y \mapsto [0, 7]]$   
 $[x \mapsto [7, \infty], y \mapsto [0, \infty]]$



# 如何拯救加宽带来的不精确?



- 对于后者，可以只在循环入口点添加加宽算子
- 但识别这样的位置比较麻烦
- 同时，循环入口点仍然可能产生不精确

```
y = 0; x = 7; x = x+1;
while (input) {
    x = 8;
    x = x+1;
    y = y+1;
}
```

采用通用加宽， $x$ 的值可能因为更新顺序不同加宽到 $+\infty$ 或 $-\infty$



# 变窄Narrowing

- 通过再次应用原始转换函数对加宽的结果进行修正

<code>y = 0; x = 7; x = x+1;</code>	$[x \mapsto [7, \infty], y \mapsto [0, 0]]$
<code>while (input) {</code>	$[x \mapsto [7, \infty], y \mapsto [0, \infty]]$
<code>x = 7;</code>	$[x \mapsto [7, 7], y \mapsto [0, \infty]]$
<code>x = x+1;</code>	$[x \mapsto [7, \infty], y \mapsto [0, \infty]]$
<code>y = y+1;</code>	$[x \mapsto [7, \infty], y \mapsto [1, \infty]]$
<code>}</code>	

加宽收敛之后的结果



# 变窄Narrowing

- 通过再次应用原始转换函数对加宽的结果进行修正

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

```
[x ↦ [8,8], y ↦ [0,0]]  
[x ↦ [7, ∞], y ↦ [0, ∞]]  
[x ↦ [7,7], y ↦ [0, ∞]]  
[x ↦ [8,8], y ↦ [0, ∞]]  
[x ↦ [7, ∞], y ↦ [1, ∞]]
```

应用一遍F



# 变窄Narrowing

- 通过再次应用原始转换函数对加宽的结果进行修正

<code>y = 0; x = 7; x = x+1;</code>	$[x \mapsto [8,8], y \mapsto [0,0]]$
<code>while (input) {</code>	$[x \mapsto [7, \infty], y \mapsto [0, \infty]]$
<code>x = 7;</code>	$[x \mapsto [7,7], y \mapsto [0, \infty]]$
<code>x = x+1;</code>	$[x \mapsto [8,8], y \mapsto [0, \infty]]$
<code>y = y+1;</code>	$[x \mapsto [8,8], y \mapsto [1, \infty]]$
<code>}</code>	

应用两遍F



# 变窄Narrowing

- 通过再次应用原始转换函数对加宽的结果进行修正

<code>y = 0; x = 7; x = x+1;</code>	$[x \mapsto [8,8], y \mapsto [0,0]]$
<code>while (input) {</code>	$[x \mapsto [8,8], y \mapsto [0, \infty]]$
<code>x = 7;</code>	$[x \mapsto [7,7], y \mapsto [0, \infty]]$
<code>x = x+1;</code>	$[x \mapsto [8,8], y \mapsto [0, \infty]]$
<code>y = y+1;</code>	$[x \mapsto [8,8], y \mapsto [1, \infty]]$
<code>}</code>	

应用三遍F





# 变窄的安全性

- 针对轮询函数F讨论安全性
- 令
  - 原数据流分析的函数为F，收敛于 $I_F$
  - 经过加宽的函数为G，收敛于 $I_G$
- 那么有
  - 因为  $I_F \sqsubseteq I_G$
  - 所以  $I_F = F(I_F) \sqsubseteq F(I_G) \sqsubseteq G(I_G) = I_G$
  - 即  $I_F \sqsubseteq F(I_G) \sqsubseteq I_G$
- 类似可得
  - $I_F \sqsubseteq F^k(I_G) \sqsubseteq I_G$
- 即变窄保证安全性

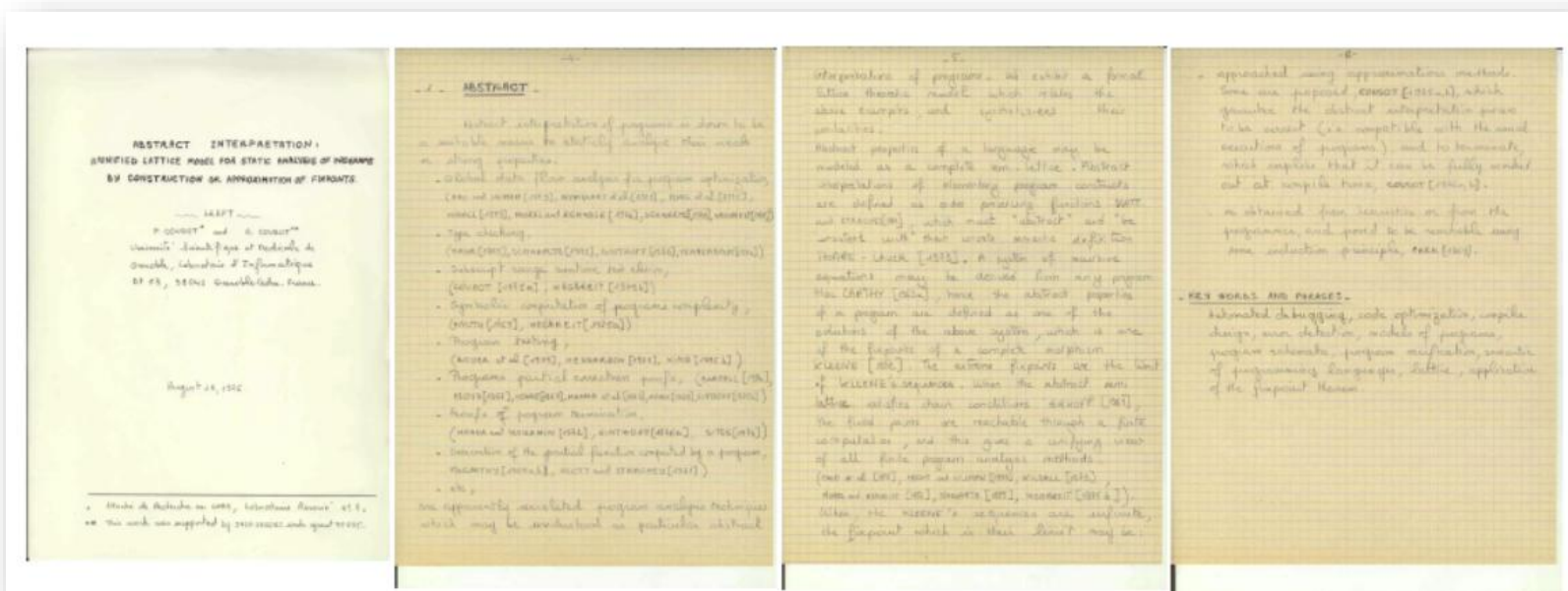


# 变窄的收敛性

- 变窄不保证收敛，也不保证终止
- 通常需要针对应用证明收敛/终止性，或者只应用固定次数 $F$

# 抽象解释


- 最早发表于POPL'77（手写的100页论文）





# 所获荣誉

## 2013年ACM SIGPLAN 程序语言成就奖

**SIGPLAN**  
To explore programming language concepts and tools focusing on design, implementation and efficient use.

Report ProblemContact Us

HomeAwardsConferencesResourcesMembershipSIGPLAN Research HighlightsStudent InformationPublicationsAnnouncement

### Programming Languages Achievement Award

Given by ACM SIGPLAN to recognize an individual or individuals who has made a significant and lasting contribution to the field of programming languages. The contribution can be a single event or a life-time of achievement. The award includes a prize of \$5,000. The award is presented at SIGPLAN's [PLDI conference](#) the following June.

#### Nominations

- [Details of the nomination and award process \(pdf\)](#).
- Please use <http://awards.sigplan.org/> to submit nominations.

### Recipients of the Achievement Award

#### 2013: Patrick and Radhia Cousot

Patrick and Radhia Cousot are the co-inventors of abstract interpretation, a unifying theory of sound abstraction and approximation of structures involved in various domains of computer science, such as formal semantics, specification, proof, and verification. In particular, abstract interpretation has had a major impact on the development of the static analysis of software. In their original work, the Cousots showed how to relate a static analysis to a language's standard semantics by means of a second, abstract semantics that makes precise which features of the full language are being modeled and which are being discarded (or abstracted), providing for the first time both a formal definition of and clear methodology for designing and proving the correctness of static analyses. Subsequently, the Cousots contributed many of the building blocks of abstract interpretation in use today, including chaotic iteration, widening, narrowing, combinations of abstractions, and a number of widely used abstract domains. This work has developed a remarkable set of intellectual tools and has found its way into practice in the form of widely used libraries and frameworks. Finally, the Cousots and their collaborators have contributed to demonstrating the utility of static analysis to society. They led the development of the AstrÉE static analyzer, which is used in the medical, automotive, and aerospace industry for verifying the absence of a large class of common programming errors in low-level embedded systems code. This achievement stands as one of the most substantial successes of program verification to date.



# 所获荣誉

## 2018年 约翰·冯诺依曼奖



### Patrick Cousot awarded John von Neumann Medal

Patrick Cousot is the recipient of the IEEE John von Neumann medal, given "for outstanding achievements in computer-related science and technology".

[Read More](#)



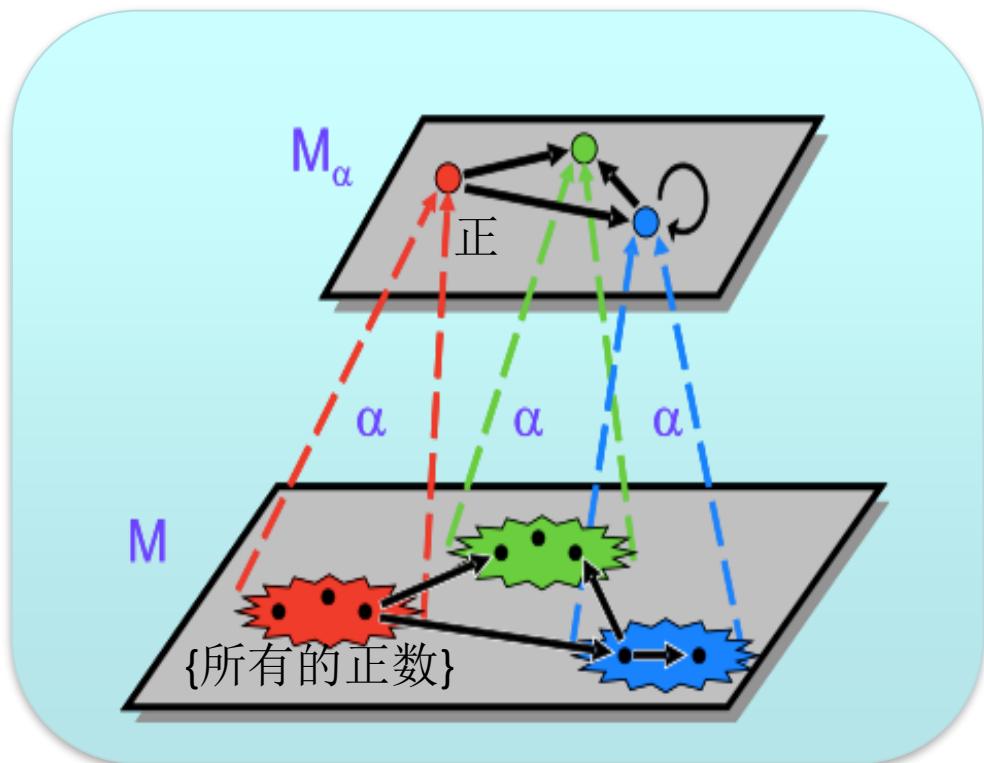
### IEEE JOHN VON NEUMANN MEDAL RECIPIENTS

2018 PATRICK COUSOT  
Professor, New York University,  
New York, New York, USA

"For introducing abstract interpretation, a powerful framework for automatically calculating program properties with broad application to verification and optimization."

# 抽象解释

- 主要解释抽象空间和具体空间的关系



抽象空间

具体空间



# 抽象解释

- 具体化函数  $\gamma$  将抽象值映射为具体值
  - $\gamma(\text{正}) = \{\text{所有的正数}\}$
  - $\gamma(\perp) = \emptyset$
  - 暂时可以把具体值想像成集合
- 抽象化函数  $\alpha$  将具体值映射为抽象值
  - $\alpha(\{\text{所有的正数}\}) = \text{正}$
  - $\alpha(\{1, 2\}) = \text{正}$
  - $\alpha(\{-1, 0\}) = \text{赧}$
- 假设抽象域上存在偏序关系  $\sqsubseteq$



# 伽罗瓦连接

## Galois Connection

- 我们称 $\gamma$ 和 $\alpha$ 构成抽象域**虚**和具体域 **$D$** 之间的一个伽罗瓦连接，记为

$$(D, \sqsubseteq) \sqsubseteq_{\alpha}^{\gamma} (\text{虚}, \sqsubseteq)$$

- 当且仅当

$$\forall X \in D, \text{甲} \in \text{虚}: \alpha(X) \sqsubseteq \text{甲} \Leftrightarrow X \sqsubseteq \gamma(\text{甲})$$





# 定理

- $(D, \subseteq) \sqsubseteq_{\alpha}^{\gamma} (\text{虚}, \sqsubseteq)$  当且仅当以下所有公式成立
- $\alpha$  是单调的:  $\forall X, Y \in D: X \subseteq Y \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$
- $\gamma$  是单调的:  $\forall \text{甲}, \text{乙} \in \text{虚}: \text{甲} \sqsubseteq \text{乙} \Rightarrow \gamma(\text{甲}) \subseteq \gamma(\text{乙})$
- $\gamma \circ \alpha$  保持或增大输入:  $\forall X \in D: X \subseteq \gamma(\alpha(X))$
- $\alpha \circ \gamma$  保持或缩小输入:  $\forall \text{甲} \in \text{虚}: \alpha(\gamma(\text{甲})) \sqsubseteq \text{甲}$



# 证明

- $\Rightarrow$ 
  - $\forall X \in \mathbf{D}: X \subseteq \gamma(\alpha(X))$ 
    - 由  $\alpha(X) \sqsubseteq \alpha(X)$  和伽罗瓦连接定义可得  $X \subseteq \gamma(\alpha(X))$
  - $\forall \text{甲} \in \mathbf{虚}: \alpha(\gamma(\text{甲})) \sqsubseteq \text{甲}$ 
    - 由  $\gamma(\text{甲}) \subseteq \gamma(\text{甲})$  和伽罗瓦连接定义可得  $\alpha(\gamma(\text{甲})) \sqsubseteq \text{甲}$
  - $\forall X, Y \in \mathbf{D}: X \subseteq Y \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$ 
    - $X \subseteq Y \subseteq \gamma(\alpha(Y)) \Rightarrow \alpha(X) \sqsubseteq \alpha(Y)$
  - $\forall \text{甲}, \text{乙} \in \mathbf{虚}: \text{甲} \sqsubseteq \text{乙} \Rightarrow \gamma(\text{甲}) \subseteq \gamma(\text{乙})$ 
    - $\alpha(\gamma(\text{甲})) \sqsubseteq \text{甲} \sqsubseteq \text{乙} \Rightarrow \gamma(\text{甲}) \subseteq \gamma(\text{乙})$



# 证明

•  $\Leftarrow$

•  $\alpha(X) \sqsubseteq \text{甲}$

•  $\Rightarrow \gamma(\alpha(X)) \subseteq \gamma(\text{甲})$       **【 $\gamma$ 的单调性】**

•  $\Rightarrow X \subseteq \gamma(\text{甲})$       **【 $X \subseteq \gamma(\alpha(X))$ 】**

•  $X \subseteq \gamma(\text{甲})$

•  $\Rightarrow \alpha(X) \sqsubseteq \alpha(\gamma(\text{甲}))$       **【 $\alpha$ 的单调性】**

•  $\Rightarrow \alpha(X) \sqsubseteq \text{甲}$       **【 $\alpha(\gamma(\text{甲})) \sqsubseteq \text{甲}$ 】**



# 函数抽象

- 给定伽罗瓦连接  $(D, \sqsubseteq) \sqsubseteq_{\alpha}^{\gamma} (\text{虚}, \sqsubseteq)$
- 给定  $D$  上的函数  $f$  和  $\text{虚}$  上的函数  $\varphi$
- $\varphi$  是  $f$  的安全抽象, 当且仅当
  - $(\alpha \circ f \circ \gamma)(\text{甲}) \sqsubseteq \varphi(\text{甲})$
  - 即  $(f \circ \gamma)(\text{甲}) \sqsubseteq (\gamma \circ \varphi)(\text{甲})$
- $\varphi$  是  $f$  的最佳抽象, 当且仅当
  - $\alpha \circ f \circ \gamma = \varphi$
- $\varphi$  是  $f$  的精确抽象, 当且仅当
  - $f \circ \gamma = \gamma \circ \varphi$
- 最佳抽象总是存在, 但精确抽象不一定存在



# 定义程序分析的安全性

- 执行踪迹（具体执行序列）：(语句编号,内存状态)构成的序列
- 程序分析：分析程序所有执行踪迹集合的属性
  - 符号分析：正常返回的执行踪迹对应变量的符号
  - 可达定值：执行踪迹集合的所有踪迹在某个节点的可达定值的并
  - 可用表达式：执行踪迹集合中所有踪迹在某个节点的可用表达式的交
  - 定义通常包括两部分：1. 单条踪迹的属性 2. 如何从单条的属性得到集合的属性
- 程序分析的安全性：在分析结果域存在某种偏序关系，与理想结果相同或者更大视为安全



# 定义程序分析的安全性

- 程序的执行踪迹集合和分析结果域构成伽罗瓦连接
  - 具体域：执行踪迹集合+集合子集关系
  - **抽象域**：分析结果+分析结果上的偏序关系
  - $\alpha$ ：踪迹集合对应的精确分析结果，定义为
    - $\alpha(X) = \sqcup_{x \in X} \beta(x)$
  - $\gamma(\text{甲}) = \{x \mid \beta(x) \sqsubseteq \text{甲}\}$

红色为特定分析需要定义的部分

- 容易证明上述元素形成伽罗瓦连接
  - $\alpha(X) \sqsubseteq \text{甲} \Rightarrow X \subseteq \gamma(\text{甲})$ ：由 $\gamma$ 定义直接可得
  - $X \subseteq \gamma(\text{甲}) \Rightarrow \alpha(X) \sqsubseteq \text{甲}$ ：两边应用 $\alpha$ ，得 $\alpha(X) \sqsubseteq \alpha(\gamma(\text{甲})) = \sqcup_{\beta(x) \sqsubseteq \text{甲}} \beta(x) \sqsubseteq \text{甲}$ 
    - 最后一步用到下页最小上界定理。给定集合 $\{\beta(x) \mid \beta(x) \sqsubseteq \text{甲}\}$ ，甲是该集合的上界， $\sqcup_{\beta(x) \sqsubseteq \text{甲}} \beta(x)$ 是最小上界。



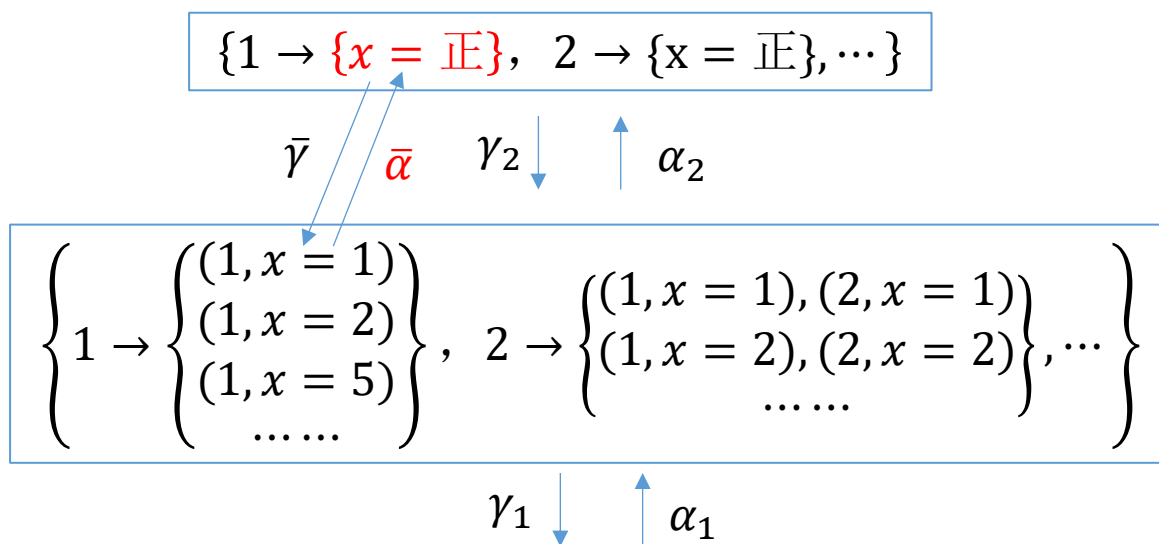
# 集合的最小上界

- 上界：给定集合 $S$ ，如果满足 $\forall s \in S: s \sqsubseteq u$ ，则称 $u$ 是 $S$ 的一个上界
- 最小上界：设 $u$ 是集合 $S$ 的上界，给定任意上界 $u'$ ，如果满足 $u \sqsubseteq u'$ ，则称 $u$ 是 $S$ 的最小上界
- 引理： $\sqcup_{s \in S} s$ 是 $S$ 的最小上界
  - 证明：
    - 根据幂等性、交换性和结合性，我们有 $\forall v \in S: (\sqcup_{s \in S} s) \sqcup v = \sqcup_{s \in S} s$ ，所以 $\sqcup_{s \in S} s$ 是 $S$ 的上界
    - 给定另一个上界 $u$ ，我们有 $\forall s \in S: s \sqcup u = u$ ， $(\sqcup_{s \in S} s) \sqcup u = (\sqcup_{s \in S} (s \sqcup u)) = u$ ，所以 $\sqcup_{s \in S} s$ 是最小上界



# 定义数据流分析的安全性

- 数据流分析在每个程序点产生一个结果，可以看做是下面多个伽罗瓦连接的复合



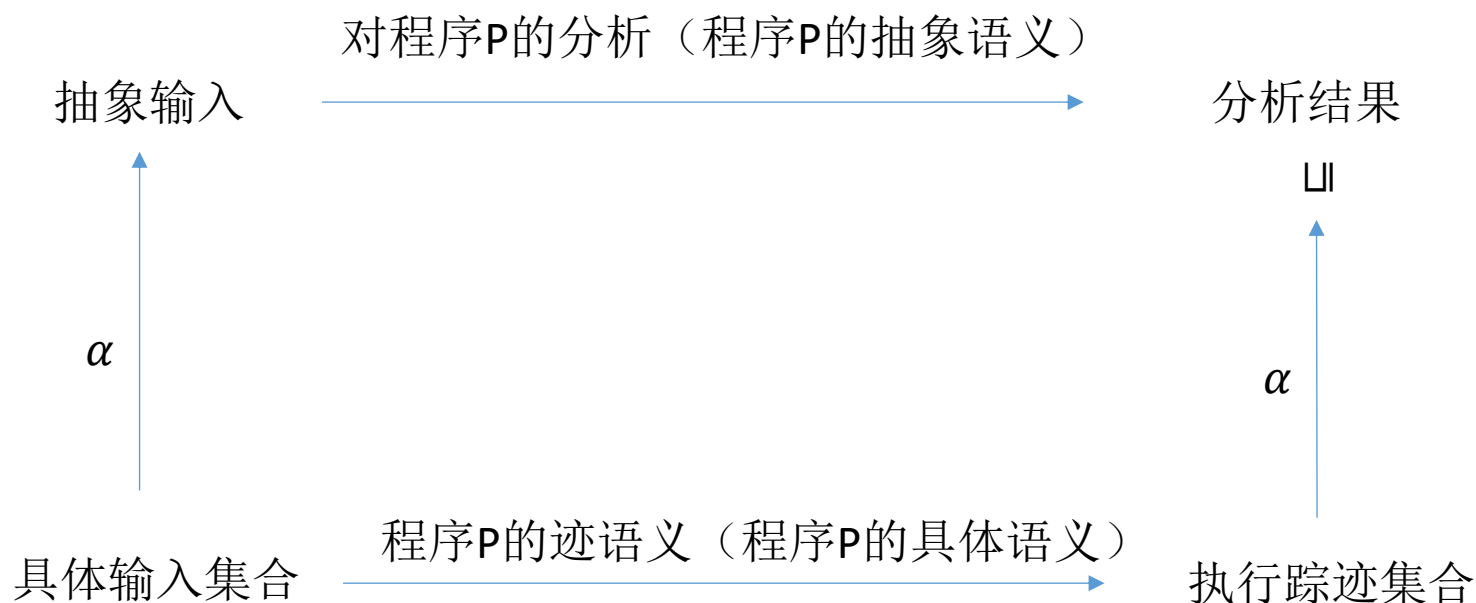
红色为需要定义的部分





# 迹语义 (Trace Semantics)

- 迹语义是一个函数，将程序和输入集合映射为执行踪迹的集合



也就是说，程序分析是对具体语义的一个安全函数抽象



# 控制流图的具体语义

- 假设每个控制流节点 $v$ 对应两个函数
  - 具体（内存状态）转换函数：
    - $trans_v: M \rightarrow 2^M$
  - 控制转移函数：
    - $next_v: M \rightarrow 2^{succ(v)}$
  - 其中 $M$ 为所有内存状态的集合
- 以上函数的返回值都是集合，对应不确定的执行，比如返回随机数
- 考虑反向分析的话，需要反向执行程序，程序的反向执行通常也是不确定的



# 控制流图的具体语义

- 定义如下单步执行函数
  - $Step(T) = \{t + (v', m') \mid$   
 $t \in T$   
 $v' \in next_{last(t).node}(last(t).mem),$   
 $m' \in trans_{v'}(last(t).mem)\} \cup T$
  - 即每次把序列加长一步
- 那么一个程序的迹语义就是 $Step^\infty$ , 定义为
  - $Step^\infty(T) = \lim_{n \rightarrow \infty} Step^n(T)$



# 控制流图的抽象语义

- 复习：之前定义过轮询函数
  - $F(\text{OUT}_{v_1}, \text{OUT}_{v_2}, \dots, \text{OUT}_{v_n}) =$   
 $(f_{v_1}(\sqcup_{w \in \text{pred}(v_1)} \text{OUT}_w),$   
 $f_{v_2}(\sqcup_{w \in \text{pred}(v_2)} \text{OUT}_w),$   
 $\dots,$   
 $f_{v_n}(\sqcup_{w \in \text{pred}(v_n)} \text{OUT}_w))$
- 数据流分析的结果为  $F^\infty(I)$
- 我们现在要证明轮询函数  $F$  是  $\text{Step}$  的安全抽象



# 标准数据流分析的安全性

- 如果任意节点 $v$ 上的抽象域转换函数 $f_v$ 满足如下条件
  - $t \in \text{Step}(\bar{\gamma}(\text{甲})) \wedge \text{last}(t).node = v$   
 $\Rightarrow t \in \bar{\gamma}(f_v(\text{甲}))$
- 那么 $F$ 是Step的安全函数抽象，即
  - $\text{Step}(\gamma(\text{甲})) \subseteq \gamma(F(\text{甲}))$
- 证明：和之前证明类似，考虑顺序、分支、合并等多种不同情况，路径都仍然在转换之后的抽象值中。略



# 小结： 基于抽象解释设计程序分析

- 程序的具体语义：
  - 反复应用某种具体单步执行函数得到程序的踪迹集合
- 程序的抽象语义：
  - 针对问题设计抽象域
  - 设计抽象单步执行函数，是具体执行函数的安全抽象
  - 证明抽象单步执行函数收敛
    - 通常基于单调性+半格高度有限



# 作业:

- 对于下面程序，如果我们在条件分支的地方加上节点根据条件压缩抽象值，采用今天课上讲的加宽算子进行区间分析，每条语句对应的OUT值是什么？如果加上变窄，对应的OUT值是什么？
  1. `x=1;`
  2. `while (x < 100) {`
  3. `x++;}`
  4. `skip;`



# 参考资料

- 《编译原理》 第9章
- Lecture Notes on Static Analysis
  - <https://cs.au.dk/~amoeller/spa/>
- A Gentle Introduction to Abstract Interpretation
  - Patrick Cousot
  - TASE 2015 Keynote speech
- 抽象解释及其在静态分析中的应用
  - 陈立前
  - SWU-RISE Computer Science Tutorial
- 《Introduction to Static Analysis》 Rival and Yi